UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Planning and Scheduling in Temporally Uncertain Domains

Andrea Micheli

Advisor
Dr. Alessandro Cimatti
Head of Embedded Systems Unit,
Fondazione Bruno Kessler, Trento

Co-Advisor
Dr. Marco Roveri
Fondazione Bruno Kessler, Trento

January 2016

Abstract

Any form of model-based reasoning is limited by the adherence of the model to the actual reality. Scheduling is the problem of finding a suitable timing to execute a given set of activities accommodating complex temporal constraints. Planning is the problem of finding a strategy for an agent to achieve a desired goal given a formal model of the system and the environment it is immersed in. When time and temporal constraints are considered, the problem takes the name of temporal planning.

A common assumption in existing techniques for planning and scheduling is controllability of activities: the agent is assumed to be able to control the timing of starting and ending of each activity. In several practical applications, however, the actual timing of actions is not under direct control of the plan executor.

In this thesis, we focus on this temporal uncertainty issue in scheduling and in temporal planning: we propose to natively express temporal uncertainty in the model used for reasoning. We first analyze the state-of-the-art on the subject, presenting a rationalization of existing works. Second, we show how Satisfiability Modulo Theory (SMT) solvers can be exploited to quickly solve different kinds of query in the realm of scheduling under uncertainty. Finally, we address the problem of temporal planning in domains featuring real-time constraints and actions having duration that is not under the control of the planning agent.

Keywords
[Temporal Planning with Uncontrollable Durations, Scheduling under Uncertainty, Temporal Problems with Uncertainty, Temporal Uncertainty, Satisfiability Modulo Theory]

Acknowledgments

Many people deserve my gratitude for their support and their contribution to this thesis.
First of all, I want to thank my advisor, Alessandro Cimatti, and my co-advisor, Marco Roveri, for the help, the guidance and the support they always provided me during the PhD. They taught me what good research is and how to do it, how to approach a problem and how to think critically. I will never thank them enough for these lessons and their friendship.
Then, I would like to thank David E. Smith for giving me the possibility of working in a thrilling environment such as NASA Ames for six months; it has been a wonderful and enriching experience. I am deeply thankful to him, to Minh Do and to Jeremy Frank for all the interesting discussions and the inputs they provided.
I have to thank all my close friends from the Embedded Systems Unit in FBK, their support and their glee brightened my days: Marco Gario, Alessandro Mariotti, Cristian Mattarei, Sergio Mover and Gianni Zampedri.
Thanks to Daniela, Jessica, Marta, Matteo and Samuel for being my closest friends and always encouraging me and listening to me during our Saturday nights.
Finally, this thesis is dedicated to my family: I am who I am thanks only to their example and love. To my mother for her everyday courage, to my father for his wisdom, to my brother for his jokes and glee and to my girlfriend Silvia for her love.

Ringraziamenti

Devo ringraziare molte persone per il loro supporto e il loro contributo a questa tesi.
Prima di tutto, voglio ringraziare il mio supervisore, Alessandro Cimatti, e il mio co-supervisore, Marco Roveri, per l'aiuto, la guida e il supporto che mi hanno sempre dato durante questo periodo di dottorato. Mi hanno insegnato cos'è e come si fa la buona ricerca, come approcciare un problema e come pensare criticamente. Non li ringrazieró mai abbastanza per queste lezioni e per la loro amicizia.
Vorrei inoltre ringraziare David E. Smith per avermi dato la possibilitá di lavorare in un ambiente elettrizzante come NASA Ames per sei mesi; è stato un periodo meraviglioso e un esperienza arricchente. Sono profondamente grato a lui, a Minh Do e a Jeremy Frank per tutte le interessanti discussioni e gli stimoli che mi hanno dato.
Ringrazio tutti i miei amici dell' unitá di Embedded Systems in FBK, il loro supporto e la loro allegria hanno illuminato le mie giornate: Marco Gario, Alessandro Mariotti, Cristian Mattarei, Sergio Mover e Gianni Zampedri.
Grazie a Daniela, Jessica, Marta, Matteo e Samuel per essere i miei migliori amici e per incoraggiarmi ed ascoltarmi sempre durante le nostre serate.
Infine, questa tesi è dedicata alla mia famiglia: io sono chi sono solo grazie al loro esempio e al loro amore. A mia madre per il suo coraggio che dimostra ogni giorno, a mio padre per la sua saggezza, a mio fratello per i suoi scherzi e la sua allegria e a Silvia per il suo affetto.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Model-based reasoning is a powerful and well-developed area of Artificial Intelligence that is concerned with automatic techniques for synthesizing some knowledge from a formal model of the reality. AI Planning is one of the oldest and most studied forms of model-based reasoning: given a mathematical model of the initial world state and of the applicable actions to change it, the objective is to synthesize a strategy (a plan), to drive the system and the world in a desired goal state. Planning is arguably one of the most fundamental forms of intelligence: devising strategies to achieve desired goals is a basic behavior common to all intelligent forms.

Model-based techniques are very powerful, because complex knowledge can be elaborated from very simple models, but there is an intrinsic limitation: the quality of the result for a real-world situation is always limited by the adherence of the model to the reality. In fact, formal models are always abstractions of the real world, and such an abstraction can be more or less coarse. Clearly, the more fine-grained and detailed a model is, the more complex and intractable the reasoning problem becomes.

The focus of this thesis is on temporal planning and scheduling techniques, that are AI planning algorithms that can deal with models having real-time constraints (such as synchronizations and deadlines) and that consider the precise timing of the activities. A lot of research has been devoted to these issues over the years, but many planning systems still assume that the duration of each activity as well as all the constraints are under control of the planning system that can freely schedule its activities, without any uncertainty.

In real world situations, however, the duration of an activity is not always *controllable*. For example, the duration of a car trip from San Francisco to Los Angeles is not under the complete control of the driver, because it also depends on the traffic and the weather conditions. In this kind of situation, different approaches are possible for planning. One possibility is to disregard uncertainty, estimating beforehand the duration of the trip; if, during execution, the trip is taking longer or shorter than expected, a new plan needs to be generated. Another idea is to come up with a plan that does not commit to a specific duration of the trip, but tries to be as general as possible in either a formal or a best-effort way. In this thesis, instead, we focus on a dedicated modeling of the uncertainty in the trip. We explicitly model the trip as an activity that has *uncontrollable* duration, assuming that minimal and maximal bounds for the trip duration are given. This amounts to decide a strategy for the journey that is guaranteed to achieve the goal regardless of the traffic conditions, assuming that the duration of the trip will stay in the modeled bounds. Differently from other works, we are not interested in the probability distribution of the trip duration, as we want to provide a plan that is guaranteed to work for every possible duration of the trip within the given bounds, not to maximize probabilistic expectation.

Dealing with temporal uncertainty, guaranteeing that the synthesized plan is valid for every possible concrete execution of uncontrollable entities, is useful in various applications domains, in particular where safety of the controlled system is a primary concern and where no fail-safe condition exist. In particular, this kind of issues have been studied for space applications such as satellites and exploration rovers.

The thesis addresses the problem of temporal uncertainty from a scheduling and a planning point of view. We consider scheduling as a special form of a planning problem in which the set of actions to be executed is known a-priori and only the timing needs to be synthesized. On the contrary, planning in its generality is concerned with the problem of synthesizing both the actions needed to achieve a desired goal and their timing. Throughout the thesis we consider techniques that deal with a continuous and dense model of time: durations and time instants are modeled as real numbers assuming no quantization of time, unless explicitly specified.

1.1 Contributions and Publications

This thesis contributes the state-of-the-art in different directions. We devise efficient techniques to schedule activities in presence of temporal uncertainty as well as planning techniques for temporal planning with uncertain durations. In this section, we give a high-level overview of the thesis contributions and for each of them we list our peer-reviewed publications that are relevant for the presented work.

1. We survey the relevant state-of-the-art, proposing a novel categorization schema for the different approaches. At the time of writing, this part of the thesis is unpublished.

2. We approach the problem of strong controllability for disjunctive temporal networks with uncertainty and we propose a set of novel encodings of the problems in the framework of Satisfiability Modulo Theory. We empirically show that these encodings can effectively solve the problem faster than dedicated techniques, leveraging recent ad-

vances in the SMT technology. This part of the work has been initially published in [CMR12a] and an extended journal version appears in [CMR14].

3. We tackle the weak controllability problem for disjunctive temporal networks with uncertainty. We propose a novel characterization of the notion of strategy for a given network and we describe a number of algorithms for strategy synthesis that exploit SMT solvers for quantitative reasoning. We also demonstrate encodings for deciding if a given network is weakly controllable. This has been initially published in [CMR12b] and an extended journal version appears in [CMR15a].

4. We discuss the open problem of dynamic controllability for disjunctive temporal networks with uncertainty. We show a reduction from the dynamic controllability problem of temporal networks to a reachability game in a Timed Game Automaton. Since complete algorithms exist for the latter problem, we obtain the first sound and complete dynamic controllability algorithm for disjunctive temporal networks with uncertainty as well as several other kinds of networks that the reduction can accommodate. This research line has been presented in [CHMR14] and in [CHM$^+$14]. Moreover, an extended journal paper appears in [CHM$^+$16]: we consider non-determinism in disjunctive temporal networks and propose an extension of the Timed Game Automata reduction to solve the dynamic controllability problem. We also present native algorithms for the validation and the synthesis of dynamic strategies that appear in [CMR16].

5. In the planning context, we deal with the problem of Strong Temporal Planning with Uncontrollable Durations, proposing a portfolio of techniques to deal with the landscape of sub-classes of the problem. First, we extend an existing temporal planner to deal with temporal

uncertainty, leveraging our previous contribution on the strong controllability of disjunctive temporal networks with uncertainty. This research has been published in [CMR15b]. Then, we present a general translation from a very expressive language with uncertainty in the duration of actions to a temporal planning problem without uncertainty. The compilation is such that each plan in the target model corresponds to a strong plan and vice-versa. This research appears in [MDS15].

6. Finally, we focus on the strong temporal planning problem for timeline-based planning, providing a novel formalization of the problem and a theoretical formalization of a bounded-horizon solution. This formalization is described in [CMR13].

7. As a side-product of the work on this thesis, we contributed in the definition of the ANuML language, that extends the syntax and semantics of the Action Notation Modeling Language (ANML) to express several kinds of uncertainty. This extension is currently part of the official draft of the ANML manual.

1.2 Experimental Evaluations

Throughout the thesis, we present several empirical evaluations of the proposed techniques and approaches. In order to allow for experiment replication and for future comparison, all the experimental data is available. All the software we developed and the benchmark instances are available online at `http://www.mikand.net/thesis`.

Almost all the tools we developed are written in the Python programming language and leverage the use of SMT solvers. For this reason, we started an open-source project with some colleagues to develop an effective library in Python for simplifying the development of SMT-based programs. The project, currently running and used by many people around the world, is called PySMT [GM15] and can be found at `http://www.pysmt.org`. The tools we developed for this thesis are based on this library.

1.3 Structure of the Thesis

The thesis is structured as follows. The next chapter introduces the logical notations and the notions needed to understand the rest of the thesis. The thesis is divided in three Parts, each dealing with one aspect of the general problem at hand.

In part I, we survey and organize the state-of-the-art in scheduling and in planning with a focus on temporal uncertainty. We propose a general model that theoretically subsumes and categorizes all the analyzed works.

In part II, we formally introduce the family of temporal networks that have been developed to model and reason about temporal uncertainty when the set of activities to be carried on is bounded and known a-priori. We then propose several SMT encodings to efficiently solve the strong controllability problem for disjunctive temporal networks, several SMT-based algorithms to solve the weak controllability problem. Finally, we propose a new, general framework that accommodates a wide range of temporal networks and we present a reduction from the dynamic controllability for such networks to Timed Game Automata (TGA). Thanks to this reduction we can solve a previously open problems and synthesize a dynamic strategy in a closed-form.

In part III, we analyze the problem of planning in presence of temporal uncertainty. We first present a formal planning language to express temporal uncertainty in the duration of the actions: we develop a number of techniques to address the problem of Strong Temporal Planning with

Uncontrollable Durations. We experimentally evaluate the merits of each technique on a number of benchmark problems. Then, we apply a similar idea to the world of timeline-based planning, proposing a formalization of the problem in first order logic.

Finally, in chapter 14 we conclude the thesis drawing our conclusions and highlighting directions for future work.

Chapter 2

Background

In this chapter, we introduce the basic notation and notions that are used throughout the rest of the thesis. In section 2.1 we introduce a basic first-order logic language and notation, while in section 2.2 we present a quick overview of the modern Satisfiability Modulo Theory framework. Then, in section 2.3 we introduce the Timed Game Automata framework and, finally, in section 2.4 we discuss ways to canonically represent and manipulate time regions using Difference Bound Matrices and Federations.

2.1 Technical Preliminaries

Our setting is standard first order logic [Kle67]. The first-order signature is composed of constants, variables, function symbols, Boolean variables, and predicate symbols. A term is either a constant, a variable, or the application of a function symbol of arity n to n terms. A theory constraint (also called a theory atom) is the application of a predicate symbol of arity n to n terms. An atom is either a theory constraint or a Boolean variable. A literal is either an atom or its negation. A clause is a finite disjunction of literals. A formula is either true ($\top$), false ($\bot$), a Boolean variable, a theory constraint, the application of a propositional connective ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) of arity n to n formulae, or the application of a quantifier ($\forall$, $\exists$) to an individual variable and a formula. If t_1 and t_2 are terms, and ϕ is a formula, an if-then-else (ITE) term is $ite(\phi, t_1, t_2)$. The semantics of an ITE term is the usual if-then-else semantics from programming languages. For example, the term $ite(x > y, x, y)$ where x and y are numeric variables, corresponds to the maximum between x and y. An ITE term $ite(\phi, t_1, t_2)$ occurring in a formula ψ can be rewritten by substituting each occurrence with a fresh variable v and by conjoining $(\neg\phi \vee (v = t_1)) \wedge (\phi \vee (v = t_2))$. See [KSJ09] for a thorough discussion. We use $x, y, v, \ldots$ for variables, and $\vec{x}, \vec{y}, \vec{v}, \ldots$ for vectors of individual variables. Terms and formulae are referred to as expressions. Formulae are denoted with Greek letters: $\phi, \psi, \ldots$. Let $\vec{x}$ be a vector of variables, we indicate the i-th variable in the vector with x_i. We write $\phi(x)$ to highlight the fact that x occurs in ϕ, and $\phi(\vec{x})$ to highlight the fact that the free variables of ϕ are variables in $\vec{x}$. We indicate with $Q\vec{x}.\phi(\vec{x})$ the formula $Qx_1.Qx_2.\ldots Qx_n.\phi(x_1, \ldots, x_n)$, where $Q \in \{\forall, \exists\}$.

Let $\phi(\vec{x}) \doteq \bigwedge_i \phi_i(\vec{x}_i)$ be a conjunction of formulae. We write $\phi(\vec{x})|_{\vec{y}}$ to represent the conjunction of the $\phi_i(\vec{x}_i)$ in which at least one variable of $\vec{y}$ occurs in $\vec{x}_i$.

Substitution is defined in the standard way [Kle67]. We write $\phi[s/t]$ for the substitution of every occurrence of term t in ϕ with term s. Let $\vec{t}$ and $\vec{s}$ be vectors of terms having the same length, we write $\phi[\vec{s}/\vec{t}]$ for the parallel substitution of every occurrence of t_i in ϕ with s_i.

We use the standard semantic notions of interpretation and satisfiability. We call *satisfying assignment* or *model* of a formula $\phi(\vec{x})$ a total function μ that assigns to each x_i an element of its domain such that the formula $\phi[\mu(\vec{x})/\vec{x}]$ evaluates to $\top$ provided an interpretation of function symbols. A formula $\phi(\vec{x})$ is *satisfiable* if and only if it has a satisfying assignment and an interpretation.

Following standard naming, we say that a formula is in Conjunctive Normal Form (CNF) if it is expressed as a conjunction of disjunctions of atoms or negations of atoms: $\bigwedge_{i=1}^{h} \bigvee_{j=1}^{k} l$, l being an atom or the negation of an atom (usually called literal). Each disjunction in a CNF formula is called a clause. It is well known that each formula can be reduced to an equi-satisfiable CNF formula of linear size [dlT90].

Another important normal form is the Disjunctive Normal Form (DNF): a formula is in DNF if it is expressed as a disjunction of conjunctions of literals: $\bigvee_{i=1}^{h} \bigwedge_{j=1}^{k} l$, l being an atom or the negation of an atom. Each formula can be reduced to a DNF formula; however, the size of the DNF formula is in general exponential.

Finally, we consider the Negation Normal Form (NFF). A formula is in NNF if the negation operator ($\neg$) is only applied to atoms, and the only other Boolean operators present in the formula are conjunctions and disjunctions. Any quantifier-free formula can be reduced to an equivalent linear-size NNF formula.

2.2 Satisfiability Modulo Theories

Given a formula ϕ, satisfiability is the problem of finding a satisfying assignment for ϕ and an interpretation for the functional symbols in ϕ. This problem is approached in propositional logic with enhancements of the DPLL algorithm [DLL62]: the formula is converted into an equi-satisfiable one in Conjunctive Normal Form (CNF); then, a satisfying assignment is incrementally built, until either all the clauses are satisfied, or a conflict is found, in which case back-jumping takes place (i.e. certain assignments are undone). Keys to efficiency are heuristics for the variable selection, and learning of conflicts [MMZ$^+$01].

Given a first-order formula ψ expressed in a decidable background theory T, *Satisfiability Modulo Theory* (SMT) [BSST09] is the problem of deciding whether ψ is satisfiable. For example, consider the formula $(x \leq$

$y) \wedge ((x+3 = z) \vee (z \geq y))$ in the theory of real arithmetic ($x, y, z \in \mathbb{R}$, and the symbols $\leq$, $+$, $=$ and $\geq$ are interpreted in the usual way). The formula is satisfiable and a satisfying assignment is $\{x := 5,\ y := 6,\ z := 8\}$. The theory of real arithmetic interprets "3" as the real number 3 and $+, =, <, >, \leq, \geq$ as the usual mathematical functions and relations.

There exist several theories of practical interests: *Equality and Uninterpreted Functions*, *Linear Arithmetic* over the Reals and the Integers, *Non-Linear Arithmetic*, *Difference Logic*, *Bit Vectors*, *Arrays* and others. In this thesis, we concentrate on the theory of linear arithmetic over the real numbers ($\mathcal{LRA}$) because it offers a natural and convenient way to model continuous, dense time. A formula in $\mathcal{LRA}$ is an arbitrary Boolean combination, a universal ($\forall$) or an existential ($\exists$) quantification, of atoms in the form $\sum_i a_i x_i \bowtie c$ where $\bowtie \in \{>, <, \leq, \geq, \neq, =\}$, every x_i is a real variable, every a_i is a real constant and c is also a real constant. For brevity, given two real constants l, u such that $l \leq u$, we denote with $t \in [l, u]$ the formula $l \leq t \wedge t \leq u$. With a slight abuse of notation, we allow l or u to be $-\infty$ or $+\infty$, respectively. The semantics is obtained by simply removing the constraint containing ∞ as it is tautological. For example, $t \in [2, \infty]$ simply becomes $t \geq 2$; $t \in [-\infty, 5]$ simply becomes $t \leq 5$ and $t \in [-\infty, \infty]$ becomes $\top$.

Real Difference logic ($\mathcal{RDL}$) is the fragment of $\mathcal{LRA}$ where all the atoms have the form $x_i - x_j \bowtie c$. We denote with $\mathcal{QF\_LRA}$ and $\mathcal{QF\_RDL}$ the quantifier-free fragments of $\mathcal{LRA}$ and $\mathcal{RDL}$, respectively.

An SMT solver [BSST09] is a decision procedure which solves the satisfiability problem for a formula expressed in a decidable subset of First-Order Logic. The most efficient implementations of SMT solvers use the so-called "lazy approach", where a SAT solver is tightly integrated with a T-solver, that is demanded to decide conjunction of constraints in the theory T. The role of the SAT solver is to enumerate the truth assignments to the

Boolean abstraction of the first-order formula. The Boolean abstraction has the same Boolean structure of the first-order formula, but "replaces" the predicates which contain theory information with fresh Boolean variables. The Boolean abstraction of $(x \leq y) \wedge ((x + 3 = z) \vee (z \geq y))$ is $a \wedge (b \vee c)$, where a, b, c are fresh Boolean variables. The T-solver is invoked when the SAT solver finds a satisfying assignment for the Boolean abstraction: the satisfying assignment to Boolean abstraction maps directly to a conjunction of T atoms, which the T-solver can handle. If the conjunction is satisfiable also the original formula is satisfiable. Otherwise the T-solver returns a conflict set which identifies a reason for the unsatisfiability. Then, the negation of the conflict set is learned by the SAT solver in order to prune the search. Examples of solvers based on the "lazy approach" are MathSAT [BCF$^+$08, CGSS13], Z3 [dMB08], Yices [DdM06b] and OpenSMT [BPST10]).

2.2.1 SMT Notation

In this thesis, we present algorithms that use different features provided by modern SMT solvers, such as optimization [ST12]. In the algorithm pseudcode, we indicate with the prefix "SMT." the functions that are related with SMT solving. In particular, the function SMT.declareRealVar(v) declares an SMT variable named v of real type. SMT.solve($\phi(\vec{x})$) is a function that checks the satisfiability of the formula $\phi(\vec{x})$ and returns "SAT" if and only if the formula is satisfiable, otherwise the function returns "UNSAT". SMT.getModel() returns a satisfying assignment to the formula that was checked using SMT.solve if the answer was "SAT". Finally, the function SMT.solveMaximizing($\phi(\vec{x})$, $h(\vec{x})$) behaves like SMT.solve($\phi(\vec{x})$) but generates the model that maximizes the evaluation of the function $h(\vec{x})$.

In an incremental setting [CGSS13, dMB08], we assume a stateful SMT solver that has the following capabilities. SMT.ASSERT($\phi(\vec{x})$) conjoins the formula $\phi(\vec{x})$ to the state of the SMT solver, without performing any solving operation. SMT.PUSH() records a backtrack point in the sate of the SMT solver in a stack. The last recorded state can be restored by calling the SMT.POP() function. Finally, when using incrementality we assume that the SMT.SOLVE function can be called without arguments to check the satisfiability of the conjunction of the currently asserted set of formulae.

2.2.2 Quantifiers in $\mathcal{LRA}$

Traditionally, SMT solvers have been focused on solving quantifier-free formulae, but recently techniques to deal with quantifiers on selected theories have been developed and implemented. For the sake of this thesis we are interested in dealing with quantifiers in the $\mathcal{LRA}$ theory. Some solvers (e.g. Z3) natively support $\mathcal{LRA}$ quantifiers, but others (e.g. MathSAT) are still limited to the quantifier-free fragment.

The $\mathcal{LRA}$ theory admits quantifier elimination[1], this means that from any $\mathcal{LRA}$ formula it is possible to derive a *logically-equivalent*, quantifier-free $\mathcal{LRA}$ formula (in $\mathcal{QF\_LRA}$).

Several automated techniques for quantifier elimination exist for the $\mathcal{LRA}$ theory: Fourier-Motzkin [Sch98] and Loos-Weispfenning [LW93] are two well-known examples that we will use in this thesis.

Fourier-Motzkin Elimination

In order to decide the satisfiability of a conjunction of constraints in linear real arithmetic, we can apply the *Fourier-Motzkin elimination* (FME) tech-

[1]A theory T is said to admit quantifier elimination, if for every quantified formula ϕ in T, there exist a quantifier-free formula ϕ' that is logically equivalent to ϕ. It has been proven that $\mathcal{LRA}$ admits quantifier elimination [Sch98].

nique. The method was discovered in 1826 by Fourier and re-discovered by Motzkin in 1936.

Consider the following formula where $\{x_1, ..., x_n\}$ is a set of real variables ($x_i \in \mathbb{R}$) and $a_{i,j}, b_i \in \mathbb{R}$ are real coefficients.

$$\exists x_e. \bigwedge_{i=1}^{m} (\sum_{j=1}^{n} a_{i,j} x_j) \leq b_i$$

The elimination method allows the elimination of the variable x_e and obtain a new system without x_e that is equi-satisfiable with respect to the original one.

The basic principle of the method consists in the projection onto the x_e dimension of the polytope described by the system

$$\bigwedge_{i=1}^{m} (\sum_{j=1}^{n} a_{i,j} x_j) \leq b_i.$$

A detailed description of the method is beyond the scope of this thesis; the interested reader can consult Schrijver's thorough description [Sch98] and Kessler's efficient implementation [Kes96].

If we are given a general formula $\exists x_e.\phi(\vec{X})$, we can naïvely apply this technique by computing the Disjunctive Normal Form of $\phi(\vec{X})$ and apply the Fourier-Motzkin technique on each disjunct separately, because the existential quantification distributes over the $\vee$. More advanced approaches are also possible [Mon08]. The computational cost of quantifier elimination id doubly exponential.

Loos-Weispfenning Elimination

Another effective approach for $\mathcal{LRA}$ quantifier elimination is the Loos-Weispfenning technique [LW93], named after Rüdiger Loos and Volker Weispfenning. The technique solves the same problem of FME but is based on a completely different mechanism. The idea behind *Loos and*

Weispfenning elimination (LWE) approach is that an existentially quantified formula

$$\exists x.\phi(x, \vec{y})$$

with free variables $\vec{y} = y_1, ..., y_n$ can be replaced by a formula $\psi(\vec{y})$

$$\psi(\vec{y}) = \phi(\bar{x}_1, \vec{y}) \vee ... \vee \phi(\bar{x}_m, \vec{y})$$

where $\bar{x}_1, ..., \bar{x}_m$ are expressed as functions of $\vec{y}$. In the case of Loos-Weispfenning elimination the number of produced disjuncts is linear, but the overall complexity bound is still doubly exponential in the number of disjuncts of $\psi(\vec{y})$.

2.3 Timed Game Automata

We now consider a formal framework for the modeling of two-players timed games, namely the Timed Game Automata framework.

A finite automaton [LP98] comprises a finite set of locations and a finite set of labeled transitions (or actions). One of the locations is called initial (or starting); a distinguished subset of locations are marked as final. Each labeled transition specifies a legal move from one location to another.

A Timed Automaton(TA) [AD94] augments a finite automaton including constraints on non-negative, real-valued variables called clocks. To differentiate clocks from regular variables we use the over-line writing: we indicate variables with letters (e.g. x) and clocks with over-lined letters (e.g. $\overline{x}$). Each transition in a TA may include temporal constraints, called guards, that disable the transition if the current clock values do not satisfy those constraints. Each transition may also include clock resets that cause specified clocks to be reset to 0 whenever the transition is taken. Finally, each location may include an invariant: a constraint specifying the conditions under which the automaton may stay in that location. Definition 1 formalizes this structure.

Definition 1 (Timed Automaton)**.** *A Timed Automaton (TA) is a tuple, $A = (L, l_0, Act, \mathcal{X}, E, Inv)$, where:*

- *L is a finite set of locations;*
- *$l_0 \in L$ is the initial location;*
- *Act is a set of actions;*
- *$\mathcal{X}$ is a finite set of real-valued clocks;*
- *$E \subseteq L \times \mathcal{H}_k^{\cap}(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times L$ is a finite set of transitions; and*
- *$Inv : L \to \mathcal{H}_k^{\cap}(\mathcal{X})$ associates an* invariant *to each location.*

Elements in $\mathcal{H}_k^{\cap}(\mathcal{X})$ are conjunctions of constraints[2] *of the form, $x \bowtie k$ or $y - x \bowtie k$, where $x, y \in \mathcal{X}$, $k \in \mathbb{R}$, and $\bowtie$ is one of $<, \leq, =, >$ or $\geq$.*

A state of a TA is a pair $\langle l, v\rangle$, where $l \in L$ and $v : \mathcal{X} \to \mathbb{R}^{|\mathcal{X}|}$ is a total assignment for the clocks.

A transition in a TA is either discrete or timed. A discrete transition $\langle l_1, v_1\rangle \xrightarrow{a} \langle l_2, v_2\rangle$ is such that $a \in Act$, $(l_1, g, a, R, l_2) \in E$, $v_1 \models g$, $v_2 \models Inv(l_2)$, $v_2(x) = v_1(x)$ if $x \notin R$ and $v_2(x) = 0$ if $x \in r$. Essentially, a discrete transition makes a move in the automaton according with the transition relation E. The move is possible if the guard is satisfied by the starting state and the resulting state is obtained by changing the location and resetting the clocks in R. A timed transition is obtained by letting a certain amount of time to pass: $\langle l_1, v_1\rangle \xrightarrow{\delta} \langle l_1, v_1 + \delta^{|\mathcal{X}|}\rangle$. However, we cannot violate the location invariant: if $\langle l_1, v_1\rangle \xrightarrow{\delta} \langle l_1, v_2\rangle$, then $v_2 \models Inv(l_1)$. A TA is a formalism that models a set of traces (called “runs”). A run is a sequence of states $\langle\langle l_0, \vec{0}\rangle, s_1, s_2, \cdots\rangle$ where $\vec{0}$ indicates the assignment of all the clocks to 0, $s_i \xrightarrow{\delta} s_{i+1}$ if i is even and $s_i \xrightarrow{a} s_{i+1}$ if i is odd.

[2]Essentially, guards are expressed as purely-conjunctive (convex) $\mathcal{QF\_RDL}$ formulae.

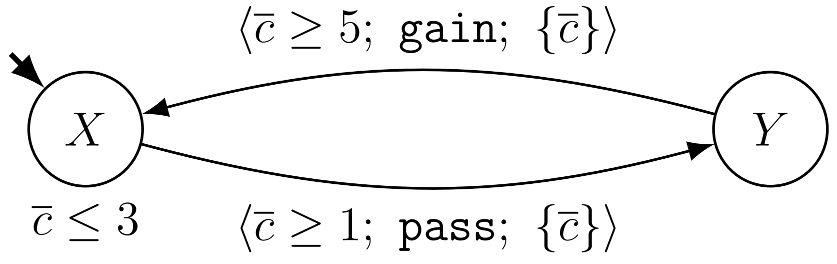

$\langle \overline{c} \geq 5;$ gain; $\{\overline{c}\}\rangle$
X
Y
$\overline{c} \leq 3$
$\langle \overline{c} \geq 1;$ pass; $\{\overline{c}\}\rangle$

Figure 2.1: A sample timed automaton. The bold arrow with no predecessor indicates the initial location X, while other arrows are the transitions.

Figure 2.1 shows a sample timed automaton. The timed automaton has one clock, $\overline{c}$. X is the initial location. X's invariant is $\overline{c} \leq 3$. Each transition has a label, $\langle \phi;\ s;\ R\rangle$, where ϕ is the guard, s is a name for the transition, and R is the set of clocks it resets. A *run* starts in the initial location, X, with $\overline{c} = 0$. X's invariant, $\overline{c} \leq 3$, and the guard, $\overline{c} \geq 1$, on the `pass` transition, together ensure that the timed automaton must take the transition from X to Y at some time when $1 \leq \overline{c} \leq 3$. When taken, that transition resets $\overline{c}$ to 0. Afterwards, the `gain` transition, whose guard is $\overline{c} \geq 5$, could be taken back to X at any time for which $\overline{c} \geq 5$. If taken, the `gain` transition also resets $\overline{c}$ to 0. However, since Y has no invariant, the timed automaton could instead remain at Y forever.

In turn, a Timed Game Automaton(TGA) generalizes a Timed Automaton by partitioning the set of transitions into controllable and uncontrollable. A TGA can be used to model a two-player game between an agent and the environment, where the agent controls the controllable transitions, and the environment controls the uncontrollable transitions. TGA are formally defined in definition 2.

Definition 2 (Timed Game Automaton)**.** *A Timed Game Automaton (TGA) is a Timed Automaton whose set of actions, Act, is partitioned into controllable* (Act_c) *and uncontrollable* (Act_u) *actions.*

Figure 2.2 shows a TGA with three locations: `ctrl`, `env` and `goal`,

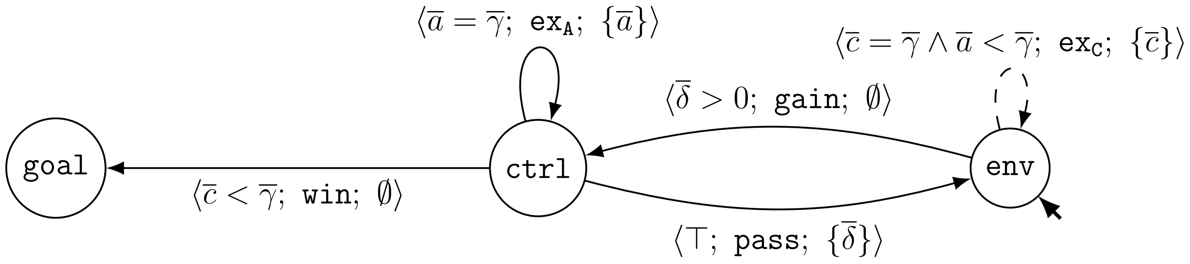

$\langle \overline{a} = \overline{\gamma};\ \texttt{ex}_{\texttt{A}};\ \{\overline{a}\}\rangle$
$\langle \overline{c} = \overline{\gamma} \wedge \overline{a} < \overline{\gamma};\ \texttt{ex}_{\texttt{C}};\ \{\overline{c}\}\rangle$
$\langle \overline{\delta} > 0;\ \texttt{gain};\ \emptyset\rangle$
goal
ctrl
env
$\langle \overline{c} < \overline{\gamma};\ \texttt{win};\ \emptyset\rangle$
$\langle \top;\ \texttt{pass};\ \{\overline{\delta}\}\rangle$

Figure 2.2: A sample Timed Game Automaton. Controllable transitions (belonging to Act_c) are solid, while uncontrollable transitions (belonging to Act_u) are dashed. Transitions are labeled with triplets representing the guard, the action label and the set of clocks to be reset.

where `env` is the initial location. It has four clocks: $\overline{a}, \overline{c}, \overline{\gamma}$ and $\overline{\delta}$. The solid arrows represent controllable transitions; the dashed arrow represents the one uncontrollable transition. For example, the transition from `ctrl` to itself has the label, $\langle \overline{a} = \overline{\gamma};\ \texttt{ex}_{\texttt{A}};\ \{\overline{a}\}\rangle$, which specifies that it can only be taken if $\overline{a}$ and $\overline{\gamma}$ have the same value; and that taking this transition resets $\overline{a}$ to 0. Consider the following possible run of this TGA. It begins at the initial location `env`, with all clocks set to 0. Five units of time later, when all clocks read 5, the agent takes the `gain` transition to `ctrl`. (The guard is satisfied; and no clocks are reset.) Then, at time 6, the agent takes the $\texttt{ex}_{\texttt{A}}$ transition, which causes $\overline{a}$ to be reset to 0. Then, at time 7, the agent takes the `pass` transition back to `env`, which resets $\overline{\delta}$ back to 0. At this point, $\overline{\delta} = 0$; $\overline{a} = 1$; and $\overline{c} = \overline{\gamma} = 7$. Thus, the environment can take the $\texttt{ex}_{\texttt{C}}$ transition from `env` to itself, resetting $\overline{c}$ to 0. Then, at time 10, the agent takes the `gain` transition back to `ctrl`, and at 11 the `win` transition to the `goal` location.

We adopt the common practice of labeling certain locations as *urgent*. An urgent location is one in which players are prevented from waiting. Making a location ℓ urgent is equivalent to: (1) introducing a new, fresh clock c that is reset by every transition entering ℓ; and (2) conjoining a new invariant, $c = 0$, to ℓ.

For any TGA, different kinds of games can be modeled [CDF+05]. In a reachability game, the controller (or agent) seeks to move the TGA into one of the *winning locations* within a finite amount of time. In the avoidance game, the controller seeks to prevent the TGA from entering a certain set of locations in a valid infinite run.

In the context of TGA games, a strategy is, in general, a mapping from partial runs to transitions: each partial run represents a history of the game that has an associated transition to be taken to win the game, each time such history is encountered. A strategy is winning if it allows the controller to invariably win the game no matter which counter-move the opponent takes. We consider *memory-less* strategies, since they have been shown to be sufficient for reachability and avoidance games [MPS95, CDF+05]. Intuitively, a memory-less strategy associates a state of the system to either an action to be executed or a special symbol λ that stands for "wait" (i.e., do nothing; wait until something changes).

Definition 3. *For a TGA,* $(L, l_0, Act, \mathcal{X}, E, Inv)$*, a* memory-less strategy *is a mapping* $f : L \times \mathbb{R}^{|\mathcal{X}|}_{>=0} \rightarrow Act_c \cup \lambda$*.*

Further details on the TGA semantics are available in [MPS95].

2.4 Clocks and Time Regions

In order to reason on TGA and to solve any kind of query on this formalism, we need ways to represent and manipulate constraints in $\mathcal{H}^{\cap}_{k}(\mathcal{X})$. The set $\mathcal{H}^{\cap}_{k}(\mathcal{X})$ is defined as the set of all possible formulae ϕ generated by the following grammar:

$$\phi ::= \top \mid \bot \mid \overline{x} \bowtie k \mid \overline{x} - \overline{y} \bowtie k \mid \phi \wedge \phi$$

Essentially, each of these formulae is just a conjunction of $\mathcal{QF\_RDL}$ atoms.

In fact, some approaches use SMT-based algorithm to analyze Timed Automata [BL11, KJN12]. Analogously to the classical Bounded Model Checking approach for Finite Automata [BCC$^+$03], these techniques work by representing paths in the TA by means of SMT formulae.

Another way of dealing with the time constraints in TA and TGA, is by using specialized data structures called Difference Bound Matrices (DBM), that have been devised to canonically represent and to manipulate the elements of $\mathcal{H}_k^{\cap}(\mathcal{X})$.

We first define a special clock $\overline{c_0}$ that always has value 0. Now, given the set of clocks $\mathcal{X}$, any constraint $\mathcal{H}_k^{\cap}(\mathcal{X})$ can be equivalently rewritten as a conjunction of terms in the form $x_i - x_j \prec k$ where each $x_i, x_j \in \mathcal{X} \cup \{\overline{c_0}\}$, $k \in \mathbb{R}$ and $\prec \in \{<, \leq\}$. Clearly, when both $x_i - x_j \prec k_1$ and $x_i - x_j \prec k_2$ constraints are present, we can simplify the formulation by taking only $x_i - x_j \prec min(k_1, k_2)$. Notably, this simplification always yields a formula with no more than $(|X| + 1) * (|X| + 1)$ constraints, independently of the input formula size.

In this simplified form, we can represent the whole constraint using a matrix Z of cardinality $|X| + 1 \times |X| + 1$ defined as follows.

$$Z[i,j] = \begin{cases} \langle k, \prec \rangle & \text{if } x_i - x_j \prec k \text{ appears in the simplified form} \\ \langle 0, \leq \rangle & \text{if } i = 0 \text{ or } i = j \\ \infty & \text{otherwise} \end{cases}$$

A pivotal property of DBMs is the existence of a canonical form. Given any DBM Z we can always derive a canonical DBM $\chi(Z)$ that is identical for each logically equivalent DBM Z'. Hence, given any two formulae $\phi, \psi \in \mathcal{H}_k^{\cap}(\mathcal{X})$ having DBMs Z_ϕ and Z_ψ respectively, if $\phi \leftrightarrow \psi$ is valid, then $\chi(Z_\phi) = \chi(Z_\psi)$. The normal form of a DBM can be computed in polynomial time in the size of the DBM; we refer the reader to [Ben02] for a thorough description of DBMs and their efficient normalization.

For the sake of this thesis, we just need to say that efficient and canonical implementation of DBM exists [Bul12] and that several manipulation operations for canonical DBM are supported. In particular, we will use the following operations.

- Conjunction ($\phi \wedge \psi$)
- Time elapse ($\phi\nearrow \doteq \exists\delta \leq 0.\phi[x \to (x+\delta) \mid x \in \overline{T}]$)
- Time rewind ($\phi\swarrow \doteq \exists\delta \geq 0.\phi[x \to (x+\delta) \mid x \in \overline{T}]$)
- Clock Reset ($\rho(\phi, \overline{x}) \doteq (\exists\overline{x}.\phi) \wedge \overline{x} = 0$)

Given a time constraint ϕ, $\phi\nearrow$ is the result of letting time pass indefinitely, while $\phi\swarrow$ is the time region from which it is possible to reach ϕ by letting time pass. The $\rho(\phi, \overline{x})$ operation unconditionally assigns the $\overline{x}$ clock to value 0 (clock reset). These are common operations in Timed Automata analysis. In addition, it is possible to check a DBM for emptiness (check if there exists at least one assignment to the clocks that satisfy the time constraint).

One last feature we exploit is the possibility of using and manipulating sets of DBM representing finite disjunctions. We call "time region" a disjunction of constraints in $\mathcal{H}_k^{\cap}(\mathcal{X})$: given a set of clocks X, the set of all possible time regions is given by all the possible formulae expressed by the following grammar:

$$\phi ::= \top \mid \bot \mid \overline{x} \bowtie k \mid \overline{x} - \overline{y} \bowtie k \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi$$

where $\bowtie \in \{<, \leq, >, \geq, =\}$.

Analogously to $\mathcal{QF\_RDL}$ formulae, time regions can be expressed as finite disjunctions of polihedra, and in particular, they can be expressed as disjunctions of DBMs. In fact a simple, yet effective way of manipulating time regions is to represent them as sets of DBMs. Such a data structure [Ben02] is called “Federation” (of DBMs). The same operations defined for DBMs are also defined for Federations and dedicated software packages exists [Bul12].

Part I

State-of-the-Art Survey

Introduction

The literature in model-based temporal planning and scheduling is vast and diverse. Several variations and declinations of the problems have been defined and studied. As a result, the landscape is quite scattered: different modeling frameworks, assumptions and execution models make it difficult to see the unifying picture, and to understand what has been done and what needs further research.

The goal of this Part is to survey the field of scheduling and planning under uncertainty: we classify existing works discriminating different assumptions and different underlying models. We first propose a general framework that can be used to classify a number of planning and scheduling techniques. Our framework assumes a generic plan is given: we do not define the plan structure at this stage, we intend a plan as a general prescription provided by a planner. An executor is demanded to send appropriate commands to a plant following the plan. The plant represents the controlled system, it receives the commands, and provides readings of its state by notifying the executor. We identified five main characteristics that can be used to classify each situation:

- Fixed or open set of actions to be executed (Scheduling vs. Planning)
- Presence or absence of non-determinism in the outcome of activities
- Full or no observability of the domain variables at run-time
- Presence or absence of temporal uncertainty

- Full or no observability of the duration of actions

Depending on these execution assumptions we can provide a classification of a wide area of planning and scheduling tasks. In this survey part, we only consider “synthesis” techniques, that are techniques to produce a schedule or a plan for a given problem, disregarding other types of problems such as schedule/plan validation or model validation. For each technique in the literature, we provide a short description surveying the major characteristics.

There are other surveys covering various aspects of AI planning and scheduling. The closest to what we discuss in this Part is the paper by Ingrand and Malik [IG14]. Its focus is on aspects of temporal reasoning and planning that are specifically important for robotics with a wide overview of the different components of a deliberation system. Compared to this Part, the survey has some works in common, but our focus is on temporal uncertainty in general and we do not limit ourselves to robotic deliberation planning.

Structure of this part. This Part of the thesis is structured as follows. In chapter 3 we describe and formalize our execution model. The abstract model yields a classification table that we use throughout the Part. We then survey existing works in the realm of scheduling (chapter 4) and of temporal planning (chapter 5). In chapter 6 we discuss relevant extensions of the problem that are orthogonal to the discussed model, but are important for their literature coverage or for the practical applicability of planning and scheduling techniques. Finally, in section 6.4 we summarize the part, highlighting several possible lines of research.

Chapter 3

Execution Model

In this section, we provide a unifying framework in which different flavors of scheduling and temporal planning problems can be mapped.

The framework is designed to abstract a wide range of real-world applications and to subsume many views of what a planning system is supposed to do. We identify a number of features that can be enabled or disabled yielding different classes of problems. This will in turn result in a classification table that we use in the rest of the Part to classify the various problems and techniques.

Our theoretical execution model is composed of two interacting components: a *plant* and a *plan executor*. The plant represents the physical agent together with the environment it is immersed in, while the plan executor is the component demanded to send commands to the plant and (possibly) receive observations of the plant status and events.

At an abstract level, the plant is an open system that can evolve in many possible ways, it can be (partially) controlled by specifying *commands* in time. It can be configured to provide interrupt-style signals to the executor when something of interest changes. For example, it can be instructed to raise a signal whenever the temperature surpasses a given threshold.

In the following, we detail the two components and we derive a classifi-

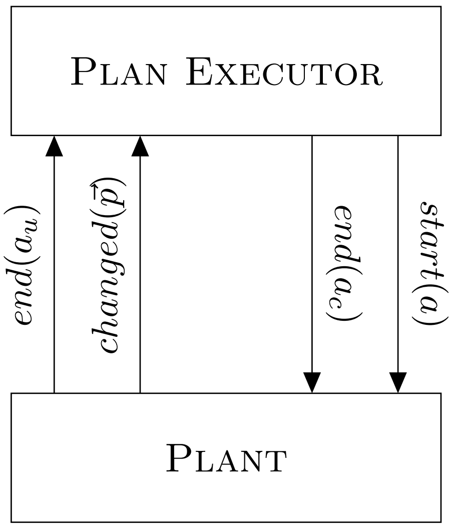


Figure 3.1: Visualization of the interface between the plant and the executor.

cation of the plan kinds that are sensible for this model.

3.1 Plant Interface

First, we describe the interface between the Plant and the Executor. Figure 3.1 provides a pictorial representation of the interface.

We assume that the plant has an embedded concept of "activity". An activity is a process that must be initiated using a starting command and runs for a certain amount of time. It can produce effects on the plant and it may require some conditions during its executions to be correctly executed. The effects of an activity can be non-deterministic: the changes on the plant can be expressed as relations (not functions) of the plant state. For example, an activity "roll a dice" has an outcome that is a-priori unknown regardless of our complete knowledge of the state of the plant. However, it might be possible to observe the outcome of the activity (after its termination). More practically, many activities can have a nominal outcome or several fault conditions that can non-deterministically arise and yield diverse consequences.

A first distinction between activities is embodied in our framework. In fact, we differentiate controllable and uncontrollable activities: a controllable activity can be started and ended by dedicated commands, while an uncontrollable activity can be started, but its duration is not under the control of the plant and therefore it is terminated by the environment.

The plant can accept in input two possible commands:

- `start(a)`. Meaning that an activity a must be immediately started
- $\texttt{end}(\texttt{a}_\texttt{c})$. Meaning that the started controllable activity a_c must be immediately terminated.

We disallow the possibility of terminating an uncontrollable activity by a command.

The plant generates a number of stimuli for the executor. In particular, the plant wakes up the executor each time an uncontrollable activity terminates, providing the information about which activity terminated ($\texttt{end}(\texttt{a}_\texttt{u})$). Moreover, the plant can be configured to interrupt the executor each time a Boolean expression expressed on the plant variables changes its truth value. For example, the plant can wake up the scheduler each time the temperature of the room goes below a certain threshold. We call these stimuli *predicates*. The number of predicates must be *finite* and expressed as computable functions of the plant state. We indicate with $\vec{p}$ the set of predicates. Each time a predicate changes its value, all the new predicate values are transmitted to the executor (indicated with a $changed(\vec{p})$ event). This feature provides a mean for the partial *observation* of the plant state: while it may prevent the full observation of the internal plant state, it allows the executor to be notified each time something relevant for the plan changes in order to react accordingly.

We highlight that the interaction between the plant and the executor happens in real time, there is no possibility of delaying a command in the

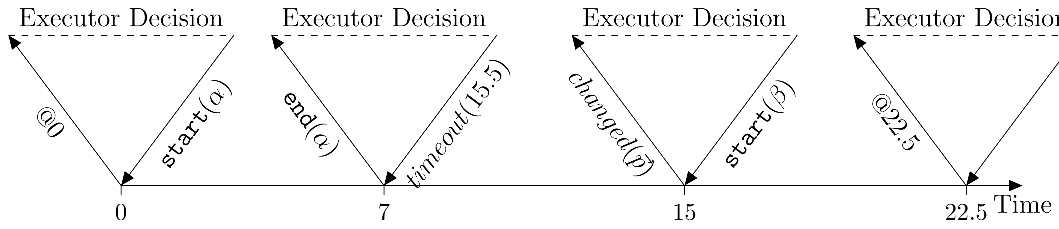


Figure 3.2: Visualization of the timing of a possible execution. The x axis represents time. Each time the executor is woken up by a signal from the plant the computation happens in no time and a command for the plant is produced, or a timeout alarm is set, so that the executor will wake up at a specific time.

plant nor any anticipation or delay in the observation. The executor must take care of accommodating the prescriptions in the plan given this limited capabilities.

At this stage, we do not detail the goal of the plan, limiting ourselves to consider it a checkable condition on a finite trace of the plant.

3.2 Plan Executor

The plan executor is demanded to send to the plant the commands in time following the prescription of the plan, possibly taking into account the observations coming from the plant. Internally, the executor can be seen as any program or function, mapping a history of the observation into a stream of commands for the plant. We do not consider the computation time of the executor, for this reason we imagine an execution schema as depicted in figure 3.2, in which the time flow is interleaved with the computation time of the executor.

We assume that the executor can send commands only when it receives a stimulus either from the plant or from a timer. The latter, can be seen as an internal alarm clock that wakes up the system when a pre-specified deadline is met. For example, at time 3 the executor can decide to sleep for 5 time units. At this point, if no stimulus is produced by the plant, at time 8 the executor wakes up and can send a command. If no wake up deadline is set by the executor, it sleeps until a stimulus is received by the plant (an uncontrollable activity terminates or a predicate transitions from one truth value to another).

We define a number of classes also for the executor depending on whether it is allowed to observed past events or if it has a complete knowledge of the future of the plant.

3.3 Formal Model

We formalize the Plant $\mathcal{P}$ as a set of traces over a set of variables V. This formalization keeps the whole generality of a plant evolution, without committing to a specific model.

We assume the set of variables V of the plant is given, and for each variable $v \in V$ we define the domain of v (written $Dom(v)$) as the set of possible values for v.

We first define a trace as a function that maps a variable f and a non-negative real number representing a time into a value v for f (with $v \in Dom(f)$). We also employ a special value: $\lightning$.

Definition 4 (Timed Trace)**.** *Given a set of variables V, a timed trace is a function $T : V \times \mathbb{R}^+ \to \bigcup_{v \in V} Dom(v) \cup \{\lightning\}$ such that for any $v \in V$ and any $t \in \mathbb{R}^+$, $T(v,t) \in Dom(v)$. We write $T(t)$ with $t \in \mathbb{R}^+$ to indicate the assignment $\{v = T(v,t) \mid v \in V\}$.*

In this setting, a plant is a (possibly infinite) set of traces. In fact, each trace represent a possible temporal evolution. In this view, we clearly subsume branching time; in fact, a temporal situation that can evolve in different ways is simply represented as the collection of possible paths having a common prefix.

Definition 5 (Plant)**.** *A plant $\mathcal{P}$ defined over a set of variables V is a set of traces over V.*

The plant can evolve by itself (for example, exogenous events and non-determinism are instances of this behavior) but it can also be controlled via activities.

We assume that for each possible activity a, a special variable $v_a \in V$ exists with Boolean domain: $\{\top, \bot\}$. This variable is set to $\top$ the moment activity a is started and is reset to $\bot$ when it ends. In this way, we can monitor each activity in a timed trace. Activities may change the status of the plant deterministically or non-deterministically.

We consider an interface of the plant that allows the executor to start any activity at any time and to terminate a controllable activity at any time. Moreover, the plant will provide some observations to the executor. In particular, it provides an instantaneous notification of the termination of an uncontrollable activity and a notification whenever a finite and pre-defined set of predicates $\vec{p}$ defined over V changes its value.

Definition 6 (Interface)**.** *The interface of a plant $\mathcal{P}$ over V is a tuple $\langle A_c, A_u, \vec{p}\rangle$, where A_c is a set of controllable activities, A_u is a set of uncontrollable activities and $\vec{p}$ is a finite set of predicates over V.*

A command is then an input to the interface, and we define a history of commands as a set of time-stamped commands.

Definition 7 (Command)**.** *Given a plant interface $\langle A_c, A_u, \vec{P}\rangle$, a command C is either start(a) with $a \in A_c \cup A_u$ or end(a_c) with $a_c \in A_c$.*

Definition 8 (Command History)**.** *Given a plant interface $\langle A_c, A_u, \vec{p}\rangle$, a command history H_C is a finite set of tuples $\langle C, t\rangle$, where C is a command for the interface and $t \in \mathbb{R}^+$.*

Similarly, an observation is a possible output of the interface, and we define a history of observations as a set of time-stamped observations.

Definition 9 (Observation). *Given a plant interface $\langle A_c, A_u, \vec{p}\rangle$, an observation O is either changed($\vec{p}$) or end(a_u) with $a_u \in A_u$.*

Definition 10 (Observation History). *Given a plant interface $\langle A_c, A_u, \vec{p}\rangle$, an observation history H_O is a finite set of tuples $\langle O, t\rangle$, where O is an observation for the interface and $t \in \mathbb{R}^+$.*

Given a history of observations, the set of traces characterizing the plant can be pruned to be limited to the one matching the history of observation. Similarly, given a history of commands.

Definition 11 (Restricted Observation Traces). *Given a plant interface $\langle A_c, A_u, \vec{p}\rangle$ for plant $\mathcal{P}$ and an observation history H_O, we define the traces restricted to H_O as a subset $T_{\mathcal{P}}(H_O)$ of $\mathcal{P}$ where each timed trace $T \in T_{\mathcal{P}}(H_O)$ is such that:*

- *For each $\langle end(a), t\rangle \in H_O$, $T(v_a, t) = \top$ and $T(v_a, t+k) = \bot$ for any $k \in [0, \epsilon]$ with a sufficiently small $\epsilon > 0$.*
- *For each $\langle changed(\vec{p}), t\rangle \in H_O$, $T(t) \models \vec{p}$ and $T(t-k) \not\models \vec{p}$ for any $k \in [0, \epsilon]$ with a sufficiently small $\epsilon > 0$.*
- *For each $t \in \mathbb{R}^+$ and each $a \in A_u$ such that $T(v_a, t) = \top$ and $T(v_a, t+k) = \bot$ for any $k \in [0, \epsilon]$ with a sufficiently small $\epsilon > 0$, $\langle end(a), t\rangle \in H_O$.*
- *For each time $t \in \mathbb{R}^+$ in which the predicates $\vec{p}$ have truth values $\vec{x}$ ($T(t) \models \vec{p}_i$ if and only if $\vec{x}_i$ is true), and at time $t-k$ have truth values $\vec{y}$, $\vec{x} \neq \vec{y}$ for any $k \in [0, \epsilon]$ with a sufficiently small $\epsilon > 0$, $\langle changed(\vec{p}), t\rangle \in H_O$.*

Definition 12 (Restricted command traces). *Given an interface $\langle A_c, A_u, \vec{p}\rangle$ for plant $\mathcal{P}$ and a command history H_C, we define the traces restricted to H_C as a subset $T_{\mathcal{P}}(H_C)$ of $\mathcal{P}$ where each timed trace $T \in T_{\mathcal{P}}(H_C)$ is such that:*

- *For each* $\langle end(a), t\rangle \in H_C$*,* $T(v_a, t) = \top$ *and* $T(v_a, t+k) = \bot$ *for any* $k \in [0, \epsilon]$ *with a sufficiently small* $\epsilon > 0$*.*

- *For each* $\langle start(a), t\rangle \in H_C$*,* $T(v_a, t) = \top$ *and* $T(v_a, t-k) = \bot$ *for any* $k \in [0, \epsilon]$ *with a sufficiently small* $\epsilon > 0$*.*

- *For each* $t \in \mathbb{R}^+$ *and each* $a \in A_c$ *such that* $T(v_a, t) = \top$ *and* $T(v_a, t+k) = \bot$ *for any* $k \in [0, \epsilon]$ *with a sufficiently small* $\epsilon > 0$*,* $\langle end(a), t\rangle \in H_C$*.*

- *For each* $t \in \mathbb{R}^+$ *and each* $a \in A_c \cup A_u$ *such that* $T(v_a, t) = \top$ *and* $T(v_a, t-k) = \bot$ *for any* $k \in [0, \epsilon]$ *with a sufficiently small* $\epsilon > 0$*,* $\langle start(a), t\rangle \in H_C$*.*

Intuitively, a trace yields an observation history if the observations of that trace in time exactly produce the elements of the observation history (each activity termination and each predicate change happen exactly at times indicated in the observation history and no other event is present in the trace nor in the observation history). Analogously, a command history is compatible with a trace if each activity start or end in the trace coincides with a start command in time and no other command is present in the trace nor in the command history.

We assume that an impossible input infers a trace that always assigns $\lightning$ for each variable after the first inconsistent point, so restricted traces are never empty sets.

We can now define the executor as a function that is invoked each time the plant signals a change through the interface to produce a command to be immediately executed. Such a function takes in input a plan and maps a prefix of the observation history into a command. Intuitively, this is a way of modeling an "infinite recall": the plan executor never forgets the past observations and can always rely on them for taking decisions. We also model the possibility of having in input a "prediction" representing a knowledge on the future happenings of the system.

We start by defining an observation history slice as an observation history that is limited up to a specific point in time.

Definition 13 (Observation History Slice). *Given an observation history H_O and a real number $k \in \mathbb{R}^+$, the slice of H_O up to k is the set $H_O[k] \doteq \{\langle x, t\rangle | \langle x, t\rangle \in H_O, t \leq k\}$.*

Next, we formalize a prediction. A prediction represents knowledge on the future outcomes of the plant: given any command history, it returns a (possibly complete) observation history that are the predicted future observations of the plant, if the provided command history will be imposed on the plant. An example where such a prediction is needed is when before executing an estimation on the activity duration is provided and the reasoner assumes the estimation is correct. The executor can query the prediction to project possible futures and pick the most favorable one.

Definition 14 (Prediction). *A prediction is a function ρ that maps a command history H_C in an observation history H_O.*

We can now define the executor as follows.

Definition 15 (Plan Executor). *The plan executor is a function exec that maps a plan π, an observation history H_O, a prediction ρ and a time $t \in \mathbb{R}^+$ into a command C or in timeout(k) with $k \in \mathbb{R}^+$.*

For every plan π, each prediction ρ, each time $t \in \mathbb{R}^+$ and each pair of observation histories H_O^1, H_O^2 if $H_O^1[t] = H_O^2[t]$ then $exec(\pi, H_O^1, \rho, t) = exec(\pi, H_O^2, \rho, t)$.

Intuitively, a plan executor $exec(\pi, H_O, \rho, t)$ is a function that is invoked each time the executor is woken up. The function takes the observation history to "remember" past happenings, a prediction ρ as additional knowledge on the future outcomes of the plant and the absolute time t. The function produces a command C to be immediately executed in the plant (via the interface) or a $timeout(k)$ decision, that signals the execution to do nothing for k time units and then re-invoke the executor, unless a predicate changes in the plant or an uncontrollable activity terminates.

The condition in the definition ensures that the executor ignores any observation in H_O later than t, hence the only future knowledge that is available to the executor comes from the prediction ρ.

3.4 Plant Classification

Given this plant-executor abstraction, we are now able to define two dimensions of interest that we use to classify the different kinds of planning and scheduling problems.

First, we define a deterministic plant as a plant that, given a command history, exhibits exactly one trace. Intuitively, a deterministic plant is a plant that can be completely controlled via the interface input.

Definition 16 (Determinism). *A plant is said to be deterministic if and only if for every possible command history H_C, $|T_{\mathcal{P}}(H_C)| = 1$.*

If the plant is non-deterministic, we define two sub-classes of uncertainty. A plant is said to be duration-only uncertain if the only thing that is not under the control of the executor are the duration of activities: this means that once a duration for each activity is fixed, the behavior of the plant is completely deterministic.

Definition 17 (Duration uncertainty). *Given a command history H_C, we define the duration uncertainty of H_C as the set $U(H_C) \doteq \{\langle a_u, t\rangle \mid a_u \in A_u, \langle start(a_u), t\rangle \in H_C\}$.*

Definition 18 (Duration-only uncertainty). *A plant is said to be duration-only uncertain if $A_u \neq \emptyset$ and there exists a deterministic plant $\mathcal{P}'$ such that for every possible command history H_C and each $T \in T_{\mathcal{P}}(H_C)$, $T_{\mathcal{P}'}(H_c \cup \{\langle end(a_u), t + \delta\rangle \mid \langle a_u, t\rangle \in U(H_C), \delta \in \mathbb{R}^+\}) = \{T\}$.*

If the plant is non-deterministic, but all the activities are controllable, then we say it is discrete-only uncertain, because the uncertainty comes from exogenous effects or from non-deterministic activity effects.

Definition 19 (Discrete-only uncertainty). *A plant is said to be discrete-only uncertain if it is not deterministic and $A_u = \emptyset$.*

By complement, we say that a plant is fully uncertain if it is neither duration-only nor discrete-only uncertain, but is non-deterministic.

Definition 20 (Fully uncertain). *A plant is said to be fully uncertain if it is not deterministic and it is neither duration-only nor discrete-only uncertain.*

Table 3.1 shows a first classification of plant types based on the categories we formalized so far. We titled the two dimensions "Activity Duration", representing the temporal uncertainty on the duration of activities, and "Plant Evolution" representing the discrete uncertainty.

3.5 Plan Executor Classification

At this point, we introduce a second dimension in our classification. We consider the plant interface $(A_c, A_u, \vec{p})$ and we distinguish different assumptions on the executor capabilities. In particular, we classify executors by their dependence or independence from the observations and the observation kind.

An executor as per definition 15 is a function that maps an observation history, a prediction and an absolute time into a command (or in a

		Activity Duration	
		Controllable	**Uncontrollable**
Plant Evolution	**Deterministic**	Deterministic	Duration-only
	Non Deterministic	Discrete-only	Fully uncertain

Table 3.1: Classification of plant types.

timeout(k) wait decision). As such, it is in principle possible to feed the executor with a prediction that completely predicts the future happenings in the plant. This models the fact that in some cases the executor has an estimator or a sensor that "projects-away" the uncertainty before the actual execution. In this way, the executor has access to a prediction of the consequences of its choices on the plant and can reason accordingly. However, this kind of prediction is not always available, therefore we first discriminate situations where this is available from situations where it is not. Formally, we impose this discrimination on the executor dependence or independence on the prediction ρ instead of defining a class of executors that do not take ρ in input, because this simplifies the formalism.

Definition 21 (Dynamic Executor)**.** *A plan executor exec is said to be dynamic if for every plan π, every observation history H_O, every time $t \in \mathbb{R}^+$ and for each pair of predictions ρ^1 and ρ^2, $exec(\pi, H_O, \rho^1, t) = exec(\pi, H_O, \rho^2, t)$.*

The definition imposes that no matter what is the input to the executor, the prediction ρ is always ignored. We call "weak" an executor that is not dynamic and has access to a complete knowledge of the future happenings through ρ.

In the following, we discriminate three cases for the executor, namely time-strong, time-dynamic and time-weak. We define a time-strong executor as an executor that does not observe the duration of uncontrollable activities (it ignores the $end(a_u)$ signal).

Definition 22 (Time-Projection)**.** *Given an observation history H_O, its time projection $\lambda_t(H_O)$ is an observation history $\{\langle x, t\rangle | \langle x, t\rangle \in H_O, x = end(a_u)\}$. Given a prediction ρ its projection $\lambda_t(\rho)$ is the function $\rho \circ \lambda_t$.*

Intuitively, the time projection of an observation history masks all the predicate changed events from the history leaving only the uncontrollable action end. Similarly, also the prediction is projected by removing discrete events from the function images.

Definition 23 (Discrete-Projection)**.** *Given an observation history H_O, its discrete projection $\lambda_d(H_O)$ is an observation history $\{\langle x, t\rangle | \langle x, t\rangle \in H_O, x = changed(\vec{p})\}$. Given a prediction ρ its projection $\lambda_d(\rho)$ is the function $\rho \circ d$.*

Analogously, the discrete projection leaves only predicate change information, removing all the uncontrollable ends timings.

Definition 24 (Time-Strong Executor)**.** *A plan executor exec is said to be time-strong if for every plan π, each observation history H_O, each prediction ρ and each time t, $exec(\pi, H_O, \rho, t) = exec(\pi, \lambda_d(H_O), \lambda_d(\rho), t)$.*

Intuitively, a time-strong executor ignores any $end(a_u)$ observation, considering only discrete events.

On this line, we can define an executor that "remembers" all the past observations and is able to exploit this knowledge to take decisions as follows.

Definition 25 (Time-Dynamic Executor)**.** *A plan executor exec is said to be time-dynamic if for every plan π, each observation history H_O, each* *time t and each pair of predictions ρ^1 and ρ^2 such that $\lambda_d(\rho^1) = \lambda_d(\rho^2)$, $exec(\pi, H_O, \rho^1, t) = exec(\pi, H_O, \rho^2, t)$.*

Intuitively, a time-dynamic executor is allowed to depend on past happening (stored in H_O) of any kind, but it cannot use any information on the future timing of actions. Nonetheless it is allowed to use future information on the predicates (stored in λ_d).

It is easy to see that any time-strong executor is also time-dynamic.

Proposition 3.1. *Any time-strong executor exec is time-dynamic.*

Proof. Consider any two predictions ρ^1, ρ^2 such that $\lambda_d(\rho^1) = \lambda_d(\rho^2)$. Since *exec* is a time-strong executor, $exec(\pi, H_O, \rho^1, t) = exec(\pi, H_O, \rho^2, t)$ for any π, H_O, and t; hence *exec* is time-dynamic. $\square$

These two definitions split the landscape of duration-uncertain plants in three classes: time-strong, time-dynamic and time-weak. The last class is the one in which we do not impose any particular assumption on the executor, allowing for the "clairvoyance" of the observations.

Analogously to the temporal uncertainty case, we define three classes that split the landscape of non-deterministic plants. We define an undet-strong executor as an executor that makes no use of the discrete observations $changed(\vec{p})$.

Definition 26 (Undet-Strong Executor)**.** *A plan executor exec is said to be undet-strong if for every plan π, each observation history H_O, each prediction ρ and each time t, $exec(\pi, H_O, \rho, t) = exec(\pi, \lambda_t(H_O), \lambda_t(\rho), t)$.*

We define an undet-dynamic executor as an executor that only observe past predicate observation, without observing the future discrete happenings.

Definition 27 (Undet-Dynamic Executor)**.** *A plan executor exec is said to be time-dynamic if for every plan π, each observation history H_O, each*

			Activity Duration			
			Controllable	**Uncontrollable**		
				No Observation	**Dynamic Observation**	**Future Observation**
Plant Evolution	**Deterministic**		Deterministic Plant	Duration-only Plant, Time-Strong Executor	Duration-only Plant, Dynamic Executor	Duration-only Plant, Weak Executor
	Non-Deterministic	**No Observation**	Discrete-only Plant, Undet-Strong Executor	Fully-Uncertain Plant, Undet-Strong Time-Strong Executor	Fully-Uncertain Plant, Undet-Strong Time-Dynamic Executor	Fully-Uncertain Plant, Undet-Strong Time-Weak Executor
		Dynamic Observation	Discrete-only Plant, Undet-Dyanmic Executor	Fully-Uncertain Plant, Undet-Dyanmic Time-Strong Executor	Fully-Uncertain Plant, Undet-Dyanmic Time Dyanmic Executor	Fully-Uncertain Plant, Undet-Dyanmic Time-Weak Executor
		Future Observation	Discrete-only Plant, Undet-Weak Executor	Fully-Uncertain Time-Strong Plant, Undet-Weak Executor	Fully-Uncertain Plant, Undet-Weak Time Dynamic Executor	Fully-Uncertain Plant, Undet-Weak Time-Weak Executor

Table 3.2: Classification table showing the landscape of possible classes of plants end executors. The table reports the name of each class obtained by combining the plant type and the executor type.

time t and each pair of predictions ρ^1 and ρ^2 such that $\lambda_t(\rho^1) = \lambda_t(\rho^2)$, $exec(\pi, H_O, \rho^1, t) = exec(\pi, H_O, \rho^2, t)$.

Also in this case, it is easy to see that any undet-strong executor is also undet-dynamic.

Proposition 3.2. *Any undet-strong executor exec is undet-dynamic.*

Proof. Consider any two predictions ρ^1, ρ^2 such that $\lambda_t(\rho^1) = \lambda_t(\rho^2)$. Since *exec* is a undet-strong executor, $exec(\pi, H_O, \rho^1, t) = exec(\pi, H_O, \rho^2, t)$ for any π, H_O, and t; hence *exec* is time-dynamic. $\square$

Given these definitions, we can now introduce a new dimension in our classification by distinguishing executor assumptions as shown in table 3.2. We distinguish situations in which uncontrollable activity duration is not observed from situations in which it is observed dynamically from situations in which it is observed without constraints (time-strong vs. time-dynamic vs. non time-dynamic). Similarly, we have three classes for undet-strong vs. undet-dynamic vs non undet-dynamic. For the rest of this thesis we will use this schema to classify planning and scheduling problems.

3.6 Discussion

3.6.1 Partial Observability

An important issue arising when non-determinism of any kind is considered is partial observability. In a real system, not all the relevant variables are directly observed by appropriate sensors, either for cost and implementation reasons or simply because some quantities are not measurable in a sufficiently small time. Partial observability limits the entities that can be observed and forces the execution to take decision with incomplete information. In our model, we handles this concept in the non-determinism formalization: the configurable set of predicates limits the amount of information available to the executor on the discrete state of the system. Also for timing information a similar idea is applicable: we could divide uncontrollable activities in observable and non-observable and provide only the observable durations to the executor.

For the sake of this thesis, partial observability is not needed: we limit ourselves to the complete observability of both the discrete and timed information of the system. We only discriminate on the timing when the information becomes available to the executor. Nonetheless, we believe that an extension of this formal model in the direction of partial-observability is important and interesting, and we consider this as future work.

3.6.2 Weak Executors and Predictions

For the sake of the classification summarized in table 3.2, we do not need to impose any constraint on predictions, we simply discriminate executors (and plans) that make use or ignore such information. In fact, the prediction formalization in definition 14, simply gives a syntax without imposing the plant to actually adhere to the knowledge stored in the prediction itself.

In practical applications and problems, however, the prediction is usually assumed to be aligned with the plant evolution. For example, in chapter 10 we discuss the weak controllability problem and we assume to know in advance the action uncontrollable durations. This has theoretical relevance, but also practical applications: durations can be seen as parameters of the problem that get instantiated before starting the execution.

The prediction definition we gave is very general and comprehensive; for example, it allows the modeling of an executor that knows about the future outcomes of activities depending on their starting time and ordering. In existing formalisms and applications, however, the "weak" information is much more limited. In weak controllability, the information is a fixed and total assignment of durations and of non-deterministic outcomes to uncontrollable activities. In the planning case, no formalisms exist in the literature, but we imagine that also in that case it make more sense to assume that the executor is given a prediction of the action durations and outcome that is independent of the command history (i.e. the duration of each activity, even if repeated, is independent of the followed plan). Nonetheless, the classification table and the formalization accommodate even more complex cases.

Chapter 4

Scheduling Classification

The landscape of scheduling problems and techniques for planning is wide and diverse. We focus on models and techniques used in AI temporal reasoning disregarding resource constraints (we briefly discuss the modeling and use of resources in section 6.2).

In general, a scheduling problem is expressed as a finite set of entities (usually activities or time points) subject to constraint of various nature that needs to be allocated in time. This is fully compatible with our execution model: the plant is demanded to execute the activities and to observe their outcome or duration. The plan executor sends the command to the plant according to a plan that is the outcome of a scheduler. In this view, a scheduler is a specialized kind of planner that can reason only on a pre-fixed set of activities instead of generating the set of activities from a theoretical model.

In this Chapter, we analyze several kinds of scheduling problems with a particular focus on temporal networks. We highlight that this survey of the scheduling literature is not intended to be comprehensive: we focus on models and techniques relevant for AI temporal planning. For each formalism under analysis, we will provide a description of the most successful reasoning techniques. Table 4.1 provides an overview of this survey

			Activity Duration			
			Controllable	**Uncontrollable**		
				No Observation	**Dynamic Observation**	**Future Observation**
Plant Evolution	**Deterministic**		Consistency of TNs, Point Algebra, Allen Algebra	Strong Controllability of TNUs	Dynamic Controllability of TNUs	Weak Controllability of TNUs
	Non-Deterministic	**No Observation**	Strong Consistency of CTNs			
		Dynamic Observation	Dynamic Consistency of CTNs		Dynamic Controllability of CTNUs	
		Future Observation	Weak Consistency of CTNs			

Table 4.1: Overview of the surveyed scheduling problems.

according to the classification table we introduced in chapter 3.

4.1 Qualitative Scheduling

Two types of temporal reasoning problems emerged over time: qualitative and quantitative. Qualitative scheduling deals with precedence and relative timing of intervals but it cannot express metric constraints such as deadlines and absolute time, that are considered in quantitative scheduling.

4.1.1 Point Algebra

Point Algebra (PA) [SV98] is a formalism that models relative timing of events. The basic elements of this algebra are time points: a time point is a time-valued variable that represent an event such as the starting or the termination of an activity. Given a pair of time points a and b, either of three relations are possible: a before b ($a < b$), a after b ($a > b$) or a equals b ($a = b$). Let $\bowtie_{PA} \doteq \{<, >, =\}$ be the set of such relations, and let $R \doteq 2^{\bowtie_{PA}}$ be the set of qualitative constraints.

$\circ$	$<$	$>$	$=$
$<$	$<$	$\{<,>,=\}$	$<$
$>$	$\{<,>,=\}$	$>$	$>$
$=$	$<$	$>$	$=$

Table 4.2: PA composition table.

In this setting, $\bowtie_{PA}$ is the tautological constraint and $\emptyset$ is the unsatisfiable constraint; the other elements of R are interpreted as the disjunction of the constraints appearing in each set.

We can define the composition of the relations in $\bowtie_{PA}$ as shown in table 4.2.

Definition 28 (Point Algebra)**.** *The Point Algebra is defined as* $\langle R, \cup, \circ \rangle$. *It is an algebra because:*

- *it is closed under* $\cup$ *and under* $\circ$
- $\cup$ *is associative and commutative with* $\emptyset$ *as identity element*
- $\circ$ *is associative with* $\{=\}$ *as identity*
- $\cup$ *and* $\circ$ *are distributive*

Given any set of constraints expressed in PA, we can check if it is satisfiable in polynomial time by applying path consistency on the network induced by the set of constraints [GNT04].

4.1.2 Allen Algebra

Allen's algebra [All83] is a different formalism that is based on intervals instead of time points. The basic constrained elements are variables representing right-open intervals. The algebra is constructed from the 13 basic relations listed in table 4.3.

Similarly to PA, Allen's algebra is defined on the power-set of the basic relations that are interpreted disjunctively. For example, $A\{m, <\}B\}$ means that A either meets or comes before B. Also Allen's algebra needs a composition operation ($\circ$) that is defined according to table 4.4.

Formally, the algebra can be defined as follows.

Definition 29 (Allen's Algebra). *Allen's algebra is a tuple* $\langle R, \cup, \circ \rangle$, *where* $\bowtie_{AA} \doteq \{<, >, =, m, mi, s, si, f, fi, d, di, o, oi\}$ *and* $R \doteq 2^{\bowtie_{AA}}$.

Allen's algebra is strictly more expressive than PA, and the satisfiability problem is NP-hard [SV98]. Nonetheless, the algebra is limited by its qualitative nature: it is not possible to express metric constraints such as quantitative deadlines or duration constraints. In more recent works, especially in planning applications, Allen algebra has been extended to support metric constraints and employed as basic constraint formalism for domain description languages [FJ03, CCF$^+$09]. However, the use of metric temporal networks is far more common in AI planning systems.

4.2 Temporal Networks

Another model for expressing time constraints and knowledge is the Temporal Network[1] (TN). Differently from Allen's algebra, the basic unit for the temporal network modeling framework are time points (instead of intervals), but differently from PA, Temporal Networks allow for the use of metric constraints. Different kinds of constraints have been introduced

[1]Some authors use the term "Temporal Problem" instead of "Temporal Network". We think this is confusing as it mixes the model of the temporal knowledge with the query to be answered. Hence, we take the following view: "Temporal Network" is used to indicate the modeling of the temporal knowledge, while the word"problem" is used to indicate the query (or queries) that is asked on the temporal network.

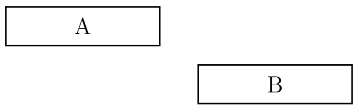
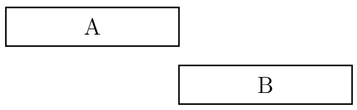
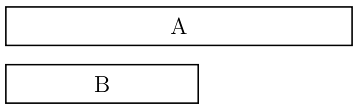
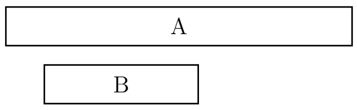
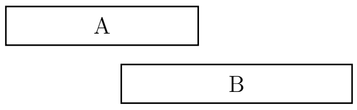
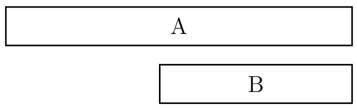
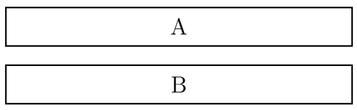
Timeline representation	Relation	Relation symbol
	A before B B after A	$A < B$ $B > A$
	A meets B A met-by B	$A\, m\, B$ $B\, mi\, A$
	A started-by B B starts A	$A\, si\, B$ $B\, s\, A$
	A contains B B during A	$A\, di\, B$ $B\, d\, A$
	A overlaps B B overlapped-by A	$A\, o\, B$ $B\, oi\, A$
	A finished-by B B finishes A	$A\, fi\, B$ $B\, f\, A$
	A equals B	$A = B$

Table 4.3: Symbolic notation for Allen relations.

$\circ$	$<$	$>$	d	di	o	oi	m	mi	s	si	f	fi
$<$	$<$	$\emptyset$	$< o\ m\ d\ s$	$<$	$<$	$< o\ m\ d\ s$	$<$	$< o\ m\ d\ s$	$<$	$<$	$< o\ m\ d\ s$	$<$
$>$	$\emptyset$	$>$	$> oi\ mi\ d\ f$	$>$	$> oi\ m\ d\ f$	$>$	$> oi\ mi\ d\ f$	$>$	$> oi\ mi\ d\ f$	$>$	$>$	$>$
d	$<$	$>$	d	$\emptyset$	$< o\ m\ d\ s$	$> oi\ mi\ d\ f$	$<$	$>$	d	$> oi\ mi\ d\ f$	d	$< o\ m\ d\ s$
di	$< o\ m\ di\ fi$	$> oi\ di\ mi\ si$	$o\ oi\ d\ di =$	di	$o\ di\ fi$	$oi\ di\ si$	$o\ di\ fi$	$oi\ di\ si$	$di\ fi\ o$	di	$di\ si\ oi$	di
o	$<$	$> oi\ di\ mi\ si$	$o\ d\ s$	$< o\ m\ di\ di$	$< o\ m$	$o\ oi\ d\ di =$	$<$	$oi\ di\ si$	o	$di\ fi\ o$	$d\ s\ o$	$< o\ m$
oi	$< o\ m\ di\ fi$	$>$	$oi\ d\ f$	$> oi\ mi\ di\ si$	$o\ oi\ d\ di =$	$> oi\ mi$	$o\ di\ fi$	$>$	$oi\ d\ f$	$oi > mi$	oi	$oi\ di\ si$
m	$<$	$> oi\ di\ mi\ si$	$o\ d\ s$	$<$	$<$	$o\ d\ s$	$<$	$f\ fi =$	m	m	$d\ s\ o$	$<$
mi	$< o\ m\ di\ fi$	$>$	$oi\ d\ f$	$>$	$oi\ d\ f$	$>$	$s\ si =$	$>$	$d\ f\ oi$	$>$	mi	mi
s	$<$	$>$	d	$< o\ m\ di\ fi$	$< o\ m$	$oi\ d\ f$	$<$	mi	s	$s\ si =$	d	$< m\ o$
si	$< o\ m\ di\ fi$	$>$	$oi\ d\ f$	di	$o\ di\ fi$	oi	$o\ di\ fi$	mi	$s\ si =$	si	oi	di
f	$<$	$>$	d	$> oi\ mi\ di\ si$	$o\ d\ s$	$> oi\ mi$	m	$>$	d	$> oi\ mi$	f	$f\ fi =$
fi	$<$	$> oi\ di\ mi\ si$	$o\ d\ s$	di	o	$oi\ di\ si$	m	$si\ oi\ di$	o	di	$f\ fi =$	fi

Table 4.4: Composition table for Allen's algebra (excluding equality).

in the formalism to accommodate a wide range of characteristics in the model, but the common characteristic of all the formalism is the use of metric constraints. In fact, temporal networks offer a viable solution for modeling and reasoning on situations where deadlines and durations are involved.

4.2.1 Temporal Networks and Consistency

Many applications require the scheduling of a set of activities over time, subject to constraints of various nature. In Artificial Intelligence, and in particular in planning, this kind of temporal knowledge is often expressed as a *Temporal Network* (TN), where each activity is associated with two time points, representing the start time and the end time, and time points are subject to constraints.

The first class of temporal networks we analyze models knowledge of a temporal situation in which a set of homogeneous time points are subject to some metric temporal constraints.

Simple Temporal Network

The first Temporal Network model was proposed by Dechter et al. in [DMP91a]. The paper presents the notorious Simple Temporal Network formalism in which a finite set of time points is subject to binary difference constraints.

Definition 30 (STN [DMP91a]). *A Simple Temporal Network (STN) is a tuple $\langle \mathcal{T}, \mathcal{C} \rangle$ where $\mathcal{T}$ is a finite set of time points and $\mathcal{C}$ is a finite set of constraints of the form $t_1 - t_2 \in [l, u]$, with $t_1, t_2 \in \mathcal{T}$ and $l, u \in \mathbb{R} \cup \{+\infty, -\infty\}$.*

Intuitively, an STN represents a set of time-valued variables that need to be scheduled fulfilling all the constraints in $\mathcal{C}$. STNs are very popular in Temporal Planning as a way to represent the temporal constraints of a selected sequential plan. A common way of encoding a problem naturally expressed as intervals into an STN is by exploding each interval into its starting and its termination time points. An STN is a way of encoding a piece of knowledge on a set of time points. Given such a knowledge base, the network can be queried for *consistency*.

An STN is consistent if there exists an assignment $\mu : \mathcal{T} \rightarrow \mathbb{R}$, such that each constraint $c \in \mathcal{C}$ is satisfied by substituting each time point $t \in \mathcal{C}$ with $\mu(t)$ in c. An assignment μ that witnesses consistency is called "consistent schedule". Consistency corresponds to the first column in table 3.2, because no temporal uncertainty is present in plain temporal networks.

Intuitively, an STN can be seen as a region in the space $\mathbb{R}^{|\mathcal{T}|}$, and a consistent schedule is a point belonging to such a region. The problem is consistent if the region is non-empty. Moreover, it is easy to see that such a region is convex: the problem admits no disjunction and each constraint corresponds to the intersection of two half-spaces; therefore, the resulting region is the intersection of finitely many half-spaces.

The popularity of STN is given by two main factors. First, many useful scheduling problems can be accommodated in this framework. Second, checking the consistency of an STN is a tractable problem and efficient algorithms have been devised for both the decision problem and the consistent schedule synthesis problem.

As explained in the original paper that introduced STNs [DMP91a], the consistency problem can be stated as an all-pair-shortest path problem in a graph $G \doteq \langle V, E\rangle$, where $V \doteq \mathcal{T}$ and $E \subseteq V \times \mathbb{R} \times V$ is defined as follows.

- $\langle x, u, y\rangle \in E$ if $(y - x \in [l, u]) \in \mathcal{C}$ and $u \neq \infty$.
- $\langle x, -l, y\rangle \in E$ if $(y - x \in [l, u]) \in \mathcal{C}$ and $l \neq -\infty$.

G is called the distance graph of the STN. If G has no negative cycles, then the STN is consistent, otherwise it is not. A consistent schedule can be extracted from the all-pairs shortest paths by fixing an arbitrary time point $z \in \mathcal{T}$ to 0 and computing all the others assignments as the length of the path from z to each other node. Any all-pair shortest path algorithm (supporting negative edges) can be used to solve the problem, common choices are the classical Floyd-Warshall and the Johnson's algorithms [CSRL01], but other specialized techniques have been devised [XC03, PdWvdK12].

These techniques can efficiently solve the problem in a monolithic setting, in which a single STN is given and a single solution must be returned. However, the STN can be also solved in an incremental setting, in which network constraints can be added or removed and the network is queried for consistency multiple times in different states [CO96].

Disjunctive Temporal Network

A popular extension of STN, still belonging to the topmost leftmost cell of table 4.1, is the Disjunctive Temporal Network [SK00].

Definition 31 (DTN [SK00])**.** *A Disjunctive Temporal Network (DTN) is a tuple $\langle \mathcal{T}, \mathcal{C} \rangle$ where $\mathcal{T}$ is a finite set of time points and $\mathcal{C}$ is a finite set of constraints of the form $\bigvee_{j=1}^{h} t_{1,j} - t_{2,j} \in [l_j, u_j]$, with $t_{1,j}, t_{2,j} \in \mathcal{T}$ and $l_j, u_j \in \mathbb{R} \cup \{+\infty, -\infty\}$.*

A DTN extends an STN by allowing disjunctions in the set of constraints. Similarly to STN, also for DTN the relevant query is consistency defined analogously to STN, but in this case the region represented by the network is no longer guaranteed to be convex. This is because we allowed disjunctions, hence it is possible to describe regions with "holes" and even disjoint solution spaces.

Since the solution space for a DTN is no longer convex, the complexity of the problem immediately jumps to NP-hardness. In fact, all the efficient algorithms for STN rely on the convexity to quickly check consistency; given a DTN, instead, search is needed to deal with the Boolean structure of the constraints. Several techniques have been presented in the literature to solve the DTN consistency problem, that is relevant for many applications thanks to the high expressiveness of the formalism. A dedicated approach is implemented in the Epilitis solver [TP03] that uses a number of constraint propagation techniques to solve the DTN consistency problem. The TSAT++ solver [ACG99]. is a highly specialized SMT-solver for the $\mathcal{QF\_RDL}$ logic that is suitable to solve this problem.

Temporal Constraint Satisfaction Networks

There exists a third class of Temporal Networks that stays in between STNs and STNs. Temporal Constraint Satisfaction Problems (TCSN) have been introduced in [DMP91a] as a disjunctive generalization of STNs. Differently from DTN, a TCSN does not allow arbitrary disjunctions, but is limited to "binary" disjunctions, meaning that the variables appearing in each disjunctive constraint must be exactly two. It is thus impossible to express disjunctions involving three or more time points.

Definition 32 (TCSN [DMP91a]). *A Temporal Constraint Satisfaction Problems (TCSN) is a tuple $\langle \mathcal{T}, \mathcal{C} \rangle$ where $\mathcal{T}$ is a finite set of time points and $\mathcal{C}$ is a finite set of constraints of the form $\bigvee_{i=0}^{h} t_1 - t_2 \in [l_i, u_i]$, with $t_1, t_2 \in \mathcal{T}$ and $l_i, u_i \in \mathbb{R} \cup \{+\infty, -\infty\}$.*

Intuitively each TCSN constraint requires the temporal distance between two variables to lie within a disjunction of intervals. Even this restricted form of a DTN is NP-hard [DMP91a]. The consistency problem for TCSN is the same as for STN and for DTN: decide the existence (or find) an assignment to all the time points that fulfills all the problem constraints.

Minimal Networks

Apart for consistency, a common operation to be performed on temporal networks is the construction of the so-called minimal network.

Definition 33 (Minimal Network)**.** *A Temporal Network is minimal if $\mathcal{C}$ contains exactly one constraint for each pair of time points $t_1, t_2 \in \mathcal{T}$. Each constraint c is such that for pair of values v_1, v_2 satisfying c, there exists a consistent schedule for the problem that assigns $t_1 = v_1$ and $t_2 = v_2$.*

Intuitively, a minimal network is a formulation of the problem that allows the scheduling of a time point in a backtrack-free manner, requiring only a minimal propagation of information between two successive executions of time points. We do not enter in the details of minimal networks definitions and computation, we refer the interested reader to [DMP91a, PdWvdK12] for the STN case, to [DMP91a] for the TCSN case and to [BD11] for the DTN case.

4.2.2 Temporal Uncertainty

The temporal networks we presented so far and Allen's algebra have in common the determinism of the plant and of the scheduler. In fact, no uncertainty is modeled in the formalism nor in the problem being solved. For these reason, these formalisms all correspond to the deterministic-plant cell in table 4.1.

Despite the success of "plain" Temporal Networks, many interesting and hard problems are beyond the expressive power of STNs, TCSNs and even DTNs. In fact, all these networks rely on the assumption that all the time points are *controllable*: meaning that we can assign a time value to each of them. In real applications, however, some points might not be under the control of the scheduler, because they represent, for example, the termination of an activity having an uncontrollable duration. This scenario is common in planning, where time points are used to mark starting and termination times of activities. For example, the car trip time between two cities is an activity that can be started at any time, but whose duration significantly depends on the amount of traffic and the weather conditions.

In order to reason in this kind of situations, Temporal Networks with Uncertainty (TNU) have been presented [VF99a, PVYS07].

TNUs are TNs in which the time points are divided in two classes: controllable (also called "free") ($\mathcal{T}_f$) and uncontrollable ($\mathcal{T}_u$); and constraints are divided in requirements (free constraints) and assumptions (contingent links).

Intuitively, controllable time points are used to represent time instants that are decided by the scheduler, while uncontrollable time points represent events that can only be observed. Similarly, free constraints are the requirements that the scheduler aims to fulfill, while contingent links are the assumptions under which the uncontrollable time points might happen.

Differently from plain temporal networks, consistency is not the only query that can be asked for a TNU. In the literature, three different kinds of "controllability"[2] have been presented: strong, weak and dynamic [VF99b].

A network is *strongly controllable* if there is a solution that consists of a fixed, unconditioned, non-reactive assignment to each controllable time point that is guaranteed to satisfy the free constraints in the network regardless of how the uncontrollable time points turn out while respecting the contingent links. Such a solution corresponds to a time-triggered program, where activities are started at fixed times that are determined in advance of execution. This problem corresponds to the second column in table 3.2, characterized by the time-strong executor. In strong controllability, in fact, the executor is not allowed to depend on the activity duration in any way.

[2]In the CTN [TVP03] literature, they are called strong, weak and dynamic consistency, but the concept is the same.

In sharp contrast, a network is *weakly controllable* if there is a solution strategy that assigns values to the controllable time points as a function of the uncontrollable time points. Although the values for the uncontrollable time points need not be known when solving the problem, this version of controllability presumes that all such information is provided to the executor in advance of execution. In some sense, we can see a weak strategy as a clairvoyant program that schedules the controllable time points given a forecast of the uncontrollables outcome. Weak controllability corresponds to the fourth column in table 3.2, characterized by the time-weak executor. In weak controllability, in fact, the executor is informed a-priori of the duration of all the activities, and it can depend on such *prediction*.

Finally, a network is *dynamically controllable* if it has a dynamic execution strategy that can react to uncontrollable time points observations, but only those that have occurred in the past. In other words, the values that the execution strategy assigns to the controllable time points may depend on uncontrollable events, but only if that information has already been observed in real time. It cannot depend on advance knowledge of future uncontrollables. Typically, the execution strategy must be able to deal with the branching that derives from delays, and may interleave the start times of activities with the observation of uncontrollable time points. Dynamic controllability is the third column of the classification table, because the executor can depend on past observation (the observation history H_O in our formalism), but has no access to any prediction.

In the following we present three classes of TNU that are extensions of the STN, TCSN and DTN networks. For each class we present existing techniques for solving the controllabilities problems we explained.

Simple Temporal Networks with Uncertainty

A Simple Temporal Networks with Uncertainty (STNU) is a data structure for representing and reasoning about temporal knowledge in domains where some time points are controlled by the executor (or agent) while others are controlled by the environment[3]. Analogously to STNs, all temporal constraints in an STNU are binary and convex.

Definition 34 (STNU [VF99a]). *An STNU is a tuple $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ where:*

1. *$\mathcal{T}$ is a set of real-valued variables, called time points, that is partitioned into the sets, $\mathcal{T}_c$ and $\mathcal{T}_u$, of controllable and uncontrollable time points;*
2. *$\mathcal{C}$ is a set of binary constraints, each of the form, $t_1 - t_2 \in [l, u]$, for some $t_1, t_2 \in \mathcal{T}$ and $l, u \in \mathbb{R} \cup \{+\infty, -\infty\}$; and*
3. *$\mathcal{L}$ is a set of contingent links, each of the form, $\langle b, l, u, e \rangle$, where $b \in \mathcal{T}_c$, $e \in \mathcal{T}_u$, and $l, u \in \mathbb{R}$ with $0 < l < u$.*

A contingent link, $\langle b, l, u, e \rangle$, represents a temporal interval from b to e whose duration is uncontrollable, but bounded by $e - b \in [l, u]$. b is called the activation time point of the uncontrollable time point e and ins indicated with $\alpha(e)$.

STNUs have been successfully used in many application contexts [GL94, MMV01, CYF+15]; in general, they are useful to model a fixed set of activities or tasks (some of which having uncontrollable duration) subject to constraints. For example, STNUs can be used to represent temporal plans in AI planning.

The STNU framework has been presented in [VF99a], where the authors prove that the strong controllability problem is tractable and give an algorithm for reducing any given STNU to an STN having only controllable

[3]The agent and environment are not part of the formal STNU semantics; they are used here for expository convenience.

time points, in such a way that any consistent schedule for the STN is a solution for the strong controllability problem. This approach (that we indicate as FargierVidal) is based on removing all the uncontrollable time points and all the contingent links, and to substitute each occurrence of an uncontrollable time points in the free constraints according to a substitution table.

Given an STNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, the method returns an STN $\langle \mathcal{T}_c, \mathcal{C}' \rangle$ where the rewritten constraints $\mathcal{C}'$ are defined as follows. Given an uncontrollable time point e, we write l_e to indicate l and u_e to indicate u if $\langle \alpha(e), l, u, e \rangle \in \mathcal{L}$.

$$\mathcal{C}' \doteq \{\rho(c) \mid c \in \mathcal{C}\}$$

where:

- $\rho(t_1 - t_2 \in [l, u]) \doteq t_1 - t_2 \in [l, u]$ if $t_1, t_2 \in \mathcal{T}_c$;
- $\rho(t_1 - t_2 \in [l, u]) \doteq \alpha(t_1) - t_2 \in [l - l_{t_1}, u + u_{t_1}]$ if $t_1 \in \mathcal{T}_u$ and $t_2 \in \mathcal{T}_c$;
- $\rho(t_1 - t_2 \in [l, u]) \doteq t_1 - \alpha(t_2) \in [l + u_{t_2}, u + l_{t_2}]$ if $t_2 \in \mathcal{T}_u$ and $t_1 \in \mathcal{T}_c$;
- $\rho(t_1 - t_2 \in [l, u]) \doteq \alpha(t_1) - \alpha(t_2) \in [l + l_{t_1} - u_{t_2}, u - l_{t_2} + u_{t_1}]$ if $t1, t_2 \in \mathcal{T}_u$.

The intuition is that each uncontrollable time point is substituted by enlarging the bounds of the constraints it is involved into by considering the worst-case situations.

Concerning weak controllability, [VF99a] gives a co-NP algorithm for deciding the problem, but no synthesis algorithm for weak strategies is present in the literature. The algorithm follows from the following theorem (proven in [VF99a]) that gives an exponential way of checking weak controllability: it suffices to check the consistency of all the possible combinations of extreme values of contingent links.

Theorem 4.1 (Weak Controllability on Bounds)**.** *An STNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ is weakly controllable if and only if for any $\omega \in \{l_1, u_1\} \times \cdots \times \{l_n, u_n\}$ with*

$\mathcal{L} \doteq \{\langle b_1, l_1, u_1, e_1\rangle, \cdots, \langle b_1, l_1, u_1, e_1\rangle\}$, *the STN* $\langle \mathcal{T}, \mathcal{C} \cup \{e_i - b_i \in [\omega_i, \omega_i] \mid i \in [1, n]\}\rangle$ *is consistent.*

Finally, dynamic controllability has been widely studied for the STNU class. Starting from the original formulation in [VF99a] that proposed an exponential game-based solution, Morris, Muscettola and Vidal showed a pseudo-polynomial algorithm in [MMV01]. Then, Morris and Muscettola refined the idea into a polynomial algorithm [MM05] that has been further optimized by Morris in [Mor06] obtaining an $O(n^4)$ algorithm. The semantics of dynamic controllability has been studied by Hunsberger [Hun09] that proposed a view in which the solution strategy cannot react to uncontrollable decisions in zero time, but needs a non-null time to elaborate and react to the observation. These works are based on the definition of enriched networks of constraints that are propagated till a fixed point is reached following carefully designed rules. If the network does not enter in an inconsistent state, then the problem is provably dynamically controllable but no strategy synthesis is performed by these algorithms. [Mor06] and [Hun14] show how to write a reasoning algorithm that takes in input a propagated network and decides when to execute controllable time points at run-time. The problem with such algorithms is that they propagate constraints at run-time each time a time point is scheduled or observed. The complexity of the technique is polynomial.

Finally, we mention [Mor14] in which Morris proposed an algorithm for transforming a dynamically controllable STNU in an executable form (called dispatchable) that can be used for on-line execution with a minimal reasoning effort.

TCSNU and DTNU

Analogously to DTN, an important way of extending STNUs is to include disjunctive constraints. Disjunctions frequently arise in practice.

For example, workflows in the healthcare domain frequently involve activities whose executions cannot overlap due to conflicting requirements. An STNU cannot accommodate such disjunctive constraints. However, similarly to the Disjunctive Temporal Network (DTN) [SK00], *Disjunctive Temporal Networks with Uncertainty* (DTNUs) [PVYS07] can accommodate arbitrary disjunctions in the free constraints and binary disjunctions in the contingent links.

Definition 35 (DTNU [PVYS07]). *A DTNU is a tuple $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, where:*

1. *$\mathcal{T}$ is a set of time points, partitioned into controllable ($\mathcal{T}_c$) and uncontrollable ($\mathcal{T}_u$);*

2. *$\mathcal{C}$ is a set of* free constraints*: each constraint c_i is of the form, $\bigvee_{j=1}^{D_i} t_{1,j} - t_{2,j} \in [l_{i,j}, u_{i,j}]$, for some $t_{1,j}, t_{2,j} \in \mathcal{T}$ and $l_{i,j}, u_{i,j} \in \mathbb{R} \cup \{+\infty, -\infty\}$; and*

3. *$\mathcal{L}$ is a set of* contingent links*: each $l_i \in \mathcal{L}$ is of the form, $\langle b_i, \mathcal{B}_i, e_i \rangle$, where $b_i \in \mathcal{T}_c$, $e_i \in \mathcal{T}_u$, and $\mathcal{B}_i$ is a finite set of pairs $\langle l_{i,j}, u_{i,j} \rangle$ such that $0 < l_{i,j} < u_{i,j} < \infty$, $j \in [1, E_i]$ (E_i being $|B_i|$); and for any distinct pairs, $\langle l_{i,j}, u_{i,j} \rangle$ and $\langle l_{i,k}, u_{i,k} \rangle$ in $\mathcal{B}_i$, either $l_{i,j} > u_{i,k}$ or $u_{i,j} < l_{i,k}$.*

Generalizing the contingent links in an STNU, a contingent link $\langle A, \mathcal{B}, C \rangle$ in a DTNU represents a temporal interval from A to C whose duration, $C - A$, is uncontrollable, but guaranteed to lie within a union of disjoint intervals. In particular, if $\mathcal{B} = \{\langle l_1, u_1 \rangle, \cdots, \langle l_n, u_n \rangle\}$, then $C - A$ is guaranteed to fall somewhere within the set $[l_1, u_1] \cup \cdots \cup [l_n, u_n]$. Although contingent durations in a DTNU can be disjunctive in this way, the execution semantics ensures that the choices made by the environment for distinct contingent durations are independent.

This is useful to model periodic activities whose windows of opportunity have certain degrees of uncertainty. An STNU is the particular case of

DTNU in which all the $\mathcal{B}$ sets are singletons and $D_i = 1$ for each constraint belonging to $\mathcal{C}$.

An important subclass of DTNU (apart for STNU) is the Temporal Constraints Satisfaction Networks with Uncertainty (TCSNU), that analogously to TCSN limits the free constraints to be binary. Each free constraint in a TCSNU is in the form $\bigvee_{j=0}^{D_i} t_1 - t_2 \in [l_{i,j}, u_{i,j}]$.

Strong controllability. The strong controllability problem for DTNU has been addressed in [PVYS07]. The PVYS algorithm[4] can be described in terms of two nested enumerations. At the highest level, it explicitly enumerates every possible way (hereafter refereed to as *contingent choice*) to satisfy the contingent constraints. Intuitively, a contingent choice corresponds to picking, for each contingent link, one disjunct. For each contingent choice, PVYS obtains a *simple-natured* DTNU. In turn, the solution space (i.e. the set of strong schedules) of each simple-natured DTNU is represented as a DTN. The DTNs thus obtained are intersected, and result into a DTN that represents the solution space for the original DTNU. The innermost enumeration is used to convert each simple-natured DTNU, associated with the contingent choice μ_c, to the corresponding DTN. More specifically, PVYS explicitly enumerates every possible *free choice*, i.e. every possible way to satisfy each free constraint. Each free choice μ_f, in combination with μ_c, yields an STNU, which can be efficiently reduced to an STN by FargierVidal. All the STNs are then combined, by disjunction, into the DTN representing the solution space for the contingent choice μ_c.

Consider the following DTNU example, with $\mathcal{L} \doteq \{cl_1, cl_2, cl_3\}$ and

[4] The name PVYS is formed from the initials of the authors, and is used to refer to this algorithm throughout the thesis.

$\mathcal{C} \doteq \{cf_1, cf_2, cf_3\}$.

$$\begin{aligned} cl_1 &\doteq \langle b_1, \{\langle 10, 20\rangle, \langle 30, 40\rangle, \boldsymbol{\langle 70, 80\rangle}\}, e_1\rangle \\ cl_2 &\doteq \langle b_2, \{\boldsymbol{\langle 5, 8\rangle}\}, e_2\rangle \\ cl_3 &\doteq \langle b_3, \{\langle 1, 5\rangle, \boldsymbol{\langle 10, 15\rangle}\}, e_3\rangle \end{aligned}$$

$$\begin{aligned} cf_1 &\doteq (\mathbf{x_1} - \mathbf{x_2} \in [\mathbf{10}, \mathbf{30}]) \vee (x_1 - x_3 \in [3, 4]) \\ cf_2 &\doteq (\mathbf{x_3} - \mathbf{x_5} \in [\mathbf{10}, \infty)) \\ cf_3 &\doteq (x_2 - x_4 \in [20, 20]) \vee (\mathbf{x_1} - \mathbf{x_4} \in [\mathbf{10}, \mathbf{10}]) \end{aligned}$$

A possible contingent choice of contingent links is shown in blue, and a free choice is highlighted in red. A logical characterization of the algorithm is given below. Given a DTNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, the algorithm computes the final DTN as follows.

$$\bigwedge_{\mu_c \in \textsc{Choice}(\mathcal{L})} \bigvee_{\mu_f \in \textsc{Choice}(\mathcal{C})} \textsc{FargierVidal}(\mathcal{T}, \mu_f, \mu_c)$$

where the two calls to Choice are used to produce the free and contingent choices, and embody the two enumerations. Intuitively, the external conjunction iterates over the "blue" choices, while the internal disjunction iterates over the "red" ones.

The pseudo-code of PVYS, reported in algorithm 1 (continued in algorithms 2 and 3 for space constraints), has several optimizations with respect to the high-level view proposed above.

The top-level function DTNU-SC considers the problem constraints divided into three sets: C_S contains the simple constraints (i.e. constraints not containing disjunctions), and the disjunctive constraints that are defined over controllable variables only; C_C contains the disjunctive contingent links; C_E contains the other disjunctive free constraints. The procedure makes use of four additional data structures: the STN A, the STNU A_c, and the DTNs G and H.

Algorithm 1 PVYS algorithm (taken from Peintner et al. [PVYS07])

```
1: procedure DTNU-SC($A$, $A_C$, $C_S$, $C_C$, $C_E$)
2:     $S \coloneqq \emptyset$
3:     if $C_S = \emptyset$ then
4:         $G \coloneqq$ MinimalNetwork($A$)
5:         $S \coloneqq$ ALL-PATHS-SC($A$, $A_C$, $C_C$, $C_E$, $G$)
6:     else
7:         $C_i \coloneqq$ SelectVariable($C_S$)
8:         $C'_S \coloneqq C_S - \{C_i\}$
9:         for $c_{ij} \in$ Disjuncts($C_i$) do
10:            $A'_C \coloneqq A_C \cup c_{ij}$
11:            $A' \coloneqq A \wedge$ SCTransform($A'_C$, $c_{ij}$)
12:            if IsConsistent($A'$) then
13:                $S \coloneqq$ DTNU-SC($A'$, $A'_C$, $C'_S$, $C_C$, $C_E$)
14:                if $S \neq \emptyset$ then
15:                    return $S$
16:                end if
17:            end if
18:        end for
19:    end if
20:    return $S$
21: end procedure
```

In procedure DTNU-SC, the algorithm selects a combination of one disjunct for each constraint in C_S and accumulates the result in A. The constraints are rewritten one by one, using the approach by Fargier and Vidal: the function SCTransform takes a constraint and a STNU, and rewrites that constraint with respect to the STNU eliminating uncontrollable time points.

For each combination of disjuncts, the function ALL-PATHS-SC checks if the choice of free disjuncts satisfies all the contingent constraints by considering each possible combination of contingent disjuncts separately. For each combination, it accumulates the rewriting in the STN A and invokes

Algorithm 2 ALL-PATHS-SC sub-procedure of PVYS algorith

```
 1: procedure ALL-PATHS-SC($A$, $A_C$, $C_C$, $C_E$, $G$)
 2:     if $C_C = \emptyset$ then
 3:         $G := G \wedge$ SATISFY-Ce($A$, $A_C$, $C_E$)
 4:     else
 5:         $C_i :=$ SelectVariable($C_C$)
 6:         $C'_C := C_C - \{C_i\}$
 7:         for $c_{ij} \in$ Disjuncts($C_i$) do
 8:             $A'_C := A_C \cup c_{ij}$
 9:             $A' := A \wedge$ SCTransform($A'_C$, $c_{ij}$)
10:             if IsConsistent($A'$) then
11:                 $G :=$ ALL-PATHS-SC($A'$, $A'_C$, $C'_C$, $C_E$, $G$)
12:                 if $G = \emptyset$ then
13:                     return $\emptyset$
14:                 end if
15:             else
16:                 return $\emptyset$
17:             end if
18:         end for
19:     end if
20:     return $G$
21: end procedure
```

the function SATISFY-Ce that computes a DTN with all the possible solutions for the remaining free constraints. All the DTNs are accumulated by conjunction in G until either G becomes empty, meaning that there exist no solution that works for all the contingent disjunct combinations, or it contains at least on solution that is compatible with all the combinations and is returned. The algorithm terminates when a solution is found, or when all the combinations of C_S have been explored. The intermediate checks for consistency (via the function IsConsistent) are used for early-termination.

Algorithm 3 SATISFY-CE sub-procedure of PVYS algorithm

```
1: procedure SATISFY-CE($A$, $A_C$, $C_E$)
2:     $H = \emptyset$
3:     if $C_E = \emptyset$ then
4:         $H :=$ MINIMALNETWORK($A$)
5:     else
6:         $C_i :=$ SELECTVARIABLE($C_E$)
7:         $C'_E := C_E - \{C_i\}$
8:         for $c_{ij} \in$ DISJUNCTS($C_i$) do
9:             $A' := A \wedge$ SCTRANSFORM($A'_C$, $c_{ij}$)
10:            if ISCONSISTENT($A'$) then
11:                $H := H \vee$ SATISFY-CE($A'$, $A'_C$, $C'_E$)
12:            end if
13:        end for
14:    end if
15:    return $H$
16: end procedure
```

Weak and dynamic controllability. No algorithms or techniques exist in the literature for solving the dynamic controllability problem of the DTNU problem class. [VVPYS10] presents an algorithm for dynamic controllability that is limited to the TCSNU class and is based on a Meta-CSP exploration and a decision procedure for the weak controllability of DTNUs. Both these approaches are mainly theoretical and are limited to the decision problem. This means that they are not concerned with the synthesis of a strategy (in our execution model, a plan) for scheduling the time points, but only in verifying that such a plan exists at all.

4.2.3 Discrete Non-Determinism

Following the classification in table 4.1, we covered the deterministic row, and we are now left with the non-deterministic one. In scheduling, activities are often decoupled from their effect as the constraints are used to express them. If the effect of an activity is non-deterministic, however, we need ways to reason on the possible outcomes and their impact on future activities. In this section we discuss the existing literature on network with and without temporal uncertainty having support for non-deterministic outcomes.

Conditional Temporal Networks

Conditional Temporal Networks have been introduced to model situations in which part of the constraints are activated or de-activated depending on a Boolean, non-deterministic run-time observation [TVP03].

CTNs extend Temporal Networks by introducing Boolean conditions that are attached to special time points, called observation nodes. A *scenario* specifies the truth values of the Boolean conditions that have been observed so far. Propositional *labels* comprising conjunctions of (positive or negative) propositional letters can be attached to time points and temporal constraints in a CTN. A time point with a propositional label ℓ is only executed in scenarios where ℓ is true; a constraint labeled by ℓ only applies in scenarios where ℓ is true.

Definition 36 (Labels [TVP03])**.** *Given a set P of propositional letters, a* label *is any (possibly empty) conjunction of (positive or negative) literals from P. The* label universe *of P, denoted by P^*, is the set of all labels with literals drawn from P.*

Definition 37 (CSTN [TVP03])**.** *A* Conditional Simple Temporal Network *(CSTN) is a tuple, $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$, where:*

1. *P is a set of propositional letters;*
2. *$\mathcal{T}$ is a set of time points;*
3. *$\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time points;*

4. *$\mathcal{O} : P \to \mathcal{OT}$ is a bijection that assigns observation time points to propositional letters;*

5. *$L : \mathcal{T} \to P^*$ is a function assigning labels to time points; and*

6. *$\mathcal{C}$ is a set of* labeled *temporal constraints of the form, $\langle t_1 - t_2 \in [l, u], \ell\rangle$, where $t_1, t_2 \in \mathcal{T}$, $l, u \in \mathbb{R} \cup \{\infty, -\infty\}$, and $\ell \in P^*$.*

Intuitively, a CSTN is an STN in which each temporal constraint may be associated with a Boolean label and labels are observed by some specific time points. The idea is that constraints are enabled (and shall be satisfied) only in scenarios where the constraint label is satisfied. Using feature, we can enable or disable part of the temporal network depending on the scenario.

Analogously to the uncontrollable durations for a TNU, the scenario defines the non-deterministic part of the problem and may or may not be observable. Three possible queries are defined for a given CSTN.

Strong consistency is the problem of finding a single, fixed schedule that fulfills all the constraints in every possible scenario. This means that without observing the Boolean labels we want to find an assignment to the time points that fulfills all the constraints in every possible scenario. It is possible to prove [TVP03] that a CTN is strongly consistent if and only if the corresponding STN without uncertainty (obtained by disregarding the labels of each constraint) is consistent. The intuition is that strong consistency is essentially the problem of fulfilling all the constraints because we assume to have no observation on the labels. Hence, this means that we want a single assignment that fulfills all the constraints (a consistent schedule).

Weak consistency is the problem of finding a strategy for scheduling all the time points assuming given any possible scenario. This is the analogous of weak controllability of temporal networks.

Similarly to dynamic controllability, a CSTN is dynamically consistent if there is a runtime strategy for scheduling the time points fulfilling all the constraints in every scenario, assuming that only the past part of the scenario is known.

Apart for proving that strong consistency coincides with plain consistency, [TVP03] presents algorithms for weak and dynamic consistency checking: weak consistency is checked by considering every possible scenario and solving the consistency problem for each of them. Instead, dynamic controllability is reduced to an exponential-size DTN that represents the space of all the solutions. The problem is dynamically consistent if and only if the resulting DTN is consistent, and a propagation algorithm is presented to execute the CSTN dynamic strategy at runtime, but this requires runtime propagation of constraints on a DTN. No explicit synthesis technique is available in the CSTN literature.

In addition to CSTNU, it is easy to extend definition 37 to add disjunctive constraints analogously to TCSN and DTN, obtaining CTCSN and CDTN. The algorithms presented in [TVP03] are general enough to cover also these disjunctive cases, but no other work in the literature covers them.

Conditional STNU

Recently, some research has been devoted to combine the expressiveness of CTN with TNU. The resulting formalisms natively express both temporal uncertainty and discrete non-determinism in a single, compact temporal reasoning problem.

A CSTNU augments an STNU to include propositional letters that represent Boolean conditions whose truth values are observed in real time, during execution (analogously to CSTN).

Definition 38 (CSTNU [HPC12])**.** *A* Conditional Simple Temporal Network with Uncertainty *(CSTNU) is a tuple,* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \mathcal{L} \rangle$*, where:*

1. $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ *is a CSTN, and*

2. *ignoring any labels,* $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ *is an STNU.*

The CSTNU formalism is useful when the modeled situation has multiple evolution scenarios depending on some observation at runtime. While STNUs are suitable for representing temporal plans, CSTNUs can be used to model temporal conditional plans.

In principle, 9 possible queries can be checked on a CSTNU as shown in table 4.1 derived from our execution model. In fact, we could check if a CSTNU is dynamically controllable but strongly consistent, meaning that we would look for a solution that is fixed and unobserving in the space of the label scenario, but dynamic in the uncontrollable durations.

However, the only problem definition in the literature is in [HPC12] that defines "dynamic controllability" of CSTNU as the problem of checking the existence of a runtime strategy for both the durations and the labels. The paper defines the problem and introduces a solution technique based on constraint propagation that is sound but incomplete.

The other eight queries can be obtained by combining strong, weak and dynamic consistency and controllability, but to best of our knowledge they are not defined in the literature.

Chapter 5

Planning Classification

Temporal planning is the problem of synthesizing a plan for achieving a specified goal given a formal description of the domain and the initial situation. Intuitively, planning is an abstract reasoning task that involves a mathematical representation of the plant and is devoted to the synthesis of a strategy for bringing the plant in a desired state in time. Differently from the scheduling problems we presented so far, the planning problem instances do not specify the set of actions to be executed a-priori, it is the task of the planner to find a suitable set of actions and to find a good schedule to achieve the high-level goal.

The planning literature is vast and heterogeneous: different planning languages and representation formalisms exists. In this chapter we survey the existing planning techniques for temporal and non-deterministic domains.

Also, the domain of planning can be partitioned using the same schema we introduced in table 3.2: depending on the *model* of the domain and the type of plan that needs to be generated we recognize the various features we discussed in chapter 3. The resulting classification is reported in table 5.1 where we list the problems that have been addressed in the literature in the realm of temporal planning.

			Action Duration			
			Controllable	**Uncontrollable**		
				No Observation	**Dynamic Observation**	**Future Observation**
Plant Evolution	**Deterministic**		Temporal Planning		Dynamic Temporal Planning with Duration Uncertainty	
	Non-Deterministic	**No Observation**				
		Dynamic Observation			Probabilistic Temporal Planning	
		Future Observation				

Table 5.1: View of the currently available approaches with respect to discrete non-determinism and temporal actions.

In the following, we survey existing techniques for the problems reported in table 5.1.

5.1 Temporal Planning without Uncertainty

A vast amount of work has been devoted in the development of techniques for temporal planning. In its simpler form, temporal planning can be seen as the extension of classical planning in which actions are no longer assumed to be instantaneous. In fact, while classical planning models actions as atomic, instantaneous changes of the world, temporal planning takes into account the *duration* of such actions. This extension is everything but trivial: an action having a non-zero duration can be subject to a range of issues that are not present in classical planning.

The first issue arises in the context of this extension is concurrency: while in a classical setting two actions can be executed at the same time if they are not interfering, in a temporal setting, the possible situations are much more diverse. Two actions can be executed simultaneously or (partially) overlap, effects can happen at the beginning, at the end or even during an action execution. All of this, yields a complex interaction between different actions during execution.

A second issue concerns the executability of actions. While in classical planning actions have preconditions that must be true in order to allow the correct execution, the situation is more complex in a temporal setting. An action may require some condition to be started, another condition to terminate and can also require to maintain a condition for an interval (or a sub-interval) of time during the action execution. Clearly, it is possible to check a condition before starting an action, but if the action requires a condition to be terminated, the execution is committed on a future condition before starting an action. This is problematic for both reasoning and execution: the conditions are not "local" to the starting of actions, but need to be represented and verified in time, and the executor needs to monitor conditions to ensure the plan constraints are respected.

Finally, since actions are immersed in time, temporal constraints between actions may be required. For example, an action may require another action to be executed after 10 time units or to be executed during a specified period. This requires the *scheduling* of the actions; in fact temporal planning is also called *planning and scheduling*.

The rest of this section is organized as follows: we first discuss what a plan is in a temporal setting, then we present the major temporal planning languages and finally we survey the techniques that arose in the literature for solving the temporal planning problem.

5.1.1 Temporal Plans

The solution of a classical planning problem is a sequence of actions. Similarly, a possible solution for a temporal planning problem is a set of actions with an associated duration and starting time. This kind of temporal plans, called time-triggered, are simple to represent and to simulate, but are limited in the actual applicability. This is because they require the ability to start an action exactly at a given time and to terminate it exactly at a given duration: if one of this events is subject to a delay, the entire plan could be invalidated.

Definition 39. *A time-triggered plan is a finite set of triples* $\langle t, a, \delta \rangle$ *where:* $t, \delta \in \mathbb{R}$ *and* a *is an action.*

Other types of temporal plans are also possible: flexible temporal plans are temporal plans in which the set of actions is fixed, while starting times and duration are not. The temporal constraints for ensuring plan validity are encoded as a Temporal Network that needs to be executed at run-time. This kind of plans are common in timeline-based planners and often used in practice. However, the execution of a flexible plan requires an executor able to efficiently solve STNs at run-time to decide which action should be started and when.

5.1.2 Planning Language Classification

In this thesis we focus on domain-independent planning, that is planning when the domain description is provided as input and is not hard-coded in the planner itself. In order to have a domain-independent planner, a planning language is needed: in fact, we need ways to express the available actions to modify the world, the constraints that are required to hold, the initial state, the goal and any other relevant information for the planning reasoning.

Two main families of temporal planing languages and techniques exist, namely action-based and timeline-based. Action-based languages use the concept of action as a primitive operator available to the agent to change the state of the system or the world. These languages are rooted in the traditional classical planning formalisms deriving from STRIPS [FN71].

Timeline-based languages are more similar to a sequential generalization of a Constraint Satisfaction Problem: a set of constraints describe the possible evolution of the system and the environment and no explicit concept of action or state is present.

In this section, we survey the planning languages available for temporal planning.

PDDL. The Planning Domain Definition Language (PDDL) is the action-based language of the International Planning Competition (IPC) and was first conceived by Drew McDermott [McD00]. The language builds upon STRIPS [FN71] and ADL [Ped89]. In its original version, PDDL is only concerned with classical planning.

Over the years, several variations and improvements of PDDL have been presented. A famous and widely-used extension is PDDL 2.1 [FL03] that, among other innovations, introduced the support for temporal planning and a limited form of continuous change. PDDL 2.1 models actions that have non-instantaneous duration: effects can happen at the beginning or at the ending of a durative action, and conditions can be expressed as instantaneous requirements for starting and for terminating an action. Moreover, PDDL 2.1 also supports `overall` conditions, that are conditions that are required to hold during the entire execution of an action.

Another version of PDDL, PDDL 2.2 [EH04], added support for Timed Initial Literals (TIL). A TIL is a proposition (or its negation) annotated with a timestamp. The intuitive semantics is that the proposition will become true at the time indicated by the timestamp. This construct is particularly useful to express predictable exogenous events such as the visibility window of a satellite, the day/night alternation and so on.

Finally, PDDL 3 [GHL^{+}09] is another relevant update of the PDDL language that introduced timed goals and preferences. While in other PDDL versions goals are just conditions to be eventually reached by the plant, PDDL 3 allows time intervals associated to each goal, in order express conditions that must be reached during specific portions of time.

NDDL. The New Domain Definition Language (NDDL) is a language for temporal planning developed by NASA [BWMB$^+$05]. The philosophy behind NDDL is profoundly different from PDDL. NDDL considers state variables that evolve in time and are (temporally) constrained by a set of rules. In this setting, no distinction is made between conditions and effects, nor between facts and goals: the problem is given by a set of rules that model the possible behaviors and a set of temporally-instantiated assertions that are the facts and the goals. The task of the planner is to fill the gaps finding a suitable evolution that has no contradictions. NDDL is the language of the EUROPA [FJ03] planning and scheduling framework.

The Advanced Planning and Scheduling Infrastructure (APSI) [CCF$^+$09] is a planner having a language very similar to NDDL, that is in use by the European Space Agency.

ANML. The Action Notation Modeling Language (ANML) is the designed successor of NDDL, and it is also developed by NASA [SFC08]. ANML is a modern, user-friendly language that tries to bridge the gap between PDDL and the timeline languages such as NDDL. ANML is an action-based, variable-value language that allows high-level modeling, complex conditions and effects, HTN decomposition and timeline-like temporal constraints. Compared to PDDL 2.1, ANML allows for conditions and effects to be specified in any sub-interval of an action and general formulae are allowed as conditions.

5.1.3 State Space Temporal Planning

State-space planning is the current de-facto standard for solving classical planning problems in the context of PDDL. This technique typically involve a best-first search (usually A* or IDA*) in the domain explicit state space. Two flavors of the technique are possible: forward and backward. In a forward search, the starting node is the initial state and the objective is to construct a sequence of states linked by actions that yield to a goal state. In the backward approach, the search is rooted in the goal state and proceeds by applying the actions in reverse order. The key ingredient to obtain good performance from this approach is the heuristic guiding the search process. In the literature, an impressive amount of work has been devoted to find effective heuristics for the classical planning problem yielding fast and robust planners.

Lifting the state-space idea to the temporal planning case, is not trivial as it may look like. In fact, while the search space for a classical planning problem is finite (but exponential in the number of propositions), the search space for a temporal planning problem is infinite due to the density of time: an action can be started at any time and even a small change in the starting time or the duration can yield a significant alteration of the plant state. Two lines of research arose in the literature in this context.

SAPA [DK03] is a temporal planner that is able to reason on a limited notion of time and resources efficiently. The idea of the approach is to extend the concept of state adding time-stamps. A state is composed of a complete assignment to the problem fluents, an absolute time t and an "event queue" of events and conditions imposed by actions that have been started at time t but that have not terminated yet. Searching in the space of such states is similar to the search in classical planning, with some modifications. In fact, once a durative action is applied, the starting conditions are checked and the starting effects are immediately applied in the generated state, but the overall conditions and the ending conditions and effects are added to the event queue as they are seen as commitments on the future.

In order to allow for the advancement of time, a special action "advanceTime" is added. The purpose of such an action is to increment the absolute time of the state without changing the fluent values. In theory, one could advance the time of any real value, but SAPA takes the view that actions can be started only in "decision epochs" that are time points in which something changes. In practice, SAPA advances the time to the earliest event in the events queue. This behavior yields a great advantage, as it removes the need to represent and reason on an infinity of possible time points where actions can be started, but surrenders completeness. In fact, as shown in [CKMW07] the approach used by SAPA (and the other "decision epoch" planners) is able to find plans with limited forms of concurrency.

Another line of research studying the integration of classical state-space planning and scheduling is polarized toward the use of a symbolic representation of time and temporal constraints with an explicit representation of the discrete state of the system. In particular, the key idea in this context is to explode each durative action in two instantaneous actions (called "snap actions") representing the starting and ending time points of the original activity[1]. The overall approach is depicted in figure 5.1.

The general idea is to abstract away temporal information from the domain, leaving only the snap actions (with an implicit constraint that the ending action must be executed after the starting one). This abstraction,

[1]This approach is rooted in the PDDL 2.1 formalism that only allows effects at the starting and ending time points of an action; for this reason only two time points for each action are needed. In ore expressive formalisms, we need one snap action for each time point in which an effect can happen or a durative condition starts or ends.

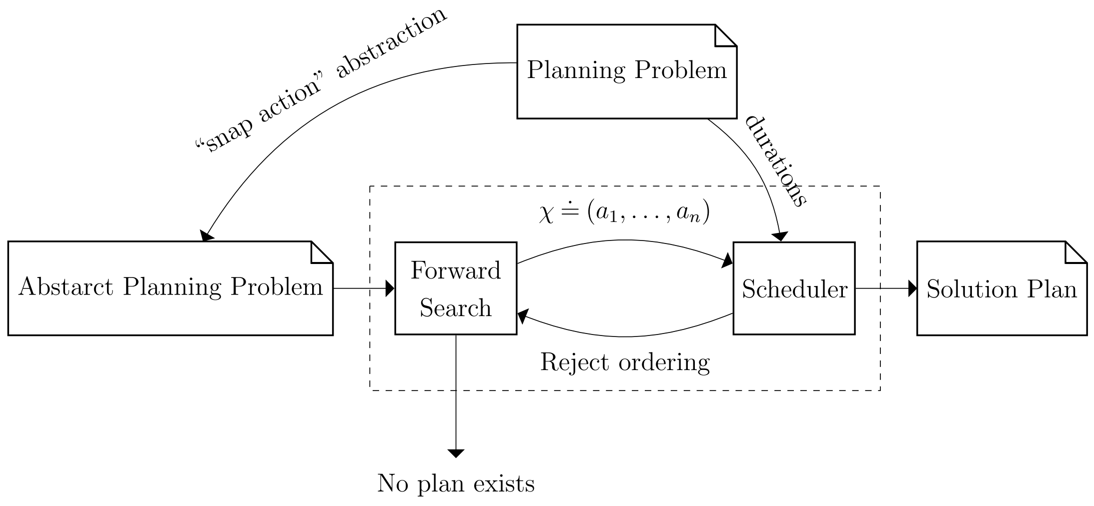


Figure 5.1: High-level architecture of a Forward State Space Temporal Planner. The original planning problem (above) is abstracted in a classical planning problem that is fed into a planner. Each time a plan χ is generated by the forward search, it must be enriched by adding the duration and precedence constraints and checked for schedulability. If it is schedulable, the plan is returned, otherwise the forward search is asked for another plan. If no plan remains in the abstract space, then no temporal plan is possible.

leaves us with a classical planning problem that can be solved by a forward search planner. However, each time a discrete plan χ is generated from the forward search, it must be checked for temporal feasibility. This is done in two steps, as detailed in algorithm 4. Each action in χ is considered a time point, then each pair of snap actions is constrained to the duration of their original activity. Moreover, we need to ensure that the ordering between the time points is such that no cause-effect relation established by the forward search is violated incompatibly with the domain description. For this reason, a set of precedence constraints are added to the Simple Temporal Network representing the plan. The network is then checked for consistency. If it is found consistent, then a plan based on the consistent schedule of the network, is created and returned; otherwise, the discrete plan χ is rejected and the forward search is required to find another plan. If no plans are left in the discrete space, then no plan exists for the temporal planning problem.

Algorithm 4 The Forward State Space Temporal Planning (FSSTP) Framework

```
 1: procedure FSSTP($\mathcal{P}$)
 2:     for all abstract plan $\chi$ generated while solving $abstract(\mathcal{P})$ do
 3:         $D := \textsc{Durations}(\chi, \mathcal{P})$
 4:         $P := \textsc{Precedences}(\chi, \mathcal{P})$
 5:         if $\mu := \text{TP.}\textsc{Solve}((\chi, D \cup P))$ then
 6:             if $\textsc{IsComplete}(\chi)$ then
 7:                 return $\textsc{BuildTemporalPlan}(\mu, \chi)$
 8:             else
 9:                 $\textsc{continue}(\,)$
10:             end if
11:         else
12:             $\textsc{reject}(\chi)$
13:         end if
14:     end for
15:     return $\perp$
16: end procedure
```

The general schema resembles a typical abstraction-refinement loop, but for this planning problem we need to be careful on some details. First of all, the forward search must be slightly adapted with respect to the one employed in a classical planner. In fact, the states cannot be cached as simple assignments of values to the propositions, but the plan leading to the state must be also part of the state representation. This is to avoid problems when a state is reached by a plan that is not schedulable first, but can also be reached by a schedulable plan. Another point is termination: this schema is guaranteed to find a temporal plan if it exists assuming that all the plans of any length are eventually generated by the forward search, but it is not guaranteed to terminate if no plan exists. In fact, there could be infinitely many plans of increasing length leading to a goal and it might be the case that none of them is schedulable, yielding an infinite loop. Another customization of the search, implicitly imposes the `overall` conditions (that must be kept from the starting to the ending of a durative action) without explicitly stating them as axioms in the abstracted planning problem. This guarantees faster reasoning on this kind of constraints.

Finally, there is an interesting point about the precedence constraints. The forward search produces a total order on the snap actions that lead to a discrete state fulfilling the goal. The job of the scheduler is to relax this total order without invalidating any action condition. For doing so, the scheduler encodes a set of needed precedence constraints in the STN that is then checked for consistency. This order lifting is an interesting point of the approach that can be done in different ways. The simplest approach is to keep the total order intact, but this can cause a high number of rejections. Another possibility is to analyze the plan and add only the needed precedence constraints for keeping the causal-effect relation produced by the forward search intact: this yields a partial order in the STN.

A series of planners implements these ideas. Crikey [CFH$^+$09] was the first implementation of this approach, it uses a peculiar three-actions representation (one action was used for the overall constraint) and a partial order lifting derived from [VPC90]. Its successor, Crikey3 [CFLS08] employs a total order lifting instead.

TEMPO [CKMW07] uses an approach similar to Crickey, but avoids the explicit snap-actions encoding by keeping a record of the started actions in the state representation as an “agenda” of started actions.

POPF [CCFL10] also uses a partial order lifting, but has many optimizations on the heuristic for the abstract search and used two snap actions only. Finally, COLIN [CCFL12] uses a total order lifting for the time points and a linear programming problem instead of an STN to accommodate linear continuous change of fluents in addition to temporal constraints.

Other temporal planners can be classified as state-space.

TALPlanner [KD00] (Temporal Action Logic Planner) is a forward chaining planner. Similarly to SAPA, TALPlanner also uses the decision epoch simplification that can deal with a limited form of concurrency. The peculiarity of TALPlanner is the use of Temporal Action Logic as a mean to represent the domain, the goals and also the plan. The planner heavily relies on domain specific knowledge expressed by the user in the form of Temporal Action Logic formulae.

SGPlan [CWwH06] uses a divide-et-impera technique by first solving sub-goals and by then combining the results in a global plan.

TLPlan [BA01] and TP4 [HG01] are another examples of forward chaining planners that use approaches similar to the one adopted by SAPA.

5.1.4 Plan Space Temporal Planning

Another relevant approach to solve temporal planning problem is to search in the space of plans. This approach was studied also in the context of classical planning [GNT04] but recently it was outperformed by state-space search. In temporal planning, the relationship between the two approaches in not so well defined yet. Many practical planners employ plan-space search as their engine.

The general algorithm for plan-space search is reported in algorithm 5. Intuitively, the algorithm searches in the space of partial plans: a partial plan is a set of actions subject to a set of constraints. Starting from the empty plan, the algorithm incrementally develops a plan that satisfies the goals by solving one "flaw" at a time using a series of "refinements". A flaw, in the partial order planning jargon, is either an unsatisfied action precondition, an unsatisfied goal or a violation of a causal link [GNT04]. This schema is completely equivalent to the classical planning case, but this is due to the fact that the INCONSISTENT, SELECTFLAW and the re-

Algorithm 5 The Plan Space Temporal Planning framework

```
procedure PSTP($\mathcal{P}$)
    Solve($\emptyset$, $\mathcal{P}$)
end procedure
procedure Solve($\pi$, $\mathcal{P}$)
    if Inconsistent($\pi$) then
        return $\perp$
    end if
    $f$ := SelectFlaw($\pi$, $\mathcal{P}$)
    if $f = \emptyset$ then
        return $\pi$
    end if
    for all possible refinement $\pi'$ of $f$ do
        $r$ := Solve($\pi'$, $\mathcal{P}$)
        if $r \neq \perp$ then
            return $r$
        end if
    end for
    return $\perp$
end procedure
```

finement generator procedures hide all the complexity deriving from the introduction of time. Clearly, Inconsistent must be able to perform a schedulability check analogously to the one performed via STN in algorithm 4. Another point is the way precedence constraints and causal links are stored: often in the temporal setting the ordering constraints are substituted by a STN that keeps track of the metric constraints for the partial plan.

The first planner of this kind was ZENO [PW94]. ZENO was an ambitious attempt to deal with a very expressive problem specification including time and a wide variety of resources. Differently from SAPA and other state-space planners, ZENO is provably sound and complete for temporal planning. ZENO implements an uniform representation of time and resource constraints in the form of a linear programming problem and uses the Simplex algorithm to check for inconsistencies.

VHPOP [YS03] is another successful planner based on this scheme. VHPOP supports both grounded and lifted reasoning and offers a variety of plan selection and flaw-selection heuristics. VHPOP uses STNs to keep track of the temporal information in the partial plans.

ASPEN [CRK$^+$00] is a complex and practical framework developed by NASA that allows very expressive plan domains to be analyzed. ASPEN uses iterative repair [ZDDD93] to plan and re-plan. This is similar, even if not equivalent, to plan-space search.

Timeline planning. A different, yet very successful, realm of planning techniques is the one of the so-called timeline-based planners. Differently from the systems we presented so far, timeline planners use a problem representation that is more similar to a Constraint Satisfaction Problem (CSP) than to a classical planning domain. In fact, timeline planners do not distinguish between preconditions and effects, nor between facts and goals: they simply impose constraints on variables that need to be satisfied in order for the action to be applied. The philosophy behind this idea is that temporal planning is a natural extension of scheduling to control dynamic systems.

One can imagine a timeline planner as a technique to automatically fill a Gantt chart using some rules. Some activities are imposed on the chart, those are facts and goals, the job of the planner is to fill the chart using some user-provided rules, the domain specification.

The Heuristic Scheduling Testbed (HSTS) [MSCD91] was the first system of this kind. It formalized the concept of timeline and introduced a number of innovations and opened a new area of research for planning and scheduling. IxTeT [GL94] is another system based on this approach that has been used in many practical applications. We will discuss IxTeT also in the next section, as one incarnation of IxTeT also supports temporal uncertainty.

EUROPA [FJ03] is the successor of HSTS and provides a rich framework for the modeling and the solution of timeline planning problems. Europa is more than a domain-independent planner, it is a software architecture that allows the development of planning and scheduling applications also supporting optimization. EUROPA is an open-source project that is still running.

APSI [CCF$^+$09], is a framework similar to EUROPA in use by the European Space Agency.

Verfaille et al. [VPL10] proposed a similar CSP-based approach for space satellites scheduling. The formalisms used in this work is a middle-ground between scheduling and planning, because these timelines cannot describe generally evolving systems, but only a finite number of activities subject to temporal constraints.

5.1.5 Planning as Satisfiability

Following an approach similar to the successful SATPlan [KS92] for classical planning, it is possible to leverage the expressiveness of the SMT framework to address the temporal planning problem. The first research on this topic can be found in [SD05].

The idea is to encode all the valid plans of finite length k as a single SMT formula. If the formula is unsatisfiable, there are no plan of length k, otherwise a model of the formula yields a valid plan.

The use of arithmetic theories provides an easy way of encoding time; however, this kind of planners are still under-represented in the literature and in the available set of tools.

We do not report here the details of the encoding, we refer the reader to [Rin15b] and [Rin15a] for a recent an in-depth discussion.

The use of arithmetic theories provides an easy way of encoding time; however, this kind of planners are still under-represented in the literature and in the available set of tools.

5.1.6 Planning Graph Derivations

GraphPlan [BF97] is an influential and fundamental technique in classical planning. Unsurprisingly, several works went in the direction of extending the GraphPlan algorithm for the temporal planning case.

The Temporal Graphplan (TGP) [SW99] algorithm extends the GraphPlan idea to the temporal case by generalizing the mutual exclusion concept to deal with durative actions.

Local search Planning Graph (LPG) [GSS03] uses Temporal Action Graphs (TA-graphs) as search nodes for a stochastic local search. Each node intuitively corresponds to an abstract plan that can be refined, if needed, in several ways. If the search finds a non-spurious node it terminates and a plan can be extracted from the TA-graph.

TPSys [GFL02] is another tool that uses an extension of the planning graph for the temporal case. Similarly to TGP, it also represent the temporal information in an enriched form of planning graph.

The Linear Programming and Graph Plan (LPGP) [LF03] tool combines a planning graph search for the abstraction of the domain obtained by discarding temporal information with linear programming, that is used to solve the temporal constraints.

5.2 Beyond Temporal Planning

As clearly evident from table 5.1, there are few works in the literature that are beyond temporal planning according to our classification scheme.

In fact, the existence of non-deterministic action effects is a well-studied problem in planning without time, but to the best of our knowledge the temporal case is an unexplored problem in the literature.

IxTeT [GL94] is a system that is able to reason on uncontrollable action durations, by incrementally searching for a plan expressed in the form of an STNU. At each step, the planner checks if the STNU is dynamically controllable and backtracks if it is not. In this way, IxTeT is able to produce a dynamic plan that is guaranteed to achieve the goal given an executor that is able to schedule an STNU (for example using the Morris algorithm [Mor06]). Compared to the formalization we gave in chapter 3, IxTeT does not address the full "Duration-only Plant, Dynamic Executor" class, because the plans generated by IxTeT have a fixed set of actions to be executed: only the timing of actions is dynamic. On the other side, the general form of the "Duration-only Plant, Dynamic Executor" class allows the executor to observe the action durations and to possibly execute *different of actions* depending on the observation. Nonetheless, IxTeT is a very important and successful planner in this context.

A recent planner that takes an approach similar to IxTeT to deal with uncontrollable action durations is FAPE [DBMIG14]. FAPE is a planning architecture designed to work in a closed loop with an executor: the planner is used as a deliberation system that produces a (possibly dynamic) plan and is invoked each time the conditions for which the plan was generated change. FAPE can use different planning algorithms to produce a plan structure that is then scheduled by a dynamic controllability solver. If the plan is dynamically controllable, then a constraint network is passed to the executor that has the responsibility of scheduling the actions.

Recent research ignored the uncertainty dimensions we proposed focusing on temporal planning in the controllable deterministic case. The practical problems arising from uncontrollability of the action durations and the non-deterministic effects are dealt in practice with a re-planning scheme. A planner is employed as an on-line component that is required to generate a new plan form the current state of the system each time the executor monitor realizes that something is non behaving as predicted by the previous plan. This scheme can be seen as an optimistic planning that is iterated each time something goes wrong.

Apart for the aforementioned techniques, no other works addressed the issue in the realm of qualitative uncertainty. Instead, some related work is present in the field of probabilistic planning, that is when temporal uncertainty and uncontrollability are not considered as pure non-determinism, but probabilities distributions are attached to the model. This is very different from the planning model we discussed so far and form the scope of this thesis. In fact, probabilistic planning associates a probability distribution on the possible action durations and on the possible non-deterministic outcomes and aims at producing a policy that maximizes a metric function (usually the likelihood of plan success). In this field, the Probabilistic Temporal Planning problem [MW08] can be imagined to address the problem of dynamic controllability for uncontrollable action duration and dynamic observation for non-determinism.

Planners such as MOP [ATZ04], DUR [MW08] and Prottle [LAT05] have been developed to solve this problem, but their analysis is beyond the scope of this survey.

Chapter 6

Extensions

In this part, we analyzed and classified the landscape of scheduling and planning techniques when time is considered. We focused on two dimensions, namely the action durations and the presence or absence of non-determinism, combined with the executor observation capabilities. However, this classification has several possible extensions in orthogonal directions. In this section, we list and briefly discuss some of such features that have been presented in the literature.

6.1 Flexibility

Our execution model focuses on the concept of controllability that is a long standing problem in AI planning and is recognized as a still-open challenge [BDM$^+$02]. Nonetheless, several approaches addressed the executability issues of temporal planning in a different way, called flexibility.

Flexibility is often used as a synonymous of least commitment planning in the temporal case. The idea is to generate a plan in a fully deterministic setting without instantiating the time points, but leaving to the executor the task to schedule them. In this way, instead of producing a single time-triggered plan, the planner produces a bag of plans that differ only for the timing of the actions. The executor can then (incrementally) select which plan to apply, possibly adapting its decision to contingencies such as a delay or a tightening of a deadline observed at runtime. Usually, a (Simple) Temporal Network is used to compactly represent the set of plans to be executed.

This sounds similar to the reason why controllability has been introduced in the first place, but we remark two important differences.

1. There is no formal guarantee that a flexible plan will adapt to a runtime situation different from the nominal case. In fact, no modeling of the uncertainty is present and the planner only tries to limit commitments in a greedy way: in general, no formal guarantees are given.

2. Flexibility is very useful in practice as it limits the planner commitments and allow for soundly tackling behaviors that arise in real world but are out of the modeled reality. Hence, flexibility, differently from controllability, addresses behaviors arising when the model is not aligned with the real world.

Therefore, we argue that controllability and flexibility are two orthogonal concepts: controllability guarantees plan executability and goal achievement with respect to the behaviors encoded in the formal model, flexibility tries to limit the commitment of the plan by greedily dealing with as many non-nominal behaviors as possible.

Finally, we highlight that the two ideas are not self-contradicting: it is possible to conceive a flexible controllable plan, and we believe this is an important future development that can be pursued.

6.2 Resources and Continuous Change

One of the most discussed and analyzed feature of a planning problem is the presence of resources. This is a very well-understood concept in schedul-

		Resource change	
		Discrete	**Continuous**
Dynamics	**Consumable**		
	Reusable		

Table 6.1: Classification of resources.

ing and many planners offer dedicated language support and algorithmic features to deal with resources.

A resource can be defined generally as “something needed in order to achieve an action” [GNT04]. In practice, a resource is any tool, fuel, asset, machine with non-unlimited availability that is needed/consumed to perform an action.

Resources are usually modeled as values that change in time. At a high level, different classes of resources are possible as shown in table 6.1.

First of all, a resource can be reusable or consumable. A reusable resource is a quantity that is “borrowed” by an action and it is restored once the action is terminated, while consumable resources are not restored after action termination: they are employed and need to be replenished in order to return to the original value. Examples of reusable resources are machines and tools, but also the grid-power in a house (you cannot use too many appliances, otherwise you exceed the allowed watts, but once you shut down one appliance, some watts are freed to be used by another appliance). Examples of consumable resources are batteries that are depleted during operations, the fuel in a car that is consumed while driving, but also raw materials that are transformed in products by a factory. Consumable resources can be possibly replenished by specific actions: for example, a refuel action in a gas station allows the fuel in a car to be increased.

Another dimension of distinction for resources is given by rate at which a resource is changed: a resource can be changed discretely or continuously. If the value of a resource in time can be described as a piecewise-constant function the resource is said to be discrete, otherwise it is continuous (and the behavior in time is described by a generic total function). For example, the number of available cars in a car-renting shop is a discrete resource, while the amount of fuel in a car is a continuous resource.

In addition, similarly to time and action effects, resources can be uncontrollable. For example, the amount of energy produced by a solar panel is a function of time that does not entirely depend on the planner decisions, but also on uncontrollable factors such as weather conditions. This is somehow similar to temporal uncertainty, but resources might have behaviors that are more convoluted and less easy to manage than time.

6.3 Optimality

We defined the planning and scheduling problems as the problems of finding a plan or a schedule fulfilling some requirements, however many plans and schedules can fulfill the requisites, hence one may be interested in obtaining the one that maximizes or minimizes one or more objective functions while still being a valid plan or schedule.

The scheduling community is very oriented towards optimality, while the planning community is more oriented towards satisfying techniques (where optimality is sought but not guaranteed).

In temporal planning a significant amount of work has been devoted to optimize the make-span of plan, that is the total completion time. Nonetheless, PDDL 2.1 introduced the possibility of specifying custom metrics to be optimized.

Optimality is another orthogonal concept with respect to controllability and non-determinism, in fact, it is for example possible to define an optimal controllable plan.

6.4 Temporally Extended Goals

We never described nor formalized which kind of goal we pursue in a planning problem. The simpler and most common kind of goal is reachability, that is bringing the system to a state fulfilling a desired property. However, this is not the only goal of practical interest. For example, one may want to express constraints at intermediate points of the execution or to maintain a property for a period of time.

To this extent, some features and techniques have been developed especially in the field of temporal planning.

A simple, yet very powerful, extension to reachability goals are durative goals that allow each goal to be associated with an interval specifying when the goal must be reached. Durative goals can be modeled as constraints in PDDL 3.0, while languages such as NDDL and ANML natively support them.

An extension to durative goals are trajectory constraints that have been introduced in PDDL 3.0 that use temporal specifiers to constrain the set of valid plans. It is for example possible to force a proposition to stay true in a given interval or to achieve goals in a given order [GHL$^+$09].

State-Of-The-Art Survey Conclusions

In this part of the thesis, we proposed an execution model to classify existing scheduling and planning techniques and problems along two directions: temporal uncontrollability and discrete non-determinism. We summarized the existing literature on the subject locating each work in our classification table.

Evidently, the realm of planning application is much more concentrated on few cases than the scheduling one that covers almost all the cases. This opens the perspective for several research lines.

First, several scheduling problems are open for expressive classes of temporal networks (e.g. DTNU). Second, planning under temporal uncertainty is largely an open and interesting problem.

In this thesis, we contribute techniques that fill some of these gaps.

Part II

Disjunctive Scheduling under Temporal Uncertainty

Introduction

In this part, we present some novel techniques to deal with the problem of scheduling in presence of temporal uncertainty. We already introduced the main existing formalisms and existing techniques in chapter 4: here we recall the definitions and proceed in the description and the analysis of the new techniques.

As we discussed in section 4.2.1, several kinds of temporal networks have been identified, depending on the nature and structure of the constraints [DMP91b, PVYS07]. If the network is expressible as a pure conjunction of constraints over distances of time points, then we have the so-called *Simple Temporal Network* (STN). A more complex class is *Temporal Constraint Satisfaction Network* (TCSN), where the temporal distance between two time points can be constrained to lie in the union of disjoint intervals. Constraints in TCSNs can be seen as a restricted form of Boolean combinations. When arbitrary Boolean combinations are allowed, we have a *Disjunctive Temporal Network* (DTN).

A temporal network is said to be *consistent* if there exists an assignment for the time points, such that all the constraints are satisfied [DMP91b]. Such an assignment is called a *schedule*, and it corresponds to sequential time-triggered programs.

In many practical cases, however, the durations of some activities are uncontrollable. TNs are thus extended with *uncertainty* in the duration of activities, thus obtaining the classes of STNU, TCSNU and DTNU [VF99b].

Given a temporal network with uncertainty (TNU), three different problems can be addressed, namely weak, dynamic and strong controllability [VF99b]. *Weak controllability* concerns the existence of a strategy that schedules each activity, as a function of all the uncontrollable durations. The executor is assumed to know the duration of the uncontrollable activities before the execution starts (this property is sometimes known as "clairvoyance"). In *dynamic controllability*, a solution is a strategy, similarly to weak controllability, but each decision is constrained to depend on past events only. In *strong controllability*, we disallow any runtime observation, and we require a fixed schedule for the activities that is independent of the uncertainty. As in the case of consistency, we look for a schedule. However, the schedule only determines the start of all the activities, and the end of the activities that are controllable, and must satisfy the constraints *for all* the durations of the uncontrollable activities. If such a schedule exists, the problem is said to be *strongly controllable* [VF99b].

In this part, we focus on the scheduling problem of temporal networks with disjunctions and uncertainty. In fact, while STN (U) scheduling is a generally well-established area, solutions for the DTN (U) problem class are less studied. Nonetheless, many application domains, such as production planning and mission critical robotics, require the expressive power of disjunctive constraints to be modeled naturally [MNPW98].

We refer the reader to chapter 4 for a thorough overview of existing approaches. In this part, we make the following contributions.

1. We experiment with several encoding of the consistency problem into the SMT framework and we report very encouraging results.

2. We re-visit the problem of strong controllability for the DTNU problem class, providing a set of SMT encodings of the problem that are empirically evaluated to show their efficiency with respect to the state-

of-the-art.

3. We address the weak controllability decision problem for the DTNU problem class by reducing it to SMT modulo $\mathcal{LRA}$. This accounts for the first implementation of a decision procedure for this problem.

4. We address the open problem of weak strategy extraction for STNUs and DTNUs. We provide a portfolio of algorithms for both the classes and show their empirical performance.

5. We tackle the dynamic controllability problem for the DTNU problem class. First, we provide a reduction of the problem to a reachability game in a linear-sized TGA. This not-only is the first solution technique for the open problem, but it also provides the first dynamic solution in closed form. Second, we exploit the ideas behind the formal TGA encoding to provide a more efficient, dedicated solution technique.

Structure of this part. In chapter 7 we formally define several classes of temporal networks.

In chapter 8 we revisit the consistency problem for temporal networks and we show how the SMT framework can be used to solve the problem and build consistent schedules.

In chapter 9 we discuss the strong controllability problem and we propose novel solution techniques based on the encoding of the problem in the SMT framework.

In chapter 10 we introduce the weak controllability problem, in its two declinations: decision problem and strategy synthesis problem. We show two quantified SMT encodings for the decision problem and a portfolio of algorithm to synthesize weak strategies given a problem.

In chapter 11 we analyze the last form of controllability, namely dynamic controllability. We discuss an approach to solve the dynamic controllability problem for the most general class of temporal networks, namely DTNU.

Finally, in section 11.5 we review the part contributions and highlight possible future work.

Chapter 7

Temporal Networks Formalization

In this chapter, we formalize various classes of temporal networks proposed in the literature. In particular, we focus on disjunctive temporal networks with uncertainty (DTNU). We recall the needed concepts from chapter 4, adding the technical details needed to present the rest of the part.

We start by proposing a small example of a temporal network with uncertainty. Suppose we have two activities A and B. Activity A has duration of at least 7 units and at most 8 units or at least 10 units and at most 11 units, depending on a controllable decision. Activity B is uncontrollable, meaning that the actual duration is not decidable by the solver, but we can assume that it is at least 8 units and at most 11 units. We require that activity B must start after activity A and both activities must end within 20 units. The situation is depicted in figure 7.1.

A Temporal Network (TN) is a formalism that is used to represent temporal constraints over time-valued variables representing time points. Two families of TNs have been presented in literature over the years: TN without uncertainty, in which all the time points can be freely assigned by the solver [DMP91b, SK00] and TN with uncertainty (TNU), in which only some of the time points can be assigned by the solver, while the others are intended to be assigned by an adversary [VF99b, PVYS07]. As such,

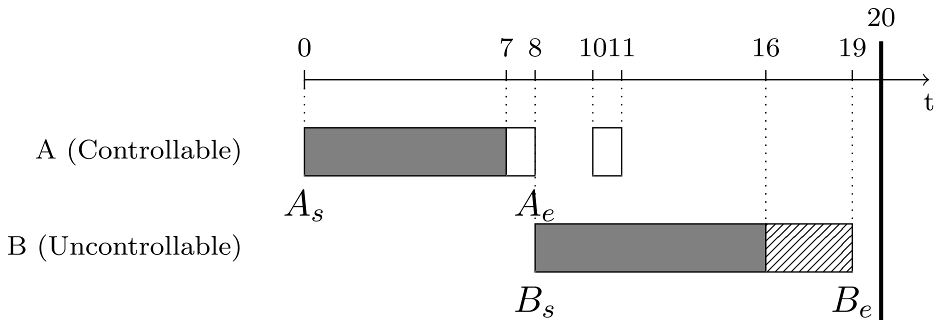


Figure 7.1: Schema of a possible temporal situation in the running example. Activities are depicted in time, filled regions are used to indicate the minimal guaranteed duration of an activity, the region in which uncontrollable event B_e can happen is striped, while the region in which A_e can be scheduled is the union of the two white rectangles. The problem deadline is indicated with the solid line at time 20.

TNUs can be seen as a form of game between the solver and an adversarial environment. In this part we focus on the DTNU class. We recall the DTNU definition from chapter 4.

Definition 40 (DTNU [PVYS07]). *A DTNU is a tuple $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, where:*

1. *$\mathcal{T}$ is a set of time points, partitioned into controllable ($\mathcal{T}_c$) and uncontrollable ($\mathcal{T}_u$);*

2. *$\mathcal{C}$ is a set of* free constraints*: each constraint c_i is of the form, $\bigvee_{j=1}^{D_i} t_{1,j} - t_{2,j} \in [l_{i,j}, u_{i,j}]$, for some $t_{1,j}, t_{2,j} \in \mathcal{T}$ and $l_{i,j}, u_{i,j} \in \mathbb{R} \cup \{+\infty, -\infty\}$; and*

3. *$\mathcal{L}$ is a set of* contingent links*: each $l_i \in \mathcal{L}$ is of the form, $\langle b_i, \mathcal{B}_i, e_i \rangle$, where $b_i \in \mathcal{T}_c$, $e_i \in \mathcal{T}_u$, and $\mathcal{B}_i$ is a finite set of pairs $\langle l_{i,j}, u_{i,j} \rangle$ such that $0 < l_{i,j} < u_{i,j} < \infty$, $j \in [1, E_i]$ (E_i being $|B_i|$); and for any distinct pairs, $\langle l_{i,j}, u_{i,j} \rangle$ and $\langle l_{i,k}, u_{i,k} \rangle$ in $\mathcal{B}_i$, either $l_{i,j} > u_{i,k}$ or $u_{i,j} < l_{i,k}$.*

Intuitively, time points belonging to $\mathcal{T}_c$ are time decisions that can be controlled by the solver, while time points in $\mathcal{T}_u$ are under the control of the environment. A similar subdivision is imposed on the constraints: free constraints $\mathcal{C}$ are constraints that the solver is required to fulfill, while contingent links ($\mathcal{L}$) are the assumptions that the environment will fulfill. As in previous work [VF99b, PVYS07], we consider only contingent links that start with a controllable time point. Thus, each uncontrollable time point e_i is constrained by exactly one contingent link to a controllable time point b_i called the activation time point [1] of e (indicated with $\alpha(x)$).

Within the framework of DTNU, we can only express uncertainty on the duration of activities (i.e. we cannot express uncertainty on whether an activity could occur or not, nor on its discrete outcome). Contingent links are used to model the possible durations of the uncontrollable activities, while uncontrollable time points represent the uncontrollable ending time of activities. We remark that the Temporal Network model of time is continuous, and we explicitly avoid any discretization seeking for a real-valued solution.

Any temporal network is defined over a set of time points, namely variables representing time instants. A temporal situation such as the one in our running example can be encoded in a temporal network by using two time points to represent the starting and ending time of each activity. Therefore, in order to model the running example with a temporal network we need a total of four time points A_s, A_e, B_s and B_e representing the start and end of activity A and B respectively (figure 7.1). B_e is the only uncontrollable time point, as we can control the starting and end time of A but we cannot control the duration of B. The only contingent constraint is the constraint on the duration of B. The rest of the network is composed of requirements that have to be fulfilled in any possible situation, and can be translated in three free constraints. The resulting

[1]This formulation assumes a complete independence between contingent links. This means that this formalism cannot express assumptions of interdependence between uncontrollable durations.

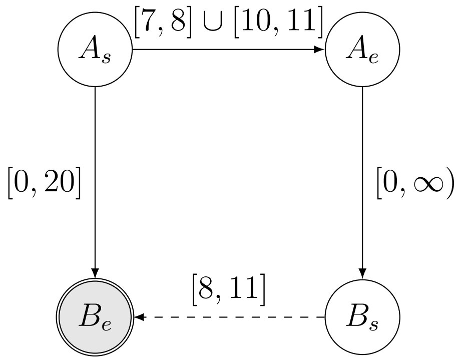


Figure 7.2: Graphical representation of the TNU model derived from the running example description. Each node of the graph is a time point, doubly circled nodes are uncontrollable the others are controllable, solid arrows are free constraints and dashed arrows are contingent constraints.

temporal network is then $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ where $\mathcal{T}_c = \{A_s, A_e, B_s\}$, $\mathcal{T}_u = \{B_e\}$, $\mathcal{C} = \{B_e - A_s \in [0, 20], B_s - A_e \in [0, \infty), A_e - A_s \in [7, 8] \vee A_e - A_s \in [10, 11]\}$ and $\mathcal{L} = \{\langle B_e, \{\langle 8, 11 \rangle\}, B_s \rangle\}$. Figure 7.2 shows a graphical visualization of the resulting TNU.

For the sake of this part, we define a *TN without uncertainty* as a TNU $\langle \mathcal{T}, \mathcal{C}, \emptyset \rangle$, in which the set of uncontrollable time points $\mathcal{T}_u$ is empty and the set of contingent links $\mathcal{L}$ is also empty. Excluding the $\emptyset$ in the tuple, this coincides with the Definitions of STN, TCSN and DTN (see section 4.2.1).

Depending on the generality of the constraints in $\mathcal{C}$ and the maximal cardinality of the sets $\mathcal{B}_i$ of the elements of $\mathcal{L}$, three classes of TNUs are possible [PVYS07]. Definition 40 in its general form identifies *Disjunctive Temporal Network with Uncertainty* (DTNU) [PVYS07]. If each constraint contained in $\mathcal{C}$ is defined on at most two time points, the resulting network is a *Temporal Constraint Satisfaction Network with Uncertainty* (TCSNU).

If each constraint in $\mathcal{C}$ has exactly one disjunct and each $\mathcal{B}_i$ has exactly one element, we obtain a *Simple Temporal Network with Uncertainty* (STNU).

Similarly, we can define the corresponding TNs without uncertainty

(DTN [SK00], TCSN, and STN [DMP91b]). Following the classification of Peintner et al. [PVYS07], we also say that a network is *simple-natured* if each $\mathcal{B}_i$ has exactly one element.

Given a TNU, values for controllable time points can be decided, namely they can be scheduled in time by an executor, while an uncontrollable time point e_i just happens after its activation time point b_i has been scheduled. The only assumption is that the i-th contingent link will be satisfied by the values of b_i and e_i. Given this intuitive meaning, we rephrased the concept of situation for a TNU [VF99b] for the DTNU problem class.

Definition 41 (Situation). *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU and let m be the number of uncontrollable time points ($|\mathcal{T}_u| = m$).*

The space of situations for P is a set of tuples $\Omega_P \doteq S_1 \times \cdots \times S_m$, where $S_i \doteq \bigcup_{j=1}^{E_i} [l^c_{i,j}, u^c_{i,j}]$. A situation is an element ω of Ω_P, and we write ω_{e_i} to indicate the value of the $i-th$ element of the tuple (i.e. the duration of the contingent link for e_i).

Intuitively, a situation is a choice of the actual duration for each activity with uncontrollable duration.

Given a situation, we define the *projection* of a TNU as the TN obtained fixing the duration of each contingent link.

Definition 42. *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU and let $\omega \doteq \langle \omega_1, \ldots, \omega_{|\mathcal{T}_u|} \rangle$ be a situation in Ω_P. The projection P_ω of the network P with respect to the situation ω is the TN $\langle \mathcal{T}, \mathcal{C}', \emptyset \rangle$, where $\mathcal{T} = \mathcal{T}_c$, $\mathcal{C}' \doteq \mathcal{C} \cup \{e_i - b_i \in [\omega_i, \omega_i] \mid \langle b_i, \mathcal{B}_i, e_i \rangle \in \mathcal{L}\}$.*

Intuitively, the projection P_ω is the network without uncertainty in which each uncontrollable duration has been fixed to a given value.

In the following, we will discuss one query at the time for the general disjunctive case, starting from the consistency of DTNs.

Chapter 8

Consistency

We first focus on the consistency problem, i.e. the case in which there is no uncontrollability.

We define an assignment to the time points as a total function from time points to real values. Given a TN without uncertainty, checking *consistency* corresponds to deciding the existence of an assignment that fulfills all the constraints of the network. We call such an assignment a *consistent schedule*, and we say that the TN is *consistent*. Checking the consistency of a TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ is defined as checking the consistency of the TN without uncertainty $\langle \mathcal{T}, \mathcal{C} \cup \rho(\mathcal{L}), \emptyset \rangle$, where $\rho(\mathcal{L})$ is the set of constraints obtained by considering each contingent link as a requirement constraint. Formally, $\rho(\mathcal{L}) \doteq \{\rho(x) \mid x \in \mathcal{L}\}$ and

$$\rho(\langle b, \mathcal{B}, e \rangle) \doteq \bigvee_{\langle l,u \rangle \in \mathcal{B}} e - b \geq l \wedge e - b \leq u \ .$$

The running example in figure 7.2 is consistent, and a consistent schedule is for example $\mu = \{A_s = 0, A_e = 11, B_s = 11, B_e = 20\}$. Intuitively, when checking consistency of a TNU, the behavior of the environment is assumed to be “cooperative” with the solver. Hence, checking the consistency of a temporal network amounts to checking whether the conjunction of the constraints admits a model. Therefore, the consistency problem can be reduced to checking the satisfiability of a quantifier-free formula modulo the $\mathcal{LRA}$ theory. The temporal network is consistent if and only if the corresponding SMT formula is satisfiable, and any satisfying assignment for the formula corresponds to a consistent schedule for the network.

8.1 Consistency Encodings

In this thesis, consistency checking plays the role of back-end for controllability solutions for TNUs. We present several SMT encodings, that turn out to have different performance in the solvers, depending on the nature of the constraints. We exploit the characteristics of such encodings to improve the performances of the approach we propose to solve the strong and weak controllability problems.

8.1.1 Naïve Encoding

In the following, we assume that a temporal network without uncertainty $P = \langle \mathcal{T}, \mathcal{X}, \emptyset \rangle$ is given. The first encoding in SMT of the consistency problem can be directly obtained as follows: for every time point in $x \in \mathcal{T}$ we introduce a real SMT variable[1] x, and we denote with $\vec{T}$ the vector of such variables (imposing an arbitrary order); each constraint $c \in \mathcal{C}$ is directly mapped on the corresponding SMT formula (indicated by $[\![c]\!]$) by keeping the Boolean structure of the constraint and substituting each time point with the corresponding SMT variable.

$$[\![\bigvee_{j=1}^{D_i} x_{i,j} - y_{i,j} \in [\ell_{i,j}, u_{i,j}]]\!] \doteq \bigvee_{j=1}^{D_i} \left(x_{i,j} - y_{i,j} \geq \ell_{i,j} \wedge x_{i,j} - y_{i,j} \leq u_{i,j}\right) \quad (8.1)$$

[1]We use no graphical distinction for time points and SMT variables, because we always have a one-to-one correspondence. It shall be clear from the context whether we are referring to the time point (when we discuss about constraints) or to the SMT-variables (when we work with formulae).

The resulting encoding is given by the SMT formula shown in equation (8.2).

$$\bigwedge_{c \in \mathcal{C}} [\![c]\!] \tag{8.2}$$

Proposition 8.1. *The temporal network P is consistent if and only if equation* (8.2) *is satisfiable (and a model of the formula yields a strong schedule for P).*

Proposition 8.1 is justified by the fact that equation (8.2) is the conjunction of the formalization of the network constraints, and by definition, checking consistency amounts to checking the existence of a model of the constraints. This formalization is such that a consistent schedule can be extracted from a model of the encoding formula by interpreting the value assigned to each SMT variable as a time value of the corresponding time point.
Equation (8.2) is already a working SMT encoding. It is linear in the size of the original TN, but does not exploit any knowledge on the structure of the network, and is thus referred to as *naïve encoding*. In particular, we notice that the resulting SMT formula is not in Conjunctive Normal Form (CNF)[2].

In the running example this encoding amounts to checking the satisfiability of the conjunction of all the constraints as follows.

$$\begin{aligned}
&(B_e - B_s \geq 8) \wedge (B_e - B_s \leq 11) \wedge \\
&(B_e - A_s \geq 0) \wedge (B_e - A_s \leq 20) \wedge \\
&(B_s - A_e \geq 0) \wedge \\
&((A_e - A_s \geq 7) \wedge (A_e - A_s \leq 8)) \vee ((A_e - A_s \geq 10) \wedge (A_e - A_s \leq 11))
\end{aligned}$$

In the rest of this chapter, we introduce three optimizations: the switch encoding (applicable to any DTN), the switch encoding with mutual exclusion and the hole encoding (both dedicated to the TCSN sub-class).

[2]Since most efficient SMT solvers work by combining a SAT and a T-solver, a CNF formulation of the problem is an advantage that prevents the solver for computing a (possibly less efficient) CNF by itself.

8.1.2 Switch Encoding

The *switch encoding* performs a CNF conversion of the formula in equation (8.2) by means of a polarity-based CNF labeling conversion [dlT90]. To this extent, we introduce $\sum_{c_i \in \mathcal{C}} D_i$ Boolean "switch" variables $s_{i,j}$ (with $j \in [1, D_i]$), and the resulting encoding is the one in equation (8.3).

$$\bigwedge_{c_i \in \mathcal{C}} \left(\left(\bigwedge_{j=1}^{D_i} \left((\neg s_{i,j} \vee (x_{i,j} - y_{i,j} \geq \ell_{i,j})) \wedge (\neg s_{i,j} \vee (x_{i,j} - y_{i,j} \leq u_{i,j})) \right) \right) \wedge \bigvee_{j=1}^{D_i} s_{i,j} \right) \tag{8.3}$$

where $c_i \doteq \bigvee_{j=1}^{D_i} x_{i,j} - y_{i,j} \in [\ell_{i,j}, u_{i,j}]$.

We remark that the encoding uses the switch variables $s_{i,j}$ as implicants for the disjuncts of the i-th constraint, in fact $s_{i,j} \rightarrow (x_{i,j} - y_{i,j} \in [\ell_{i,j}, u_{i,j}])$ can be equivalently rewritten as $(\neg s_{i,j} \vee (x_{i,j} - y_{i,j} \geq \ell_{i,j})) \wedge (\neg s_{i,j} \vee (x_{i,j} - y_{i,j} \leq u_{i,j}))$.

The following theorem states the correctness of the encoding; the proof is in appendix A.1.

Theorem 8.1 (Switch Correctness). *The temporal network P is consistent if and only if equation* (8.3) *is satisfiable (and a consistent schedule can be derived from any model of equation* (8.3)*).*

This encoding is also linear in the size of the original TN network, and it directly produces a CNF formula. We notice that the clauses involving theory atoms are binary; furthermore, if a switch variable is assigned to false, the corresponding clauses are satisfied without any theory reasoning. These factors have a positive impact on the performance of the SMT solver.

In the running example this encoding is as follows.

$$\begin{aligned}
&(B_e - B_s \geq 8) \wedge (B_e - B_s \leq 11) \wedge \\
&(B_e - A_s \geq 0) \wedge (B_e - A_s \leq 20) \wedge \\
&(B_s - A_e \geq 0) \wedge \\
&(\neg s_1 \vee (A_e - A_s \geq 7)) \wedge \\
&(\neg s_1 \vee (A_e - A_s \leq 8)) \wedge \\
&(\neg s_2 \vee (A_e - A_s \geq 10)) \wedge \\
&(\neg s_2 \vee (A_e - A_s \leq 11)) \wedge \\
&(s_1 \vee s_2)
\end{aligned}$$

8.1.3 Switch Encoding with Mutual Exclusion

If we focus on the TCSN class, we can exploit the network structure to further improve our encodings. In particular, we assume that the disjuncts in each constraint are mutually exclusive, otherwise two or more disjuncts can be merged together by simply taking the union of the intervals they represent. For example, a constraint $(a - b \in [10, 20]) \vee (a - b \in [30, 35]) \vee (a - b \in [15, 25])$ can be simplified to $(a - b \in [10, 25]) \vee (a - b \in [30, 35])$ by merging the first and the last disjuncts. Formally[3], this means that each TCSN constraint c_i is composed of disjuncts of the form $x_i - y_i \in [\ell_{i,j}, u_{i,j}]$, where x_i and y_i are time points, and for all j, $\ell_{i,j} \leq u_{i,j}$ and $u_{i,j} < \ell_{i,j+1}$.

If we use the previous encodings, it is left to the solver (in particular to the theory solver) to discover this mutual exclusion property. We can strengthen the switch encoding by statically adding mutual exclusion constraints of the form $(\neg s_h \vee \neg s_k)$, with $h \neq k$. Adding this information to the encoding is a form of static learning, and it can guide the Boolean search by pruning branches that are unsatisfiable in the theory. The *switch encoding with mutual exclusion* is presented in equation (8.4) and its correctness is

[3]Note that in TCSN the constraints are binary, i.e. each constraint relates exactly two variables.

stated by theorem 8.2 that is proven in appendix A.1.

$$\bigwedge_{c_i \in \mathcal{C}} \left(\bigwedge_{j=1}^{D_i} \Big((\neg s_{i,j} \vee (x_{i,j} - y_{i,j} \geq \ell_{i,j})) \wedge (\neg s_{i,j} \vee (x_{i,j} - y_{i,j} \leq u_{i,j})) \Big) \wedge \right.$$
$$\left. \bigvee_{j=1}^{D_i} s_{i,j} \wedge \bigwedge_{j=1}^{D_i} \bigwedge_{k=j+1}^{D_i} (\neg s_{i,j} \vee \neg s_{i,k}) \right) \tag{8.4}$$

Theorem 8.2 (Mutex Switch Correctness)**.** *If P is a TCSN, P is consistent if and only if equation* (8.4) *is satisfiable (and a model of equation* (8.4) *yields a consistent schedule).*

This encoding is in CNF, but its size is quadratic in the size of the TN because of the added term $\bigwedge_{j=1}^{D_i} \bigwedge_{k=j+1}^{D_i} (\neg s_{i,j} \vee \neg s_k)$.

In the running example this encoding is as follows.

$$\begin{aligned}
&(B_e - B_s \geq 8) \wedge (B_e - B_s \leq 11) \wedge \\
&(B_e - A_s \geq 0) \wedge (B_e - A_s \leq 20) \wedge \\
&(B_s - A_e \geq 0) \wedge \\
&(\neg s_1 \vee (A_e - A_s \geq 7)) \wedge \\
&(\neg s_1 \vee (A_e - A_s \leq 8)) \wedge \\
&(\neg s_2 \vee (A_e - A_s \geq 10)) \wedge \\
&(\neg s_2 \vee (A_e - A_s \leq 11)) \wedge \\
&(s_1 \vee s_2) \wedge \\
&(\neg s_1 \vee \neg s_2)
\end{aligned}$$

8.1.4 Hole Encoding

A different encoding for the TCSN problem class is obtained by considering the intervals that define each TCSN constraint. Without loss of generality,

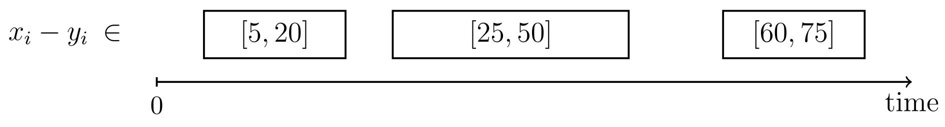


Figure 8.1: Graphical view of an example TCSNU constraint. The difference between two time points is constrained to lay within a disjunction of three intervals in time.

we assume that the intervals are disjoint and ordered: for each constraint $c_i = \bigvee_{j=1}^{D_i}(x_i - y_i) \in [\ell_{i,j}, u_{i,j}]$, we require that $u_{i,j} < \ell_{i,j+1}$. The idea is then to consider two adjacent intervals in the constraint and exclude the "holes" between intervals, a hole being an open interval $(u_{i,j}, \ell_{i,j+1})$. The result is the *hole encoding* reported in equation (8.5). The encoding correctness in stated in theorem 8.3 that is proven in appendix A.1.

$$\bigwedge_{c_i \in \mathcal{C}} \Bigg((x_i - y_i \geq \ell_{i,1}) \wedge (x_i - y_i \leq u_{i,D_i}) \wedge \\ \Big(\bigwedge_{j=1}^{D_i - 1} (x_i - y_i \leq u_{i,j}) \vee (x_i - y_i \geq \ell_{i,(j+1)}) \Big) \Bigg) \tag{8.5}$$

Theorem 8.3 (Hole Encoding Correctness). *If P is a TCSN, P is consistent if and only if equation* (8.5) *is satisfiable (and a model of equation* (8.5) *yields a consistent schedule for P).*

Consider for example figure 8.1, depicting the constraint $(x - y \in [5, 20]) \vee (x - y \in [25, 50]) \vee (x - y \in [60, 75])$. The hole encoding of this constraint is $((x - y) \geq 5) \wedge ((x - y) \leq 20 \vee (x - y) \geq 25) \wedge ((x - y) \leq 50 \vee (x - y) \geq 60) \wedge ((x - y) \leq 75)$.
This encoding is linear in the size of the original TN, does not introduce any additional variable, and, most importantly, results in a 2-CNF formula. These properties are noteworthy and will be exploited in the following sections.

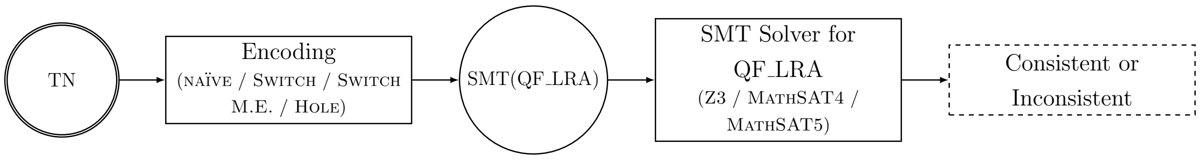

TN
Encoding
(naïve / Switch / Switch M.E. / Hole)
SMT(QF_LRA)
SMT Solver for QF_LRA
(Z3 / MathSAT4 / MathSAT5)
Consistent or Inconsistent

Figure 8.2: Graphical representation of the developed tool-chain.

In the running example this encoding is as follows.

$$\begin{aligned}
&(B_e - B_s \geq 8) \wedge (B_e - B_s \leq 11) \wedge \\
&(B_e - A_s \geq 0) \wedge (B_e - A_s \leq 20) \wedge \\
&(B_s - A_e \geq 0) \wedge \\
&(A_e - A_s \geq 7) \wedge \\
&((A_e - A_s \leq 8) \vee (A_e - A_s \geq 10)) \wedge \\
&(A_e - A_s \leq 11)
\end{aligned}$$

Finally, we notice that equation (8.5) is logically equivalent to equation (8.2) (in the applicable case of TCSN), while equations (8.3) and (8.4) are only equi-satisfiable to it, because of the added switch variables. The solution to the temporal network is still obtained directly from any satisfying assignment, gathering the values for the variables in $\vec{X_c}$.

8.2 Experimental Evaluation

In this section, we experimentally evaluate the consistency encodings we proposed.

We developed a tool that automatically encodes the various classes of temporal problems as SMT problems. The tool, depicted in figure 8.2, generates SMT ($\mathcal{QF\_LRA}$) encodings, that can then be solved by MathSAT4 [BCF+08], MathSAT5 [CGSS13] or Z3 [dMB08].

We used a set of randomly-generated benchmarks. Consistency problems are generated using the random generator presented in [ACG99]. The

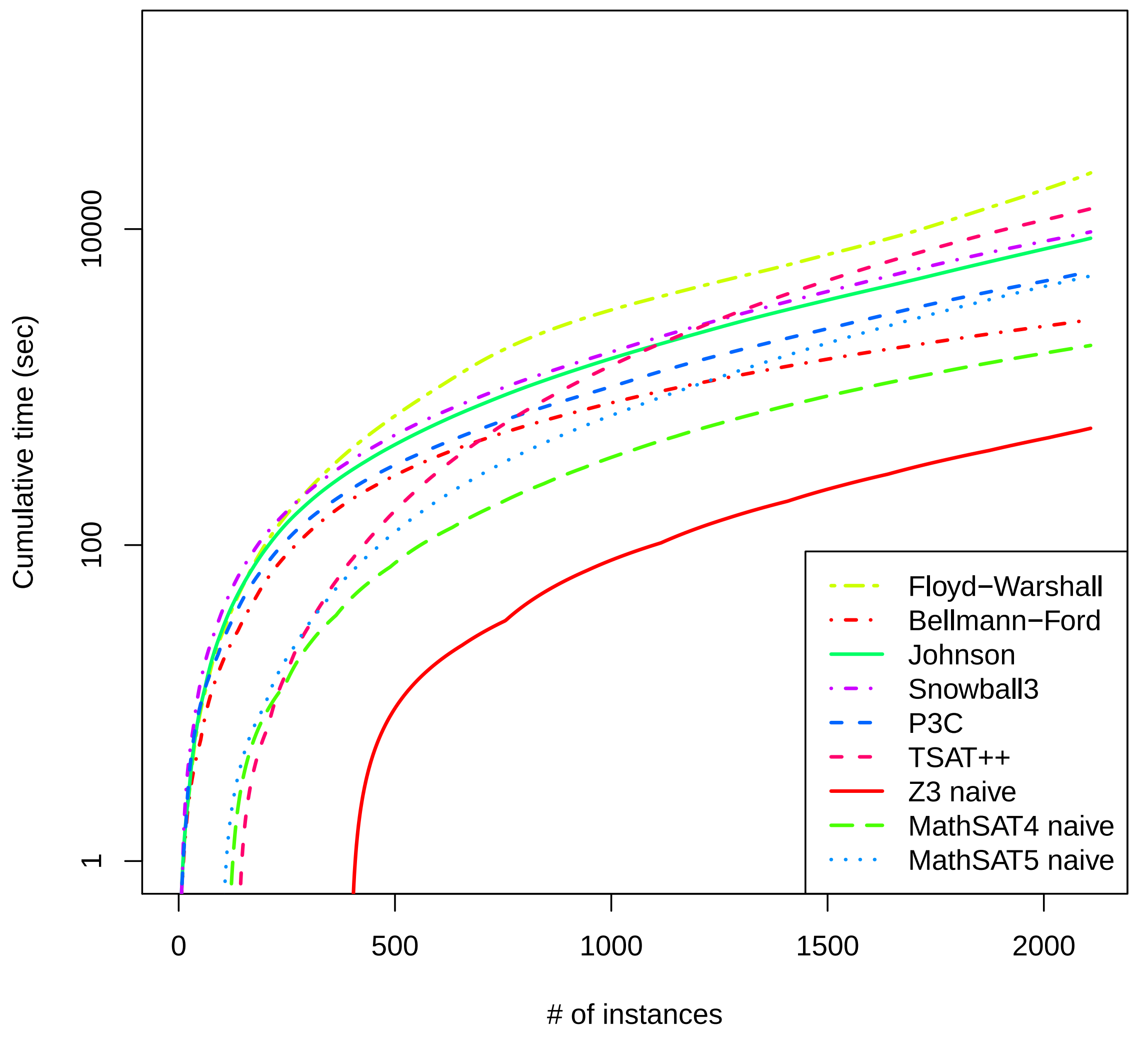


Figure 8.3: Results for consistency experimental evaluation of STN.

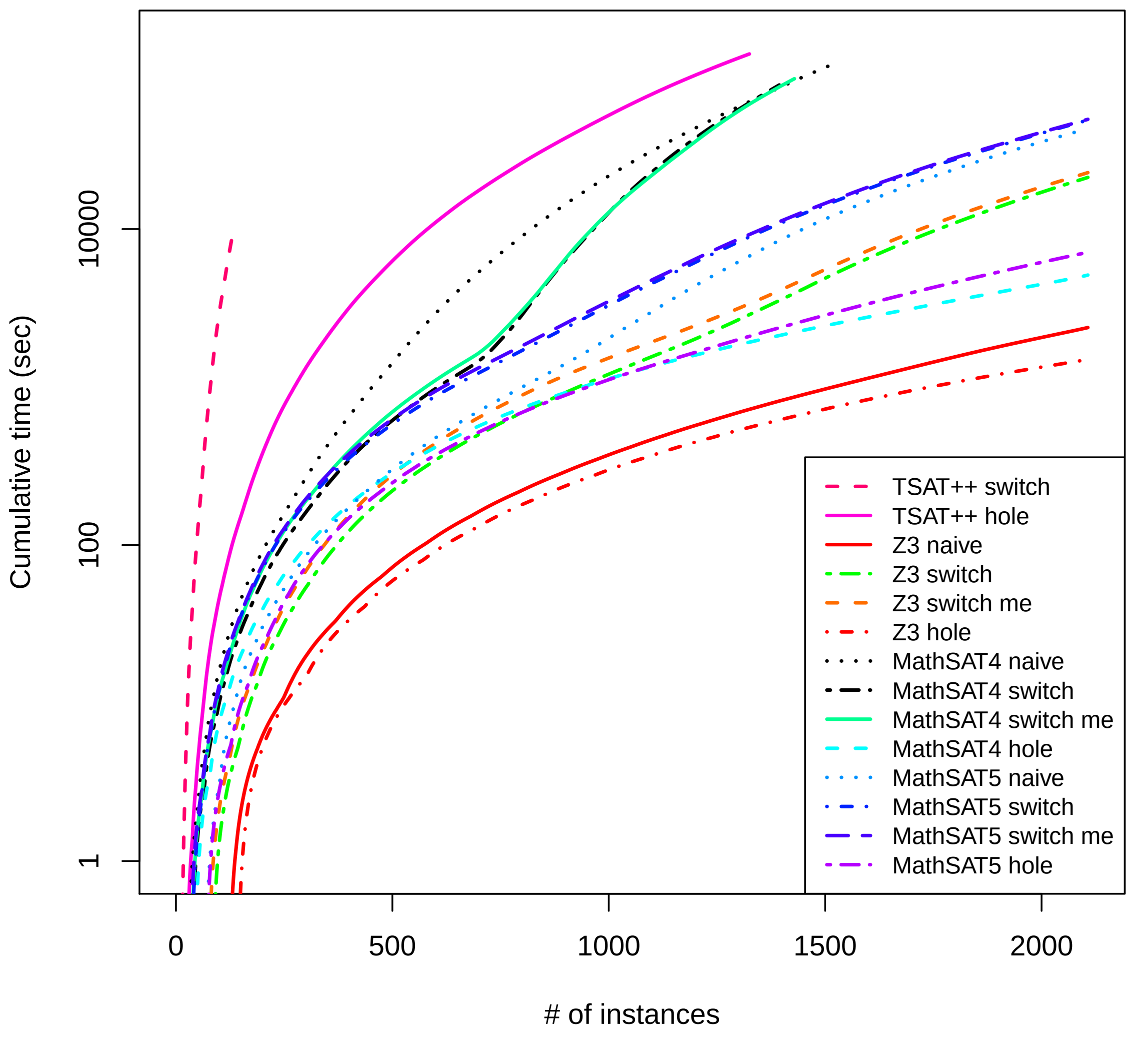


Figure 8.4: Results for consistency experimental evaluation of TCSN.

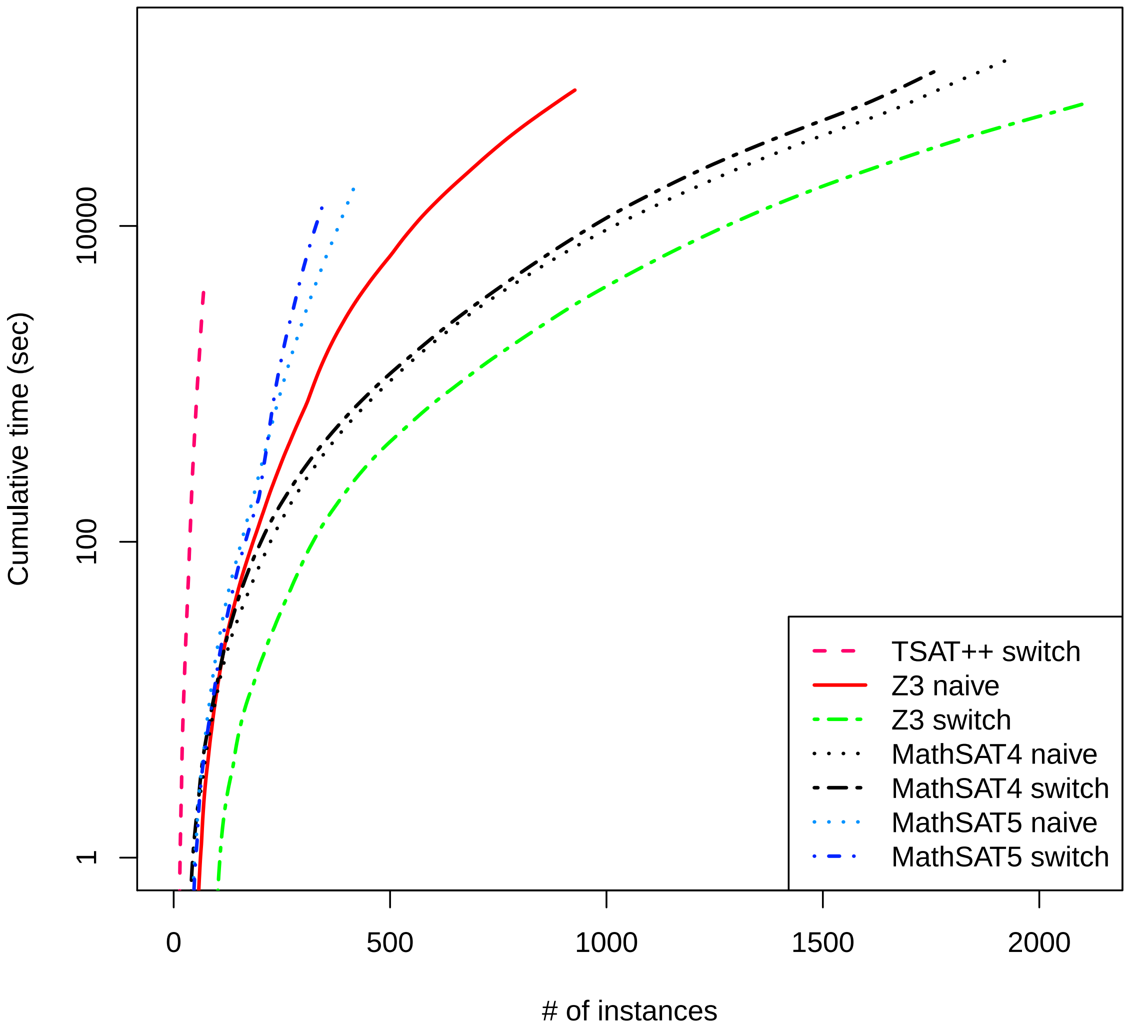


Figure 8.5: Results for consistency experimental evaluation of DTN.

benchmark set contains 2108 instances for each problem class (STN, TCSN and DTN). We used random instance generators because they are typically used in literature, and because they can be easily scaled to stress the solvers.

We performed all our experiments on a machine running Scientific Linux 6.0, equipped with two quad-core Xeon processors @ 2.70GHz. We considered a memory limit of 2GB and a time-out of 300 seconds. The benchmarks and the tool are available as indicated in section 1.2.

8.2.1 Results

For consistency problems, we analyzed the performance of the various SMT solvers on the various encodings. We also compared our tool chain with the other available solvers for TN without uncertainty, namely the Snowball3 [PdWvdK12] tool, that implements many algorithms for the case of STN (i.e Floyd-Warshall, Bellman-Ford, Johnson and Snowball3), and TSAT++ [ACG99], that is able to solve STN, TCSN and DTN problems.

The results for consistency problems are reported in figures 8.3 to 8.5. The cactus plot reports the number of solved instances in the horizontal axis and the cumulative time for each approach in logarithmic scale on the vertical axis. For example, MathSAT4 takes about 100 seconds to solve the easiest 500 STN instances. For STN problems, we compared the naïve encoding with various algorithms available in the Snow-Ball3 [PdWvdK12] tool, and with TSAT++ [ACG99]. (In the case of STN, the other encodings coincide with the naïve encoding.) In TCSN and DTN, we tested all the applicable encodings with all the SMT solvers under analysis and with TSAT++. The plots show that the SMT approach is competitive with dedicated techniques. MathSAT4 implements a dedicated algorithm for the theory of difference logic [CM06], and is thus faster than MathSAT5, that uses a general purpose algorithm for

$\mathcal{LRA}$ [DdM06a]. All solvers perform better on problems with Hole encoding. This encoding produces a formula that has just one real variable for every time point and has at most two literals per clause: this simplifies the SMT search procedure by augmenting the number of unit propagations, and by reducing the size of the search space. TSAT++ is outperformed by the other (more modern) SMT solvers.

Chapter 9

Strong Controllability

In this chapter, we focus on strong controllability for a TNU, where the environment is adversarial [VF99b]. Strong controllability consists in deciding the existence of an assignment to controllable time points that fulfills the free constraints under any assignment of uncontrollable time points that satisfies the contingent constraints. Such an assignment is called a *strong schedule* of the network. A TNU for which there exists a strong schedule is said to be *strongly controllable.* Consider again the running example, the network is strongly controllable and a strong schedule is $\mu' = \{A_s = 0, A_e = 8, B_s = 8\}$.

If a TNU is strongly controllable, it is also consistent [VF99b]. However, the converse does not hold in general. Consider for example a variation to the running example in figure 7.2 in which the deadline is moved from 20 time units to 17 time units. The network is consistent because a consistent schedule is $\{A_s = 0, A_e = 7.5, B_s = 8, B_e = 16\}$. However, this version of the problem is not strongly controllable because the activity B cannot be started before 7 (that is the minimal duration of activity A that must precede B): since B is uncontrollable with duration in $[8, 11]$, it may be the case that the execution of the activity takes more than 10 time units, thus exceeding the deadline.

Strong controllability is an important problem, because it results in a schedule that is satisfactory under all possible uncertainties. Clearly, a strong schedule can yield a longer time-span compared to a dynamic strategy. However, dynamic information may not be available, e.g. due to the lack of sensors. Furthermore, most algorithms for dynamic execution require run-time reasoning [Hun10a]. This may be incompatible with some operational settings: for example, in mission-critical systems, validating the run-time reasoner to the required level of assurance may be prohibitively hard. Furthermore, the computational resources available during execution may be too limited for a dynamic approach. Examples of such application domains can be found in production planning and in mission critical robotics, for which strong controllability is a very relevant problem. We also remark that, in the same domains, the expressiveness of disjunctive constraints (compared to simple temporal problems) is often necessary [MNPW98].

We now formally define the concept of strong controllability.

Definition 43. *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be a TNU. P is strongly controllable if there exists an assignment μ for $\mathcal{T}_c$ such that for each situation $\omega \in \Omega_P$, μ is a consistent schedule for the projection P_ω.*

In this chapter, we propose a comprehensive and effective approach for solving the strong controllability problem for TNUs in the most general form including arbitrary disjunctions. We tackle the strong controllability problem of TNUs by reduction to SMT. The resulting problem can be then solved by efficient SMT solvers. First, we show how to encode a TNU into the theory of quantified linear real arithmetic ($\mathcal{LRA}$) and, by leveraging the specific nature of the problem, we optimize the encoding by reducing the scope of quantifiers. The resulting formula can be solved by any SMT solver for (quantified) $\mathcal{LRA}$. Each encoding we present is satisfiable if and only if the temporal problem is strongly controllable, and such that a model of each encoding yields a solution for the original problem. Second, we present a general reduction procedure from strong controllability to consistency, based on the eager application of quantifier elimination techniques. The resulting formulae can be directly fed into any SMT solver for the quantifier-free $\mathcal{LRA}$ theory ($\mathcal{QF\_LRA}$). This gives the first general comprehensive solver for strong controllability of TNUs. Third, we generalize the results by Vidal and Fargier [VF99b], originally stated for STNU, to the subclass of (simple-natured) TCSNU. In this way, we avoid the use of expensive general purpose quantifier elimination techniques, with significant performance improvements.

The proposed approach has been implemented in a solver based on state-of-the-art SMT techniques. To the best of our knowledge, this is the first implemented solver for strong controllability of TNUs. We carried out a thorough experimental evaluation, over a large set of benchmarks. We analyze the merits of the various encodings, and demonstrate the overall feasibility of the approach. We also compare the proposed approaches with state-of-the-art algorithms on consistency problems. SMT solvers turned out to be competitive with, and often outperform, the best known dedicated solving techniques. Finally, we compared our approach with the only other algorithm to check strong controllability for DTNU [PVYS07]. The results show that the symbolic techniques we propose can dramatically outperform the enumerative approach in [PVYS07].

9.1 Encoding Strong Controllability in SMT

In the following, we assume that a TNU $P = \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ is given. We derive a number of encodings of the problem using the expressive power of the SMT framework.

9.1.1 Encodings into Quantified $\mathcal{LRA}$

As in the previous section, we assume each time point is associated with an SMT variable. The encoding in equation (9.1) is a direct logical mapping of the notion of strong controllability; we call this encoding *direct encoding.* We indicate with $\vec{T_c}$ and $\vec{T_u}$ the SMT variables corresponding to $\mathcal{T}_c$ and $\mathcal{T}_u$, respectively.

$$\forall \vec{T_u}. [\![\rho(\mathcal{L})]\!] \rightarrow \bigwedge_{c_i \in \mathcal{C}} [\![c_i]\!] \tag{9.1}$$

Proposition 9.1. *The TNU P is strongly controllable if and only if equation* (9.1) *is satisfiable (and a model of equation* (9.1) *yields a strong schedule for P).*

Proposition 9.1 is directly obtained by formalizing the definition of strong controllability. Intuitively, equation (9.1) is satisfiable if and only if there exists an assignment to the controllable variables $\mathcal{T}_c$ such that, for all assignments to the uncontrollable variables $\mathcal{T}_u$ (that is, for each situation) satisfying the contingent links $\mathcal{L}$, the free constraints $\mathcal{C}$ are also satisfied[1]. In the above formula, the controllable variables are implicitly existentially quantified. In case of satisfiability, the SMT solver returns a satisfying assignment to the controllable variables that is exactly a strong schedule.

In the running example, this encoding is as follows.

$$\begin{aligned}\forall B_e. \Big(\Big(& (B_e - B_s \geq 8) \wedge (B_e - B_s \leq 11) \Big) \rightarrow \\ & \Big(B_e - A_s \geq 0 \wedge B_e - A_s \leq 20 \wedge B_s - A_e \geq 0 \wedge \\ & \big((A_e - A_s \geq 7 \wedge A_e - A_s \leq 8) \vee (A_e - A_s \geq 10 \wedge A_e - A_s \leq 11) \big) \Big) \Big)\end{aligned}$$

[1]Here we assume that the contingent links are not contradictory, otherwise the implication will be automatically true. However, the non-contradiction of contingent links is true by construction in our definition of temporal problem, as no relationship between different contingent links is possible.

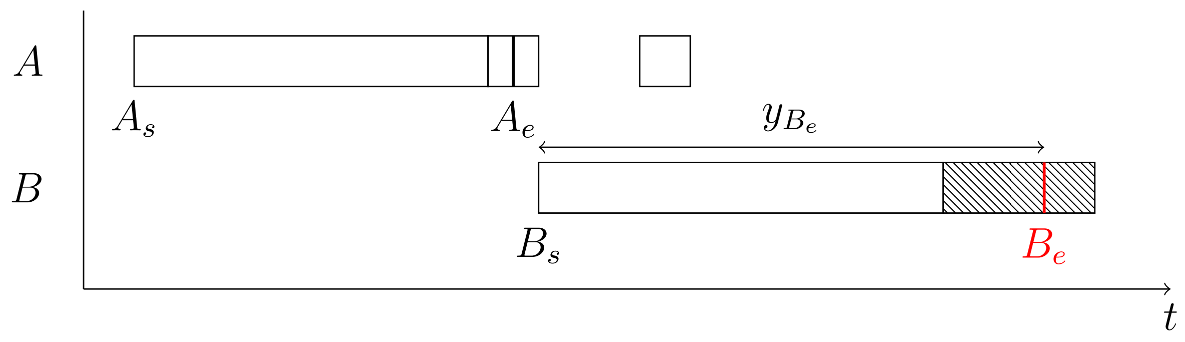


Figure 9.1: The running example seen from an activities point of view to explain the encoding of the problem. The striped region is the uncontrollable space, namely where the uncontrollable time point B_e can be scheduled given the decision on the related controllable time point (B_s). The value of y_{B_e} in the shown situation is seen as the actual duration of the B activity.

In order to enable further simplifications, we notice that contingent constraints depend both on controllable and uncontrollable time points, and we re-code the problem as follows.

We encode each uncontrollable time point e_i in terms of the time difference with its starting time point $b_i \doteq \alpha(e_i)$ by means of an uncontrollable duration variable y_{e_i}. Intuitively, if we take an activity view, y_{e_i} measures the duration of the i-th activity. For every contingent link $l_i = \langle b_i, \mathcal{B}_i, e_i \rangle$ with $\mathcal{B}_i = \{\langle \ell_{i,1}, u_{i,1} \rangle, \langle \ell_{i,E_i}, u_{i,E_i} \rangle\}$, let $y_{e_i} \in \mathbb{R}$ be the uncontrollable offset variable associated to e_i such that $\bigvee_{j=1}^{E_i}(y_{e_i} \in [\ell_{i,j}, u_{i,j}])$. y_{e_i} represents the duration of the interval $[b_i, e_i]$ that is constrained by the i-th contingent link. We are thus symbolically encoding a situation $\omega \doteq (\omega_1, \ldots, \omega_{|\mathcal{T}_u|})$ in which y_{e_i} models the value of ω_i. Figure 9.1 gives a pictorial representation of this encoding interpreted at the activity level.

Definition 44. *Given a TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, let $\vec{Y}_u$ be the vector of uncontrollable duration variables $\langle y_{e_1}, y_{e_2}, \ldots, y_{e_m} \rangle$, with $\mathcal{T}_u \doteq \{e_1, e_2, \cdots, e_m\}$. We define the encoding of the problem as a tuple $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ where*

$$\Gamma(\vec{Y}_u) \doteq \bigwedge_{i=1}^{m} \bigvee_{j=1}^{E_i} (y_{e_i} \in [l_{i,j}, u_{i,j}])$$

$$\begin{aligned}
\vec{T}_c &= \langle A_s, A_e, B_s \rangle \\
\vec{Y}_u &= \langle y_{B_e} \rangle \\
\Gamma(\vec{Y}_u) &= (y_{B_e} \geq 8) \wedge (y_{B_e} \leq 11) \\
\Psi(\vec{T}_c, \vec{Y}_u) &= (A_e - A_s \in [7, 8] \vee A_e - A_s \in [10, 11]) \wedge \\
&\quad ((B_s + y_{B_e}) - A_s \in [0, 20]) \wedge \\
&\quad (B_s - A_e \in [0, \infty])
\end{aligned}$$

Figure 9.2: The encoding of the example TCSNU of figure 7.2.

and

$$\Psi(\vec{T}_c, \vec{Y}_u) \doteq \bigwedge_{c \in \mathcal{C}} c[(\alpha(e_1) + y_{e_1})/e_1][(\alpha(e_2) + y_{e_2})/e_2] \ldots [(\alpha(e_m) + y_{e_m})/e_m].$$

Intuitively, $\Gamma(\vec{Y}_u)$ is the formula representing the conjunction of all the contingent links after the re-coding, and $\Psi(\vec{T}_c, \vec{Y}_u)$ is the conjunction of all the free constraints rewritten in terms of $\vec{T}_c$ and $\vec{Y}_u$.

We remark that the use of this encoding yields two consequences. First, thanks to the redefinition of each e_i in terms of y_{e_i}, we managed to encode the contingent links in terms of $\vec{Y}_u$ only, therefore they are independent of the values of the controllable time points ($\vec{T}_c$). Intuitively, $\Gamma(\vec{Y}_u)$ encodes the set of all possible situations (Ω_P) for the given problem P: each model of $\Gamma(\vec{Y}_u)$ corresponds to a situation ω. Second, the constraints in this formulation are expressed in the $\mathcal{LRA}$ theory (the original formulation was expressed in the $\mathcal{RDL}$ fragment of $\mathcal{LRA}$). This encoding applied to the STNU problem in figure 7.1 is shown in figure 9.2.

From here on, we assume an encoded problem $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ is given. In this setting, the strong controllability problem consists in finding a value for $\vec{T}_c$ that satisfies the free constraints $\Psi(\vec{T}_c, \vec{Y}_u)$ under any possible value of $\vec{Y}_u$ that satisfies $\Gamma(\vec{Y}_u)$.

The strong controllability encoding in equation (9.1) can be re-coded as an $\mathcal{LRA}$ formula in the free variables $\vec{T}_c$ as follows.

$$\forall \vec{Y}_u . \big(\Gamma(\vec{Y}_u) \rightarrow \Psi(\vec{T}_c, \vec{Y}_u) \big) \tag{9.2}$$

We call this encoding *Offset Encoding.* This formulation corresponds to a quantified SMT problem in $\mathcal{LRA}$, and still requires a solver that supports quantified formulae, but the part of the encoding representing the contingent link is now dependent on $\vec{Y}_u$ only. The following theorem states the correctness of this encoding, the relative proof can be found in appendix A.2.

Theorem 9.1 (Offset Encoding Correctness)**.** *The TNU P is strongly controllable if and only if equation* (9.2) *is satisfiable (and a strong schedule can be extracted from any of its models).*

In the running example, this encoding is as follows.

$$\forall y_{B_e} . \Bigg((y_{B_e} \geq 8 \ \wedge \ y_{B_e} \leq 11) \rightarrow$$
$$\Big(B_s + y_{B_e} - A_s \geq 0 \ \wedge \ B_s + y_{B_e} - A_s \leq 20 \ \wedge \ B_s - A_e \geq 0 \ \wedge$$
$$\Big(\big(A_e - A_s \geq 7 \ \wedge \ A_e - A_s \leq 8 \big) \vee \big(A_e - A_s \geq 10 \ \wedge \ A_e - A_s \leq 11 \big) \Big) \Big) \Bigg)$$

The main problem in the previous encodings is the scope of the universal quantifier. Since the computational cost of quantification is very high, we can rewrite the offset encoding in equation (9.2) in order to obtain a possibly more efficient encoding. Let us assume that $\Psi(\vec{T}_c, \vec{Y}_u)$ is written as a conjunction of H formulae $\psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h})$, where $\vec{T}_{c_h} \subseteq \vec{T}_c$ and $\vec{Y}_{u_h} \subseteq \vec{Y}_u$ are the variables used in the formula ψ_h. This assumption can be easily satisfied by converting $\Psi(\vec{T}_c, \vec{Y}_u)$ in CNF using any[2] consistency encoding

[2]If the used encoding introduces additional variables, those are existentially quantified and extend the model of equation (9.1) by preserving the satisfiability and the strong schedules encoded in the models.

we presented in chapter 8.

$$\Psi(\vec{T_c}, \vec{Y_u}) = \bigwedge_{h=1}^{H} \psi_h(\vec{T_{c_h}}, \vec{Y_{u_h}})$$

We have that $\bigwedge_h \forall \vec{Y_u}.(\neg\Gamma(\vec{Y_u}) \vee \psi_h(\vec{T_{c_h}}, \vec{Y_{u_h}}))$ can be equivalently rewritten to $\bigwedge_h \forall \vec{Y_{u_h}}.(\neg\Gamma(\vec{Y_u})|_{Y_{u_h}} \vee \psi_h(\vec{T_{c_h}}, \vec{Y_{u_h}}))$, and we obtain the following *distributed encoding.* We recall that $\phi|_{\vec{x}}$ is a notation meaning the restriction of the conjunction ϕ to the conjuncts that are defined on at least one variable of $\vec{x}$ (see section 2.1 for the details).

$$\bigwedge_h \forall \vec{Y_{u_h}}. \Big(\neg\Gamma(\vec{Y_u})|_{Y_{u_h}} \vee \psi_h(\vec{T_{c_h}}, \vec{Y_{u_h}})\Big) \tag{9.3}$$

The following theorem states the correctness of this encoding, the proof is reported in appendix A.2.

Theorem 9.2 (Distributed Encoding Correctness)**.** *If the TNU P is consistent, it is strongly controllable if and only if equation* (9.3) *is satisfiable (and each model yields a strong schedule).*

The size of the produced (quantified) formula is linear with respect to the original TNU. This encoding still requires a solver that supports quantified formulae, and contains as many quantifiers as conjuncts in $\Psi(\vec{T_c}, \vec{Y_u})$. However, each quantification is now restricted to the offset variables $Y_{u_h} \subseteq Y_u$ occurring in each conjunct ψ_h. This encoding also limits the scope of the universal quantifiers, which turns out to be beneficial in practice. Intuitively, this is related to the fact that a number of quantifier eliminations in $\mathcal{LRA}$ on smaller formulae may be much cheaper than a single, monolithic quantifier elimination over a large formula.

If we use the hole encoding to obtain the CNF formula for the free constraints, the running example formulation of this encoding is as follows.

$$\begin{aligned}
&\big(\forall y_{B_e}.(y_{B_e} < 8 \lor y_{B_e} > 11 \lor B_s + y_{B_e} - A_s \geq 0)\big) \land \\
&\big(\forall y_{B_e}.(y_{B_e} < 8 \lor y_{B_e} > 11 \lor B_s + y_{B_e} - A_s \leq 20)\big) \land \\
&(B_s - A_e \geq 0) \land (A_e - A_s \geq 7) \land \\
&\big((A_e - A_s \leq 8) \lor (A_e - A_s \geq 10)\big) \land (A_e - A_s \leq 11)
\end{aligned}$$

9.1.2 Encodings into Quantifier-Free $\mathcal{LRA}$

In order to exploit solvers that do not support quantifiers, we propose an encoding of strong controllability into a quantifier-free SMT ($\mathcal{LRA}$) formula. This is obtained by resorting to an external procedure for quantifier elimination.

We rewrite equation (9.3) as $\bigwedge_h \neg(\exists \vec{Y}_{u_h}.(\Gamma(\vec{Y}_u)|_{Y_{u_h}} \land \neg\psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h})))$, in order to apply a procedure for the elimination of existential quantifiers (e.g. Fourier-Motzkin [Sch98]). In the following we refer to each conjunct after quantifier elimination as $\psi_h^\Gamma(\vec{T}_{c_h})$ ($\psi_h^\Gamma(\vec{T}_{c_h})$ is then a quantifier-free formula).

$$\psi_h^\Gamma(\vec{T}_{c_h}) \leftrightarrow \neg(\exists \vec{Y}_{u_h}.(\Gamma(\vec{Y}_u)|_{Y_{u_h}} \land \neg\psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h})))$$

The resulting encoding, reported in equation (9.4), is called *eager for-all elimination encoding.* Theorem 9.3 states the correctness of the encoding and is proven in appendix A.2.

$$\bigwedge_h \psi_h^\Gamma(\vec{T}_{c_h}) \tag{9.4}$$

Clearly, this approach moves most of the computation complexity from the solving of the resulting formula to the encoder. In fact, the encoder needs now to solve a number of costly quantifier eliminations.

Theorem 9.3 (EFE Encoding Correctness)**.** *The TNU P is strongly controllable if and only if equation* (9.3) *is satisfiable (and a strong schedule can be extracted from a model of equation* (9.3)*).*

In the running example, this encoding is as follows.

$$\begin{aligned}
&\big(\neg\exists y_{B_e}.(y_{B_e} \geq 8 \wedge y_{B_e} \leq 11 \wedge (B_s + y_{B_e} - A_s < 0)\big) \wedge \\
&\big(\neg\exists y_{B_e}.(y_{B_e} \geq 8 \wedge y_{B_e} \leq 11 \wedge B_s + y_{B_e} - A_s > 20)\big) \wedge \\
&(B_s - A_e \geq 0) \wedge (A_e - A_s \geq 7) \wedge \\
&\big((A_e - A_s \leq 8) \vee (A_e - A_s \geq 10)\big) \wedge (A_e - A_s \leq 11)
\end{aligned} \tag{9.5}$$

For the simple-natured TCSNU class, it is not necessary to apply a general purpose quantifier elimination procedure. Given the specific nature of the constraints and the limitation to convex contingent constraints, only few cases are possible, and for each of them we use a pattern-based encoding, that in essence pre-computes the result of quantifier elimination. This result can be thought of as generalizing to simple-natured TCSNUs the result proposed by Fargier and Vidal [VF99b] for the case of STNU. We start from the distributed encoding of equation (9.3), where the each sub-formula ψ_h is generated by the hole encoding. We treat each sub-formula as a separate existential quantification problem, and provide static results for each case. The final result is logically equivalent to the corresponding $\psi_h^\Gamma(\vec{T}_{c_h})$ in equation (9.4).

Each conjunct under analysis results from the encoding of a free constraint in the TCSNU over variables v and w, with D intervals. Let t be $v - w$. The encoding results in two unit clauses ($t \geq l_1$ and $t \leq u_D$), and in $D-1$ binary clauses in the form $(t \leq u_i) \vee (t \geq l_{i+1})$.

The static elimination procedure must deal with eight possible cases, depending on v and w being controllable or uncontrollable[3]. The eight possible clause patterns are shown in table 9.1. For unit clauses, we proceed as in the work by Fargier and Vidal [VF99b]: the first four rows of table 9.1 report these results.

[3]The possible cases are actually sixteen but $v - w \geq k$ can be rewritten as $w - v \leq -k$, thus halving the possibilities.

Clause pattern	**Quantification Result** ($\psi_h^\Gamma(\vec{T}_{c_h})$)
$(b_i - b_j) \geq k$	$(b_i - b_j) \geq k$
$(e_i - b_j) \geq k$	$(b_i - b_j) \geq k - L_i$
$(b_i - e_j) \geq k$	$(b_i - b_j) \geq k + U_j$
$(e_i - e_j) \geq k$	$(b_i - b_j) \geq k - L_i + U_j$
$(b_i - b_j) \leq k_1 \vee$ $(b_i - b_j) \geq k_2$	$(b_i - b_j) \leq k_1 \vee (b_i - b_j) \geq k_2$
$(e_i - b_j) \leq k_1 \vee$ $(e_i - b_j) \geq k_2$	$((b_i + L_i - b_j > k_1) \vee (b_i + U_i - b_j \leq k_1)) \wedge$ $((b_i + L_i - b_j < k_1) \vee (b_i + L_i - b_j \geq k_2))$
$(b_i - e_j) \leq k_1 \vee$ $(b_i - e_j) \geq k_2$	$((b_i - b_j - L_j < k_2) \vee (b_i - b_j - U_j \geq k_2)) \wedge$ $((b_i - b_j - L_j > k_2) \vee (b_i - b_j - L_j \leq k_1))$
$(e_i - e_j) \leq k_1 \vee$ $(e_i - e_j) \geq k_2$	$((b_i + U_i - b_j - U_j > k_1) \vee (b_i + U_i - b_j - L_j \leq k_1)) \wedge$ $((b_i + U_i - b_j - U_j < k_1) \vee (b_i + L_i - b_j - U_j \geq k_1)) \wedge$ $((b_i + L_i - b_j - L_j < k_2) \vee (b_i + L_i - b_j - U_j \geq k_2)) \wedge$ $((b_i + L_i - b_j - L_j > k_2) \vee (b_i + L_i - b_j - L_j \leq k_2))$

Table 9.1: Static quantification for simple-natured TCSNUs. For each clause pattern deriving from a hole-encoding of free constraints, the corresponding $\psi_h^\Gamma(\vec{T}_{c_h})$ is presented, assuming that if e_i is an uncontrollable time point, b_i is its corresponding controllable time point that relates to it with the i-th contingent link $\langle b_i, \{\langle L_i, U_i\rangle\}, e_i\rangle$.

The rest of the table present the results for the disjunctive binary clauses. The static quantification is possible by knowing that the contingent links are in the shape $e_i - b_i \in [L_i, U_i]$ and thus each possible free constraint clause can be parametrized and resolved upfront.

During the encoding of a given problem, we can now generate the set of clauses using the hole encoding, search in the table which is the applicable pattern and instantiate the resulting $\psi_h^\Gamma(\vec{T}_{c_h})$ (The quantifier free formula that is equivalent to $\neg(\exists \vec{Y}_{u_h}.(\Gamma(\vec{Y}_u)|_{Y_{u_h}} \wedge \neg\psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h}))))$ that form a set of clauses to be conjoined to obtain a sound and complete SMT encoding for strong controllability.

In the running example, equation (9.5) can be equivalently transformed using this technique as follows.

$$(B_s - A_s \leq 9) \wedge (A_s - B_s \leq 8) \wedge (B_s - A_e \geq 0) \wedge (A_e - A_s \geq 7) \wedge$$
$$((A_e - A_s \leq 8) \vee (A_e - A_s \geq 10)) \wedge (A_e - A_s \leq 11)$$

The constraint $(B_s - A_s \leq 9)$ comes from the $B_e - A_s \leq 20$ clause using the third rule, while $(A_s - B_s \leq 8)$ is derived from $B_e - A_s \geq 0$ using the second rule.

In order to explain the intuition of the rules, let us show, as an example, why the constraint $(B_s - A_s \leq 9)$ comes from the $B_e - A_s \leq 20$. By rewriting the constraint $B_e - A_s \leq 20$, we get $B_s + y_{B_e} - A_s \leq 20$. This inequality must hold for any $y_{B_e} \in [8, 11]$, because y_{B_e} is uncontrollable. A fortiori, it must hold for $y_{B_e} = 11$, that is the worst case for an upper bound constraint[4]. Therefore, we obtain $B_s + 11 - A_s \leq 20$, that is $B_s - A_s \leq 9$.

The construction described above can be used in equation (9.4). This specialized quantification technique results in a 2-CNF formula that has linear size in the original TCSNU. This is because the size of the hole encoding is linear, and for each clause, we statically resolve the quantification by creating at most four new binary clauses. This encoding spares the computational cost of quantifier elimination and produces a highly optimized $\mathcal{QF\_LRA}$ formula.

9.2 Related Work

We discussed the existing approaches for strong controllability in chapter 4, here we compare them with our approach and we discuss alternative solving techniques for our encoding.

[4]Recall that setting $y_{B_e} = 11$ means assuming the duration of activity B to be its maximum, namely it takes 11 time units.

9.2.1 The PVYS Algorithm

The first (and only) technique to solve the strong controllability problem for the DTNU problem class is proposed in [PVYS07]. We described the details of PVYS in section 4.2.2. The key difference between our approach and PVYS is in the nature of enumerations. PVYS explicitly enumerates the choices over the free constraints and contingent links. This may be costly if many disjunctive constraints are present in the problem: in fact, in the worst case, all the possible combinations of disjuncts must be analyzed. In our approach, the enumerations are carried out symbolically, and relying on the SMT infrastructure for efficiency. In this way, we inherit effective splitting heuristics, learning, and backjumping. Moreover, the early pruning mechanism in the SMT solver is able to draw conclusions from "partial" choices, where a disjunct is not (yet) chosen for each clause [BSST09]. Finally, we use more powerful quantifier elimination techniques, that are able to deal with DTNUs at once. In the section 9.3 we present a thorough empirical comparison of our encodings with PVYS.

9.2.2 Polyhedra-Based Approach

The ideas presented in this chapter are based on $\mathcal{LRA}$ formulae, and are made practical by leveraging SMT solvers. In principle, given the geometric interpretation of $\mathcal{LRA}$, the problem could be addressed by other means. In fact, each conjunction of $\mathcal{LRA}$ atoms is a Non-Necessarily Closed Convex Polyhedron (NNC-Polyhedron), and each formula over $\mathcal{LRA}$ can be seen as the (non-convex) union of finitely-many NNC-Polyhedra. In this parallelism, conjunction corresponds to intersection, disjunction to union, negation to complement, and existential quantification to projection.

Libraries for manipulating NNC-Polyhedra are available (e.g. [BHZ08, Wil93]), and could be used as a back-end for the problems described here instead of SMT solvers. In an early stage of this research, we also explored this possibility, experimenting with the Parma Polyhedra Library [BHZ08], one of the most efficient libraries available. The results we obtained were dramatically in favor of the SMT approach. We could identify various reasons for this lack of scalability. On the one side, the explicit manipulation of polyhedra disjunctions may be very costly. On the other, using polyhedra-based solvers, we are computing the entire solution space, while the SMT based approaches are only looking for one solution. Further discussion of these techniques is out of the scope of this thesis.

9.3 Experimental Evaluation

In this Section, we experimentally evaluate our approach. We describe our implementation (section 9.3.1), the experimental set-up (section 9.3.2), and the results for strong controllability (section 9.3.3). Finally, in section 9.3.4 we evaluate (our implementation of) the PVYS algorithm.

9.3.1 Implementation

We developed a tool that automatically encodes the various classes of temporal problems as SMT problems. For strong controllability problems, the tool, depicted in figure 9.3, has two flows. First, it implements the three encodings to quantified SMT ($\mathcal{LRA}$), that are then solved by Z3. Second, it generates quantifier-free SMT ($\mathcal{QF\_LRA}$) encodings, by applying eager quantifier elimination techniques. The quantifier elimination procedure in the eager for-all elimination encoding is carried out by calling one of the following procedures: the internal formula simplifier of Z3 [dMB08] (denoted EFE Z3QE); Fourier-Motzkin quantifier elimination (EFE M5FM), built on top of MATHSAT5 [CGSS13]; and the Loos-Weispfenning (EFE M5LW) procedure, also built on MATHSAT5. The resulting encodings are

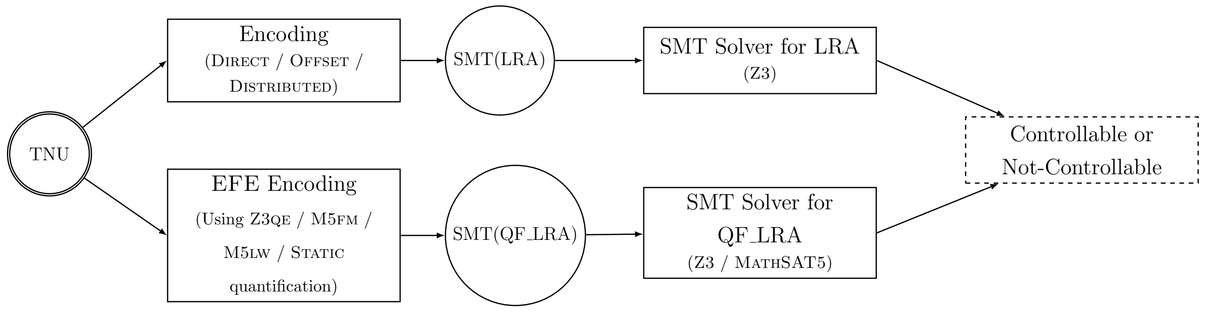

TNU
Encoding
(DIRECT / OFFSET / DISTRIBUTED)
SMT(LRA)
SMT Solver for LRA
(Z3)
EFE Encoding
(Using Z3QE / M5FM / M5LW / STATIC quantification)
SMT(QF_LRA)
SMT Solver for QF_LRA
(Z3 / MATHSAT5)
Controllable or Not-Controllable

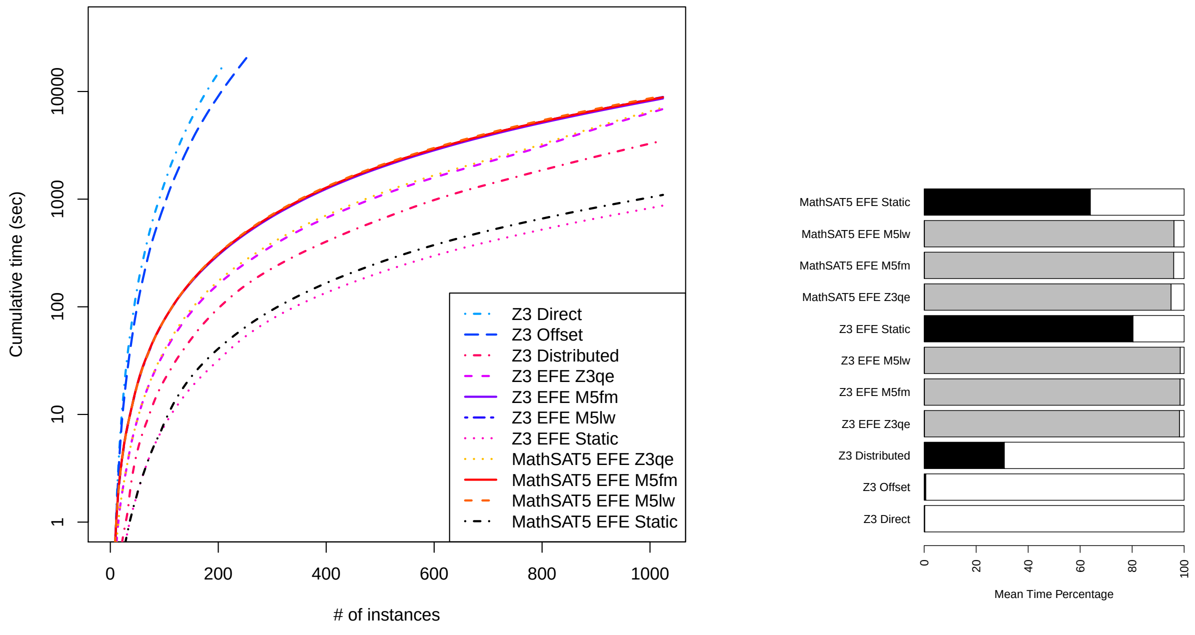
Figure 9.3: Graphical representation of the developed tool-chain.

solved using Z3 and MATHSAT5. Given that the encodings are written in SMT-LIB2 [BST+10] language, it would be straightforward to use any modern SMT solver as a back-end[5]. However, our purpose is to assess the encodings we propose, and not to compare the various SMT solvers. Z3 can be seen as a representative for solvers that support quantified theories, and MATHSAT as representative for quantifier-free solvers. We expect other solvers (e.g. YICES [DdM06b], OPENSMT [BPST10]) to exhibit a similar behavior. (See [BDM+13] for a recent summary on the performances of current state-of-the-art solvers.)

9.3.2 Experimental Set-Up

We used a set of randomly-generated benchmarks. Strong controllability problems are generated by means of an extension of the generator presented in [ACG99], where uncertainty is randomly introduced: each constraint generated by the consistency problem generator is turned in a contingent link with a given probability, and its destination node is considered as uncontrollable. The benchmark set contains 1054 simple-natured instances for each TNU class (STNU, TCSNU and DTNU). We used random instance generators because they are typically used in literature, and because they can be easily scaled to stress the solvers.

[5]In fact, the tool can also generate the benchmarks also in SMT-LIB1 [RLT06] format.



(a) (b)

Figure 9.4: Results for strong controllability experimental evaluation of STNU. In (a) we report a cumulative cactus plot of the results. In (b) we show a breakdown of computation time for the analyzed encodings for the STNU class: encoding time percentage (in black); quantifier elimination time percentage (in gray); solving time percentage (in white). In Eager For-all Elimination Static encodings the static quantification and the encoding time are considered together.

We performed all our experiments on a machine running Scientific Linux 6.0, equipped with two quad-core Xeon processors @ 2.70GHz. We considered a memory limit of 2GB and a time-out of 300 seconds. The benchmarks and the tool are available as indicated in section 1.2.

9.3.3 Results for Strong Controllability

To the best of our knowledge, there are no available solvers for strong controllability problems. Thus, we evaluated the different approaches we presented, to highlight the difference in performance and the respective

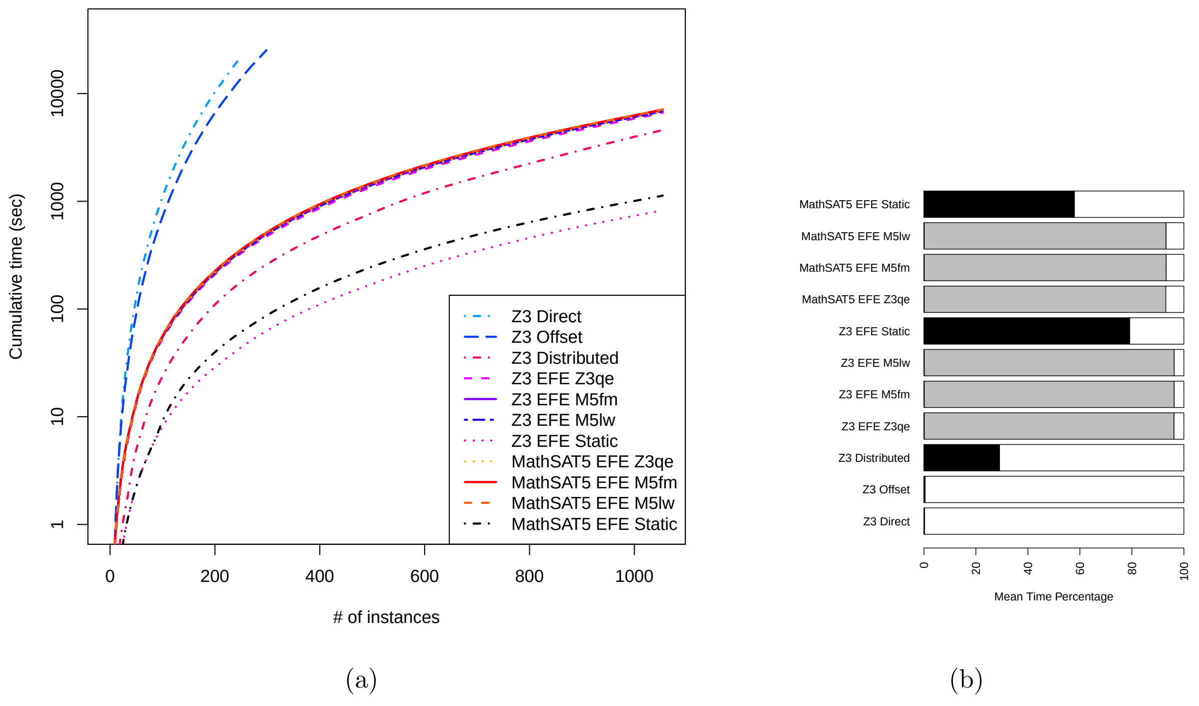
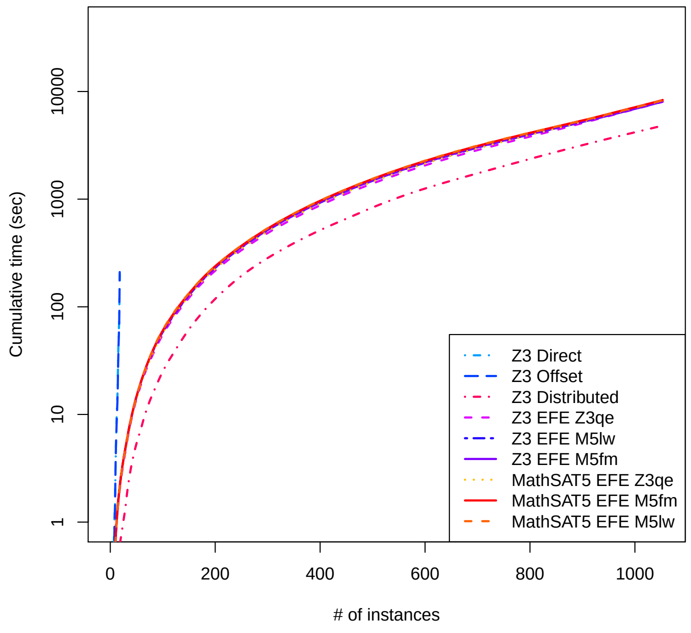


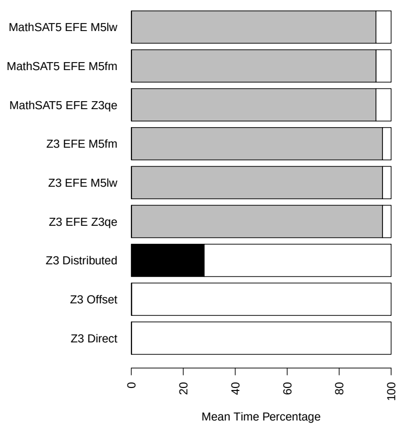
Figure 9.5: Results for strong controllability experimental evaluation of TCSNU. In (a) we report a cumulative cactus plot of the results. In (b) we show a breakdown of computation time for the analyzed encodings for the TCSNU class: encoding time percentage (in black); quantifier elimination time percentage (in gray); solving time percentage (in white). In Eager For-all Elimination Static encodings the static quantification and the encoding time are considered together.

merits. The results for strong controllability are reported in figure 9.4 for the STNU problem class and in figure 9.5 and in figure 9.6 for the TCSNU and DTNU classes, respectively. We plotted in logarithmic scale the cumulative time in seconds to solve the considered set of benchmarks. Differently from the consistency case, the total time includes the encoding time, which may be significant in the case of quantifier-free encodings.

The plots show that the Offset and Direct encodings quickly reach the resource limits, and are unable to solve all the instances. The behavior of the Distributed encoding is slightly better than the eager for-all elimination approaches. The difference can be explained in purely tech-



(a)



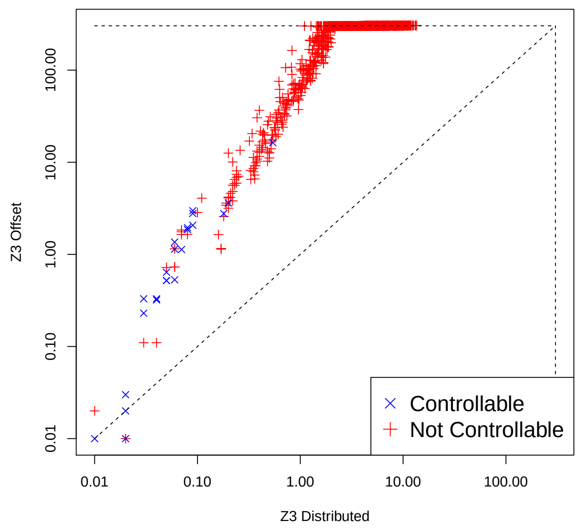
(b)

Figure 9.6: Results for strong controllability experimental evaluation of DTNU. In (a) we report a cumulative cactus plot of the results. In (b) we show a breakdown of computation time for the analyzed encodings for the DTNU class: encoding time percentage (in black); quantifier elimination time percentage (in gray); solving time percentage (in white). In Eager For-all Elimination Static encodings the static quantification and the encoding time are considered together.

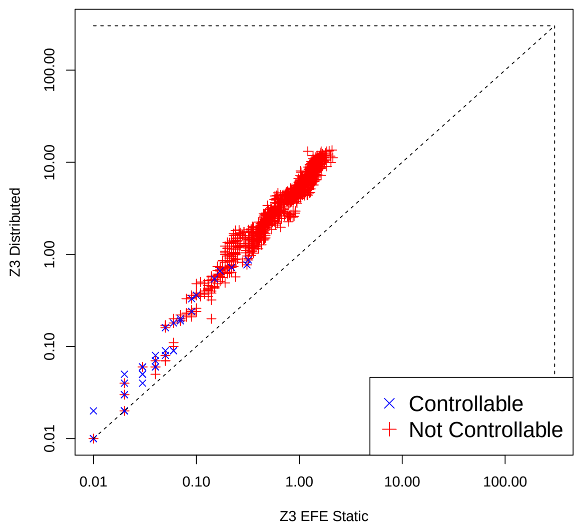
nological terms: the quantifier elimination modules are called via pipe in our implementation, while Z3, on the DISTRIBUTED encoding, performs quantifier elimination "in-memory".

We notice that the static quantification techniques (EFE STATIC), when applicable (i.e. for STNU and simple-natured TCSNU), yield a substantial improvement in performance: the expensive quantifier elimination step is avoided altogether.

In figures 9.7a and 9.7b, we report the scatter plots obtained comparing the performance of the OFFSET and DISTRIBUTED encodings, and the DISTRIBUTED and the EFE STATIC encodings using the Z3 solver. The



(a)



(b)

Figure 9.7: Scatter plot of TCSNU solving time benchmarks obtained using the Z3 solver showing the comparison of OFFSET and DISTRIBUTED encodings (a) and the DISTRIBUTED and the STATIC EAGER FOR-ALL ELIMINATION encodings (b). Controllable instances are marked with blue $\times$ signs, while not controllable instances are marked with red $+$ signs.

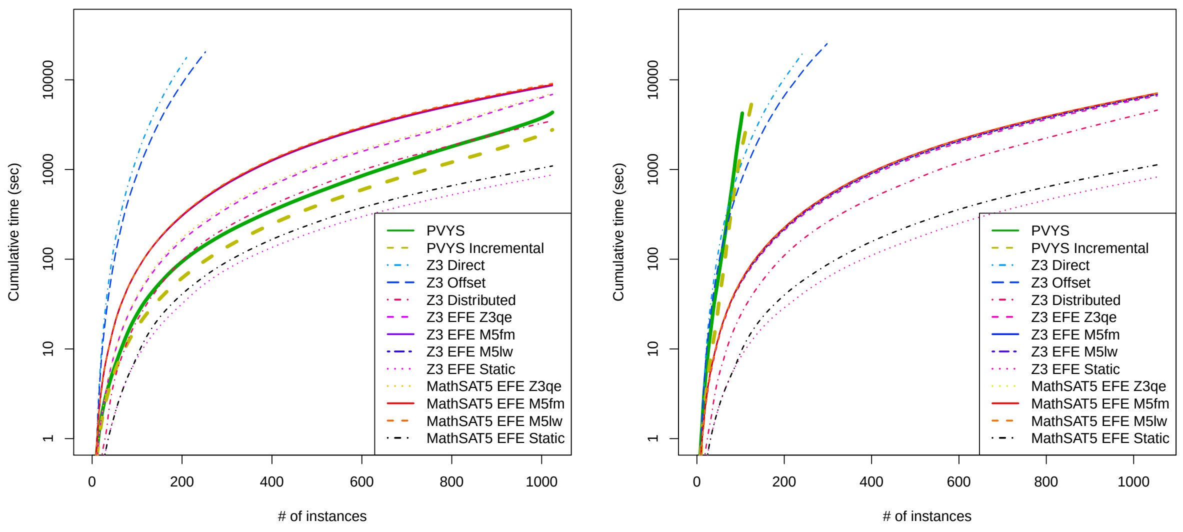
plots show how the performances are affected by the encoding, in fact the OFFSET encoding is unable to solve the most complex instances. Moreover, we see that instances that are harder to solve are the ones that are not strongly controllable for all the tested encodings. In order to assess the real gap between the OFFSET and the DISTRIBUTED encodings, we isolated three TCSNU benchmarks in which the OFFSET encoding timed out, and we tested them without time limits. Two benchmarks were solved in 1356.8 and 26353.2 seconds, while the third one was still running after 33249.8 seconds. This shows that the logical rewriting performed in the DISTRIBUTED encoding yields a very significant performance improvement; in fact, the same benchmarks are solved by the DISTRIBUTED encoding in 1.99, 3.46, and 10.992 seconds respectively. In turn, the static encoding yields a further speed-up (to 1.5, 3.02 and 9.12 seconds).

We also plotted the distribution of time consumption between encoding time, quantifier elimination and solving time (figures 9.4b, 9.5b and 9.6b). The plots highlight the fact that the quantification is the major issue in solving TNUs. The plot shows that the encoding time is absolutely negligible when quantifier elimination is applied, in fact the black part of the diagram is hardly visible. In EFE Static encodings we could not distinguish the quantifier elimination time from the encoding time because the elimination is performed together with the encoding. In approaches where the quantification is demanded to the solver, the vast majority of time is in the SMT solving, while in eager for-all elimination approaches the quantifier elimination dominates the solving time. The relatively high encoding time of Distributed encoding is mainly due to the time needed to printout the big output file in SMT-LIB format.

9.3.4 Comparison with PVYS

In this section, we compare our approach with the PVYS algorithm described in [PVYS07]. We discussed the algorithm in detail in section 4.2.2. To the best of our knowledge, no implementation is available. Therefore we implemented our own version of PVYS. The tool is written in Python, and exploits the MathSAT SMT solver to check the consistency of DTNs. The PVYS pseudo-code reported in [PVYS07] includes steps to compute the minimal network, and to check the consistency of DTNs constructed by the algorithm, but gives no details on how to push and intersect constraints. Thus, we implemented the same operations using the SMT technology.

The tool (also available as indicated in section 1.2) has been implemented in two variants. The first one directly follows the original pseudo-code; the second one exploits the incrementality feature of the MathSAT SMT solver, to gain more efficiency: instead of checking the consistency of each problem separately, it reuses information derived from previous



(a) (b)

Figure 9.8: Results for strong controllability using the PVYS implementation for the STNU (a) and TCSNU (b) problem classes.

checks whenever possible.

In figures 9.8 and 9.9, we overlay the results achieved by PVYS in the same experimental conditions of figures 9.4 to 9.6, respectively. In the STNU problem class, the results of PVYS are comparable to the SMT-based approaches. This is expected, because the implementation of PVYS uses the same SMT-based calls to FargierVidal. In the disjunctive cases, PVYS performs dramatically worse than the SMT-based approaches, due to the enumerative treatment of disjunctions. Finally, we notice that incrementality improves the performance to some extent.

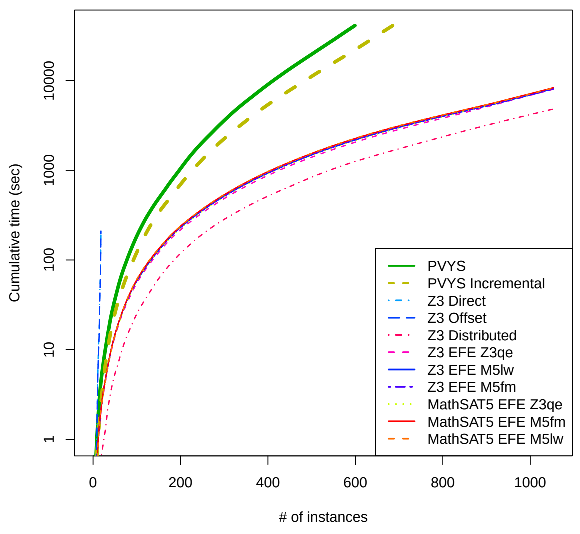


Figure 9.9: Results for strong controllability using the PVYS implementation for the DTNU problem class.

Chapter 10

Weak Controllability

Another query that can be addressed in the context of TNUs is weak controllability, that is concerned with the existence of a strategy that associates values to the controllable starting points of each activity, as a function of the uncontrollable durations. The values for the uncontrollable durations are not known at the moment of solving the problem; however, the executor is given the actual value of such durations just before the execution starts.

There are several reasons for studying weak controllability. From the temporal problems perspective, weak controllability is the conceptually interesting dual of the strong controllability problem. In addition, deciding whether a given TNU is weakly controllable may serve as a pre-check for more complex problems such as dynamic controllability. In fact, weak controllability is a necessary condition for dynamic controllability [VF99b].

From the practical standpoint, weak controllability allows for the modeling of a setting where a number of tasks is to be repeatedly executed, but with modalities that depend on some environmental parameters that become available just prior to execution. For example, an automated production line may be required to perform a set of activities, whose duration functionally depends on the measured size of the objects to be manipulated. The duration of the activities is unknown a priori, except for an upper and lower bound, but it becomes precise once the actual objects materialize. Similarly, in a multi-core processor, the power management may dynamically control the actual clock speeds, thus affecting the duration of jobs. An on-line scheduler may be required to decide the appropriate allocation based on information that may be made available by the power management unit. Another example of application is given in the setting of remote systems (such as space exploration rovers or satellites), where the degradation due to use causes many activities to change duration over time. For example, the movement speed of many components may decrease with the age of the system.

These domains share the fact that the tasks may be repeated multiple times, on platforms of limited capacity, and in conditions that can be estimated prior to execution. As such, they can be encoded as weak controllability problems. In this chapter, we tackle weak controllability for DTNUs, making the following contributions.

First, we propose a general decision procedure for the problem of weak controllability for DTNUs. The decision procedure is based on a reduction to an SMT problem for the theory of Quantified Linear Real Arithmetic ($\mathcal{LRA}$). The encoding can be thought as working by refutation: we state the existence of an assignment to uncontrollable time points that cannot be countered by any controllable assignment. This means that the SMT problem is satisfiable if and only if the TNU is not weakly controllable. The problem can thus be directly provided to an efficient SMT solver. This approach accounts for the first implemented decision procedure for weak controllability of DTNUs.

Then, we investigate the problem of on-line strategy execution, i.e. given a weakly controllable DTNU, how to repeatedly produce a suitable schedule for the controllable time points as a function of a valuation to the uncontrollable ones. We propose an approach, referred to as *implicit strategy execution*, based on the run-time execution of a solver for TN without uncertainty: any valuation to the uncontrollable durations removes the uncertainty from the problem, and thus transforms the TNU at hand into a TN. The solver is then invoked to solve the consistency problem yielding an assignment to the controllable time points. Unfortunately, this solution imposes strong requirements on the run-time: most notably, the control platform must support the execution of a solver; in addition, at each iteration it is required to solve an NP-hard problem, i.e. a DTN (without uncertainty).

This motivates the investigation of efficient run-time execution for weakly controllable TNUs. We analyze the spectrum of *explicit strategies*, expressed in a form that does not require reasoning, and can thus be directly evaluated. We consider *linear strategies*, that are strategies in which the values for the controllable time points are a linear function of the uncontrollable ones; and *piecewise-linear strategies*, that are combinations of different linear strategies, each associated with an activation condition defined over the uncontrollable time points. Linear strategies turn out not to be expressive enough in general: we prove that even for the STNU problem class, a weakly controllable instance is not guaranteed to have a linear strategy. We also prove that piecewise-linear strategies are sufficiently expressive: a piecewise-linear strategy is guaranteed to exist for every weakly controllable DTNU.

Finally, we address the synthesis problem: given a weakly controllable temporal problem, we algorithmically synthesize a function from an assignment to uncontrollable time points to an assignment to the controllable ones. We propose a number of algorithms for the synthesis of a strategy. We start by considering linear strategies, developing two algorithms to produce linear strategies for the STNU and DTNU cases. Then, we generalize to the case of piecewise-linear strategies, and we propose several algorithms for the STNU and DTNU cases.

All the proposed algorithms have been implemented in a tool for solving temporal problems under uncertainty. The tool is developed on top of, and fully leverages, state-of-the-art SMT solvers [dMB08, CGSS13]. To the best of our knowledge, this is the first implementation for weak controllability and strategy extraction. We carried out an extensive experimental evaluation on a comprehensive set of benchmarks. Our implementation, demonstrates high scalability, and is able to automatically extract strategies of significant size. The experimental evaluation highlights a dramatic speed-up in the execution of the synthesized explicit strategies.

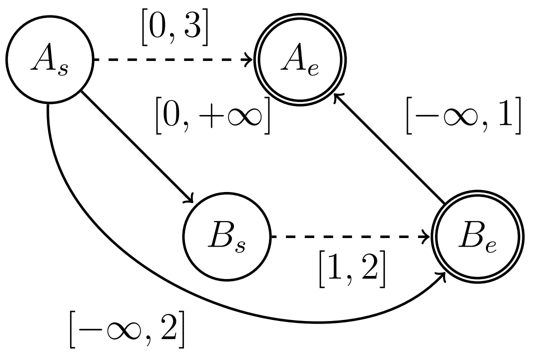
10.1 Weak Controllability Definition

We first formally define the concept of weak controllability exploiting the concept of projection from definition 42. Intuitively, the projection P_ω is the problem without uncertainty in which each uncontrollable duration has been fixed to a given value.

Definition 45. *Let $P \doteq (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be a TNU. P is weakly controllable if and only if for each situation $\omega \in \Omega_P$ the projection P_ω is consistent.*

Definition 45 captures the weak controllability concept by requiring the existence of a schedule for each situation. This definition implicitly models a strategy as a function $f : \Omega_P \to \mathbb{R}^{|\mathcal{T}_c|}$ that maps each situation ω in a schedule for the controllable time points that fulfills the constraints of the projection P_ω. If such a function exists, the problem is weakly controllable.

Weak controllability is, in terms of games, the dual of strong controllability: in strong controllability, the executor is required to make its move (i.e. all its decisions) without observing the situation (i.e. the move of the



(a)

$$\begin{aligned}
\mathcal{T}_c &= \{A_s, B_s\}\\
\mathcal{T}_u &= \{A_e, B_e\}\\
\mathcal{L} &= \{\langle A_e, \{\langle 0, 3\rangle\}, A_s\rangle\\
&\qquad \langle B_e, \{\langle 1, 2\rangle\}, B_s\rangle\}\\
\mathcal{C} &= \{B_s - A_s \in [0, +\infty),\\
&\qquad A_e - B_e \in (-\infty, 1],\\
&\qquad B_e - A_s \in (-\infty, 2]\}
\end{aligned}$$

(b)

Figure 10.1: (a) The running example of a weakly controllable STNU: nodes are time points, double-circled nodes are uncontrollable time points; contingent constraints are depicted as dashed arrows while free constraints are solid. (b) The constraint definitions for the running example.

environment); in weak controllability, the environment is required to make all its decisions before the executor.

To better explain the algorithms and encodings, we consider another example of TNU, depicted in figure 10.1. The example is composed of two activities A and B. A starts at time point A_s and ends in A_e; similarly, B starts at B_s and ends in B_e. The two activities have uncontrollable duration: A_1 has duration between 0 and 3 time units, while A_2 lasts for at least 1 and at most 2 time units. We require A_s to be scheduled before B_s ($B_s - A_s \in [0, +\infty)$), B_s before A_e ($A_e - B_s \in [0, +\infty)$), B_e to happen at most 1 time unit before A_e ($A_e - B_e \in (-\infty, 1]$) and B_e at most 2 time units after A_s ($B_e - A_s \in (-\infty, 2]$).

10.2 Deciding Weak Controllability

In this section we address the decision problem of weak controllability: given a TNU P, we want a decision procedure that answer positively if

$$\begin{aligned}
T_c &= \langle A_s, B_s \rangle \\
Y_u &= \langle y_{A_e}, y_{B_e} \rangle \\
\Gamma(\vec{Y}_u) &= (y_{A_e} \geq 0) \wedge (y_{A_e} \leq 3) \wedge (y_{B_e} \geq 1) \wedge (y_{B_e} \leq 2) \\
\Psi(\vec{T}_c, \vec{Y}_u) &= (B_s - A_s \geq 0) \wedge \\
&\quad ((A_s + y_{A_e}) - (B_s + y_{B_e}) \leq 1) \wedge \\
&\quad ((B_s + y_{B_e}) - A_s \leq 2)
\end{aligned}$$

Figure 10.2: The encoding of the example STNU of figure 10.1.

and only if P is weakly controllable.

In order to logically define weak controllability and obtain a decision encoding, we first perform some manipulations on the problem definition. As in chapter 9, we encode each uncontrollable time point e_i in terms of the time difference with its starting time point b_i by means of an uncontrollable duration variable y_{e_i}. We refer to section 9.1.1 (definition 44) for the encoding details. From here on, we assume an encoded problem $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ is given. Figure 10.2 reports the encoding of the problem depicted in figure 10.1.

Intuitively, a temporal problem is weakly controllable if there exists a strategy that maps every situation to a corresponding assignment to controllable time points, in such a way that all free constraints are satisfied. We can rephrase the concept of weak controllability presented in definition 45 as a satisfiability problem modulo the $\mathcal{LRA}$ theory as follows.

Proposition 10.1 (Weak Controllability Formalization)**.** *Let* $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ *be a TNU and let* $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ *be its encoding. P is weakly controllable if and only if the following formula is valid modulo the* $\mathcal{LRA}$ *theory.*

$$\forall \vec{Y}_u . \exists \vec{T}_c . (\Gamma(\vec{Y}_u) \rightarrow \Psi(\vec{T}_c, \vec{Y}_u)) \tag{10.1}$$

The formula in equation (10.1) is a direct formalization of the intuitive notion of weak controllability, and of the original definition in [VF99b]. The universal quantifier captures the uncertainty in the decision of the duration variables. The implication ensures that free constraints are checked only when $\Gamma(\vec{Y_u})$ is satisfied, that is only on assignments that encode situations of the original temporal problem. In fact, if $\Gamma(\vec{Y_u})$ is not satisfied, the implication is automatically satisfied. Equation (10.1) is a formula in $\mathcal{LRA}$ that is valid if and only if the problem is weakly controllable.

For example, the problem depicted in figure 10.1a is weakly controllable if and only if the following formula is valid.

$$\begin{aligned}\forall y_{A_e}, y_{B_e}.\exists A_s, B_s.(((y_{A_e} \geq 0) \wedge (y_{A_e} \leq 3) \wedge (y_{B_e} \geq 1) \wedge (y_{B_e} \leq 2)) \rightarrow \\ ((B_s - A_s \geq 0) \wedge ((A_s + y_{A_e}) - (B_s + y_{B_e}) \leq 1) \wedge \\ ((B_s + y_{B_e}) - A_s \leq 2)))\end{aligned}$$

Looking at the weak controllability formal characterization in proposition 10.1 from an SMT perspective, it is clear that we are solving the validity problem of an $\mathcal{LRA}$ formula. Any SMT solver supporting $\mathcal{LRA}$ is able to deal with such a formula directly and it can correctly solve the problem. However, due to the high computational cost of directly handling quantifiers, an optimized encoding is required.

We first rewrite the formula encoding weak controllability in proposition 10.1 by transforming the external universal quantifier into the negation of an existential one, and we consider the negation of the resulting formula. We call the resulting formula *inverted encoding.*

$$\neg\exists\vec{T_c}.(\Gamma(\vec{Y_u}) \rightarrow \Psi(\vec{T_c}, \vec{Y_u})) \tag{10.2}$$

If this formula is unsatisfiable, then the problem is weakly controllable, while if it is satisfiable, then the problem is not weakly controllable. Note that in equation (10.2) we dropped the outermost $\neg\exists\vec{Y_u}$ as any SMT problem is inherently an existential quantification and we consider the negation

$$\begin{aligned}
\neg\exists A_s, B_s.((y_{A_e} \geq 0) \land (y_{A_e} \leq 3)\land \\
(y_{B_e} \geq 1) \land (y_{B_e} \leq 2)) \rightarrow \\
((B_s - A_s \geq 0)\land \\
((A_s + y_{A_e}) - (B_s + y_{B_e}) \leq 1)\land \\
((B_s + y_{B_e}) - A_s \leq 2))
\end{aligned}$$

(a)

$$\begin{aligned}
(y_{A_e} \geq 0) \land (y_{A_e} \leq 3)\land \\
(y_{B_e} \geq 1) \land (y_{B_e} \leq 2)\land \\
\neg\exists A_s, B_s.((B_s - A_s \geq 0)\land \\
((A_s + y_{A_e}) - (B_s + y_{B_e}) \leq 1)\land \\
((B_s + y_{B_e}) - A_s \leq 2))
\end{aligned}$$

(b)

Figure 10.3: Inverted encoding (a) and assumption-extraction encoding (b) applied to the running example STNU of figure 10.1.

by reversing the interpretation of the result. Intuitively, we are searching for an assignment to the uncontrollable time points that is able to violate the free constraints under any possible strategy (it is a winning strategy for the environment). In fact, if the formula is satisfiable, each model corresponds to a situation for which no weak strategy to schedule the controllable time points exists. Therefore, differently from equation (10.1), this encoding is also helpful for debugging a non-weakly controllable problem. This encoding still requires a solver with full support of $\mathcal{LRA}$, but is able to exploit the searching power of the SMT framework and, in case of non-weak controllability, it allows for the extraction of debug information by providing a model of the formula. An example of this encoding for the running example problem is shown in figure 10.3a.

A further improvement can be achieved by limiting as much as possible the scope of the existential quantifier. To this extent, we push the existential quantifier over the implication, and thus the quantification is limited to the free constraints only (ref. as *assumption-extraction* encoding):

$$\Gamma(\vec{Y_u}) \land \neg\exists\vec{T_c}.\Psi(\vec{T_c}, \vec{Y_u}). \tag{10.3}$$

The assumption-extraction encoding for the running example problem is reported in figure 10.3b.

The following proposition states that the inverted and assumption-extraction encodings are logically equivalent: the proof can be found in appendix A.3.

Proposition 10.2 (Assumption Extraction Correctness)**.** *Equation* (10.2) *and equation* (10.3) *are logically equivalent.*

10.3 Strategies for Weak Controllability

We now consider the problem of actually executing a control strategy that is associated with a given weakly controllable TNU. A TNU is a modeling framework that represents a set of assumptions over the environment and imposes a set of requirements to be fulfilled. We consider the use-case in which a strategy for scheduling the controllable time points is repeatedly executed by reading the inputs from the environment in the form of a situation. Such a situation is generated by reading the parameters on which the uncontrollable durations depends, by means of appropriate sensors or estimators. The strategy computes an assignment to the controllable time points that fulfills the problem constraints and is then deployed to an actuator for execution.

The problem we tackle here is to automatically synthesize such a strategy: we discuss two approaches. First, we use a TN solver to do on-line reasoning, thus executing a control strategy that is *implicitly* defined in the TNU, if solvable. Then, we investigate the idea of *explicit* strategies, that can be readily executed without resorting to on-line reasoning.

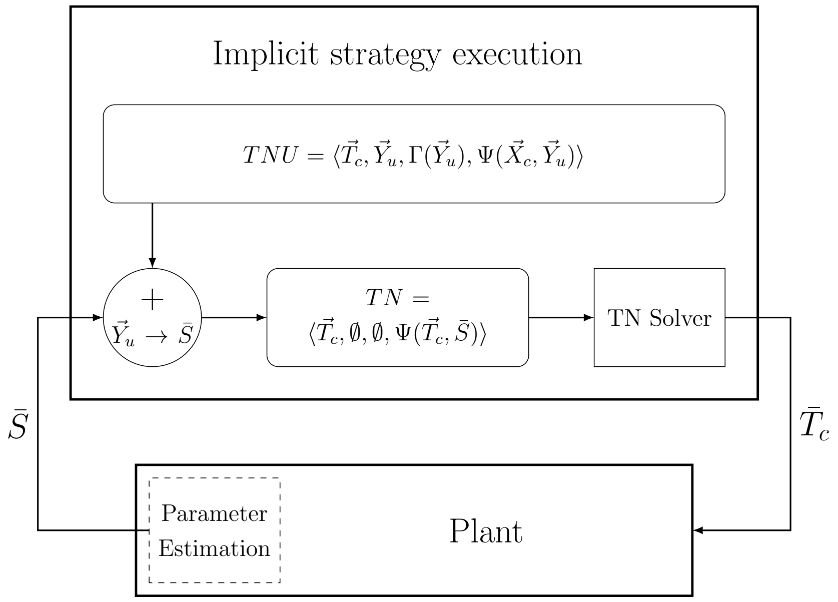

Implicit strategy execution
$TNU = \langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{X}_c, \vec{Y}_u) \rangle$
$+$
$\vec{Y}_u \to \bar{S}$
$TN =$
$\langle \vec{T}_c, \emptyset, \emptyset, \Psi(\vec{T}_c, \bar{S}) \rangle$
TN Solver
$\bar{S}$
$\bar{T}_c$
Parameter
Estimation
Plant

Figure 10.4: Schematic view of implicit strategy mechanism. The strategy is repeatedly executed once a situation is obtained by estimating the relevant parameters in the Plant. The output of the strategy is a controllable schedule (i.e. an assignment $\bar{T}_c$ to all the controllable time points). The implicit strategy works by "projecting away" the uncertainty in the TNU: the uncontrollable durations $\vec{Y}_u$ are substituted with the actual values of the situation $\bar{S}$. Then, a TN is obtained and is solved using a TN solver, yielding the assignment ($\bar{T}_c$) to the controllable time points.

10.3.1 Implicit Strategies

A way of obtaining a strategy for a weakly controllable TNU is given by definition 42 and depicted in figure 10.4: when a situation $\bar{S}$ is read[1], we eliminate the uncertainty by substituting the uncontrollable duration variables in the TNU formulation with the values obtained from the situation (obtaining a TN that is the projection of the TNU). Then, we solve the resulting temporal problem, that is now without uncertainty, and return the assignment to the controllable time points (indicated as $\bar{T}_c$) for execution. Formally, given the encoding of a TNU $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ and an assignment to all the uncontrollable durations $\bar{S}$ fulfilling $\Gamma(\bar{S})$ (a

[1]$\bar{S}$ is a vector of $|\vec{Y}_u|$ rational numbers, one for each uncontrollable duration.

situation), we can find an assignment to the controllable variables $\vec{T_c}$ by finding a model for the formula $\Psi(\vec{T_c}, \bar{S})$. This strategy requires a solver to be executed once the situation $\bar{S}$ is known.

In practice, we can implement this idea using any SMT solver by searching for a model for $\Psi(\vec{T_c}, \bar{S})$. However, this approach (called Implicit-SMT) requires one to solve a separate SMT problem for each situation. A more advanced approach is to exploit the incrementality feature of modern SMT solvers [BSST09], allowing the solver to "recycle" discovered clauses and lemmas among different situations. For this purpose, we designed an incremental approach, described in algorithm 6. Implicit-SMT-Incremental takes the encoding $\langle \vec{T_c}, \vec{Y_u}, \Gamma(\vec{Y_u}), \Psi(\vec{T_c}, \vec{Y_u})\rangle$, and initializes the SMT solver by asserting the free constraints $\Psi(\vec{T_c}, \vec{Y_u})$. Then, it enters a (possibly infinite) loop, and processes a sequence of situations $\bar{S}^1, \bar{S}^2, \cdots$. The problem description is asserted in the solver once and for all, while the situation is first asserted, and once an assignment is found, it is retracted.

The main drawback of the implicit approach is the requirement of on-line reasoning. In fact, once the situation is known, a solver is invoked to discover the assignment for the controllable time points. Solving the TN resulting from the projection of a TNU is hard in general. If the problem belongs to the STNU problem class the resulting STN can be solved in polynomial time, but for the general case of DTNU, the projection results in a DTN that is, in general, NP-hard [SK00]. In addition, having a solver as part of the run-time may require much more expensive platforms.

10.3.2 Explicit Strategies

We avoid the burden of on-line reasoning by providing techniques for the synthesis of functions that are simple and fast to execute. Consider the formalization in proposition 10.1. Interestingly, we can apply skolemiza-

Algorithm 6 Implicit strategy execution based on SMT with incrementality.

```
procedure IMPLICIT-SMT-INCREMENTAL($\Gamma(\vec{Y}_u)$, $\Psi(\vec{T}_c, \vec{Y}_u)$)
    for all $b_i \in \vec{T}_c$ do
        SMT.DECLAREREALVAR($b_i$)
    end for
    for all $y_j \in \vec{Y}_u$ do
        SMT.DECLAREREALVAR($y_i$)
    end for
    SMT.ASSERT($\Psi(\vec{T}_c, \vec{Y}_u)$)
    loop
        $\bar{S} := $ WAITFORSITUATION()
        SMT.PUSH()
        for all $y_i \in \vec{Y}_u$ do
            SMT.ASSERT($y_i = \bar{S}_i$)
        end for
        if SMT.SOLVE = SAT then
            $\mu := $ SMT.GETMODEL()
            EXCECUTETIMEPOINTS($\mu$)
        else
            $\perp$ ▷ Unreachable if the problem is weakly controllable and $S$ fulfills $\Gamma(\vec{Y}_u)$
        end if
        SMT.POP()
    end loop
end procedure
```

tion [Kle67], thus replacing the existential quantifier by means of a fresh function symbol. The following theorem formalizes this idea, the proof is reported in appendix A.3.

Theorem 10.1 (Weak Controllability Skolemization)**.** *A TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ is weakly controllable if and only if the formula*

$$\forall \vec{Y}_u . \Gamma(\vec{Y}_u) \rightarrow \Psi(f(\vec{Y}_u), \vec{Y}_u) \tag{10.4}$$

is satisfiable.

We transform the inner existential quantifier into a function f that models the weak strategy for the problem. In fact, in equation (10.4), the interpretation of the function f is exactly a strategy that solves the problem. Equation (10.4) gives a clear vision of what a strategy is: a function that gets in input the uncontrollable durations and returns an assignment to the controllable time points that fulfills all the problem constraints.

In principle, one would like to exploit this formulation to query an SMT solver, and extract, from the model, a closed form for the strategy f. However, equation (10.4) is a quantified first-order formula involving uninterpreted functions[2], that is in general undecidable.

In the following, we focus on two types of strategies: *linear strategies*, where each controllable variable is computed as a linear combination of the uncontrollable durations; *piecewise-linear strategies*, where different linear strategies are executed depending on the input situation.

From here on, we assume the encoding $\langle \vec{T_c}, \vec{Y_u}, \Gamma(\vec{Y_u}), \Psi(\vec{T_c}, \vec{Y_u})\rangle$ of a TNU is given. In general, a weak strategy is a function that maps each assignment to uncontrollable durations satisfying $\Gamma(\vec{Y_u})$ (i.e. each situation) into an assignment to the controllable time points, such that all the free constraints are satisfied.

Definition 46. *A weak strategy for a TNU is a function $f : \mathbb{R}^{|\vec{Y_u}|} \to \mathbb{R}^{|\vec{T_c}|}$ defined for every point $\vec{Y_u}$ in $\Gamma(\vec{Y_u})$ and such that $\Psi(f(\vec{Y_u}), \vec{Y_u})$ holds for every $\vec{Y_u}$ in $\Gamma(\vec{Y_u})$.*

Note that, this definition does not impose any constraint (e.g. linearity, continuity) on f other than being a function.

In definition 46, we modeled a weak strategy as a single function $f : \mathbb{R}^{|\vec{Y_u}|} \to \mathbb{R}^{|\vec{T_c}|}$, but we can equivalently consider a set of functions $f_1, \ldots, f_{|\vec{T_c}|}$ each computing a schedule for a single controllable time point given the

[2]Formally, equation (10.4) is a quantified first-order formula expressed in the theory combination of $\mathcal{LRA}$ and the theory of uninterpreted functions ($\mathcal{EUF}$) [BSST09].

situation. The two formalizations are equivalent because if there exists a unique function f, we can obtain the set of function by projection of f and vice-versa.

Let $\bar{f}(\vec{Y_u}) : \mathbb{R}^{|\vec{Y_u}|} \rightarrow \mathbb{R}^{|\vec{T_c}|}$ be a strategy. The strategy imposes a relation between the controllable time points and the uncontrollable durations: $\vec{T_c} = \bar{f}(\vec{Y})$. If such a relation is expressible as a formula in a theory $\mathcal{T}$ we can check whether $\bar{f}$ is a weak strategy for a given temporal problem by checking the existence of a point in $\Gamma(\vec{Y_u})$ that violates the free constraints.

$$\Gamma(\vec{Y_u}) \wedge \neg\Psi(\vec{T_c}, \vec{Y_u}) \wedge (\vec{T_c} = \bar{f}(\vec{Y_u})) \tag{10.5}$$

If equation (10.5) is satisfiable modulo $\mathcal{T} \cup \mathcal{QF\_LRA}$, then $\bar{f}(\vec{Y_u})$ is not a valid weak strategy, because there exists a situation for which the strategy violates the free constraints. In this case, $\mathcal{T}$ can be any theory needed to express the relation imposed by the strategy, for example it could be $\mathcal{LRA}$ or even Nonlinear Real Arithmetic. Note that, this check is very useful in practice if $\bar{f}(\vec{Y_u})$ can be expressed in $\mathcal{QF\_LRA}$ because the entire check would fit in $\mathcal{QF\_LRA}$. In the following, we describe two possible shapes of strategies, namely linear and piecewise-linear. Both the shapes can be expressed in $\mathcal{QF\_LRA}$. Therefore checking if such strategies are weak strategies for a given problem is possible by performing a single call to an SMT solver in $\mathcal{QF\_LRA}$.

Linear strategies. A linear strategy is such that the value of every controllable time point is obtained as a linear combination of $\vec{Y_u}$. Let $n \doteq |\vec{T_c}|$ and $m \doteq |T_u|$. A linear strategy can be represented with a matrix A of real coefficients of size $n \times m$ and a vector $\vec{c}$ of size n. Every controllable variable is scheduled according to a linear function of the uncontrollable durations. The strategy $f(\vec{Y_u})$ can be then expressed as $A \cdot \vec{Y_u} + \vec{c}$ in which each $b_i \in \vec{T_c}$ can be computed as $A_{i,1}y_{A_e} + \ldots + A_{i,m}y_m + c_i$. Therefore, the matrix A must have one column for every duration and the vector $\vec{c}$ contains the constant additive terms. The problem of synthesizing a linear strategy is then equivalent to the problem of finding a suitable matrix A and vector $\vec{c}$.

Piecewise-linear strategies. A more general form of strategy is the piecewise-linear strategy, that is the composition of a finite number of linear strategies. A piecewise-linear strategy is defined by cases over a finite partition of the situations (a partition of the region represented by $\Gamma(\vec{Y_u})$). For each case we have a linear strategy that is a valid weak strategy for that subset of the situations. We can compose these linear strategies by first checking in which element of the partition the observed situation belongs, and then applying the corresponding linear strategy. In this setting, a linear strategy is a particular case of a piecewise-linear strategy in which we have a partition of cardinality one.

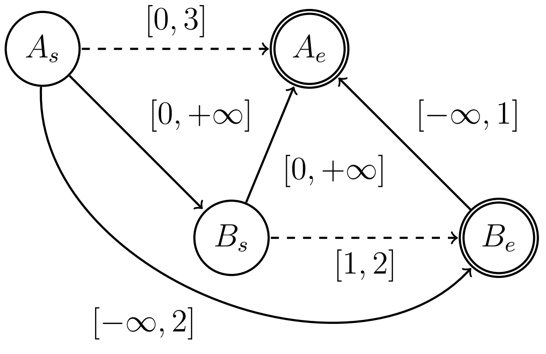
Definition 47. *A* piecewise-linear strategy *is a function*

$$f(\vec{Y_u}) \doteq \begin{cases} f^1(\vec{Y_u}) & \text{if } \eta^1(\vec{Y_u}) \\ f^2(\vec{Y_u}) & \text{else if } \eta^2(\vec{Y_u}) \\ \dots & \\ f^k(\vec{Y_u}) & \text{else if } \eta^k(\vec{Y_u}) \end{cases}$$

where each f^i is a linear strategy and $\eta^i(\vec{Y_u})$ are sub-regions of $\Gamma(\vec{Y_u})$ such that $\Gamma(\vec{Y_u}) \subseteq (\bigcup_{i=1}^{k} \eta^i(\vec{Y_u}))$.

Note that, even this kind of strategy can be directly encoded in $\mathcal{QF\_LRA}$. We call each pair $\langle f^i(\vec{Y_u}), \eta^i(\vec{Y_u})\rangle$ a "piece" of the strategy. In order to compactly represent a piecewise-linear strategy in the algorithms we abstract a piecewise-linear strategy $f(\vec{Y_u})$ as the ordered list of its pieces. For example, the strategy $f(\vec{Y_u})$ in definition 47 can be represented as the following list of pieces:

$$\langle\langle f^1(\vec{Y_u}), \eta^1(\vec{Y_u})\rangle, \langle f^2(\vec{Y_u}), \eta^2(\vec{Y_u})\rangle, \dots, \langle f^k(\vec{Y_u}), \eta^k(\vec{Y_u})\rangle\rangle.$$



(a)

$$\begin{aligned}
\mathcal{T}_c &= \{A_s, B_s\}\\
\mathcal{T}_u &= \{A_e, B_e\}\\
\mathcal{L} &= \{\langle A_e, \{\langle 0,3\rangle\}, A_s\rangle,\\
&\qquad \langle B_e, \{\langle 1,2\rangle\}, B_s\rangle\}\\
\mathcal{C} &= \{B_s - A_s \in [0,+\infty),\\
&\quad (*)\ A_e - B_s \in [0,+\infty),\\
&\qquad A_e - B_e \in (-\infty,1],\\
&\qquad B_e - A_s \in (-\infty,2]\}
\end{aligned}$$

(b)

Figure 10.5: (a) The modified running example STNU: the problem is weakly controllable, but does not have any linear strategy. (b) The constraint definitions for the modified running example.

Following definition 47, no continuity requirement is imposed on piecewise-linear strategies. Continuity is not required by the weak controllability definition and is not a useful requirement for our setting, as we assume that the parameters yielding the situation are fully specified before scheduling the problem.

Linearity is not enough. A linear strategy is very useful in practice: it is compact to represent and easy to evaluate. In fact, it can be represented using just a matrix and a vector; moreover, given an assignment to the uncontrollable duration, we can compute the resulting assignment to the controllable variables by means of a single matrix multiplication. In general, unfortunately, a weakly controllable TNU is not guaranteed to have such a strategy. In fact, even the STNU class of problems is not guaranteed to admit such a strategy for every weakly controllable instance. The following theorem states that there exists a weakly controllable STNU

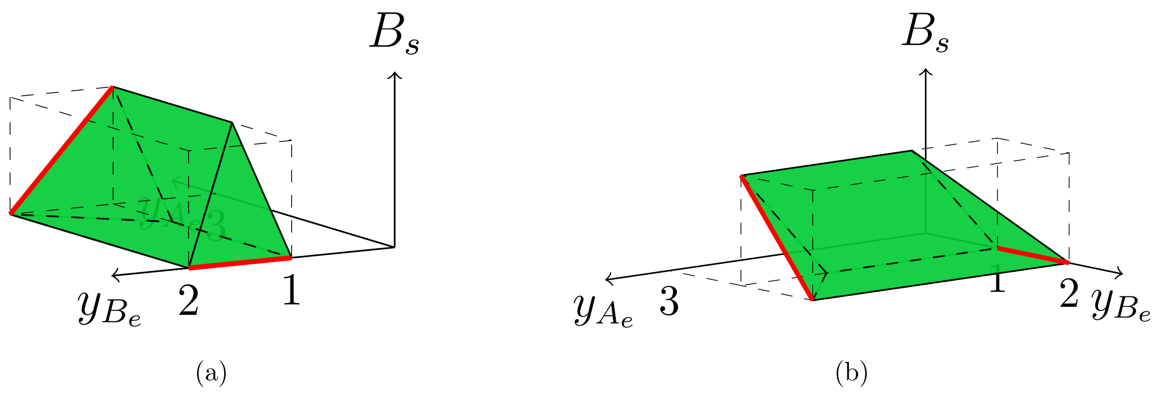


Figure 10.6: (a) The region of feasibility of the STNU in figure 10.2 with $A_s = 0$ in the space of B_s, y_{A_e} and y_{B_e}, depicted from two different angles.

without any linear strategies.

Theorem 10.2 (Linearity Insufficiency for STNU)**.** *There exists an STNU that is weakly controllable and does not have any linear strategy.*

Proof. Let us consider the STNU depicted in figure 10.5 obtained by adding the constraint $A_e - B_s \in [0, +\infty)$ to the running example in figure 7.2. In the following we show that this STNU is weakly controllable, but there exists no linear strategy.

The problem is weakly controllable, because we can apply the following piecewise-linear weak strategy.

$$\begin{pmatrix} A_s \\ B_s \end{pmatrix} = f(y_{A_e}, y_{B_e}) \doteq \begin{cases} \begin{pmatrix} 0 \\ y_{A_e} - y_{B_e} - 1 \end{pmatrix} & \text{if } (y_{B_e} \leq y_{A_e} - 1) \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{otherwise} \end{cases}$$

This strategy corresponds to the following assignments.

$$A_s = 0$$

$$B_s = \begin{cases} y_{A_e} - y_{B_e} - 1 & \text{if } (y_{B_e} \leq y_{A_e} - 1) \\ 0 & \text{otherwise} \end{cases}$$

This strategy clearly covers the entire uncontrollable space because it is a total function. Given this strategy, the free constraints are always satisfied: assuming $\Gamma(y_{A_e}, y_{B_e})$ and $y_{B_e} > y_{A_e} - 1$, the formula $\Psi(\vec{T}_c, \vec{Y}_u)$ reduces to

$$\begin{aligned} \Psi(f(y_{A_e}, y_{B_e}), y_{A_e}, y_{B_e}) \doteq\ & (0 - 0 \geq 0) \wedge ((0 + y_{A_e}) - 0 \geq 0) \wedge \\ & ((0 + y_{A_e}) - (0 + y_{B_e}) \leq 1) \wedge \\ & ((0 + y_{B_e}) - 0 \leq 2) \\ \Leftrightarrow\ & (y_{A_e} \geq 0) \wedge (y_{A_e} - y_{B_e} \leq 1) \wedge (y_{B_e} \leq 2). \end{aligned} \tag{10.6}$$

The atoms $(y_{A_e} \geq 0)$ and $(y_{B_e} \leq 2)$ follow from the assumptions of $\Gamma(y_{A_e}, y_{B_e})$ while the atom $(y_{A_e} - y_{B_e} \leq 1)$ is entailed by the condition of the piece: $y_{B_e} > y_{A_e} - 1$.

Considering the other piece, namely the case $y_{B_e} \leq y_{A_e} - 1$, we obtain the following.

$$\begin{aligned} \Psi(f(y_{A_e}, y_{B_e}), y_{A_e}, y_{B_e}) \doteq\ & (y_{A_e} - y_{B_e} - 1 - 0 \geq 0) \wedge \\ & ((0 + y_{A_e}) - (y_{A_e} - y_{B_e} - 1) \geq 0) \wedge \\ & ((0 + y_{A_e}) - (y_{A_e} - y_{B_e} - 1 - y_{B_e}) \leq 1) \wedge \\ & ((y_{A_e} - y_{B_e} - 1 + y_{B_e}) - 0 \leq 2) \\ \Leftrightarrow\ & (y_{B_e} \leq y_{A_e} - 1) \wedge (y_{B_e} \geq 1) \wedge (1 \leq 1) \wedge (y_{A_e} \leq 3) \end{aligned} \tag{10.7}$$

The atoms $(y_{B_e} \geq 1)$ and $(y_{A_e} \leq 3)$ follow from the assumptions of $\Gamma(y_{A_e}, y_{B_e})$ while the atom $(y_{B_e} \leq y_{A_e} - 1)$ is exactly the condition of the piece we are considering.

We now show that no linear strategy exists for the given problem. For the sake of contradiction, let us suppose that a linear strategy exists for the problem. Let $\bar{f}(\vec{Y}_u) \doteq A \cdot \vec{Y}_u + \vec{c}$ be such a linear strategy. Then,

$A_s \doteq A_{1,1}y_{A_e} + A_{1,2}y_{B_e} + c_1$ and $B_s \doteq A_{2,1}y_{A_e} + A_{2,2}y_{B_e} + c_2$. If $\bar{f}(\vec{Y_u})$ is a valid weak linear strategy, it must fulfill the problem constraints in all the situations. Let us consider four particular situations, namely $\omega_1 = \langle 0, 1\rangle$ (that is, $y_{A_e} \doteq 0$ and $y_{B_e} \doteq 1$), $\omega_2 = \langle 0, 2\rangle$, $\omega_3 = \langle 3, 1\rangle$ and $\omega_4 = \langle 3, 2\rangle$.

We can now build the following system obtained by instantiating each constraint of $\Psi(A_s, B_s, y_{A_e}, y_{B_e})$ in each of the four picked situations, and by substituting each b_i with its strategy definition.

$$\begin{cases}
(A_{2,1}\cdot 0 + A_{2,2}\cdot 1 + c_2) - (A_{1,1}\cdot 0 + A_{1,2}\cdot 1 + c_1) \geq 0 \\
(A_{1,1}\cdot 0 + A_{1,2}\cdot 1 + c_1 + 0) - (A_{2,1}\cdot 0 + A_{2,2}\cdot 1 + c_2) \geq 0 \\
(A_{1,1}\cdot 0 + A_{1,2}\cdot 1 + c_1 + 0) - (A_{2,1}\cdot 0 + A_{2,2}\cdot 1 + c_2 + 1) \leq 1 \\
(A_{2,1}\cdot 0 + A_{2,2}\cdot 1 + c_2 + 1) - (A_{1,1}\cdot 0 + A_{1,2}\cdot 1 + c_1) \leq 2 \\
(A_{2,1}\cdot 0 + A_{2,2}\cdot 2 + c_2) - (A_{1,1}\cdot 0 + A_{1,2}\cdot 2 + c_1) \geq 0 \\
(A_{1,1}\cdot 0 + A_{1,2}\cdot 2 + c_1 + 0) - (A_{2,1}\cdot 0 + A_{2,2}\cdot 2 + c_2) \geq 0 \\
(A_{1,1}\cdot 0 + A_{1,2}\cdot 2 + c_1 + 0) - (A_{2,1}\cdot 0 + A_{2,2}\cdot 2 + c_2 + 2) \leq 1 \\
(A_{2,1}\cdot 0 + A_{2,2}\cdot 2 + c_2 + 2) - (A_{1,1}\cdot 0 + A_{1,2}\cdot 2 + c_1) \leq 2 \\
(A_{2,1}\cdot 3 + A_{2,2}\cdot 1 + c_2) - (A_{1,1}\cdot 3 + A_{1,2}\cdot 1 + c_1) \geq 0 \\
(A_{1,1}\cdot 3 + A_{1,2}\cdot 1 + c_1 + 3) - (A_{2,1}\cdot 3 + A_{2,2}\cdot 1 + c_2) \geq 0 \\
(A_{1,1}\cdot 3 + A_{1,2}\cdot 1 + c_1 + 3) - (A_{2,1}\cdot 3 + A_{2,2}\cdot 1 + c_2 + 1) \leq 1 \\
(A_{2,1}\cdot 3 + A_{2,2}\cdot 1 + c_2 + 1) - (A_{1,1}\cdot 3 + A_{1,2}\cdot 1 + c_1) \leq 2 \\
(A_{2,1}\cdot 3 + A_{2,2}\cdot 2 + c_2) - (A_{1,1}\cdot 3 + A_{1,2}\cdot 2 + c_1) \geq 0 \\
(A_{1,1}\cdot 3 + A_{1,2}\cdot 2 + c_1 + 3) - (A_{2,1}\cdot 3 + A_{2,2}\cdot 2 + c_2) \geq 0 \\
(A_{1,1}\cdot 3 + A_{1,2}\cdot 2 + c_1 + 3) - (A_{2,1}\cdot 3 + A_{2,2}\cdot 2 + c_2 + 2) \leq 1 \\
(A_{2,1}\cdot 3 + A_{2,2}\cdot 2 + c_2 + 2) - (A_{1,1}\cdot 3 + A_{1,2}\cdot 2 + c_1) \leq 2
\end{cases}$$

This system can be rewritten as follows.

$$\begin{cases}
-A_{1,2} + A_{2,2} - c_1 + c_2 \geq 0 \\
A_{1,2} - A_{2,2} + c_1 - c_2 \geq 0 \\
-A_{1,2} + A_{2,2} - c_1 + c_2 \geq 0 \\
A_{1,2} - A_{2,2} + c_1 - c_2 \geq -1 \\
-2A_{1,2} + 2A_{2,2} - c_1 + c_2 \geq 0 \\
2A_{1,2} - 2A_{2,2} + c_1 - c_2 \geq 0 \\
-2A_{1,2} + 2A_{2,2} - c_1 + c_2 \geq -3 \\
+2A_{1,2} - 2A_{2,2} + c_1 - c_2 \geq 0 \\
-3A_{1,1} - A_{1,2} + 3A_{2,1} + A_{2,2} - c_1 + c_2 \geq 0 \\
3A_{1,1} + A_{1,2} - 3A_{2,1} - A_{2,2} + c_1 - c_2 \geq -3 \\
-3A_{1,1} - A_{1,2} + 3A_{2,1} + A_{2,2} - c_1 + c_2 \geq 1 \\
3A_{1,1} + A_{1,2} - 3A_{2,1} - A_{2,2} + c_1 - c_2 \geq -1 \\
-3A_{1,1} - 2A_{1,2} + 3A_{2,1} + 2A_{2,2} - c_1 + c_2 \geq 0 \\
3A_{1,1} + 2A_{1,2} - 3A_{2,1} - 2A_{2,2} + c_1 - c_2 \geq -3 \\
-3A_{1,1} - 2A_{1,2} + 3A_{2,1} + 2A_{2,2} - c_1 + c_2 \geq 0 \\
-3A_{2,1} - 2A_{2,2} + 3A_{1,1} + 2A_{1,2} + c_1 - c_2 \geq 0
\end{cases}$$

The system admits no solution in the real numbers. Therefore there exists no linear strategy for the given problem as there exists no assignment to the coefficients that are able to fulfill the four situations at the same time. □

In order to graphically explain the reason why no linear strategy exists for the given problem, we plotted the space of free constraints of the STNU problem in the space (y_{A_e}, y_{B_e}, B_s) regions in figures 10.6a and 10.6b (without loss of generality, we assigned $A_s = 0$ as we can always freely assign a reference controllable time point thanks to the $\mathcal{RDL}$ property of shifting solutions). The plot clearly shows that there exists no linear strategy for B_s. Considering the vertex $\langle 0, 1\rangle$ in the space (y_{A_e}, y_{B_e}), a linear solution must contain point $\langle 0, 1, 0\rangle$ as it is the only feasible point for the vertex $\langle 0, 1\rangle$. Similarly, considering $\langle 0, 2\rangle$ we must include $\langle 0, 2, 0\rangle$; considering $\langle 3, 1\rangle$ we must include $\langle 3, 1, 1\rangle$ and for $\langle 3, 2\rangle$ the linear solution must include the point $\langle 3, 2, 0\rangle$. However, no linear solution can exist because no plane contains all the four points. In fact, the only plane containing $\langle 0, 1, 0\rangle$, $\langle 0, 2, 0\rangle$ and $\langle 3, 2, 0\rangle$ is $B_s = 0$, but this plane does not contain the point $\langle 3, 1, 1\rangle$.

We can also exploit the encodings for the decision problem to show that the STNU in figure 10.5 is weakly controllable. The inverted SMT encoding of equation (10.2) for the example problem is as follows.

$$\begin{aligned}\neg\exists A_s, B_s.((y_{A_e} \geq 0) \wedge (y_{A_e} \leq 3) \wedge (y_{B_e} \geq 1) \wedge (y_{B_e} \leq 2)) \rightarrow \\ ((B_s - A_s \geq 0) \wedge ((A_s + y_{A_e}) - B_s \geq 0) \wedge \\ ((A_s + y_{A_e}) - (B_s + y_{B_e}) \leq 1)) \wedge \\ ((B_s + y_{B_e}) - A_s \leq 2)\end{aligned} \tag{10.8}$$

This formula can be shown to be unsatisfiable by any $\mathcal{LRA}$ SMT solver. Therefore the problem is indeed weakly controllable. The unsatisfiability of the formula is also shown by equations (10.6) and (10.7) that provide a witness strategy for the existential quantifier, making the formula false. In fact, if $(y_{B_e} > y_{A_e} - 1)$ holds, equation (10.8) is unsatisfiable because $A_s = 0$ and $B_s = 0$ is a model of equation (10.6). Similarly, if $(y_{B_e} \leq y_{A_e} - 1)$ holds, equation (10.8) is unsatisfiable because $A_s = 0$ and $B_s = y_{A_e} - y_{B_e} - 1$ is a model of equation (10.7).

Piecewise-linear strategy is enough. We now prove that a piecewise-linear strategy always exists for any weakly controllable TNU.

Theorem 10.3 (Piecewise Strategy Existence)**.** *For any given TNU P, if P is weakly controllable, then P admits a piecewise-linear strategy.*

Proof. Let $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u)\rangle$ be the encoding of P. Since P is weakly controllable, we know that

$$\forall \vec{Y}_u . \exists \vec{T}_c . \Gamma(\vec{Y}_u) \rightarrow \Psi(\vec{T}_c, \vec{Y}_u)$$

is valid.

We want to prove that there exists a piecewise-linear strategy f such that

$$\forall \vec{Y}_u . (\Gamma(\vec{Y}_u) \wedge \vec{T}_c = f(\vec{Y}_u)) \rightarrow \Psi(\vec{T}_c, \vec{Y}_u)$$

is valid.

By construction, both $\Gamma(\vec{Y}_u)$ and $\Psi(\vec{T}_c, \vec{Y}_u)$ are formulae in $\mathcal{QF\_LRA}$ and hence they geometrically correspond to the union of finitely many closed convex polyhedra (the polyhedra are closed because all the inequalities are non-strict by problem definition).

We now show that from each face we can extract a linear strategy that correctly work for a sub-region of $\Gamma(\vec{Y}_u)$. By combining these linear strategies for all the faces of the polyhedron we obtain a weak strategy for P.

Without loss of generality we can assume $\Psi(\vec{T}_c, \vec{Y}_u)$ being a bounded set (meaning that it can be completely contained in a ball of finite radius). This is because we already have bounds for all the uncontrollable variables in $\Gamma(\vec{Y}_u)$ (because of the assumptions in definition 40) and we can always add upper and lower bounds on controllable variables as follows. Since the problem is weakly controllable, let $g(\vec{Y})$ be any weak strategy. For each variable $x \in \vec{T}_c$, let $u_x \doteq max(\{g(\vec{Y}_u) \mid \vec{Y}_u \models \Gamma(\vec{Y}_u)\})$ and $l_x \doteq min(\{g(\vec{Y}_u) \mid \vec{Y}_u \models \Gamma(\vec{Y}_u)\})$. We can then add the following constraint to the problem without altering its weak controllability: $x \in [l_x, u_x]$.

Let $\phi^1(\vec{T}_c, \vec{Y}_u), \cdots, \phi^w(\vec{T}_c, \vec{Y}_u)$ be the formulae corresponding to the faces of $\Psi(\vec{T}_c, \vec{Y}_u)$. Each face $\phi^z(\vec{T}_c, \vec{Y}_u)$ is a convex polyhedron and can be expressed as a system of inequalities $A(\vec{T}_c|\vec{Y}_u) \leq b$ with at least one inequality satisfied as an equality. From this system is easy to extract a linear strategy $f^z(\vec{Y}_u)$ by reducing the augmented matrix $(A|b)$ into reduced row echelon form and applying substitution to extract the relation between $\vec{T}_c$ and $\vec{Y}_u$ in closed form.

For each face $\phi^z(\vec{T}_c, \vec{Y}_u)$ we define its projection $\chi^z(\vec{Y}_u) \doteq \exists \vec{T}_c.\phi^z(\vec{T}_c, \vec{Y}_u)$. Since $\mathcal{QF\_LRA}$ admits quantifier elimination, also $\chi^z(\vec{Y}_u)$ can be expressed as a $\mathcal{QF\_LRA}$ formula, and geometrically corresponds to a finite union of convex polyhedra.

Therefore, we can build the piecewise-linear weak strategy f defined as follows.

$$f(\vec{Y}_u) \doteq \begin{cases} f^1(\vec{Y}_u) & \text{if } \chi^1(\vec{Y}_u) \\ f^2(\vec{Y}_u) & \text{else if } \chi^2(\vec{Y}_u) \\ \cdots & \\ f^k(\vec{Y}_u) & \text{else if } \chi^k(\vec{Y}_u) \end{cases}$$

Clearly, $\forall \vec{Y}_u.(\bigvee_{i=1}^{w} \chi^i(\vec{Y}_u)) \rightarrow \Gamma(\vec{Y}_u)$ because we know that P is weakly controllable and we assumed $\Psi(\vec{T}_c, \vec{Y}_u)$ to be bounded (being bounded, the projection of all the faces corresponds to the projection of the polyhedral union itself). In addition, each $f^z(\vec{Y}_u)$ applied to a point in $\chi^z(\vec{Y}_u)$ yields a point belonging to a face of the polyhedron, hence belonging to $\Psi(\vec{T}_c, \vec{Y}_u)$. Thus, the strategy is a valid weak strategy for P. $\square$

We are interested in generating strategies that can be efficiently executed once the situation is known. Given this requirement, linear strategies are very helpful, because they are compact (the size is quadratic in the number of time points) and can be executed by performing a linear computation in the size of the strategy. Piecewise-linear strategies are also helpful because they can be executed in linear time in the size of the strategy as they require only a case switch before applying the linear executor.

	Strategy Type	
	Linear	**Piecewise-Linear**
Convex (STNU)	VertexEncoding (section 10.4.1) IncrementalWeakening (section 10.4.1)	SimplexesDecomposition (section 10.4.2) LazyExpansion (section 10.4.2)
Disjunctive (DTNU)	NRA Encoding (section 10.4.3)	SkinCrawler (section 10.4.4) ConvexRegionEnumerator (section 10.4.4)

Table 10.1: Overview of the developed algorithms with references to the section that describes each of them.

10.4 Synthesis of strategies for Weak Controllability

The problem of synthesizing weak strategies can be classified along two dimensions; we distinguish between (i) convex (STNU) vs. disjunctive (DTNU) temporal problems and (ii) linear vs. piecewise-linear strategies. Table 10.1 summarizes this classification and indicates the algorithms we developed for each problem class.

All the algorithms assume that the given problem is weakly controllable, but it is not known in advance whether the problem admits a linear strategy. Thus, the algorithms listed in the "Linear" column of table 10.1 return $\perp$ in case no linear strategy exists. The others are guaranteed to find a piecewise-linear strategy. In the rest of this section we analyze each combination of temporal problem class and strategy type separately.

10.4.1 Linear Strategies for STNU

In the following, we discuss two algorithms that are able to synthesize linear strategies for a given STNU problem. They both leverage the convexity in the constraints of the STNU problem class.

Vertex Encoding

If the problem is an STNU, then the free constraints represent a convex space: given any two points in the space of free constraints, any point in the line connecting these two points is also a solution. Following this idea we can generalize the result of weak controllability on bounds in [VF99b] (see section 4.2.2 and theorem 4.1) to the search of linear strategies. We consider all the vertexes of the uncontrollable space $\Gamma(\vec{Y}_u)$ that, by definition of $\Gamma(\vec{Y}_u)$, are the elements of the set $V_\Gamma \doteq \{l^c_{1,1}, u^c_{1,1}\} \times \cdots \times \{l^c_{m,1}, u^c_{m,1}\}$. We then search for a hyperplane that satisfies the free constraints in all these vertexes. Such a hyperplane constitutes a linear strategy, because the solution space is convex. The following theorem formalizes this idea, and is proven in appendix A.3.

Theorem 10.4 (Vertex Encoding Correctness). *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be an STNU, $\langle \vec{T}_c, \vec{Y}_u, \Gamma(\vec{Y}_u), \Psi(\vec{T}_c, \vec{Y}_u) \rangle$ be its encoding and let $\bar{f} : \mathbb{R}^{|Y_u|} \rightarrow \mathbb{R}^{|X_c|}$ be a linear strategy. If $\bar{f}$ fulfills $\Psi(\vec{T}_c, \vec{Y}_u)$ in all the vertexes $v_i \in V_\Gamma$, then $\bar{f}$ is a weak linear strategy for P.*

Based on this insight, the idea is to create a single formula that encodes the problem with a symbolic strategy in all the vertexes of the uncontrollable region. The encoding is obtained instantiating the problem constraint in all the vertexes $v_i \in V_\Gamma$ and by enforcing a single hyperplane to contain all of them. If such a hyperplane exists, then it is a valid linear strategy for the entire problem.

Algorithm 7 shows the pseudo-code for extracting a linear strategy with such encoding. We create a matrix A and a vector $\vec{c}$ of real SMT variables representing the coefficients of the linear strategy. The function VERTEX-ASSIGNMENTS generates all the vertexes of the convex polyhedron corresponding to $\Gamma(\vec{Y}_u)$. In order to achieve this result if $\Gamma(\vec{Y}_u)$ is generated as in definition 44, it suffices to generate all the possible combinations of as-

Algorithm 7 Vertex Encoding

```
1: procedure VertexEncoding($\Gamma(\vec{Y}_u)$, $\Psi(\vec{T}_c, \vec{Y}_u)$)
2:     for all $b_i \in \vec{T}_c$ do
3:         SMT.declareRealVar($c_i$)
4:         for all $y_j \in \vec{Y}_u$ do
5:             SMT.declareRealVar($A_{i,j}$)
6:         end for
7:     end for
8:     $\phi(A, \vec{c}) \coloneqq \top$
9:     for all $\bar{p} \in$ VertexAssignments($\Gamma(\vec{Y}_u)$) do
10:        $\phi(A, \vec{c}) \coloneqq \phi(A, \vec{c}) \wedge \Psi(A \cdot \bar{p} + \vec{c}, \bar{p})$
11:    end for
12:    if SMT.solve($\phi(A, \vec{c})$) = SAT then
13:        $(A, \vec{c}) \coloneqq$ SMT.getModel()
14:        return $f(\vec{Y}_u) \doteq A \cdot \vec{Y}_u + \vec{c}$
15:    else
16:        return $\perp$
17:    end if
18: end procedure
```

signments of contingent links bounds. We remark that each $\bar{p}$ is a vector of constants, and therefore the only variables occurring in the formula $\phi(A, \vec{c})$ are the coefficients of the linear strategy $f(\vec{Y}_u) \doteq A \cdot \vec{Y}_u + \vec{c}$. The function SMT.solve checks the satisfiability of the given formula using an SMT solver, while SMT.getModel returns the produced model in case of SAT answer.

We presented this algorithm for an encoded problem as formalized in definition 44, but the same idea can be applied when $\Gamma(\vec{Y}_u)$ and $\Psi(\vec{T}_c, \vec{Y}_u)$ are simply conjunctive $\mathcal{QF\_LRA}$ formulae (so that they represent convex polyhedra). The only modification needed for the algorithm is to change the VertexAssignments so that it is able to produce the vertexes of general formulae. For doing this we can employ well known techniques for enumerating the vertexes of a convex polyhedron [AF92].

Consider for example the STNU problem in figure 7.2. The resulting problem admits a linear strategy. The encoding obtained by the application of algorithm 7 is as follows.

$$
\begin{aligned}
&((A_{1,2} \cdot 0 + A_{2,2} \cdot 1 + c_2) - (A_{1,1} \cdot 0 + A_{2,1} \cdot 1 + c_1) \geq 0) \wedge \\
&(((A_{1,1} \cdot 0 + A_{2,1} \cdot 1 + c_1) + 0) - ((A_{1,2} \cdot 0 + A_{2,2} \cdot 1 + c_2) + 1) \leq 1) \wedge \\
&(((A_{1,2} \cdot 0 + A_{2,2} \cdot 1 + c_2) + 1) - (A_{1,1} \cdot 0 + A_{2,1} \cdot 1 + c_1) \leq 2) \\
\wedge \\
&((A_{1,2} \cdot 0 + A_{2,2} \cdot 2 + c_2) - (A_{1,1} \cdot 0 + A_{2,1} \cdot 2 + c_1) \geq 0) \wedge \\
&(((A_{1,1} \cdot 0 + A_{2,1} \cdot 2 + c_1) + 0) - ((A_{1,2} \cdot 0 + A_{2,2} \cdot 2 + c_2) + 2) \leq 1) \wedge \\
&(((A_{1,2} \cdot 0 + A_{2,2} \cdot 2 + c_2) + 2) - (A_{1,1} \cdot 0 + A_{2,1} \cdot 2 + c_1) \leq 2) \\
\wedge \\
&((A_{1,2} \cdot 3 + A_{2,2} \cdot 1 + c_2) - (A_{1,1} \cdot 3 + A_{2,1} \cdot 1 + c_1) \geq 0) \wedge \\
&(((A_{1,1} \cdot 3 + A_{2,1} \cdot 1 + c_1) + 3) - ((A_{1,2} \cdot 3 + A_{2,2} \cdot 1 + c_2) + 1) \leq 1) \wedge \\
&(((A_{1,2} \cdot 3 + A_{2,2} \cdot 1 + c_2) + 1) - (A_{1,1} \cdot 3 + A_{2,1} \cdot 1 + c_1) \leq 2) \\
\wedge \\
&((A_{1,2} \cdot 3 + A_{2,2} \cdot 2 + c_2) - (A_{1,1} \cdot 3 + A_{2,1} \cdot 2 + c_1) \geq 0) \wedge \\
&(((A_{1,1} \cdot 3 + A_{2,1} \cdot 2 + c_1) + 3) - ((A_{1,2} \cdot 3 + A_{2,2} \cdot 2 + c_2) + 2) \leq 1) \wedge \\
&(((A_{1,2} \cdot 3 + A_{2,2} \cdot 2 + c_2) + 2) - (A_{1,1} \cdot 3 + A_{2,1} \cdot 2 + c_1) \leq 2)
\end{aligned}
$$

The encoding is satisfiable and a possible model (encoding a linear strategy) is reported in equation (10.9).

$$
A = \begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix} \quad \vec{c} = \begin{pmatrix} 0 \\ 2 \end{pmatrix} \tag{10.9}
$$

Therefore, the assignments for the controllable time points A_s and B_s are $A_s \doteq 0$ and $B_s \doteq -y_{B_e} + 2$.

Note that, this approach leads to an exponential blowup in the size of the SMT problem, caused by the fact that the number of vertexes is $2^{|Y_u|}$.

Incremental Weakening

In order to limit the exponential blowup of the previous encoding to the worst case only, we developed another approach called "*incremental weakening*", that tries to limit the number of coefficients to search for and to reduce the amount of variables that are used in the linear strategy. This idea amounts to finding a matrix A in which some (possibly many) columns are null vectors; in fact, if the i-th column is null in A, the strategy does not depend on the actual values of y_i. In the limit case in which A is the null matrix, the strategy degenerates to an assignment of constant values to each controllable time points, and thus to a strong controllability solution.

We start by solving a relaxed problem, in which no uncontrollable duration is observed. This coincides with the definition of a strong controllability problem. If a solution is found, the strong assignment is a valid weak linear strategy for the problem, because strong controllability implies weak controllability. Otherwise, a subset of the uncontrollable durations $\vec{p} \subseteq \vec{Y}_u$ is picked and marked as "usable" by the strategy. The algorithm then tries to build a linear strategy that uses uncontrollable durations in $\vec{p}$ only. In this way, we are limiting the observations available to our strategy. Using the previous algorithm, we build the coefficients for the p-th column of the matrix A and we encode the problem as in the previous algorithm, limiting the exponential explosion only to the durations marked as "usable". If the algorithm fails to find a linear strategy for a particular set of "usable" durations, a different subset of the durations is picked and the approach is iterated, until all the uncontrollable durations are marked as "usable" and the encoding coincides with the previous approach.

The pseudo-code of this method is reported in algorithm 8. The func-

Algorithm 8 Incremental Weakening

```
1: procedure IncrementalWeakening($\Gamma(\vec{Y}_u)$, $\Psi(\vec{T}_c, \vec{Y}_u)$)
2:     repeat
3:         $O := \textsc{GetUsableDurations}(\vec{Y}_u)$
4:         $N := \{y \in \vec{Y}_u \mid y \notin O\}$
5:         $\eta(\vec{T}_c, \vec{O}) := \textsc{SC\_Encode}(\Gamma_{|N}(\vec{N}),\ \Psi_{|X_c \cup N}(\vec{T}_c, \vec{N}))$
6:         $f(\vec{O}) := \textsc{VertexEncoding}(\Gamma_{|\vec{O}}(\vec{O}),\ \eta(\vec{T}_c, \vec{O}))$
7:         if $f(\vec{O}) \neq \bot$ then
8:             return $\textsc{AddNullColumns}(f(\vec{O}))$
9:         end if
10:    until $O = Y_u$
11:    return $\bot$
12: end procedure
```

tion GetUsableDurations returns a heuristically computed subset of $\vec{Y}_u$ that constitutes the set of "observed" durations. The function is stateful as it is assumed to return a different subset at each call. The termination of the algorithm requires that this function eventually returns the entire $\vec{Y}_u$ that exits the repeat loop fulfilling the condition at line 9. In this termination condition, the algorithm behaves like the VertexEncoding procedure executed on the entire problem. The function SC_Encode produces the encoding of a strong controllability problem in SMT, using any of the encodings we discussed in chapter 9. This encoding is used to prevent the observation of non-used durations, leaving the others untouched. The function VertexEncoding is the function described in algorithm 7. If the VertexEncoding function returns a strategy that works for a subset of the uncontrollable duration variables, we return the same linear strategy completed by the function AddNullColumns. The function adds columns of 0s in the positions of the durations that were not used. This guarantees that the strategy is independent of the actual values of those durations.

This algorithm tries to abstract the problem by limiting the set of "usable" durations in a strategy, and refines the abstraction if no linear strategy is found. The process is iterated until a strategy is found or the entire set of durations is marked as "usable".

As shown in the previous section, if we consider the running example in figure 7.2, we can derive a strategy in which y_{A_e} is never observed. The advantage of IncrementalWeakening over the previous algorithm is that if we choose to use y_{B_e} but not y_{A_e} in our strategy we can get to the same result reported in equation (10.9) with a smaller and simpler encoding.

This algorithm depends on the heuristic used for selecting the "usable" durations. In fact, the number of cycles of the algorithm directly depends on the heuristic.

In our experiments, we implemented a heuristic based on a topological sorting of the uncontrollable time points. The heuristic first generates all the singleton subsets and, if the algorithm is not terminated, considers prefixes of the topological order of increasing size until all the durations are marked as "usable" and the algorithm terminates.

10.4.2 Piecewise-Linear Strategies for STNU

In the following, we present two algorithms for extracting a piecewise-linear strategy for a given weakly controllable STNU.

Simplexes Decomposition

A direct approach to extract a piecewise-linear strategy consists in partitioning the region of the uncontrollable durations in a set of m-simplexes (hyper-tetrahedra in m dimensions) with $m = |Y_u|$. In geometry, a k-simplex is the generalization of a triangle to k-dimensions. A k-simplex is a k-dimensional polytope which is the convex hull of $k+1$ linearly independent (i.e. not aligned) vertexes. For example, a 2-simplex is a triangle and a 3-simplex is a tetrahedron. We consider these polyhedra because they can be used to triangulate more complex regions [HA96]. The following theorem states the existence of a linear strategy in any simplex contained in the uncontrollable space, the proof is in appendix A.3.

Theorem 10.5 (Simplex Strategy Existence). *Let P be an encoded weakly controllable STNU $\langle \vec{T_c}, \vec{Y_u}, \Gamma(\vec{Y_u}), \Psi(\vec{T_c}, \vec{Y_u})\rangle$. For each $|Y_u|$-simplex $\sigma(\vec{Y_u})$ such that $\sigma(\vec{Y_u}) \subseteq \Gamma(\vec{Y})$ there exists a valid weak linear strategy f such that $\forall \vec{Y_u}.((\sigma(\vec{Y_u}) \wedge \vec{T_c} = f(\vec{Y_u})) \rightarrow \Psi(\vec{T_c}, \vec{Y_u}))$ is valid.*

In order to exploit theorem 10.5 we need to be able to split the uncontrollable space into simplexes. Doing so would allow us to split the problem of finding a piecewise-linear strategy for the whole problem in the problem of finding linear strategies for each simplex and then combine them.

The uncontrollable region $\Gamma(\vec{Y_u})$ is a hyper-rectangle, and the minimum number of simplexes needed to cover a hyper-rectangle is an open mathematical problem [HA96]. However, it is known that any hyper-rectangle in m dimensions can be split in a factorial number of simplexes ($m!$). For example, a rectangle can be split in 2 triangles, and a rectangular cuboid can be covered by 6 tetrahedrons.

Given $\Gamma(\vec{Y_u})$, we can obtain all the simplexes using the following idea. Suppose that the bounds for all the uncontrollable variables are $[0, 1]$. Then, let R be the region satisfying the sequence of inequalities $y_{p_1} \leq y_{p_2} \leq \cdots \leq y_{p_m}$ where $(p_1, \ldots, p_m)$ is a permutation of $(1, \ldots, m)$ and m is the number of uncontrollable duration variables. It can be shown that R is a simplex and that the simplexes generated for all the permutations form a partition of the uncontrollable space [HA96]. In the general case, when we can have arbitrary bounds, we apply the very same idea, permuting

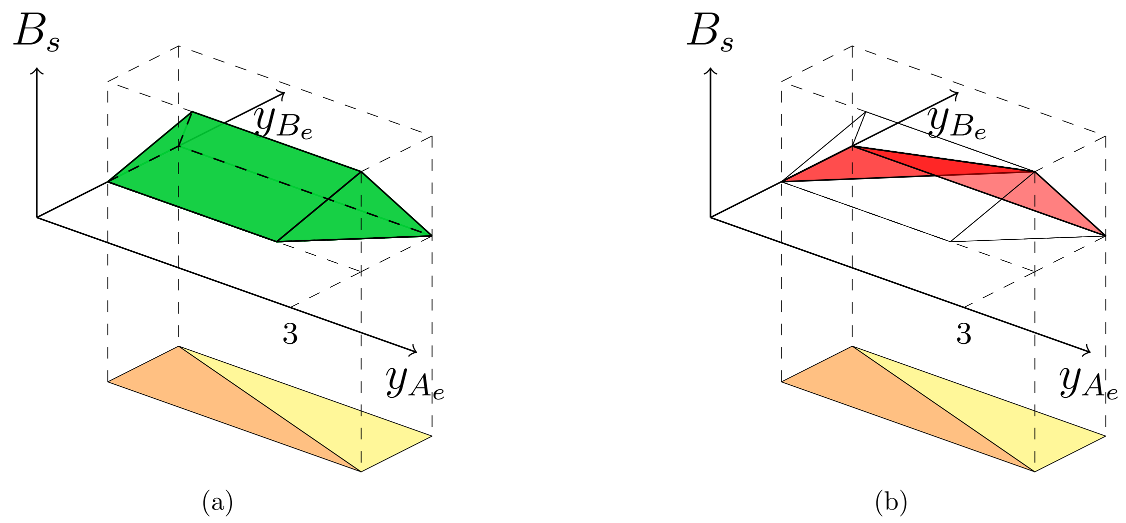


Figure 10.7: Plot of the running example problem (with A_s assigned to 0) with a partition of the space of uncontrollable durations. The space of uncontrollable durations is split in two triangles, depicted in yellow and orange. In (a) we plot the space of the solutions, while in (b) we draw in red a possible piecewise-linear strategy obtained by using a linear strategy for each triangle.

the variables and considering the inequalities arising from considering the concrete lower/upper bounds.

Using this approach, we have to enumerate all the permutations of the uncontrollable variables. Thus, the number of considered simplexes is factorial in the number of uncontrollable variables (i.e. $|Y_u|!$).

For each simplex it is possible to find a linear strategy separately, by enforcing a hyperplane to satisfy the problem constraints in all the simplex vertexes. In figure 10.7 we depicted an example of this idea for the running example problem.

Algorithm 9 shows the pseudo-code for extracting a piecewise linear strategy by enumerating all the simplexes and finding a linear strategy for every simplex. The computational complexity of this algorithm is factorial due to the enumeration of all the ($|Y_u|!$) simplexes.

In the pseudo-code, the function GetMaximalSimplexes enumerates

Algorithm 9 Simplexes Decomposition strategy extraction algorithm

```
1: procedure SimplexesDecomposition($\Gamma(\vec{Y_u})$, $\Psi(\vec{T_u}, \vec{Y_u})$)
2:     $f :=$ GetEmptyStrategy()
3:     for all $\sigma(\vec{Y_u}) \in$ GetMaximalSimplexes($\Gamma(\vec{Y_u})$) do
4:         $f_{sub} :=$ VertexEncoding($\sigma(\vec{Y_u})$, $\Psi(\vec{T_u}, \vec{Y_u})$)
5:         $f :=$ AddPieceToStrategy($f$, $(\sigma(\vec{Y_u}), f_{sub})$)
6:     end for
7:     return $f$
8: end procedure
```

a sequence of $|Y_u|$-simplexes needed to cover the $\Gamma(\vec{Y_u})$ polyhedron, while VertexEncoding returns a linear strategy suitable for the given simplex[3]. We obtain the resulting piecewise-linear strategy f by adding a piece for each simplex by means of the function AddPieceToStrategy.

Consider the problem in figure 10.5; the algorithm works as follows. We first consider the simplex with vertexes $\{\langle 0, 1\rangle, \langle 0, 2\rangle, \langle 3, 1\rangle\}$ in the space of y_{A_e} and y_{B_e}. A possible linear strategy in this simplex is $f_1 = A_1 \cdot \vec{Y_u} + \vec{c}_1$ as follows.

$$A_1 = \begin{pmatrix} 0 & 0 \\ \frac{1}{3} & 0 \end{pmatrix} \quad \vec{c}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Then, the algorithm considers the second maximal simplex having vertexes $\{\langle 0, 2\rangle, \langle 3, 1\rangle, \langle 3, 2\rangle\}$. A possible linear strategy in this simplex is $f_2 = A_2 \cdot \vec{Y_u} + \vec{c}_2$ as follows.

$$A_2 = \begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix} \quad \vec{c}_2 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

The algorithm combines such strategies in a valid piecewise-linear weak

[3]Formally, $\langle \vec{T_c}, \vec{Y_u}, \sigma(\vec{Y_u}), \Psi(\vec{T_u}, \vec{Y_u})\rangle$ is not a well formed encoding of an STNU, because $\sigma(\vec{Y_u})$ is not in the shape prescribed by definition 44. Nevertheless, the VertexEncoding algorithm can deal with any convex uncontrollable region.

strategy f as follows.

$$\begin{pmatrix} A_s \\ \\ B_s \end{pmatrix} = f(y_{A_e}, y_{B_e}) \doteq \begin{cases} f_1 & \text{if } (y_{B_e} \leq -\frac{1}{3} y_{A_e} + 2) \\ f_2 & \text{otherwise} \end{cases}$$

Lazy Expansion

To overcome the complexity limitation of the previous approach we developed a second technique, called *Lazy Expansion*, that first selects a simplex in the uncontrollable region and finds a linear strategy in that simplex. Second, we symbolically compute the region of the uncontrollable durations that is satisfied by the computed strategy. In this way, we perform a "widening" of the portion of the uncontrollable space that can be satisfied using the computed linear strategy. This widened region is guaranteed to cover at least the simplex, but it might be larger. We then associate the computed strategy to the resulting region. Finally, we search a new simplex in the remaining part of the space of uncontrollable durations. The algorithm terminates when the space of uncontrollable durations is completely covered. The idea behind the approach is to generalize the strategy found for a particular simplex to cover a wider potion of the space of the uncontrollable durations. The algorithm lazily picks a simplex from the region of the uncontrollable durations to be covered and gets a strategy that is able to cover that particular simplex. We then generalize the applicability of the returned strategy and proceed until we completely cover the uncontrollable space. The main advantage of this algorithm with respect to the previous one is that it is not forced to enumerate all the possible simplexes, because the computed strategy once found is exploited in all the possible points of the space where it is applicable.

Algorithm 10 shows the pseudo-code for extracting a piecewise linear strategy exploiting lazy expansion. The function GetUncoveredSim-

Algorithm 10 Lazy piecewise-linear strategy extraction

```
1: procedure LazyExpansion($\Gamma(\vec{Y_u})$, $\Psi(\vec{T_c}, \vec{Y_u})$)
2:     $f$ :=GetEmptyStrategy()
3:     $\eta(\vec{Y_u}) := \Gamma(\vec{Y_u})$
4:     for all $y_j \in \vec{Y_u}$ do
5:         SMT.declareVar($y_i$, $\mathbb{R}$)
6:     end for
7:     while SMT.solve($\eta(\vec{Y_u})$) do
8:         $\sigma(\vec{Y_u})$ := GetUncoveredSimplex($\eta(\vec{Y_u})$)
9:         $f_{sub}$ := VertexEncoding($\sigma(\vec{Y_u})$, $\Psi(\vec{T_c}, \vec{Y_u})$)
10:        $o(\vec{Y_u}) := \Psi(f_{sub}(\vec{Y_u}), \vec{Y_u})$
11:        $f$ := AddPieceToStrategy($f$, $(\eta(\vec{Y_u}) \wedge o(\vec{Y_u}), f_{sub})$)
12:        $\eta(\vec{Y_u}) := \eta(\vec{Y_u}) \wedge \neg o(\vec{Y_u})$
13:    end while
14:    return $f$
15: end procedure
```

plex returns any simplex $\sigma(\vec{Y_u})$ completely contained in the uncontrollable region $\Gamma(\vec{Y_u})$. At each step we compute the widening of the simplex (called $o(\vec{Y_u})$ in algorithm 10), that is the region in which the computed linear strategy f_{sub} is applicable. In order to symbolically obtain this region, we substitute the $\mathcal{LRA}$ encoding of the strategy f_{sub} in the free constraints $\Psi(\vec{T_c}, \vec{Y_u})$. In this way, each variable $b_i \in \vec{T_c}$ corresponding to a controllable time point is replaced by the linear term that computes it according to f_{sub}, and we are left with a formula defined over $\vec{Y_u}$. Each model of such formula, is a point in the uncontrollable region for which the application of f_{sub} fulfills the free constraints of the problem. We use this procedure to create "bigger" pieces and reduce the number of iterations of the algorithm.

In general, this algorithm is not guaranteed to terminate. In fact, termination can be assured with the following two requirements. First, each region $\sigma(\vec{Y_u})$ covers a non-empty volume of the space of the uncontrollable durations. This means that the piece of strategy computed at each step must be guaranteed to cover at least the simplex that originated it (at each step $\sigma(\vec{Y_u}) \models_{\mathcal{LRA}} o(\vec{Y_u})$). The second requirement is to have progression, that is we disallow infinite decomposition chains for finite regions. If we avoid empty regions and infinite subdivisions of finite regions, we will eventually get to the empty region, and thus to unsatisfiability and termination.

In our implementation, we do not guarantee termination. However, the algorithm correctly terminated in all our experiments and in many cases it performed much faster than the SimplexesDecomposition algorithm. One possibility to guarantee the termination would be to hybridize this algorithm with the SimplexesDecomposition approach by bounding the number of loops of the LazyExpansion procedure or to take a portfolio approach.

One key issue for the efficiency of this approach resides in finding good simplexes to cover a (possibly non-convex) region $\eta(\vec{Y_u})$. Defining what a "good" simplex is for the algorithm performance is a non-trivial task because the aim is to terminate with a minimum number of iterations, and thus to obtain a decomposition of $\Gamma(\vec{Y_u})$ in a minimal set of $o_i(\vec{Y_u})$ regions.

10.4.3 Linear Strategies for DTNU

We now consider the DTNU problem class and we provide algorithms for strategy synthesis starting from the linear case.

A way of computing the elements of matrix A and vector $\vec{c}$ for a linear strategy is an encoding of the constraints in the theory of Nonlinear Real Arithmetic over Polynomials ($\mathcal{NRA}$). In fact, we can compactly express the properties of each entry of A and $\vec{c}$ by imposing constraints on polynomials. Equation (10.10) is an encoding into $\mathcal{NRA}$ for extracting a linear strategy for any TNU problem in a single check.

$$\begin{aligned}
&\exists A_{1,1}, \ldots, A_{1,m}, c_1,\\
&\quad \cdots \\
&A_{n,1}, \ldots, A_{n,m}, c_n . \forall \vec{Y_u}. \Big(\Gamma(\vec{Y_u}) \rightarrow \Psi[A \cdot \vec{Y_u} + \vec{c}/\vec{X_c}](\vec{Y_u})\Big)
\end{aligned} \tag{10.10}$$

The idea is to let the solver search for the $A_{i,j}$ and c_i coefficients of the linear combination of $\vec{Y_u}$ that represent the set of hyper-planes that are strategies for each $b_i \in \vec{X_c}$. If the solver reports unsatisfiable, it means that no linear strategy exists for the given problem. This approach directly follows from the definition of linear strategy and is applicable to the entire spectrum of temporal problems with uncertainty because no assumption on the convexity of the search space is made.

As an example we show the encoding of equation (10.10) applied to the TNU in figure 10.5, to remark the fact that no linear strategy exists for this problem.

$$\begin{aligned}
&\exists A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}, c_1, c_2 . \forall y_{A_e}, y_{B_e}.\\
&((y_{A_e} \geq 0) \wedge (y_{A_e} \leq 3) \wedge (y_{B_e} \geq 1) \wedge (y_{B_e} \leq 2)) \rightarrow\\
&(((A_{2,1} y_{A_e} + A_{2,2} y_{B_e} + c_2) - (A_{1,1} y_{A_e} + A_{1,2} y_{B_e} + c_1) \geq 0) \wedge\\
&\;(((A_{1,1} y_{A_e} + A_{1,2} y_{B_e} + c_1) + y_{A_e}) - (A_{2,1} y_{A_e} + A_{2,2} y_{B_e} + c_2) \geq 0) \wedge\\
&\;(((A_{1,1} y_{A_e} + A_{1,2} y_{B_e} + c_1) + y_{A_e}) - ((A_{2,1} y_{A_e} + A_{2,2} y_{B_e} + c_2) + y_{B_e}) \leq 1) \wedge\\
&\;(((A_{2,1} y_{A_e} + A_{2,2} y_{B_e} + c_2) + y_{B_e}) - (A_{1,1} y_{A_e} + A_{1,2} y_{B_e} + c_1) \leq 2))
\end{aligned}$$

The running example problem is an STNU, but the approach presented here is more general. In fact, given a procedure that is able to decide $\mathcal{NRA}$ formulae with arbitrary disjunctions we can deal with DTNU as well. For example, the Cylindrical Algebraic Decomposition (CAD) procedure [CH91] can deal with this kind of formulae, even though the computational complexity is extremely high: the problem is in fact triply exponential.

Algorithm 11 Skin-Based Strategy Extraction for DTNU

```
1: procedure SkinCrawler($\Gamma(\vec{Y_u})$, $\Psi(\vec{X_c}, \vec{Y_u})$)
2:     $f$ := GetEmptyStrategy()
3:     for all $(o(\vec{Y_u}), f_{sub}) \in$ GetFaceStrategies($\Gamma(\vec{Y_u})$, $\Psi(\vec{X_c}, \vec{Y_u})$) do
4:         $f$ :=AddPieceToStrategy($f$, $(o(\vec{Y_u}), f_{sub})$)
5:     end for
6:     return $f$
7: end procedure
```

10.4.4 Piecewise-Linear Strategies for DTNU

In this section, we analyze the synthesis of a piecewise-linear strategy for a DTNU. When dealing with a DTNU, the convexity assumption holding for the STNU case is not valid anymore. We present two algorithms for the strategy synthesis in the DTNU problem class. The "skin crawler" method incrementally builds a strategy by considering the faces of the DTNU solution space considered as a polyhedron. The "convex region enumerator" approach, instead, decomposes the DTNU in a number of convex regions and applies the techniques for the STNU problem class on each of them.

Skin Crawler

An intuition that can be exploited to synthesize a weak strategy for the DTNU problem class is obtained from the proof of theorem 10.3. The idea is to iterate on the faces of $\Psi(\vec{X_c}, \vec{Y_u})$ and to project each of them in the space of $\vec{Y_u}$ until the entire $\Gamma(\vec{Y_u})$ region is covered by the projections. Such an iteration can be done efficiently by exploiting the optimization features of many modern SMT solvers.

The top-level procedure, shown in algorithm 11, iterates over the faces and extracts a linear strategy for each face, accumulating this result in a piecewise-linear strategy.

The face extraction procedure (algorithm 12) starts by extracting all the equalities from the free constraints. Since the free constraints are made of non-strict inequalities, we aim at extracting the skin of the free constraints by considering the equality $(a - b = k)$ derived from a constraint $(a - b \leq k)$. The algorithm uses an optimization procedure that maximizes the number of equalities satisfied at each step represented by the variable *satEqualities*. In this way, considering the conjunction of all the satisfied equalities, we first explore the vertexes, then the edges and finally the faces. The conjunction of equalities is actually a system of linear equalities representing a face. In order to extract a strategy from a face, we transform a conjunction of linear equalities into matrix form. As an optimization we discard systems that have dimension lower than the number of uncontrollable durations. This prevents the creation of pieces representing regions having null volume.

The algorithm termination depends only on the termination of the GetFaceStrategies procedure. The procedure is guaranteed to terminate because at each step we add a new clause to $\chi(\vec{X_c}, \vec{Y_u})$ that forces at least one equality that was positive in the found model to be false. Therefore we can have at most an exponential number of cycles with respect to the number of equalities in $Equalities(\Psi(\vec{X_c}, \vec{Y_u}))$.

Convex Region Enumerator

Finally, we can exploit the possibility of generating a strategy for the convex case by enumerating the convex regions in the space of free constraints.

This idea requires the possibility to deal with a possibly non-convex $\Gamma(\vec{Y_u})$ because the projection of a convex polyhedron space intersected with the non-convex $\Gamma(\vec{Y_u})$ can generate non-convex (and non-rectangular) regions. The LazyExpansion algorithm is able to deal with such constraints, because no requirement is imposed on the shape of $\Gamma(\vec{Y_u})$ during

Algorithm 12 Generates all the faces of $\Psi(\vec{X_c}, \vec{Y_u})$ and converts them in a linear system of equations

```
1: procedure GetFaceStrategies($\Gamma(\vec{Y_u})$, $\Psi(\vec{X_c}, \vec{Y_u})$)
2:     for all $x_i \in \vec{X_c} \cup \vec{Y_u}$ do
3:         SMT.declareVar($x_i$, $\mathbb{R}$)
4:     end for
5:     $\chi(\vec{X_c}, \vec{Y_u}) \coloneqq \Psi(\vec{X_c}, \vec{Y_u}) \wedge \Gamma(\vec{Y_u})$
6:     $satEqualities \coloneqq 0$
7:     for all $eq_i(\vec{X_c}, \vec{Y_u}) \in Equalities(\Psi(\vec{X_c}, \vec{Y_u}))$ do
8:         SMT.declareVar($eqv_i$, $\mathbb{R}$)
9:         $\chi(\vec{X_c}, \vec{Y_u}) \coloneqq \chi(\vec{X_c}, \vec{Y_u}) \wedge (eq_i(\vec{X_c}, \vec{Y_u}) \rightarrow (eqv_i = 1))$
10:        $\chi(\vec{X_c}, \vec{Y_u}) \coloneqq \chi(\vec{X_c}, \vec{Y_u}) \wedge (eq_i(\vec{X_c}, \vec{Y_u}) \vee (eqv_i = 0))$
11:        $satEqualities \coloneqq satEqualities + eqv_i$
12:    end for
13:    $faces \coloneqq \emptyset$
14:    while SMT.solveMaximizing($\chi(\vec{X_c}, \vec{Y_u})$, $satEqualities$) = SAT do
15:        $system \coloneqq \{eq_i(\vec{X_c}, \vec{Y_u}) \in Equalities(\Psi(\vec{X_c}, \vec{Y_u})) | \mu \models eq_i(\vec{X_c}, \vec{Y_u})\}$
16:        $(M\vec{t} = \vec{d}) \coloneqq$ ConvertToLinearSystem(system)
17:        $bases \coloneqq$ GetBases($M\vec{t} = 0$)
18:        if $|bases| \geq |\vec{Y_u}|$ then
19:            $(A, \vec{c}) \coloneqq$ ToLinearStrategy($M$)
20:            $o(\vec{Y}) \coloneqq \Psi(A \cdot \vec{Y_u} + \vec{c}, \vec{Y_u})$
21:            $faces \coloneqq faces \cup \{(o(\vec{Y}), A \cdot \vec{Y_u} + \vec{c})\}$
22:        end if
23:        $\chi(\vec{X_c}, \vec{Y_u}) \coloneqq \chi(\vec{X_c}, \vec{Y_u}) \wedge (\bigvee_{eq_i(\vec{X_c}, \vec{Y_u}) \in system} \neg eq_i(\vec{X_c}, \vec{Y_u}))$
24:    end while
25:    return $faces$
26: end procedure
```

the algorithm execution.

We need to enumerate a set of convex formulae $\{\mu_i(\vec{X}_c, \vec{Y}_u) | i \in [1, I]\}$ such that $(\bigvee_{i=1}^{I} \mu_i) \Leftrightarrow \Psi(\vec{X}_c, \vec{Y}_u)$. Such formulae can be obtained by computing the Disjunctive Normal Form (DNF) of the formula $\Psi(\vec{X}_c, \vec{Y}_u)$. From the practical point of view, each disjunct is either an atom of the original formula or its negation. The DNF can be efficiently computed in the SMT framework using an incremental mechanism.

Algorithm 13 shows the pseudo-code for the Convex Region Enumerator algorithm for the weak strategy synthesis of DTNU problems.

The algorithm works as follows. First, it selects any consistent temporal evolution by solving the SMT problem of the conjunction of the contingent and free constraints. Given the consistent model, the algorithm extracts the free constraints atoms that are satisfied and the atoms that are not fulfilled. The region obtained by conjoining all those atoms is a convex (non-necessarily closed) polyhedron. We compute the projection of such a polyhedron in the region of the uncontrollable durations and we compute a strategy for this quasi-STNU problem[4]. The obtained strategy is applied in the covered region that is removed from the problem together with the polyhedron. This ensures the algorithm termination, because at each step we remove a model of the Boolean abstraction from the $\rho(\vec{X}_c, \vec{Y}_u)$ formula.

In the pseudo-code, the function PROJECT performs a quantifier elimination in order to compute the projection of a given polyhedron onto the given space and is used to compute the uncontrollable region that is covered by the selected polyhedron.

Termination is not guaranteed, because we internally use the LAZY-EXPANSION algorithm that is incomplete; however, also in this case, the algorithm terminated in all the benchmarks.

[4]The problem is not a proper STNU because the projection can be non-rectangular and the constraints can contain strict inequalities. However the LAZYEXPANSION algorithm is able to deal even with such degenerated problems.

Algorithm 13 Convex Region Enumeration strategy extraction for DTNU

```
1: procedure ConvexRegionEnumerator($\Gamma(\vec{Y_u})$, $\Psi(\vec{X_c}, \vec{Y_u})$)
2:     for all $x_i \in \vec{X_c} \cup \vec{Y_u}$ do
3:         SMT.declareVar($x_i$, $\mathbb{R}$)
4:     end for
5:     $f :=$ GetEmptyStrategy()
6:     $\rho(\vec{X_c}, \vec{Y_u}) := \Psi(\vec{X_c}, \vec{Y_u}) \wedge \Gamma(\vec{Y_u})$
7:     while SMT.solve($\rho(\vec{X_c}, \vec{Y_u})$) = SAT do
8:         $\mu :=$ SMT.GetModel()
9:         $\delta(\vec{X_c}, \vec{Y_u}) := \top$
10:        for all $\alpha(\vec{X_c}, \vec{Y_u}) \in Atoms(\Psi(\vec{X_c}\vec{Y_u}))$ do
11:            if $\mu \models \alpha(\vec{X_c}, \vec{Y_u})$ then
12:                $\delta(\vec{X_c}, \vec{Y_u}) := \delta(\vec{X_c}, \vec{Y_u}) \wedge \alpha(\vec{X_c}, \vec{Y_u})$
13:            else
14:                $\delta(\vec{X_c}, \vec{Y_u}) := \delta(\vec{X_c}, \vec{Y_u}) \wedge \neg(\alpha(\vec{X_c}, \vec{Y_u}))$
15:            end if
16:        end for
17:        $o(\vec{Y_u}) :=$ Project($\delta(\vec{X_c}, \vec{Y_u})$, $\vec{Y_u}$)
18:        $f_{sub} :=$ LazyExpansion($o(\vec{Y_u})$, $\delta(\vec{X_c}, \vec{Y_u})$)
19:        $f :=$ AddPieceToStrategy($f$, ($o(\vec{Y_u}), f_{sub}$))
20:        $\rho(\vec{X_c}, \vec{Y_u}) := \rho(\vec{X_c}, \vec{Y_u}) \wedge \neg\delta(\vec{X_c}, \vec{Y_u}) \wedge \neg o(\vec{Y_u})$
21:    end while
22:    return $f$
23: end procedure
```

10.5 Experimental Evaluation

In order to empirically test the effectiveness of the proposed approaches, we implemented a tool for deciding weak controllability and synthesizing weak strategies for a TNU. Our tool is implemented in Python. It reads a TNU problem, and applies to it the portfolio of encodings and algorithms we presented in this chapter. The tool can synthesize explicit strategies as C++ functions (taking in input a situation), that can be compiled and linked in any program. We used the Z3 [dMB08] SMT solver for the weak controllability decision problem; we rely on the Python API provided by the MathSAT5 [CGSS13] SMT solver for all the strategy-synthesis techniques.

The randomly-generated benchmarks were obtained by modifying the generator of temporal problems presented in [ACG99] by introducing uncertainty in the problem: each constraint introduced by the consistency problem generator is turned into a contingent link with a given probability, and its destination node is considered as uncontrollable. We used random instance generators because they are typically used in the literature (e.g. [ACG99]), and because they can be easily scaled to stress the solvers.

We tested the decision problem encoding over a set of 2442 randomly generated DTNU, TCSNU and STNU instances, with a number of time points ranging from 6 to 20000. For the evaluation of strategy-extraction techniques, we used 1354 weakly controllable STNU benchmarks and 2112 weakly controllable DTNU instances ranging from 4 to 50 time points.

We tested the tool on a set of benchmarks described in detail below. We remark that, as far as our knowledge is concerned, there are no competitor tools or solvers able to deal with the weak controllability decision problem, nor with the synthesis of a weak strategy. Thus, in the experimental evaluation, we do not compare with any other tool or approach. All the experiments have been performed on a Scientific-Linux server equipped with two quad-core Xeon processors @ 2.70GHz. We used a memory limit of 2GB, a time-out of 300 seconds and we used sequential, single-core computation only. The tool, together with all the benchmarks and the results of the evaluation, can be obtained as indicated in section 1.2.

10.5.1 Decision Problem

The results of checking the decision problem over the set of TNUs are plotted in figure 10.8. The cactus plot (a) reports, in the horizontal axis, the number of solved instances and, on the vertical axis, the cumulative time, in logarithmic scale, taken by the SMT solver for each encoding. For example, the ASSUMPTION-EXTRACTION encoding takes about 10000 seconds to solve the easiest 750 instances. We compared the formulation of proposition 10.1 called DIRECT with the INVERTED and ASSUMPTION-EXTRACTION encodings.

The figure highlights the fact that Z3 performs much better when the ASSUMPTION-EXTRACTION encoding of the problem is considered: in fact, this approach is able to solve, in less time, a higher number of instances with respect to the INVERTED and DIRECT encodings. The DIRECT encoding performs almost identically to the INVERTED one. This behavior is due to the fact that the INVERTED encoding has the same shape as the DIRECT one. The only difference is the negation of the DIRECT encoding, that does not seem to affect the solver performance.

In figures 10.8b and 10.8c, we reported the scatter plots comparing the performances of the ASSUMPTION-EXTRACTION with the INVERTED encodings, distinguishing between weakly controllable and non weakly controllable instances. We note that, in the weakly controllable case, the ASSUMPTION-EXTRACTION encoding outperforms the INVERTED encoding in most of the benchmarks. For non weakly controllable instances, the two encodings perform similarly in terms of speed. However, the INVERTED encoding is able to solve 86 instances that are unsolvable by the ASSUMPTION-EXTRACTION encoding due to the imposed memory limit.

10.5.2 STNU Strategy Synthesis

The results for the evaluation of the strategy-extraction techniques for the 1354 STNU benchmarks are reported in figure 10.9a. The plot considers only those benchmarks that admit a linear strategy, and compares the four different approaches. The plot clearly shows that for linear strategies, the INCREMENTALWEAKENING approach outperforms all the others. The SIMPLEXESDECOMPOSITION method quickly explodes due to the factorial complexity of simplexes enumeration. Although the techniques for piecewise-linear strategy extraction are penalized as they are strictly more general than the others, the plot shows that LAZYEXPANSION approach is much faster than the SIMPLEXESDECOMPOSITION. In figure 10.9b we plotted the number of "pieces" of the strategies for the LAZYEXPANSION and SIMPLEXESDECOMPOSITION methods. The plot shows that, although for small problems the LAZYEXPANSION approach generates additional, unneeded "pieces", when the problem size increases the number of "pieces" identified by the LAZYEXPANSION method is much smaller than for the SIMPLEXESDECOMPOSITION one. In general, the LAZYEXPANSION approach has a huge gain in performance and in strategy size.

10.5.3 DTNU Strategy Synthesis

In figure 10.10 we report the results on the DTNU problem class. The plots show that the CONVEXREGIONENUMERATOR algorithm performs better than the SKINCRAWLER one. This is because of two main reasons.

First, the SkinCrawler approach solves a costly minimization problem and has to traverse all the faces of the space of free constraints while the ConvexRegionEnumerator algorithm applies the cheap LazyExpansion approach to each convex region that is generated by a single call to the SMT solver. Second, the linear strategy generated by the LazyExpansion approach is generalized and applied wherever possible, therefore if the problem allows for a linear strategy the ConvexRegionEnumerator algorithm is able to quickly synthesize it, while the SkinCrawler has to enumerate enough faces to cover the entire uncontrollable space.

There is an interesting peak on the rightmost part of the ConvexRegionEnumerator curve in the plot. This is due to a particular instance that is solved in 287.28 seconds generating a strategy with 3750 pieces (thus using the same number of iterations to terminate). This is one example in which the splitting done by the LazyExpansion approach gets lost in splitting the uncontrollable space in simplexes.

Finally, we did not experimented the effectiveness of the $\mathcal{NRA}$ encoding for two reasons. First, since a linear strategy is not guaranteed to exist for STNUs, it is also not guaranteed to exist in DTNUs. Second, the $\mathcal{NRA}$ approach needs a solver supporting the quantification over the real polynomial arithmetic (the full polynomial $\mathcal{NRA}$ theory), and to the best of our knowledge, no SMT solver fully supports this theory due to its complexity, even if the problem is decidable [CH91].

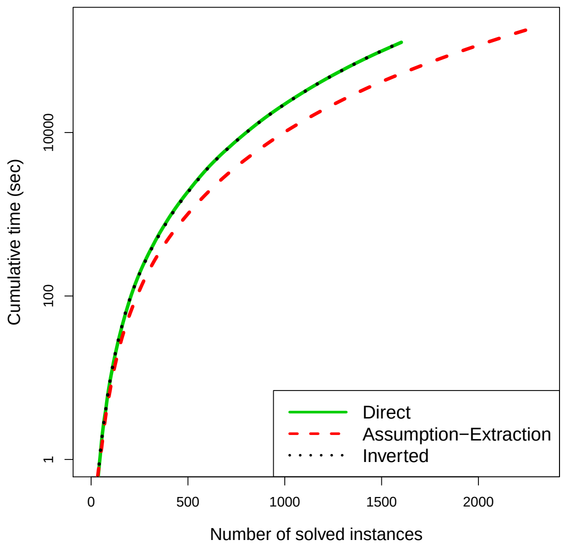
10.5.4 Strategy Execution

We proposed a number of approaches to synthesize weak strategies arguing that their execution is practically more efficient than solving the individual problems without uncertainty obtained by projecting the uncertainty away. In this section we provide experimental evidence supporting this claim on a number of STNU and DTNU instances.

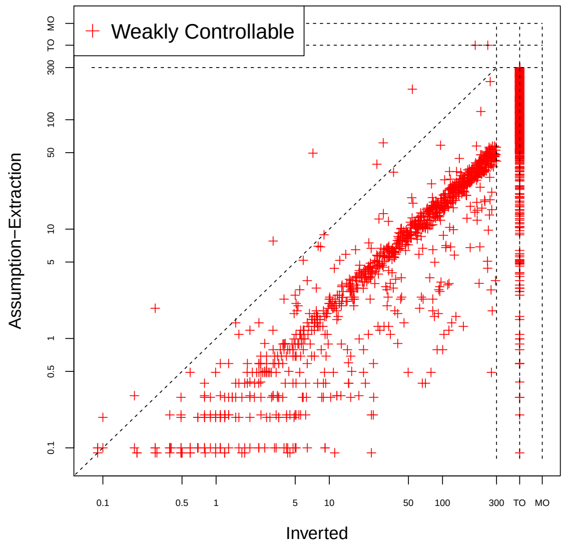
We conducted the experiment as follows. For each TNU in our benchmark set, we randomly generated 1000 situations, represented as complete assignments for the uncontrollable durations. We implemented the Implicit-SMT and Implicit-SMT-Incremental general strategies described in section 10.3.1 using the MathSAT5 SMT solver. Other TN solvers can, in principle, be employed to create implicit strategies, but in chapter 8 we show that SMT solvers are very effective in dealing with consistency problems. For this reason, and for the lack of publicly available implemented solvers, we limited our experimentation to the SMT based techniques described in section 10.3.1.

In addition, we considered three ways to compile in machine code the problem-specific strategies generated by our algorithms. We translated the linear or piecewise-linear strategy synthesized by any of our algorithms into C++ code. The translation for linear strategies is straight-forward: we create a function that takes in input a numeric value for each uncontrollable duration and we compute the output of the strategy by solving the matrix multiplication as described in section 10.3.2. Given a piecewise-linear strategy, we translate it using a sequence of `if` statements, one for each piece. The condition of each `if` is the transposition in C++ syntax of the piece condition. Each conditional statement returns the value computed by the translation of the linear strategy relative to the particular piece. We used three different datatypes to represent numeric values and perform the arithmetic operations. In particular, we used the (finite-precision) C++ `float` and `double` and the GNU-MP library for arbitrary precision arithmetic [Gt12]. The `float` and `double` datatypes are a finite-precision representation of rational numbers. As such, they suffer from both numeric stability and rounding problems that may, in principle, cause unsoundness in the strategy output. On the other hand, GNU-MP is the same library employed by the MathSAT5 SMT solver and does not suffer from any kind of numeric stability or rounding problems.

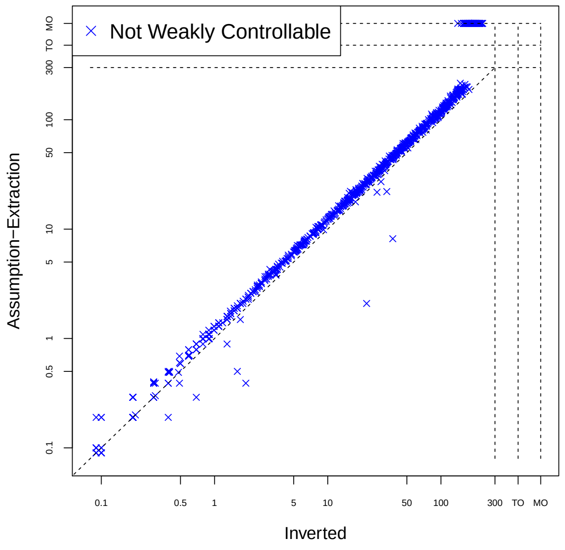
Figure 10.11 shows the results of the comparison: in our experiments the explicit strategies outperform projection-based implicit strategies. Implicit-SMT-Incremental performs better than Implicit-SMT, thanks to the incrementality feature of the SMT solver, but the explicit strategies bring a significant speedup on all the instances. Arbitrary-precision arithmetic (that fairly compares with the SMT precision) outperforms projection-based techniques by two orders of magnitude. The compiled strategies with native C++ datatypes perform even better, but the numerical stability problems can, in principle, lead to unsound results. We checked the output of each technique on each situation in order to assess the soundness, but all the results were correct. Nevertheless, studying under which condition we can guarantee that such finite-precision implementations are correct is subject of future work. We highlight that the compilations using native C++ datatypes can be translated to Boolean circuits, and this opens for the possibility of creating very efficient hardware implementations of these strategies.



(a)

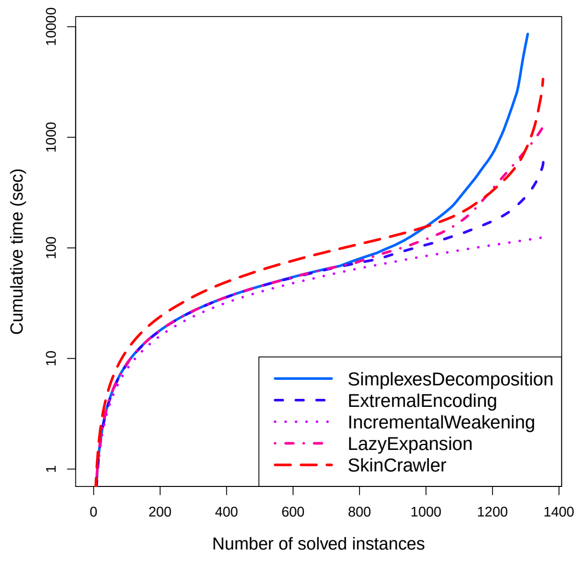


(b)

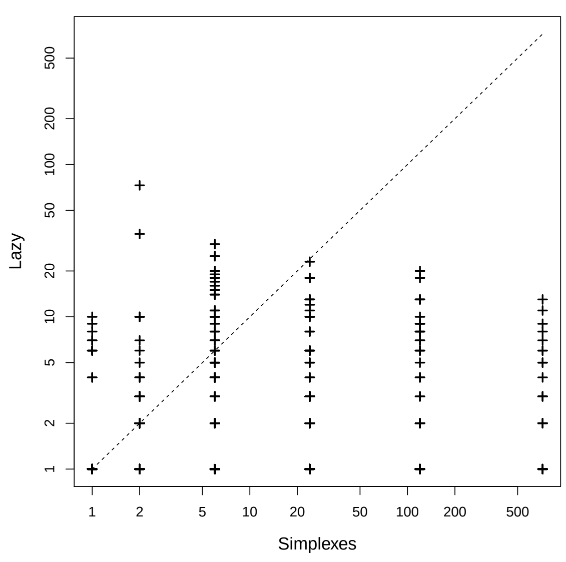


(c)

Figure 10.8: Results for the decision problem encodings solved using the Z3 SMT solver. Figure (a) reports the cumulative time (in logarithmic scale) cactus plot; Figures (b) and (c) show the scatter plots of INVERTED vs. ASSUMPTION-EXTRACTION encodings divided in weakly controllable, and not weakly controllable, respectively. The TO line denotes the instances that reached the time out, while MO indicates instances that hit the memory limit.



(a)



(b)

Figure 10.9: Results for STNU linear strategy extraction problem. In (a), we plotted the cumulative cactus plot of the strategy extraction time for the different algorithms we propose, while in (b), we compared the number of pieces for piecewise-linear algorithms expressed as the number of split regions.

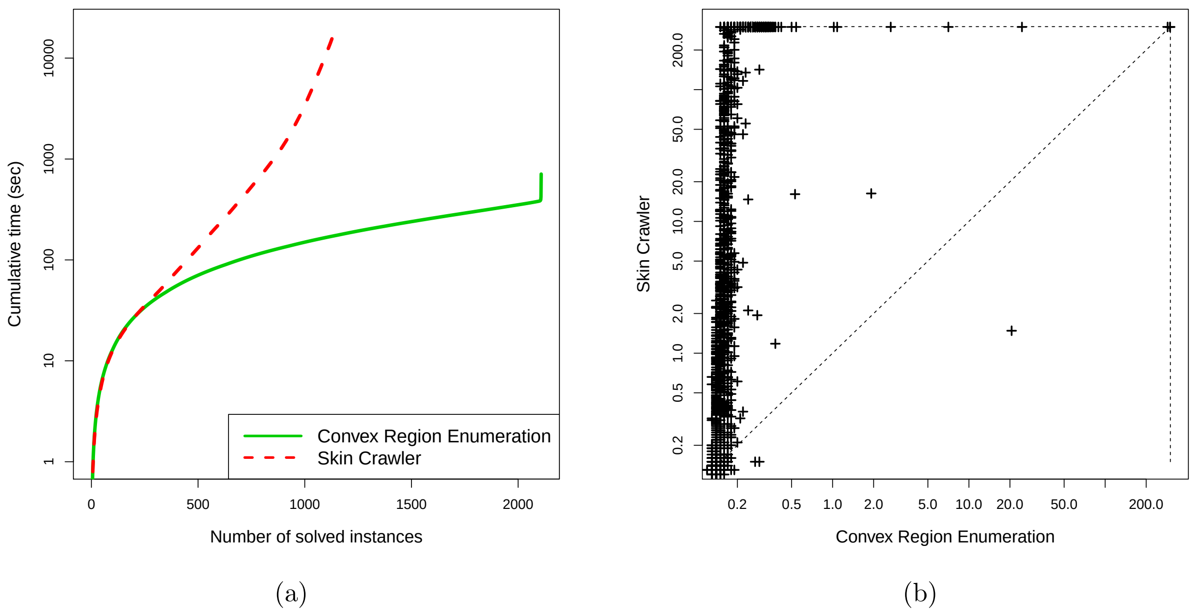


Figure 10.10: Results for strategy extraction problem in the DTNU problem class. In (a) we plotted the cumulative cactus plot of the solving time for the ConvexRegionEnumerator and the SkinCrawler algorithms while (b) is a scatter plot of the data.

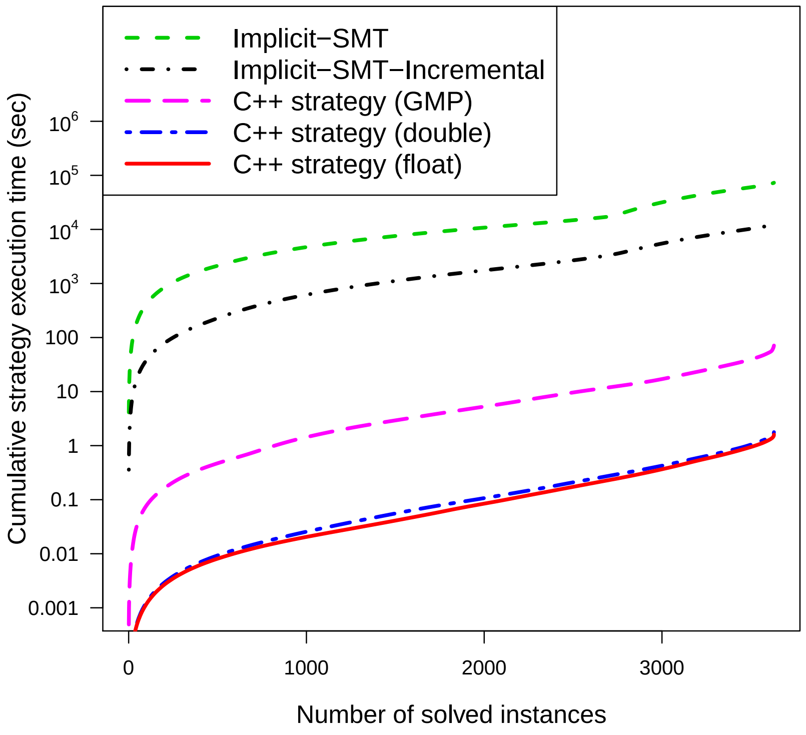


Figure 10.11: Results for strategy execution: for each problem, the generated strategy is executed on 1000 randomly generated situations. the plot considers all the STNU and DTNU randomly generated problems. The cactus plot shows the number of solved instances on the x axis and the accumulated time to solve them in the y axis.

Chapter 11

Dynamic Controllability

The last kind of query that can be addressed given a TNU is dynamic controllability. Dynamic controllability is concerned with the existence of a strategy for executing the controllable time points that depends only on past observations of the outcomes of uncontrollable durations, and that guarantees that all relevant constraints will be satisfied no matter how the durations of the contingent links turn out. Essentially, a TNU is dynamically controllable if there exists a weak strategy that, in order to decide a controllable time point at time k, does not depend on any observation past time k.

Polynomial algorithms to check the dynamic controllability of STNUs [MM05, Mor06, Hun14, Mor14] and run-time algorithms for generating an execution strategy in real-time [Hun10b, Hun13] have been presented in the literature (we discussed the state-of-the-art in dynamic controllability in section 4.2.2). Although STNUs have been successful in some domains, many other domains require a richer set of constraints and features. Disjunctive constraints often arise in practice, for example, when two activities cannot be done simultaneously, but a dynamic controllability checking algorithm has only been presented for a subclass of DTNUs [VVPYS10].

In this chapter, we make following contributions.

First, we formally define the semantics of dynamic controllability for the DTNU problem class, by generalizing the STNU semantics and we propose an algorithm to validate strategies for being valid dynamic strategies for a given DTNU.

Second, we present a novel approach for checking the dynamic controllability of DTNUs by translating the dynamic controllability problem into a reachability game on a Timed Game Automaton (TGA) [MPS95]. The reachability game can be solved using off-the-shelf software that is able to synthesize a viable execution strategy or determine that no such strategy exists [BCD$^+$07]. This results in the first sound-and-complete checking algorithm for the dynamic controllability of DTNUs. The encoding of such networks into TGA highlights important theoretical relationships between the different kinds of temporal reasoning frameworks and the TGA framework.

Third, we exploit the ideas behind the TGA encoding to develop a dedicated solving algorithm for the DTNU problem class. The algorithm is able to synthesize a strategy in form of an executable program.

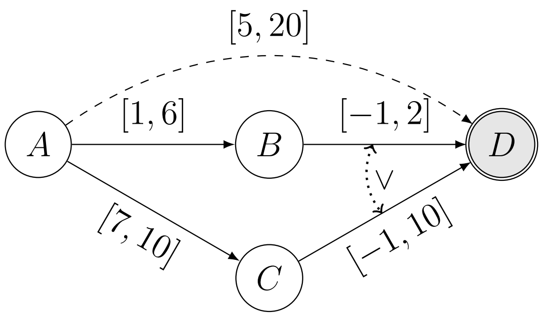
Finally, we present a comprehensive experimental evaluation of the proposed approaches.

11.1 Formalization of Dynamic Controllability

We now focus on the formal concept of dynamic controllability, which is widely viewed as the most relevant controllability level for real-world applications. Intuitively, a network is dynamically controllable if it admits a dynamic execution strategy that can react to contingent durations, but only those that have occurred in the past. In other words, the values that the execution strategy assigns to the controllable time points may depend on uncontrollable events (the execution of uncontrollable time points or the outcomes of observation nodes), but only if that information has already been observed in real time. Differently from weak strategies we presented in section 10.3, it cannot depend on any knowledge of future uncontrollables. Typically, the execution strategy must be able to deal with the branching that derives from alternative propositional outcomes, and may interleave the start times of activities with the observation of uncontrollable time points and propositional labels. In this sense, while a weak strategy is a function from the uncontrollables to the controllables, a dynamic strategy is more similar to a real-time program that gets executed and decides when a controllable time point should be scheduled. Following this analogy, the program inputs are given in real time and are the uncontrollable time point observations.

In the literature, the execution semantics for STNUs is expressed in terms of Dynamic Execution Strategies [MMV01]. For an STNU, $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, the agent seeks a strategy for executing the controllable time points in $\mathcal{T}_c \subseteq \mathcal{T}$ such that all constraints in $\mathcal{C}$ will necessarily be satisfied no matter what durations the environment "chooses" for the contingent links in $\mathcal{L}$, within their specified bounds. The decisions that constitute such a strategy can depend only on execution events that occurred in the past; however, the strategy can be dynamic in that it may react to observations of uncontrollable time points executing.

An agent's execution strategy can be compactly defined in terms of Real-Time Execution Decisions (RTEDs), where each RTED has one of two forms: `wait` or $\langle t, \chi_f \rangle$ [Hun09]. A `wait` decision can be glossed as "wait until some uncontrollable time point happens to execute." A $\langle t, \chi_f \rangle$ decision can be glossed as "if nothing happens before time t (i.e., if no uncontrollable time point happens to execute before time t), then I shall execute the (controllable) time points in the set χ_f at time t." The outcomes for an RTED specify the range of execution events that could happen



(a)

$$
\begin{aligned}
\mathcal{T}_c &\doteq \{A, B, C\}\\
\mathcal{T}_u &\doteq \{D\}\\
\mathcal{C} &\doteq \{B - A \in [1, 6], C - A \in [7, 10],\\
&\quad\; D - B \in [-1, 2] \vee D - C \in [-1, 10]\}\\
\mathcal{L} &\doteq \{\langle A, \{\langle 5, 20\rangle\}\rangle, D\}
\end{aligned}
$$

(b)

Figure 11.1: A dynamically controllable DTNU (b) and its graphical representation (a).

next. For example, a uncontrollable time point might happen to execute sometime before time t, in which case, the agent could react by adopting a new decision; or a uncontrollable time point e might happen to execute precisely at time t, in which case the time points in χ_f would be executed simultaneously with e at time t.

In the case of the DTNU in figure 11.1, the agent seeks a strategy for executing the controllable time points, A, B and C, that will guarantee that the constraints are satisfied, no matter what durations the environment happens to pick for the contingent link, $\langle A, \{\langle 5, 20\rangle\}, D\rangle$. For example, the agent might decide to execute A at time 0 and then wait. Should the environment happen to "choose" a duration of 5 for the contingent link, the agent would observe, at time 5, the execution of D. For example, the agent might then react executing B at time 6. Later, the agent will have to schedule C between time 7 and 10. In this example evolution, all the time points have been executed and all constraints in $\mathcal{C}$ are satisfied, hence the agent has succeeded. It can be checked that this DTNU is dynamically controllable (i.e., there exists a strategy for the agent that ensures success no matter how the environment behaves).

Although the execution semantics described above makes reference to the agent and the environment, execution strategies are only defined for the agent; the strategies available to the environment are only implicitly described by the sets of possible outcomes of the agent's decisions. Thus, the semantics is effectively a description of a one-player game where the outcomes of the agent's decisions are non-deterministic.

In this section we adapt the dynamic controllability semantics proposed by Hunsberger [Hun09] for the STNU problem class to the DTNU case. The semantics is expressed in terms of Real-Time Execution Decisions (RTEDs). This STNU semantics is equivalent to the one used in [MMV01], but slightly differs on the one of Morris [Mor06]. We will highlight the differences between the two semantics also for the DTNU class at the end of the section.

We first define a partial schedule as the current state of the system according to the perspective of the agent; namely, the time points that have been executed so far.

Definition 48 (Partial Schedule)**.** *A partial schedule for a DTNU, $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, is a set, PS, of assignments to time points in $\mathcal{T}$.*

- *$TPs(PS) \subset \mathcal{T}$ is the set of the time points appearing in PS;*
- *$Vals(PS) \subset \mathbb{R}$ is the set of values appearing in PS;*
- *for any $X \in TPs(PS)$, $PS(X)$ denotes the value assigned to X; and*
- *$\mathtt{now}_{PS} = \max\{v \mid v \in Vals(PS)\}$ is the time of the latest execution event in PS (if $PS = \emptyset$, we assume $\mathtt{now}_{PS} = -\infty$).*

time points in $TPs(PS)$ are said to be executed. A partial schedule is called respectful if its assignments do not violate the bounds on any contingent link.

Intuitively, a partial schedule PS assigns a real value to a subset of the time points in the network, and represents an execution history: the time points in PS are the ones that have already been executed, and the assigned values are the times at which they were executed. The time points that are not in PS are the ones that have not yet been executed.

Given a partial schedule PS, the solver must decide what to do next. Two options are possible:

1. wait for something to happen (i.e., wait for some uncontrollable time point to execute);

2. conditionally commit to executing a set of controllable time points at some time, $T_f > \texttt{now}_{PS}$.

For example, given $PS = \{\langle A, 0\rangle\}$, for which $\texttt{now}_{PS} = 0$, the solver could decide to wait until the uncontrollable time point D eventually executes. Alternatively, the executor could decide that "if nothing happens before time 6, I shall execute B at time 6" The decisions available to the solver are called Real-Time Execution Decisions (RTEDs).

Definition 49 (RTED)**.** *Let PS be a respectful partial schedule. An RTED has one of two forms:* `wait` *or $\langle T_f, \chi_f\rangle$.*

A `wait` *decision is applicable if at least one uncontrollable time point, e_i, is active in PS (i.e., e_i's activation time point $\alpha(e_i)$ has already been executed, but e_i has not).*

A $\langle T_f, \chi_f\rangle$ decision (i.e., "If nothing happens before time T_f, execute the time points in χ_f at time T_f") is applicable if $T_f > \texttt{now}_{PS}$ and χ_f is a non-empty subset of unexecuted controllable time points (i.e., $\chi_f \neq \emptyset$ and $\chi_f \cap TPs(PS) = \emptyset$).

Given a partial schedule PS and some RTED Δ, the outcome of the decision Δ typically depends on the range of possible durations for one or more contingent links, that is on the contingent situation (definition 41), as follows.

Definition 50 (Respected Situations). *A situation ω for a TNU is respected by a partial schedule PS if the durations specified in ω are consistent with not only the execution times in PS, but also the constraint that all time points that are unexecuted in PS must occur after* $\texttt{now}_{PS}$.

Note that if PS is a partial schedule that respects a situation ω, and an uncontrollable time point $e_i \notin \mathit{TPs}(PS)$ but its activation time point $\alpha(e_i) \in \mathit{TPs}(PS)$ (i.e., e_i is active in PS), then it follows that $\texttt{now}_{PS} < PS(A_i) + \omega_i$, since e_i must be executed after $\texttt{now}_{PS}$.

Definition 51 (Outcome of a `wait` decision). *Let PS be a partial schedule for which at least one uncontrollable time point is active, and let ω be a situation that is respected by PS. The outcome of the* `wait` *decision depends on:*

1. *$tnc(PS, \omega)$, the time of the next contingent execution according to PS and ω:*

   $$tnc(PS, \omega) \doteq \min\{PS(\alpha e_i) + \omega_{e_i} \mid \alpha(e_i) \in \mathit{TPs}(PS), e_i \notin \mathit{TPs}(PS)\}$$

2. *$\chi^*(PS, \omega)$, the set of uncontrollable time points that will execute next (i.e., at the time $tnc(PS, \omega)$).*

   $\chi^*(PS, \omega) \doteq \{e_i \mid \alpha(e_i) \in \mathit{TPs}(PS), e_i \notin \mathit{TPs}(PS), PS(\alpha(e_i)) + \omega_{e_i} = tnc(PS, \omega)\}$

The outcome of a `wait` *decision (written $\mathcal{O}(PS, \omega, \texttt{wait})$) is given by:*

$$\mathcal{O}(PS, \omega, \texttt{wait}) \doteq PS \cup \{\langle e_i, tnc(PS, \omega)\rangle \mid e_i \in \chi^*(PS, \omega)\}$$

Definition 52 (Outcome of a $\langle T_f, \chi_f\rangle$ Decision). *Let PS be a partial schedule for which at least one controllable time point is unexecuted, and let ω be a situation that is respected by PS. For convenience, let $t = tnc(PS, \omega)$ (or $t = \infty$ if no uncontrollable time points are active in PS), and let $\chi^* =$*

$\chi^*(PS, \omega)$. The outcome of a $\langle T_f, \chi_f\rangle$ decision (written $\mathcal{O}(PS, \omega, \langle T_f, \chi_f\rangle)$) depends on the relationship between t and T_f. In particular:

$$\mathcal{O}(PS, \omega, \langle T_f, \chi_f\rangle) \doteq PS \cup \begin{cases} \{\langle e_i, t\rangle \mid e_i \in \chi^*\} & \text{if } t < T_f \\ \{\langle x, t\rangle \mid x \in \chi_f\} & \text{if } T_f < t \\ \{\langle y, t\rangle \mid y \in \chi_f \cup \chi^*\} & \text{if } T_f = t \end{cases}$$

In the first case, some uncontrollable time points happened to execute before time T_f; in the second case, only the time points in χ_f were executed; in the third case, some uncontrollable time points happened to execute precisely at the time T_f and, thus, both uncontrollable and controllable time points were executed simultaneously.

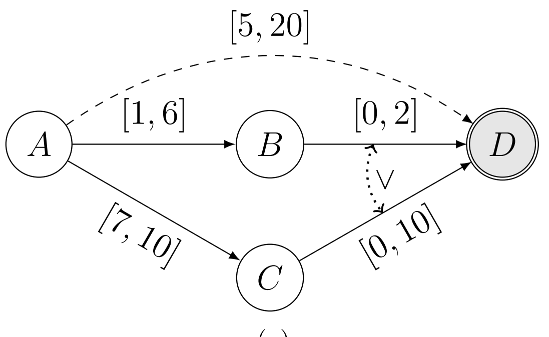
Definition 53 (RTED-based Strategy). *An RTED-based strategy for an STNU is a mapping R from respectful partial schedules to real-time execution decisions. Thus, if PS is a respectful partial schedule, then $R(PS)$ is an RTED.*

Lemma 11.1. *If R is an RTED-based strategy for a DTNU, and ω is any situation, then R and ω together determine a unique (complete) schedule, written $PS(R, \omega)$, that results from following the strategy R in the situation ω.*

Definition 54 (Dynamic Controllabilty for a DTNU). *A DTNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L}\rangle$ is dynamically controllable if there exists an RTED-based strategy R for the network such that for each situation ω, the complete schedule $PS(R, \omega)$ that results from following the strategy R satisfies all of the constraints in $\mathcal{C}$.*

11.1.1 Timestrict and Immediate-Reaction Semantics

In the literature, two slightly different versions of the semantics for dynamic controllability have been defined. The difference between the two



(a)

$$\mathcal{T}_c \doteq \{A, B, C\}$$
$$\mathcal{T}_u \doteq \{D\}$$
$$\mathcal{C} \doteq \{B - A \in [1, 6], C - A \in [7, 10], D - B \in [0, 2] \vee D - C \in [0, 10]\}$$
$$\mathcal{L} \doteq \{\langle A, \{\langle 5, 20\rangle\}\rangle, D\}$$

(b)

Figure 11.2: A DTNU derived from the one in figure 11.1 that is dynamically controllable according to the immediate-reaction semantics, but is not dynamically controllable for the timestrict semantics.

resides in the possibility for the solver to react to an observation by immediately schedule a controllable time point. [MMV01] and [Hun09] forbid this immediate reaction, while [Mor06] allows it. For example, consider the problem depicted in figure 11.2 derived from the example in figure 11.2 changing the lower bounds of the disjunctive constraint. The problem is not dynamically controllable according to the semantics in definition 54. This is because after scheduling A at time 0, the controllable time point B cannot be scheduled in until C is observed or time 9 is reached. The only viable strategy to respect the $[0, 11]$ constraint would be to wait until time 10, if C is observed during this wait *immediately* react by scheduling B, otherwise schedule B at time 9. This strategy assumes the possibility for the scheduler to immediately react to an observation (i.e. the decision of scheduling a time point and the scheduling time happen in no time).

We indicate with “timestrict” the semantics formalized by definition 54, and with “immediate-reaction” semantics allowing the agent reaction in no time, formalized as follows.

We can adapt the “timestrict” semantics by only modifying definition 49 to allow immediate reaction.

Definition 55 (Immediate Reaction RTED). *Let PS be a respectful partial schedule. An RTED has one of two forms:* `wait` *or* $\langle T_f, \chi_f \rangle$.

A `wait` *decision is applicable if at least one uncontrollable time point, e_i, is active in PS (i.e., e_i's activation time point $\alpha(e_i)$ has already been executed, but e_i has not).*

A $\langle T_f, \chi_f \rangle$ decision (i.e., "If nothing happens before or at time T_f, execute the time points in χ_f at time T_f") is applicable if $T_f \geq \texttt{now}_{PS}$ and χ_f is a non-empty subset of unexecuted controllable time points (i.e., $\chi_f \neq \emptyset$ and $\chi_f \cap TPs(PS) = \emptyset$).

Then, all the rest of the semantics stays the same. Note that this implies that if PS is a partial schedule that respects a situation ω, and an uncontrollable time point $e_i \notin TPs(PS)$ but its activation time point $\alpha(e_i) \in TPs(PS)$ (i.e., e_i is active in PS), then it follows that $\texttt{now}_{PS} \leq PS(A_i) + \omega_i$.

It is easy to see that the timestrict semantics is more demanding than the immediate-reaction one: if a problem is dynamically controllable according to the timestrict semantics, then it is also controllable for the immediate-reaction semantics. The vice-versa is not always true.

In the following, we will focus on the immediate-reaction semantics, but we will also give hints and discuss adaptations of the presented techniques for the timestrict semantics. We explicitly remark which semantics we refer to, unless there is no distinction between the two flavors.

11.2 Strategy Representation and Validation

In the practical sense, the concept of dynamic strategy for a TNU is argument of discussion. Existing approaches that solve the dynamic controllability problem, produce strategies expressed as constraint networks that need to be scheduled at run-time by an executor. These networks encode a possibly infinite number of RTEDs, but require a constraint solving algorithm running on-line with the system. This may or may not be acceptable, depending on the application at hand.

Some works aimed at reducing the amount of on-line reasoning as much as possible [Mor14]. In this chapter, we want to follow the idea we introduced for weak controllability in chapter 10: we want to automatically synthesize executable strategies.

We first propose a language to express readily executable, closed-form strategies that closely resembles the structure of a computer program. The idea is to generate strategies in a form that can be interpreted, compiled or even implemented in hardware to accommodate even more resource-constrained applications.

11.2.1 Strategy Language

We define the following language to compactly express strategies in a readily executable form.

Definition 56. *A strategy σ is recursively defined as follows.*

- `noop` *is a strategy;*
- $w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)$ *is a strategy where ψ is a time region, each $e_i \in \mathcal{T}_u$ and each σ_x is a strategy*
- $s(b); \sigma'$ *is a strategy where $b \in \mathcal{T}_c$ and σ' is a strategy.*

We defined a single wait operator that waits for a condition ψ to become true. Conditions are expressed as time regions (see section 2.4) defined over a set of clocks $\vec{T} \doteq \{\overline{x} \mid x \in \mathcal{T}\}$. We use a clock for each time point (controllable or uncontrollable). All the clocks start from time 0 and evolve as time passes. We assume that a clock is reset when the corresponding time point is executed. In this way, each clock measures the time passed since the corresponding time point was scheduled. For example, a time region $x - y \leq 2 \land x = 3$ indicates that 3 time units ago (at time $t - 3$), x was scheduled and y has been scheduled at least 1 time unit ago.

A wait action can be interrupted by the observation of an uncontrollable time point or when the waited condition becomes true (represented by the $\dashv$ symbol). For each possible outcome, the strategy prescribes a behavior, expressed as a sub-strategy. For example, if while executing a strategy $w(\psi, e_1 : \sigma_1, e_2 : \sigma_2, \dashv: \sigma_\dashv)$ the time point e_2 is observed, the wait is terminated and σ_2 is immediately executed.

In addition to wait statements, we also have the concatenation $(s(b); \sigma')$ construct that prescribes to immediately schedule the b time point (we assume no time elapses), and then proceed with the strategy σ'. The `noop` operator is a terminator signaling that the strategy is completed.
For example, a strategy σ_{ex} for the DTNU in figure 11.1, expressed in this language is as follows:

$$\begin{aligned}\sigma_{ex} \doteq s(A); w(A = 5,\ D : w(D > 0, \dashv: s(B); w(A = 7, \dashv: s(C); \texttt{noop})),\\ \dashv: s(B); w(A = 10, D : w(A \geq 7 \land D > 0, \dashv: s(C); \texttt{noop}),\\ \dashv: s(C); w(\bot, D : \texttt{noop}))).\end{aligned}$$

This example strategy is correct regardless of the dynamic controllability semantics (it is valid according to the timestrict semantics, hence it also works for the immediate-reaction semantics.). For now, we concentrate on the immediate-reaction semantics, then we will extend the results for the timestrict case in section 11.2.2.

The following theorem states that the strategy language is sufficient to capture the whole dynamic controllability semantics. The proof can be found in appendix A.6.

Theorem 11.1 (Sufficient Syntax). *A DTNU is dynamically controllable if and only if it admits a solution strategy expressible as per definition 56.*

This strategy syntax is similar to a loop-free program. In practice, one needs to represent the program allowing common strategies in different branches to be shared to avoid combinatorial explosion of the strategy size. In order to simplify the exposition, we keep the simpler tree representation. The structure of the program explicitly represent the different branches that the strategy may take: each computation path from the beginning of the strategy to `noop` is a way of scheduling the time points in a specific order.

A strategy σ needs two characteristics for being a solution to the dynamic controllability problem: *dynamicity* and *validity*, defined as follows.

A strategy σ is dynamic if it never observes future happenings and is valid if it always ends in a state where all the controllable time points are scheduled and all the free constraints are satisfied, regardless of the uncontrollable observations. Using the strategy syntax in definition 56, dynamicity can be checked syntactically: it suffices to check, for each branch of the strategy, that each wait condition ψ is defined on time points that have been already started or observed. Formally, this can be done by recursively checking the free variables of each time region ψ, as follows.

$$
\begin{aligned}
&dyn(P, \texttt{noop}) \doteq \top \\
&dyn(P, s(b); \sigma') \doteq dyn(P \cup \{b\}, \sigma') \\
&dyn(P, w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)) \doteq \\
&\qquad (FreeVars(\psi) \subseteq P) \wedge dyn(P, \sigma_\dashv) \wedge \textstyle\bigwedge_{i=1}^{n} dyn(P \cup \{e_i\}, \sigma_i)
\end{aligned}
$$

Proposition 11.1. *A strategy σ is dynamic if and only if $dyn(\emptyset, \sigma) = \top$.*

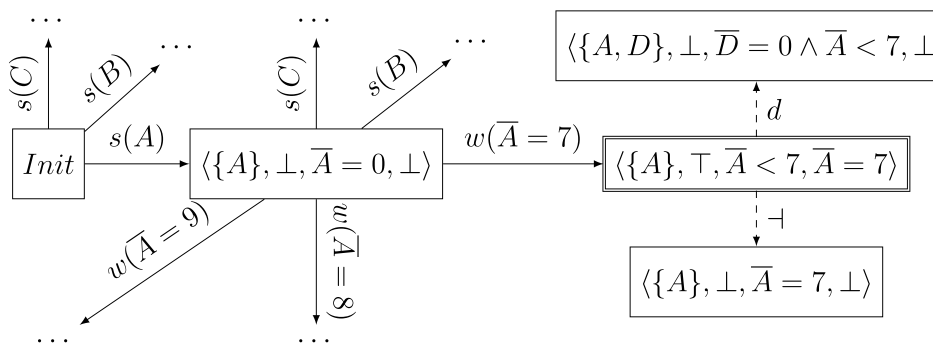
Intuitively, P keeps track of the time points that happened in the branch under analysis, and the check ensures that no time point outside P is used as a wait condition.

11.2.2 Validation

We now focus on the problem of validating a given strategy using the immediate-reaction semantics. We first present a search space that encodes every possible strategy for a given DTNU. The search space $\mathbb{S}$ is an and-or search space where the outcome of a wait instruction is an and-node (the result of a wait is not controllable by the solver), while all the other elements of the strategy language are encoded as or-nodes (they are controllable decisions). The search space is a directed graph $\mathbb{S} \doteq \langle V, E \rangle$, where E is a set of labeled edges and each node in V is a tuple $\langle P, w, \phi, \psi \rangle$ where $P \in 2^{\mathcal{T}}$ is a subset of the time points representing the time points that already happened in the past, w is a Boolean flag marking the state as a waiting state, while both ϕ and ψ are time regions. ϕ represents the set of temporal configuration in which the state can be; ψ is set only in and-nodes to record the condition that has been waited for. The graph is rooted in the node $Init \doteq \langle \emptyset, \bot, \top, \bot \rangle$. The transition relation defining the allowed moves in the space is E, defined as follows:

- $\langle P, \bot, \phi, \bot \rangle \xrightarrow{s(b)} \langle P \cup \{b\}, \bot, \rho(\phi, \overline{b}), \bot \rangle$ with $b \in \mathcal{T}_c \setminus P$;
- $\langle P, \bot, \phi, \bot \rangle \xrightarrow{w(\psi)} \langle P, \top, \omega(\phi, \psi), \psi \rangle$;
- $\langle P, \top, \phi, \psi \rangle \xrightarrow{e} \langle P \cup \{e\}, \bot, \rho(\phi, \overline{e}) \wedge uc(\overline{e}), \bot \rangle$ where $e \in \mathcal{T}_u \setminus P$ and $\alpha(e) \in P$;
- $\langle P, \top, \phi, \psi \rangle \xrightarrow{\dashv} \langle P, \bot, \psi, \bot \rangle$.

Nodes having $w = \bot$ are considered or-nodes, while the others are and-nodes. Intuitively, the first rule allows the immediate start of a controllable time point if we are not in a state resulting from a wait. The second rule allows the solver to wait for a specific condition ψ, the resulting state is an and-node because the outcome of the wait can be either a timeout ($\dashv$) or an uncontrollable time point. The last two rules explicitly distinguish



$s(C)$
$s(B)$
$s(A)$
$Init$
$\langle \{A\}, \bot, \overline{A} = 0, \bot \rangle$
$w(\overline{A} = 7)$
$\langle \{A\}, \top, \overline{A} < 7, \overline{A} = 7 \rangle$
$\langle \{A, D\}, \bot, \overline{D} = 0 \land \overline{A} < 7, \bot \rangle$
d
$\dashv$
$\langle \{A\}, \bot, \overline{A} = 7, \bot \rangle$
$w(\overline{A} = 9)$
$w(\overline{A} = 8)$

Figure 11.3: A portion of the search space $\mathbb{S}$ for the DTNU in figure 11.1. Doubly bordered nodes are and-nodes, the others are or-nodes.

these outcomes. We remark that, when a time point x is scheduled or observed, the corresponding clock $\overline{x}$ is reset and the set P keeps record of the time points that have been scheduled or observed.

Given a time region ϕ we define the time region that specifies the waiting time for a condition ψ as a time region $\omega(\phi, \psi) \doteq \phi\nearrow \land \neg(\psi\nearrow)$. This is the set of time assignments resulting from starting a wait for condition ψ starting from any assignment compatible with ϕ.

For each uncontrollable time point e, we define the $uc(e)$ time region as $\bigvee_{\langle l,u \rangle \in \mathcal{B}} \overline{\alpha(e)} \geq l \land \overline{\alpha(e)} \leq u$ where $\langle \alpha(e), B, e \rangle \in \mathcal{L}$. Intuitively, $uc(e)$ is the portion of time in which the uncontrollable time point e might be observed, according to the contingent links. We remark that the time region only depend on the clock corresponding to the activation time point $\alpha(e)$ because clocks measure the time since the corresponding time clock happened. In fact, this is a transliteration of a contingent link into a time region. This search space directly mimics the structure of our strategies, but is infinite due to the infinite number of conditions that can be waited for. Nonetheless, this space is conceptually clean and very useful to approach the validation problem. Figure 11.3 depicts the first portion of the search space $\mathbb{S}$ for the running example problem in figure 11.1.

The procedure for validating a strategy is shown in algorithm 14: the algorithm navigates the search space $\mathbb{S}$, by applying the strategy prescriptions (thus finitizing the search) and checking that each branch invariably yields to a state where all the free constraints are satisfied and all the time points are scheduled. The algorithm starts from $Init$ and recursively validate each branch of the strategy. The region Ψ, used in line 2, is defined as the region where all the free constraints are satisfied, as follows:

$$\Psi \doteq \bigwedge_{c_i \in \mathcal{C}} \bigvee_{j=1}^{D_i} l_{i,j} \leq \bar{t}_{i,2,j} - \bar{t}_{i,1,j} \leq u_{i,j}.$$

A time region implies Ψ if it satisfies all the free constraints. This is because of the implication semantics: a model satisfies the region only in it also satisfies the Ψ. We remark that clocks measure the time passed since the corresponding time point has been executed, therefore a constraint $x - y \in [l, u]$ is represented by the region $l \leq \overline{y} - \overline{x} \leq u$. The procedure executes a strategy starting from $Init$; then it recursively explores the search space enforcing the controllable decisions of the strategy σ in or-nodes (lines 3-8) and branching to explore all possible uncontrollable outcomes in and-nodes (lines 9-15). It is easy to see that the algorithm takes a number of steps that is linear in the size of the strategy because at each step the strategy gets shortened (two steps are needed to shorten a wait).

Proposition 11.2. *The validation procedure in algorithm 14 is sound and complete for the immediate-reaction semantics.*

Additional Constraints for Timestrict Semantics

When dealing with the timestrict semantics, we must prevent the executor from react in 0 time to an observation. The only action of an executor is the scheduling of a time point through a command $s(b)$. The timestrict

Algorithm 14 Dynamic Strategy Validation Procedure

```
procedure Validate($\sigma$, $\langle P, w, \phi, \psi \rangle$)
    if $P = \mathcal{T}$ then
        return ($\sigma =$ noop  and  IsValid($\phi \rightarrow \Psi$))
    end if
    if $w = \bot$ then
        if $\sigma = s(b); \sigma'$ then
            return Validate($\sigma'$, $\langle P \cup \{b\}, \bot, \rho(\phi, b), \bot \rangle$)
        else if $\sigma = w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)$ then
            return Validate($\sigma$, $\langle P, \top, \omega(\phi, \psi), \psi \rangle$)
        else
            return $\bot$                    ▷ $\sigma =$ noop, but not all time points scheduled
        end if
    else                                     ▷ $\sigma \doteq w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_\dashv)$
        for all $\langle P, w, \phi, \psi \rangle \xrightarrow{\hat{e}} \langle P \cup \{\hat{e}\}, \bot, \phi \wedge uc(\hat{e}), \bot \rangle$ do
            if $\nexists i . \hat{e} = e_i$ then
                return $\bot$                ▷ $\hat{e}$ is not handled by $\sigma$
            end if
            $c :=$ Validate($\sigma_i$, $\langle P \cup \{\hat{e}\}, \bot, \rho(\phi, e) \wedge uc(e), \bot \rangle$)
            if $c = \bot$ then
                return $\bot$
            end if
        end for
        if IsUnsatisfiable($\psi$) then
            return $\top$                    ▷ The strategy waited for $\bot$
        end if
        return Validate($\sigma_\dashv$, $\langle P, \bot, \psi, \bot \rangle$)
    end if
end procedure
```

semantics prevents the executor from scheduling a time point if no time is passed since the last observed time point.

This modifies the validation search space $\mathbb{S}$: the region in which a $s(b)$ transition is enabled is constrained to have all the clocks corresponding to the uncontrollable time points positive, so that to ensure a positive delay. In concrete, this changes the first rule of the transition relation for the search space $\mathbb{S}$ as follows.

- $\langle P, \perp, \phi, \perp\rangle \xrightarrow{s(b)} \langle P \cup \{b\}, \perp, \rho(\phi, \overline{b}), \perp\rangle$ with $b \in \mathcal{T}_c \setminus P$ and $\bigwedge_{e \in (\mathcal{T}_u \cap P} \overline{e} > 0) \subseteq \phi$;

Intuitively, this means that the strategy can start a controllable time point only if a positive amount of time has passed since any observed uncontrollable time point. In practice, this forces the strategy to wait after each observation, a start is only valid in the timeout case of a wait. This change also propagates to the algorithm, but the rest of the technique is unchanged.

11.3 Reducing Dynamic Controllability to TGA Reachability

In this section, we present a general schema to approach the dynamic controllability problem for the whole DTNU network class. The idea is to reduce the problem to a Reachability Game on a Timed Game Automaton (TGA) obtained via a *linear* encoding procedure. Since the reachability problem for TGA is decidable and algorithms have been developed to solve this problem, this reduction constitutes a viable and novel solution approach for the open problem of dynamic controllability of DTNU. Moreover, this technique can be extended to deal with discrete non-determinism [CHM$^+$16]. However, this is beyond the scope of this thesis: we only report some preliminary work in section 11.5.

This section is organized as follows. We first introduce an encoding of the STNU immediate-reaction dynamic controllability problem in TGA such that the temporal network is dynamic controllable if and only if the reachability game on the encoded TGA is solvable. We then generalize the idea to the DTNU case and to the timestrict semantics.

11.3.1 STNU to TGA

Given any STNU $P = \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, the goal is to generate a corresponding TGA $T_P = \langle L, l_0, Act, \mathcal{X}, E, Inv \rangle$, and a winning condition θ, such that the STNU P is dynamically controllable if and only if the TGA T_P admits a strategy for θ. We use controllable TGA transitions are used to model the execution of the controllable time points in P, and uncontrollable TGA transitions are used to model the execution of the controllable time points in P. We model the dynamic controllability problem as a game in which two players, the solver and the environment, are demanded to schedule time points in real time (by resetting clocks). The goal of the solver is to reach a goal location where all the constraints are satisfied, while the environment tries to prevent its success.

For this encoding, the set of locations is: $L \doteq \{\texttt{ctrl}, \texttt{env}, \texttt{goal}\}$, where `ctrl` is marked urgent. This means that time is not allowed to pass within the `ctrl` location (see section 2.3 for the details). Note that L only has three locations, regardless of the number of time points in the STNU. Intuitively, `ctrl` represents a state in which the solver can execute one or more controllable time points; `env` represents a state in which the environment can execute one or more uncontrollable time points; and `goal` represents a state in which all of the constraints have been satisfied and the game is over (and `ctrl` wins successfully scheduling all the time points fulfilling all the constraints). The initial location of the TGA is `env` (i.e., $l_0 \doteq \texttt{env}$).

The set of clocks is: $\mathcal{X} \doteq \{\overline{\gamma}, \overline{\delta}\} \cup \{\overline{x} \mid X \in \mathcal{T}\}$. All clocks start at 0.

The clock $\overline{\gamma}$ is never reset; it simply measures global time. The clock $\overline{\delta}$ is used to limit the alternation of controllable and uncontrollable scheduling in a single time instant: we require that a positive amount of time passes after each controllable decision. In this way, we prevent the environment from reacting to controllable decision in no time[1]. Finally, for each time point $x \in \mathcal{T}$, there is a corresponding clock $\overline{x}$. That clock is reset at most once each run, at the instant x is executed. It follows that any time point x has been executed if and only if $\overline{x} < \overline{\gamma}$. Since the initial state is `env`, no time point can be executed at 0. Also, after being executed, the execution time for X is forever equal to $\overline{\gamma} - \overline{x}$.

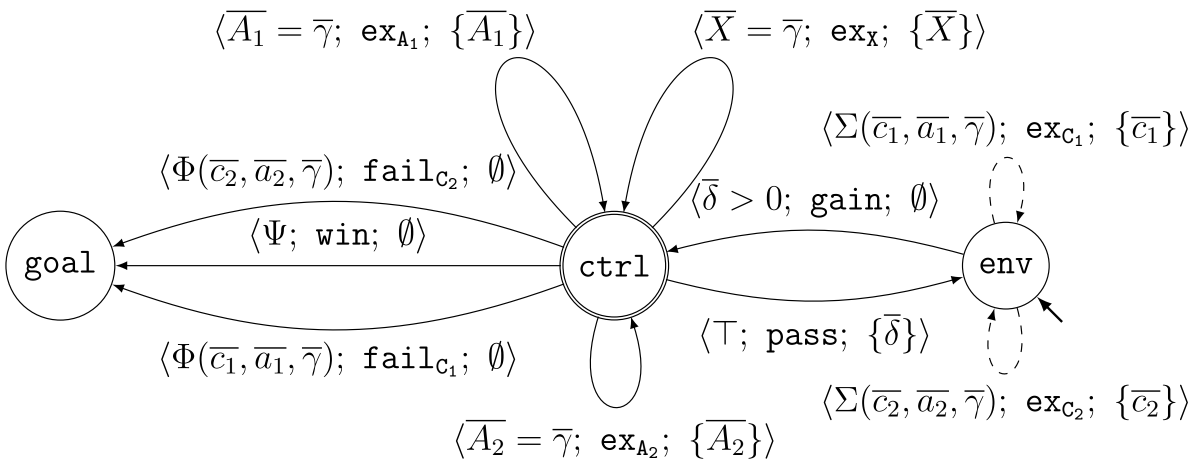
The sets of controllable and uncontrollable actions are defined as follows. First, the uncontrollable transitions (for the environment) consist of one action for each uncontrollable time point in P, as follows: $Act_u \doteq \{\mathtt{ex}_\mathtt{x} \mid x \in \mathcal{T}_u\}$. Each action in this set represents the execution of the corresponding time point. The controllable actions (for the solver) include more options: $Act_c \doteq \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$, defined as follows.

$$\mathcal{A}_1 \doteq \{\mathtt{ex}_\mathtt{x} \mid x \in \mathcal{T}_c\}$$
$$\mathcal{A}_2 \doteq \{\mathtt{fail}_\mathtt{e} \mid \langle b, l, u, e\rangle \in \mathcal{L}\}$$
$$\mathcal{A}_3 \doteq \{\mathtt{gain}, \mathtt{pass}, \mathtt{win}\}$$

$\mathcal{A}_1$ contains one execution action for each controllable time point. $\mathcal{A}_2$ contains one action for each contingent link; these actions are only enabled if the environment violates the bounds on any contingent link. `gain` and `pass` model the interplay between the execution of time points by the solver and the environment; `win` is used at the end when all time points have been executed and all constraints have been satisfied.

The transition relation, E, for the TGA encoding of an STNU is demon-

[1]We remark that we are now considering immediate-reaction semantics, in which the solver can react in 0 time. We are designing a game in which the environment is forced to execute all the uncontrollable time points of a given instant before the solver reaction.



$\langle \overline{A_1} = \overline{\gamma};\ \mathtt{ex}_{\mathtt{A_1}};\ \{\overline{A_1}\}\rangle$
$\langle \overline{X} = \overline{\gamma};\ \mathtt{ex}_{\mathtt{X}};\ \{\overline{X}\}\rangle$
$\langle \Sigma(\overline{c_1}, \overline{a_1}, \overline{\gamma});\ \mathtt{ex}_{\mathtt{C_1}};\ \{\overline{c_1}\}\rangle$
$\langle \Phi(\overline{c_2}, \overline{a_2}, \overline{\gamma});\ \mathtt{fail}_{\mathtt{C_2}};\ \emptyset\rangle$
$\langle \overline{\delta} > 0;\ \mathtt{gain};\ \emptyset\rangle$
$\langle \Psi;\ \mathtt{win};\ \emptyset\rangle$
goal
ctrl
env
$\langle \top;\ \mathtt{pass};\ \{\overline{\delta}\}\rangle$
$\langle \Phi(\overline{c_1}, \overline{a_1}, \overline{\gamma});\ \mathtt{fail}_{\mathtt{C_1}};\ \emptyset\rangle$
$\langle \Sigma(\overline{c_2}, \overline{a_2}, \overline{\gamma});\ \mathtt{ex}_{\mathtt{C_2}};\ \{\overline{c_2}\}\rangle$
$\langle \overline{A_2} = \overline{\gamma};\ \mathtt{ex}_{\mathtt{A_2}};\ \{\overline{A_2}\}\rangle$

Figure 11.4: Encoding the STNU from figure 11.1 into a TGA. Solid arrows represent controllable transitions (for the environment); dashed arrows uncontrollable transitions (for the solver). The doubly-circled `ctrl` location is urgent; the initial location is `env`.

strated in figure 11.4, using the sample STNU from figure 11.1. For each controllable time point x, there is a transition from `ctrl` to `ctrl` labeled by $\langle \overline{x} = \overline{\gamma};\ \mathtt{ex}_{\mathtt{x}};\ \{\overline{x}\}\rangle$, which represents the execution of x by the solver. The guard, $\overline{x} = \overline{\gamma}$ (meaning that x has not been executed yet), ensures that this transition will be taken at most once per run. The set, $\{\overline{x}\}$, stipulates that the clock $\overline{x}$ will be reset by this transition, signaling that x has been executed.

Similarly, for each contingent link, $\langle b, l, u, e\rangle$, there is a transition from `env` to `env` labeled by $\langle \Sigma(\overline{e}, \overline{b}, \overline{\gamma}, \overline{\delta});\ \mathtt{ex}_{\mathtt{e}};\ \{\overline{e}\}\rangle$, which represents the execution of e by Vera. The guard, is defined as follows.

$$\Sigma(\overline{e}, \overline{b}, \overline{\gamma}, \overline{\delta}) \doteq (\overline{\delta} > 0) \land (\overline{b} < \overline{\gamma}) \land (\overline{e} = \overline{\gamma}) \land (\overline{b} \geq l) \land (\overline{b} \leq u)$$

The guard ensures that this transition can only be taken when the link is currently activated and its duration would fall within $[l, u]$. We also require that the clock $\overline{\delta}$ is positive, this prevents the environment from scheduling an uncontrollable time point immediately after the solver scheduled something. In addition, for each contingent link, $\langle b, l, u, e\rangle$, there is a transition from `ctrl` to `goal` labeled by $\langle \Phi_e(\overline{b}, \overline{e}, \overline{\gamma})$; $\texttt{fail}_{\texttt{e}}$; $\emptyset\rangle$, enabling the solver to move to `goal` should the environment violate the bounds on that link by failing to execute e. Its guard is as follows.

$$\Phi_C(\overline{b}, \overline{e}, \overline{\gamma}) \doteq (\overline{b} < \overline{\gamma}) \wedge (\overline{b} > u) \wedge (\overline{e} = \overline{\gamma})$$

The intuition is that the upper bound on the duration of the contingent link has passed and the uncontrollable time point has not been scheduled.

Next, if $\vec{t}$ is the vector of clocks $\overline{x}$ such that $x \in \mathcal{T}$, the transition from `ctrl` to `goal` labeled by $\langle \Psi(\vec{t}, \overline{\gamma})$; `win`; $\emptyset\rangle$ signals the end of the game. $\Psi(\vec{t}, \overline{\gamma})$ models that all time points have been executed and all constraints are satisfied.

$$\Psi(\vec{t}, \overline{\gamma}) \doteq \bigwedge_{x \in \mathcal{T}} (\overline{x} < \overline{\gamma}) \wedge \bigwedge_{(y - x \in [l,u]) \in \mathcal{C}} ((\overline{x} - \overline{y} \geq l) \wedge (\overline{x} - \overline{y} \leq u))$$

Last, to model the interplay between the players, there are two more transitions. The transition from `env` to `ctrl` labeled by $\langle \overline{\delta} > 0$; `gain`; $\emptyset\rangle$ enables the solver to gain control for the purpose of executing some controllable time points, but only after some positive delay since the last time the controller executed something. This prevents the solver to avoid the environment to play and avoids multiple turns without time passing. The transition from `ctrl` to `env` labeled by $\langle \top$; `pass`; $\{\overline{\delta}\}\rangle$ enables the solver to immediately pass back to `env`, once it has finished executing the chosen time points. Crucially, no time elapses from the instant the system leaves `env` for `ctrl` to the instant it returns, because `ctrl` is an urgent state

The winning condition θ of the reachability game is to reach the `goal` state. A strategy for the solver beats the environment by ensuring that `goal` can be reached regardless of the possible uncontrollable moves.

Proposition 11.3. *The presented encoding is such that the STNU P is dynamically controllable if and only if the reachability game admits a winning strategy for the controller.*

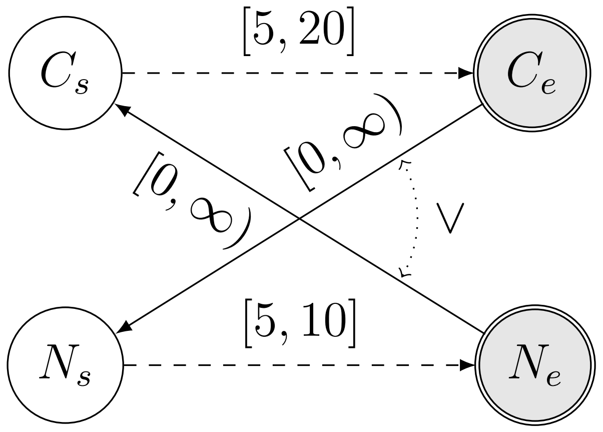


Figure 11.5: An small DTNU example used to demonstrate the DTNU-toTGA encoding of the free constraints. The DTNU corresponds to a couple of activities N and C with uncontrollable duration in $[5, 20]$ that cannot overlap.

11.3.2 DTNU to TGA

In this section, we generalize the TGA encoding we presented for the STNU problem class for the DTNU dynamic controllability problem.

DTNU generalizes the STNU framework in two different dimensions. First, the durations of contingent links can be constrained to lie within a union of disjoint intervals. Second, the free constraints can comprise Boolean combinations of difference constraints. We show how to modify the STNU-to-TGA encoding in section 11.3.1 to capture the immediate-reaction semantics of DTNUs. First, if the duration for a contingent link is constrained to lie within one of n disjoint intervals, then there will be n corresponding loops at the `env` location, where the guard for each loop effectively specifies one of the allowed intervals for that contingent duration. Second, the "all time points executed and all constraints satisfied" transition to the `goal` location is represented by alternative pathways through a sequence of locations from `env` to `goal`.

To begin, as for the STNU case, each controllable time point x will have a corresponding transition from `ctrl` to `ctrl` labeled as $\langle \overline{x} = \overline{\gamma};\ \texttt{ex}_\texttt{x};\ \{\overline{x}\} \rangle$, that represents the execution of x by the solver.

However, for each disjunctive contingent link, $\langle b, \mathcal{B}, e\rangle$, where $\mathcal{B} \doteq \{\langle l_1, u_1\rangle, \cdots, \langle l_n, u_n\rangle\}$, there are n uncontrollable transitions from `env` to `env`: $\langle \Sigma(\overline{b}, \overline{e}, \overline{\gamma}, \overline{\delta}, l_i, u_i);\ \texttt{ex}_{\texttt{e}}{}^{i};\ \{\overline{e}\}\rangle$, for $i \in [1, n]$. Each transition represents a possible interval of execution of e by the environment. The guards are defined as follows.

$$\Sigma(\overline{b}, \overline{e}, \overline{\gamma}, \overline{\delta}, l_i, u_i) \doteq (\overline{\delta} > 0) \wedge (\overline{b} < \overline{\gamma}) \wedge (\overline{e} = \overline{\gamma}) \wedge (\overline{e} \geq l_i) \wedge (\overline{e} \leq u_i)$$

The guards ensure that (one of) these transitions can be taken only when the link is currently activated and its duration would fall within one of the allowed intervals of $\mathcal{B}$. We highlight that intervals in $\mathcal{B}$ are disjoint as per definition 40.

In addition, for each contingent link, we have a controllable transition from `ctrl` to `goal` labeled as $\langle \Phi_e(\overline{b}, \overline{e}, \overline{\gamma}, max_i(u_i));\ \texttt{fail}_{\texttt{e}};\ \emptyset\rangle$ that allows the solver to win the game if the environment refuses to schedule an uncontrollable time point within the maximum allowed bound, $max_i(u_i)$. The guard is expressed by $\Phi_e(\overline{b}, \overline{e}, \overline{\gamma}, u) \doteq (\overline{b} < \overline{\gamma}) \wedge (\overline{b} > u) \wedge (\overline{e} = \overline{\gamma})$, analogously to the STNU case. The interplay between the players, governed by the `pass` and `gain` transitions, is identical to the STNU case.

Next, the TGA must accommodate the arbitrary Boolean combinations of constraints in $\mathcal{C}$. In principle, we would like to have a transition from `ctrl` to `goal` labeled as $\langle \Omega(\vec{t}, \overline{\gamma});\ \texttt{win};\ \emptyset\rangle$ that signals the end of the game, where $\Omega(\vec{t}, \overline{\gamma})$ encodes the fact that all time points have been executed and all constraints satisfied, and $\vec{t}$ is the set of all the clocks associated with the time points.

$$\Omega(\vec{t}, \overline{\gamma}) \doteq \Big(\bigwedge_{x \in \mathcal{T}} (x \leq \overline{\gamma})\Big) \wedge \Big(\bigwedge_{i=1}^{|\mathcal{C}|} \bigvee_{j=1}^{D_i} (\overline{t_{i,2,j}} - \overline{t_{i,2,j}} \geq l_{i,j}) \wedge (\overline{t_{i,2,j}} - \overline{t_{i,2,j}} \leq u_{i,j})\Big)$$

Intuitively, $\Omega(\vec{t}, \overline{\gamma})$ is a transposition of the free constraints of the DTNU, in the form of a time region. First we check that all the time points have been scheduled ($\bigwedge_{x \in \mathcal{T}}(x \leq \overline{\gamma})$), then we enforce all the constraints

However, it is not always possible to directly use the formula $\Omega(\vec{t}, \overline{\gamma})$ as a guard for the transition from `ctrl` to `goal` because the definition of a TGA restricts the language of the guards to be purely conjunctive. For this reason, we aim at building a piece of automaton (possibly adding new locations) that connects `ctrl` to `goal` in such a way that there is a path from `ctrl` to `goal` if and only if all the free constraints are satisfied. This suffices to ensure the soundness and completeness of the encoding. There are several ways in which this can be done.

Disjunctive Normal Form

A direct solution that does not introduce new locations in the TGA is to create a set of transitions from `ctrl` to `goal` such that each pathway from `ctrl` to `goal` can be taken if and only if $\Omega(\vec{t}, \overline{\gamma})$ is satisfied. This is always possible, since alternative transitions emanating from a single location are equivalent to a single transition with a disjunctive guard. Thus, all we have to do is convert $\Omega(\vec{t}, \overline{\gamma})$ into Disjunctive Normal Form (DNF) and create a separate transition from `ctrl` to `goal` for every disjunct. In this setting, negation of atomic constraints is not a problem because $\neg(\overline{y} - \overline{x} \leq k)$ is equivalent to $\overline{y} - \overline{x} > k$, which is allowed in the guards of a TGA. As for the names of the actions, we simply assign to each action a new, unique name. It is easy to see that there exists a path from `ctrl` to `goal` if and only if the free constraints are satisfied, because one disjunct of the DNF is satisfied. The main drawback of this technique is that, for a general formula, the number of disjuncts in the DNF is exponential, and thus the encoding becomes exponential in general. Nevertheless, this constitutes a sound-and-complete encoding for (constructively) deciding the dynamic controllability of DTNU.

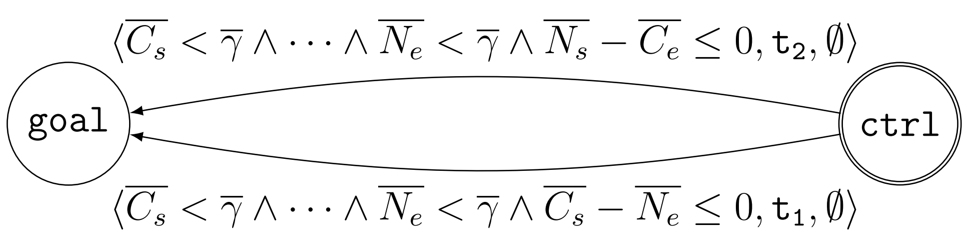

$\langle \overline{C_s} < \overline{\gamma} \wedge \cdots \wedge \overline{N_e} < \overline{\gamma} \wedge \overline{N_s} - \overline{C_e} \leq 0, \mathtt{t}_2, \emptyset \rangle$
goal
ctrl
$\langle \overline{C_s} < \overline{\gamma} \wedge \cdots \wedge \overline{N_e} < \overline{\gamma} \wedge \overline{C_s} - \overline{N_e} \leq 0, \mathtt{t}_1, \emptyset \rangle$

Figure 11.6: The DNF encoding of the guards on constraints from `ctrl` to `goal` for the sample DTNU in figure 11.5.

Considering the example in figure 11.5, the encoding of the constraints between `ctrl` and `goal` is composed of only two transitions, as Ω is already in DNF. The transitions are depicted in figure 11.6.

Negation Normal Form

If we allow for the introduction of new (urgent) locations in the TGA, we can encode $\Omega(\vec{t}, \overline{\gamma})$ linearly, thus obtaining a linear size of the overall DTNU-to-TGA encoding. The idea comes from the following observation. Suppose we have a piece of automaton that encodes a formula ϕ_1 in such a way that it is possible to move from location L_1^s to L_1^e if and only if ϕ_1 is satisfied, and suppose that we have an analogous encoding for another formula ϕ_2 with starting and ending locations L_2^s to L_2^e. We can encode the formula $\phi_1 \wedge \phi_2$ by "concatenating" the two automata. That is, we introduce a transition from L_1^e to L_2^s with the tautological guard $\top$. Now, in order to move from L_1^s to L_2^e the formula $\phi_1 \wedge \phi_2$ must be satisfied.[2] Similarly, if we consider the formula $\phi_1 \vee \phi_2$ we can introduce two extra locations $L_\vee^s$ and $L_\vee^e$ and introduce four transitions with the guard $\top$: one from $L_\vee^s$ to L_1^s, one from $L_\vee^s$ to L_2^s, one from L_1^e to $L_\vee^e$, and one from L_2^e to $L_\vee^e$. In this way, we create a "diamond" with two paths from $L_\vee^s$ to $L_\vee^e$; one path encodes ϕ_1, the other encodes ϕ_2. This construction is simple

[2]We are assuming that all the locations of the automata pieces are urgent, so the clocks are frozen and no time can elapse.

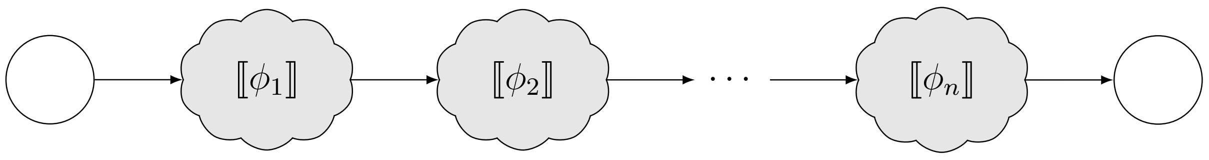


Figure 11.7: Encoding the conjunction $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$. $[\![\phi]\!]$ stands for the recursive encoding of ϕ.

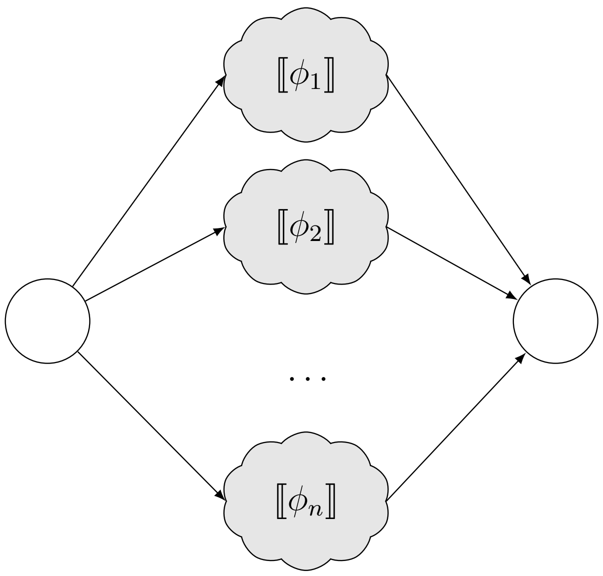


Figure 11.8: Encoding the conjunction $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$. $[\![\phi]\!]$ stands for the recursive encoding of ϕ.

and correct, even though it introduces many unneeded locations. In fact it is also possible to compress this encoding by merging locations instead of linking them with tautological transitions and it is possible to merge sequences of guards in a single conjunctive guard. However, we decided to explain the simplest version for clarity.

Now, given a rewriting of $\Omega(\vec{t}, \overline{\gamma})$ that only has disjunctions and conjunctions (but no negations) we can recursively create a piece of automaton that encodes the formula. This is done by constructing an automaton in which disjunctions and conjunctions are recursively encoded as shown in figures 11.7 and 11.8. It is well known [Kle67] that we can syntactically and linearly transform $\Omega(\vec{t}, \overline{\gamma})$ into Negation Normal Form (NNF) and given the

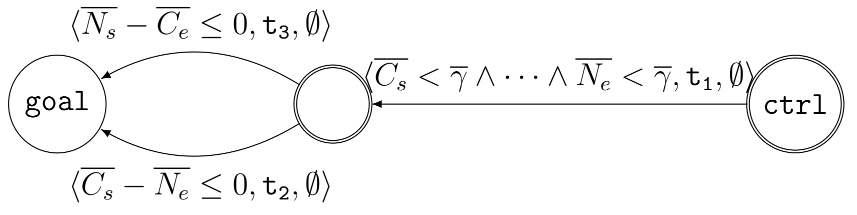

$\langle \overline{N_s} - \overline{C_e} \leq 0, \mathtt{t}_3, \emptyset \rangle$
goal
$\langle \overline{C_s} < \overline{\gamma} \wedge \cdots \wedge \overline{N_e} < \overline{\gamma}, \mathtt{t}_1, \emptyset \rangle$
ctrl
$\langle \overline{C_s} - \overline{N_e} \leq 0, \mathtt{t}_2, \emptyset \rangle$

Figure 11.9: NNF constraints between `ctrl` and `goal` for the sample DTNU in figure 11.5.

shape of the atoms of $\Omega(\vec{t}, \overline{\gamma})$ we can transform the negations of the atoms into positive atoms as before, by exploiting the fact that $\neg(\overline{y} - \overline{x} \leq \delta)$ is equivalent to $\overline{y} - \overline{x} > \delta$.

Figure 11.9 depicts the running example encoded using the NNF decomposition of the free constraints optimized by compressing the conjunction of the $\overline{x} < \overline{\gamma}$ in a single guard and by avoiding the unneeded tautological guards. In the example, Ω is already in NNF as there are no negations.

Even though the NNF decomposition is generally more succinct than the DNF, the effort of dealing with disjunctions is moved from the encoding to the TGA solver, so we are in a trade-off condition.

11.3.3 TGA Strategies

Algorithms for solving TGA reachability games such as [CDF+05], can synthesize control strategies for the controllable player. As we discussed in section 2.3, memoryless strategies are sufficient for reachability games and all the available tools produce those.

The encodings we presented, not only capture the dynamic controllability decision problem, but we have a simple way of understanding the strategies synthesized to solve the TGA in the TNU context. In fact, we encoded all the problem constraints in such a way that the clock $\overline{x}$ is reset only once in each run, and the moment it is reset (in absolute time) encodes the value assigned to the corresponding time point x. Moreover, we modeled each controllable time point with a controllable transition and each uncontrollable time point with an uncontrollable transition; therefore, the TGA strategy, that prescribes when to take the controllable transitions, encodes a dynamic strategy.

A memoryless TGA strategy can be employed in a TNU executor by providing a TGA simulator that executes our encoding controlled by the strategy. Whenever the strategy prescribes a move on a transition that resets a controllable time point, that point must be executed in the system. Each time an uncontrollable is observed in the TNU, the corresponding uncontrollable transition must be taken. This solution is far from being practical, but requires no constraint propagation at runtime. In fact, the shape of a memoryless strategy is quite distant from the syntax we proposed in definition 56. While a memoryless strategy is a mapping from pairs of locations and a time regions to TGA transitions, our language is similar to a stateful program. In the next chapter, we devise a direct technique based on the ideas of this encoding, and we present an algorithm that can extract a strategy expressed as prescribed in definition 56 from a memoryless (we call it “flat”) strategy.

11.3.4 Addressing the Timestrict Semantics

So far, we focused on the immediate-reaction semantics because it can be translated in the TGA framework more intuitively. In fact, we directly map uncontrollable time points into uncontrollable transitions and controllable time point into controllable transitions. In order to address the timestrict semantics, instead, we need to swap the roles of the players, because in the TGA semantics, the environment always has the precedence over the controller. In fact, all the difference between the two semantics is on whether or not the controller is allowed to immediately react to an uncontrollable, that is whether the controller is invoked before or after the environment move. The interplay between the players is encoded in TGA using the transitions between `ctrl` and `env` and by the use of the clock $\overline{\delta}$.

In order to encode the timestrict semantics we need to make the following modifications.

- Swap the roles of the players, the controllable transitions become uncontrollable and vice-versa.
- Transform the reachability game into a safety game: the goal of the solver is to avoid the `goal` location
- Remove the $\overline{\delta} > 0$ constraint from the $\mathtt{ex_e}$ transitions.
- Reset $\overline{\delta}$ in each controllable transition.

Moreover, we consider the TNU dynamically controllable if and only if the safety game is unsolvable: i.e. there exists no winning strategy for the controllable player. The intuition is that the controllable player is now the environment trying to avoid the scheduler from fulfilling the free constraints; if the environment has a winning strategy it means that the scheduler cannot win, otherwise the scheduler has at least a way of scheduling all the controllable time points fulfilling all the constraints.

Figure 11.10 reports the example encoded in TGA using the timestrict semantics. The important consideration is that in this encoding $\overline{\delta}$ is used to force a positive amount of time to pass between the last scheduled uncontrollable time point and the solver reaction. In fact $\overline{\delta}$ is reset in each uncontrollable time point scheduling and is required to be positive to allow the solver to schedule a controllable.

A formal proof of this encoding (due to Luke Hunsberger) is given in appendix A.5. The proof exploits the STNU dynamic controllability semantics reported in appendix A.4.

The very same modification schema can be applied also to the DTNU-to-TGA encoding obtaining a timestrict dynamic controllability decision

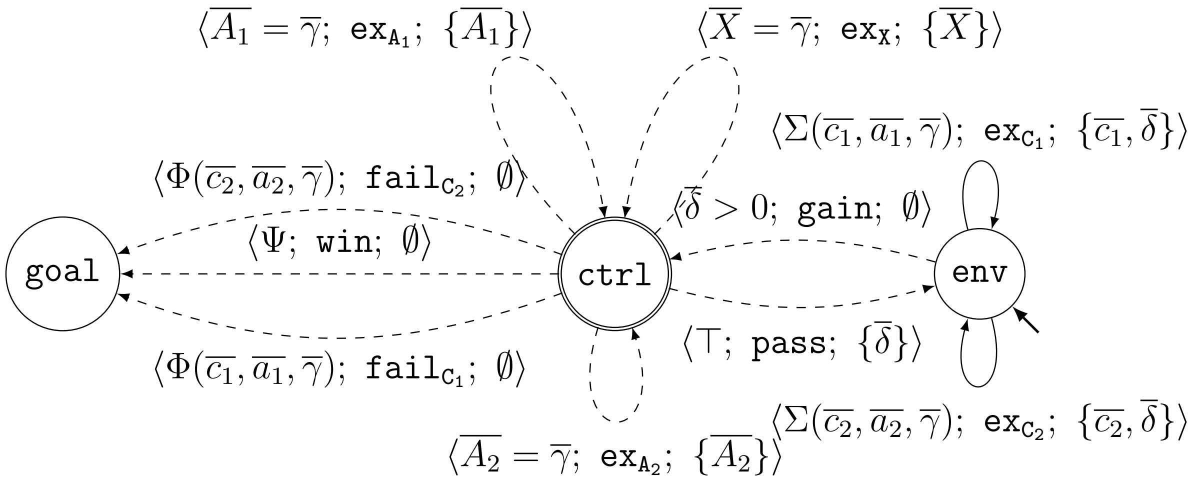

$\langle \overline{A_1} = \overline{\gamma};\ \texttt{ex}_{\texttt{A}_1};\ \{\overline{A_1}\}\rangle$
$\langle \overline{X} = \overline{\gamma};\ \texttt{ex}_{\texttt{X}};\ \{\overline{X}\}\rangle$
$\langle \Sigma(\overline{c_1}, \overline{a_1}, \overline{\gamma});\ \texttt{ex}_{\texttt{C}_1};\ \{\overline{c_1}, \overline{\delta}\}\rangle$
$\langle \Phi(\overline{c_2}, \overline{a_2}, \overline{\gamma});\ \texttt{fail}_{\texttt{C}_2};\ \emptyset\rangle$
$\langle \overline{\delta} > 0;\ \texttt{gain};\ \emptyset\rangle$
$\langle \Psi;\ \texttt{win};\ \emptyset\rangle$
goal
ctrl
env
$\langle \top;\ \texttt{pass};\ \{\overline{\delta}\}\rangle$
$\langle \Phi(\overline{c_1}, \overline{a_1}, \overline{\gamma});\ \texttt{fail}_{\texttt{C}_1};\ \emptyset\rangle$
$\langle \Sigma(\overline{c_2}, \overline{a_2}, \overline{\gamma});\ \texttt{ex}_{\texttt{C}_2};\ \{\overline{c_2}, \overline{\delta}\}\rangle$
$\langle \overline{A_2} = \overline{\gamma};\ \texttt{ex}_{\texttt{A}_2};\ \{\overline{A_2}\}\rangle$

Figure 11.10: Encoding the STNU from figure 11.1 into a TGA for the timestrict semantics. Solid arrows represent controllable transitions (for the environment); dashed arrows uncontrollable transitions (for the solver). The doubly-circled `ctrl` location is urgent; the initial location is `env`.

procedure [CHM+16].

Also this encoding can be used to produce a dynamic strategy; in fact, some tools (for example Uppaal-TIGA has this feature) can synthesize a counter-strategy as a proof of unsolvability of the game. The counter-strategy is a memoryless strategy for the uncontrollable player, that in this encoding is demanded to schedule the controllable time points. Hence, also with this encoding we can derive a dynamic strategy from the synthesis algorithm for the encoded TGA.

11.4 Synthesizing Dynamic Strategies

We now discuss a direct synthesis technique for DTNU. Differently from the encoding in the previous sections, we directly synthesize dynamic strategies that are valid by construction.

In principle, one could do a classical and-or search in the space $\mathbb{S}$: if the search terminates, the trace of the search itself is a valid strategy. However, the infinity of the space makes this choice impractical. The problem with the search space $\mathbb{S}$ is the presence of explicit waits. In fact, while the length of each path is finite, the arity of each or-node is infinite due to all the possible waits. In fact, even fixing the structure of a strategy, we can wait for any time region expressed over the set of time points that have been already observed or executed. Hence, the possibilities are infinite due to the density of time.

In order to practically synthesize a strategy, we retain the basic idea behind the search space $\mathbb{S}$ of explicitly representing the possible orderings in which the time points may happen in time, but we combine this idea with the TGA semantics to implicitly represent the controllable waits. In a TGA, time elapses inside locations until a transition (controllable or uncontrollable) is taken, causing a location change and possibly some resets. This is conceptually analogous to wait for a condition allowing to take a controllable transition, provided that the wait may be interrupted by an uncontrollable transition. We exploited this feature to let time pass in our TGA encoding: the `env` location is where all the time passes.

Analogously to the TGA encoding, we model uncontrollable time points as uncontrollable transitions with appropriate guards, and controllable time points as controllable transitions. Differently, from the previous approach we want to explicitly represent the history of the system inside each location. For this reason we have a location for each subset of $\mathcal{T}$ indicating the set of time point that have been executed so far. Unfortunately, this yields a TGA whose size is exponential in the number of time points, because there are $2^{|\mathcal{T}|}$ possible subsets of the time points. In order to address this issue, we represent this exponential TGA implicitly, by constructing transitions and constraints on-demand while exploring the symbolic state space using an algorithm derived from [CDF+05]. The implicit expansion allows us to only construct the transitions that are needed for the strategy construction process.

First of all, for each constraint $c \doteq \bigvee_{j=1}^{D_i} t_{1,j} - t_{2,j} \in [l_{i,j}, u_{i,j}]$ in $\mathcal{C}$, we define the set of occurring time points as the set $occur(c) \doteq \{t_{i,j} \mid i \in [1,2], j \in [1, D_i]\}$. Moreover, given any subset of the time points P, we define the set of active constraints as $\mathcal{C}(P) \doteq \{c \mid c \in \mathcal{C}, occur(c) \subseteq P\}$. Intuitively, given any subset of the time points P (representing the past occurrences), the active constraints are the constraints that can be verified or falsified with the knowledge of the timing of the time points in P.

The implicit TGA is defined as $\langle 2^{\mathcal{T}}, \emptyset, \mathcal{T}, \overline{T}, E, \emptyset \rangle$. Each subset of the time points is a location, the initial state is the empty set where no time points have been scheduled. We have an action label for each time point: a controllable time point yields a controllable transition, an uncontrollable corresponds to an uncontrollable transition. The set of clock variables is $\overline{T}$ as in the space $\mathbb{S}$ and the transition relation E is defined as:

- $P \xrightarrow{b,\, fc(P,b)} P \cup \{b\}$ with $b \in \mathcal{T}_c$, if $b \notin P$;
- $P \overset{e,\, uc(e)}{\dashrightarrow} P \cup \{e\}$ with $e \in \mathcal{T}_u$, if $\alpha(e) \in P$ and $b \notin P$.

where each transition implicitly resets the time point corresponding to its label and the $fc(P,b)$ and $uc(e)$ functions define the guards of the transitions.

$$fc(P,b) \doteq \begin{cases} \top & \text{iff } \mathcal{C}(P) = \emptyset \\ \bigwedge_{c_i \in \mathcal{C}(P)} \bigvee_{j \in [0, D_i]} \overline{y}_{i,j} - \overline{x}_{i,j} \in [\ell_{i,j}, u_{i,j}] & \text{otherwise} \end{cases}$$

Algorithm 15 reports the pseudo-code of the approach derived from the technique in [CDF⁺05]. We write $s_1 \overset{x}{\Rightarrow} s_2$ for either $s_1 \overset{x}{\rightarrow} s_2$ or $s_1 \overset{x}{\dashrightarrow} s_2$, where no distinction is needed. Each time an expansion is required (lines 3, 10 and, implicitly, 12) the relative portion of the search space is created. The algorithm works as follows. The *win* map records the winning portion of visited states. The *wait* set contains the transitions in the

Algorithm 15 Synthesis Algorithm

```
1: procedure SYNTHESIZE( )
2:     $wait := \emptyset; dep := \emptyset; win := \emptyset$
3:     PPUSH($wait$, $\{\langle\emptyset,\top\rangle \overset{x}{\Rightarrow} \langle P,\phi\nearrow\rangle \mid \langle\emptyset,\top\rangle \overset{x}{\Rightarrow} \langle P,\phi\rangle\}$)
4:     while $wait \neq \emptyset \wedge win[\langle\emptyset,\top\rangle] = \bot$ do
5:         $(s_1 \overset{x}{\Rightarrow} s_2) :=$ POP($wait$)                ▷ $\langle P_1,\phi_1\rangle = s_1, \langle P_2,\phi_2\rangle = s_2$
6:         if not ALREADYVISITED($s_2$) then
7:             $dep[s_2] := \{s_1 \overset{x}{\Rightarrow} s_2\}$
8:             if $P_2 = \mathcal{T}$ and ISSATISFIABLE($\phi_2 \wedge \Psi$) then
9:                 $win[s_2] := \phi_2 \wedge \Psi$
10:                PUSH($wait$, $\{s_1 \overset{x}{\Rightarrow} s_2\}$)
11:            else
12:                PPUSH($wait$, $\{s_2 \overset{x}{\Rightarrow} \langle P_3,\phi_3\nearrow\rangle \mid s_2 \overset{x}{\Rightarrow} \langle P_3,\phi_3\rangle\}$)
13:            end if
14:        else
15:            $win^* :=$ BACKPROPAGATEWINNING($s_1$)
16:            if $win^* \not\subset win[s_1]$ then
17:                $win[s_1] := win[s_1] \vee win^*$
18:                PUSH($wait$, $dep[s_1]$)
19:            end if
20:            $dep[s_2] := dep[s_2] \cup \{s_1 \xrightarrow{x} s_2\}$
21:        end if
22:    end while
23:    if $win[\langle\emptyset,\top\rangle] = \emptyset$ then
24:        return $\bot$
25:    else
26:        return MKSTRATEGY($dep$, $win$)
27:    end if
28: end procedure
```

symbolic space that need to be analyzed and is initialized with the outgoing transitions of the initial state (line 3). The Push and PPush functions insert elements in the set given as argument (the difference between the two is explained later), while the Pop function picks and removes an element from the set. The *dep* map is used to record the set of explored edges that lead to a state. The procedure keeps track of the visited states and performs two different computations depending on whether a new state is encountered or a re-visit happens (line 6). In the first case, the algorithm explores the state space forward, in the other it back-propagates winning states. In the forward expansion, the distinction between controllable and uncontrollable transitions is disregarded until a goal state is reached. When a goal state is found, the winning states are recorded in the *win* map and the transition leading to the goal is re-added to the *wait* set to trigger the back-propagation of the winning states. The back-propagation computes the set of winning states and updates the *win* table if needed. Then, all the explored edges leading to s_1 are set to be re-explored because their winning states may have changed due to this update of the winnings of s_1. Line 23 decides the controllability of the problem: if the winning set of the initial state is empty, it means that all the search space has been explored and no strategy has been found.

The following theorem states the correctness of the approach. The proof can be found in appendix A.6.

Theorem 11.2 (Correctness)**.** *With no pruning, the algorithm terminates and returns $\bot$ if and only if the TNU is not dynamically controllable.*

MkStrategy builds a strategy as per definition 56 using a forward search guided by the *dep* and *win* maps: wait conditions are extracted by the projection of the winning states.

From the algorithm search trace we can reconstruct a strategy expressed in the syntax we proposed in definition 56. We perform a forward search

Algorithm 16 Flat Strategy Extraction Algorithm

```
1: procedure MkFlatStrategy($dep$, $win$)
2:     $visited := \emptyset$; $res := \emptyset$
3:     for all $\langle s, trans\rangle \in deps$ do
4:         for all $(s_1 \xrightarrow{x} s_2) \in trans$ do
5:             if $(s_1 \xrightarrow{x} s_2) \notin visited$ then
6:                 $visited := visited \cup \{s_1 \xrightarrow{x} s_2\}$
7:                 if $win[s_1] \neq \emptyset$ and $win[s_2] \neq \emptyset$ then
8:                     $wregion :=$ PreImage($win[s2]$, $x$) $\wedge win[s1]$
9:                     $res[\langle s_1, x\rangle] := res[\langle s_1, x\rangle] \vee wregion$
10:                end if
11:            end if
12:        end for
13:    end for
14:    return $res$
15: end procedure
```

guided by the *dep* map constraining the resulting states to lie withing the winning states represented by the *win* map. Wait conditions are extracted by the projection of the winning states. The pseudo-code of this approach is reported in algorithms 16 and 17.

We first extract a flat strategy (algorithm 16). A flat strategy is a map from pairs of states and controllable time points to time regions. The intuition is that each entry $\langle state, x\rangle \rightarrow \phi$ in the map has the meaning "if you are in *state*, wait until ϕ then execute x". This is analogous to a memory-less strategy for a TGA. The flat strategy is built by re-traversing the explored search-space using the *deps* map, each controllable transition that connects winning states is considered (lines 5-7), and using a pre-image operation, we compute the subset of the starting winning states that allow this transition. With this operation, we know that it suffices to reach a winning state to win the game using this transition (scheduling this time point).

Algorithm 17 Strategy Extraction Algorithm

```
1: procedure MkStrategy($dep$, $win$)
2:     $flat := $ MkFlatStrategy($dep$, $win$)
3:     return BuildTree($flat$, $\langle \emptyset, \top \rangle$)
4: end procedure
5: procedure BuildTree($flat$, $state$)
6:     $\langle P, \phi \rangle := state$
7:     $\psi := \bot$; $\sigma_{\dashv} := \emptyset$
8:     if $\exists b. flat[\langle state, b \rangle] \neq \emptyset$ then
9:         $\psi := flat[\langle state, b \rangle]$
10:        $post := $ PostImage(state,b)
11:        $sub\_strat := $ BuildTree($flat$, $post$)
12:        $\sigma_{\dashv} := (s(b); sub\_strat)$
13:    end if
14:    $wait := \langle P, \omega(\phi, \psi) \rangle$
15:    $waiting := \emptyset$
16:    for all $wait \overset{e}{\dashrightarrow} s_2$ do
17:        $waiting[e] := $ BuildTree($flat$, $s_2$)
18:    end for
19:    if $\sigma_{\dashv} = \emptyset$ and $waiting = \emptyset$ then
20:        if $P = \mathcal{T}$ then                    ▷ All time points scheduled
21:            return noop
22:        else
23:            return WaitRemainingUncontrollables($flat$, $state$)
24:        end if
25:    else
26:        return $w(\psi, e_1 : \sigma_1, \cdots, e_n : \sigma_n, \dashv : \sigma_{\dashv}) \mid \langle e_i, \sigma_i \rangle \in waiting$
27:    end if
28: end procedure
```

The flat strategy generated by algorithm 16 is redundant because it may contain useless transitions that were explored but are not needed in an actual execution from the initial state. Moreover, it is hard to validate and to deploy. Therefore, we translate the strategy in our language (as per definition 56) in algorithm 17. The recursive procedure takes in input the flat strategy and the current state, and returns the dynamic strategy in our syntax for winning starting from that state. The algorithm is divided in three blocks. First (lines 8-13), it checks if any controllable time point is executable from this state (if there are more than one, we simply pick one of them). We recursively compute the winning strategy for that transition: this will be the timeout strategy. Then (lines 15-18), we compute the waiting region and collect all the uncontrollable time points that can happen during this wait. For each of them, we recursively compute a sub-strategy. Finally (lines 19-27), three cases are possible.

1. We scheduled all the time points, hence we can simply return a `noop` strategy.
2. Some uncontrollable time points are missing[3] and we statically build a strategy that waits for all of them.
3. Otherwise, we can combine the computed timeout strategy and condition in a wait command (line 26).

11.4.1 Ordered and Unordered states

So far, we considered the discrete component (the TGA location) of each state as a set P. This is conceptually clean but it might be a drawback. In fact, if two different paths yield to the same subset of time points, two regions of time may be merged in a single state.

[3]No controllable time points can be missing if the flat strategy is correct, the actual code contains a proper sanity check, here we omitted it.

Suppose for example that the search decides to schedule a sequence of time points: A then B then C. Later, by back-propagation of the winning states, this is proven to be insufficient to find a dynamic strategy. The search proceeds by exploring another ordering, say A then C then B, and computing the relative winning states. If we disregard the order of states, it is possible (depending on temporal constraints) that a state $\langle\{A, B, C\}, \phi\rangle$ is explored in both cases, causing its winning region to be updated twice (resulting in the disjunction of the winning of the two visits, because of line 17 of algorithm 15). This is not a correctness issue, but a lot of disjunctions are introduced, and disjunctions heavily impact the performance of time region manipulations.

A direct solution that produces a variant of the algorithm is to consider P as a sorted set, hence distinguishing between different orderings of time points. This increases the theoretical search space size (that become more than factorial in the number of time points: $\sum_{i=1}^{|\mathcal{T}|} i!$) but possibly simplifies the time region management.

This ordered exploration explicitly distinguishes states with the same set of scheduled time points but different scheduling order. In the above example, we would have two final states because $\langle\langle A, B, C\rangle, \phi\rangle$ is considered as a different state from $\langle\langle A, C, B\rangle, \phi\rangle$. Hence, the winnings of $\langle\langle A, B, C\rangle, \phi\rangle$ are not updated twice limiting the amount of disjunctions. This moves some of the complexity from temporal disjunctive reasoning to the discrete search itself.

This variation of the technique uses the very same search algorithm as before, only the state representation is different.

11.4.2 Pruning Unfeasible States

An orthogonal optimization is enabled by the use of an explicit representation (ordered or unordered) of the past time points. In fact, differently from the TGA encoding, we have the possibility of pruning unfeasible paths without manipulating time regions: we can immediately discard any path where the (ordered or unordered) set of the time points is inconsistent with the network temporal constraints.

When the algorithm adds a new, unexplored transition to the *wait* set (lines 3 and 10), the ordering of the time points resulting from the transition may be inconsistent with the free constraints or with the contingent links. For example, in the DTNU of figure 11.1, any path starting with time point b is never going to satisfy the free constraints, hence the expansion $\langle \emptyset, \top \rangle \xrightarrow{b} \langle \{b\}, \bar{b} = 0 \rangle$ can be immediately discarded avoiding a significant portion of the search.

In algorithm 15, the PPush function (short for "pruning push") is demanded to insert a set of elements in the *wait* set, but it may discard unfeasible transitions, working as a filter. This pruning greatly reduces the search space of the algorithm especially in DTNUs with many constraints.

We identified a simple, yet very effective pruning method for PPush, using the consistency checks we discussed in chapter 8.

For each transition $s_1 \overset{x}{\Rightarrow} \langle P_2, \phi_2 \rangle$, we check if, disregarding uncertainty, it is possible for the time points in P_2 to be executed before all the time points in $\mathcal{T} \setminus P_2$. For this reason, we convert the input DTNU in a DTN (without uncertainty) by considering each uncontrollable time point as controllable, and each contingent link as a free constraint[4]. We then add the following to the DTN constraints:

$$\mathcal{C}_{add} \doteq \{y - x \in [0, \infty] \mid x \in P, y \in \mathcal{T} \setminus P\}.$$

If the resulting DTN is consistent, then the transition is added to the *wait* set, otherwise it is discarded. Note that in lines 10 and 18 we do not

[4]This conversion is thoroughly discussed at the beginning of chapter 8.

use PPush because we are not exploring new transitions but re-visiting (already-checked) transitions; hence, the pruning is not needed. This pruning only removes incompatible orderings, hence it maintains soundness and completeness.

In order to perform this check we use our consistency SMT encoding, that also allows for the use of incrementality yielding very good performances.

In addition, if the search is performed using ordered states, the added constraint can be strengthened to represent the order encoded in the state.

$$\begin{aligned}\mathcal{C}_{add} \doteq & \{x_{i+1} - x_i \in [0, \infty] \mid \langle x_1, \cdots x_n \rangle = P, i \in [1, n-1]\} \\ & \cup \; \{y - x_n \in [0, \infty] \mid \langle x_1, \cdots x_n \rangle = P, y \in \mathcal{T} \setminus P\}\end{aligned}$$

Intuitively, we exploit the total order stored within states to build a stronger constraint: we force the order of time-points using $[0, \infty]$ constraints and assert that all the other (not-yet-happened) time points occur after the last time point in the ordering. This constraint is stronger than the previous one (hence, it yields a more effective pruning). Moreover, it is smaller in size: the previous constraint is quadratic in the number of time points, this one is linear.

11.5 Experimental Evaluation

In this section we evaluate the merits of the TGA encoding and the direct synthesis technique on a number of DTNU benchmarks, both considering the immediate-reaction semantics.

We implemented the DTNU-to-TGA using an automated encoder that takes in input a DTNU and outputs a TGA in the input language of the state-of-the-art TGA solver Uppaal-TIGA [BCD$^+$07]. We implemented both the flavors of the DTNU encodings discussed in section 11.3.2. In the following, we refer to the DNF encoding of section 11.3.2 as TIGA-DNF and to the NNF encoding as TIGA-NNF.

We also implemented the validation and synthesis algorithms in a tool called PyDc. The solver is written in the Python programming language and used the PyDBM [Bul12] a Python front-end to the DBM library used by Uppaal-TIGA [BCD$^+$07] for the manipulation of time regions. The pruning technique is implemented using the incremental interface of the MathSAT5 [CGSS13] SMT solver via the PySMT [GM15] Python interface.

We analyzed four versions of the synthesis algorithm: Unordered-NoH, that is the synthesis algorithm with no pruning; Ordered-NoH, that is the synthesis algorithm with no pruning that considers ordered states, Unordered-SMT and Ordered-SMT that use incremental SMT solving for pruning the unfeasible paths.

The benchmark set is the same we used for the weak controllability strategy synthesis experiments in section 10.5. It is composed of 3465 randomly-generated instances (1354 STNU, 2112 DTNU). All the networks are known to be weakly controllable, but only 2354 are dynamically controllable. The benchmarks range from 4 to 50 time points.

The results, obtained with time and memory limits set to 600s and 10GB, are shown in figures 11.11 and 11.12. All the benchmarks as well as the tools can be downloaded as indicated in section 1.2.

We first notice that our synthesis techniques are vastly superior to the TGA-based approaches as shown in the cactus plot of figure 11.11. The direct synthesis algorithm is able to solve 2543 instances, while the best TGA based approach can only solve 799 instances. We observe run-times differences between the two approaches of up to three orders of magnitude, as shown by figure 11.12a.

The cactus plot in figure 11.11, also shows that the SMT-based prun-

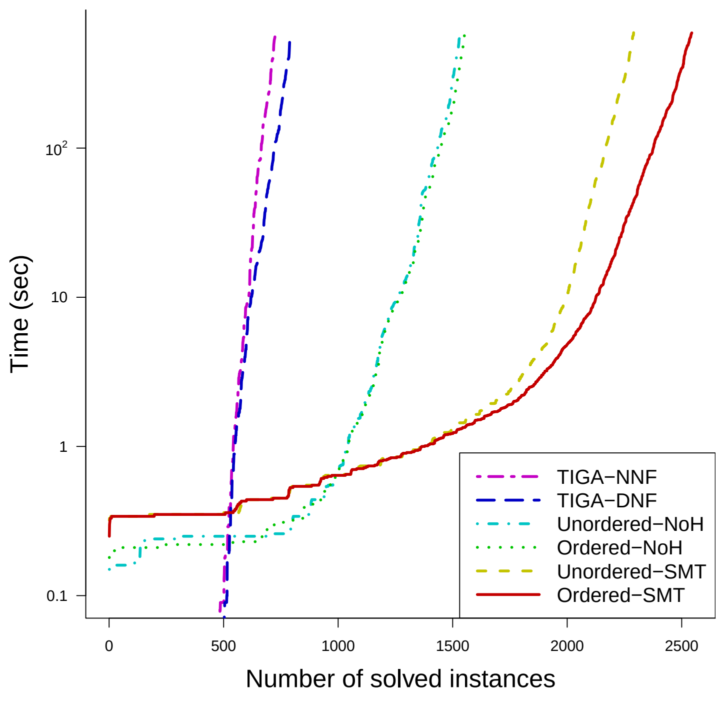
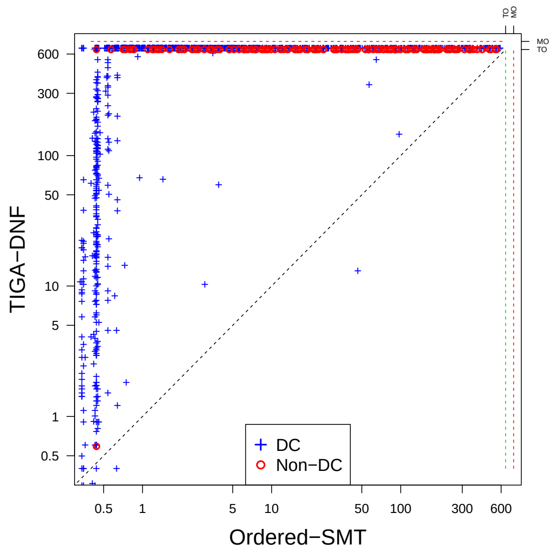


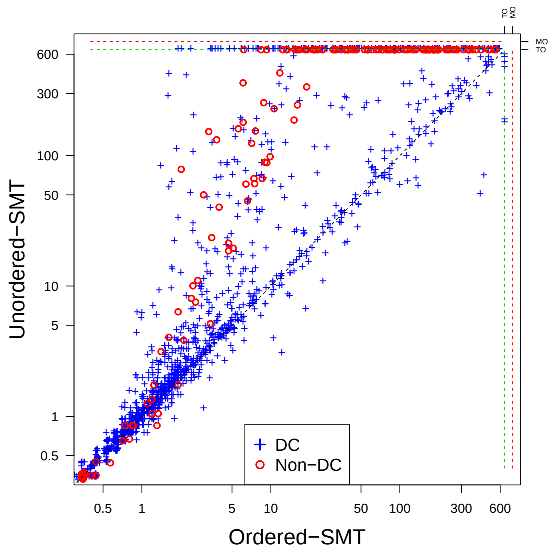
Figure 11.11: Results for the experimental evaluation. The logarithmic scale cactus plot shows the different approaches (with and without SMT pruning) against the theoretical encoding solved by the Uppaal-TIGA TGA solver.

ing yields a significant performance boost, both for the unordered case (from 1531 to 2289 solved instances), and for the ordered case (from 1552 to 2543). Nonetheless we observe how the pruning is much more effective for the ordered case, thanks to the additional strength of the pruning constraints.

Finally, the scatter plot in figure 11.12b shows that the ordered case is almost always superior to the unordered one when the SMT pruning is enabled. Further inspection shows that Unordered-SMT explores an average of 2447.9 symbolic states, compared to the 95.8 of Ordered-SMT. In the latter case, the ordering information allows the SMT solver to detect unfeasible branches much earlier than in the unordered case.



(a)



(b)

Figure 11.12: Results for the experimental evaluation. The scatters compare the ordered SMT approach with the theoretical encoding (a) and with the unordered solver (b).

Disjunctive Scheduling under Temporal Uncertainty Conclusions

In this part of the thesis, we presented several new techniques to solve a range of problem for Disjunctive Temporal Networks with Uncertainty.

Starting by encoding the consistency problems as SMT queries, we addressed the strong controllability problem by providing an effective reduction as a quantified SMT problem. For some DTNU sub-classes we also show ways to avoid the cost of quantifier elimination fully exploiting the SMT efficiency on quantifier-free formulae.

Considering the weak controllability problem, we discussed how to reduce the decision problem to an SMT query, analogously to what we did for strong controllability and we also developed a number of approaches to automatically synthesize executable weak strategies in closed form.

Finally, we solved the open problem of dynamic controllability for DTNU by designing a reduction into a TGA reachability game that can be solved by existing techniques. Moreover we pushed the efficiency of this approach by developing an algorithm that follows the TGA reduction, but optimizes it in many ways. Also in this case we resorted to the expressiveness and efficiency of SMT solvers to achieve the effectiveness on the temporal problem ah hand.

Future directions. This work can be extended in several directions.

First, according to the classification table (table 3.2) we presented in chapter 3, we only explored the "deterministic" row. A natural extension of this work is to adapt the techniques to work also for the non-deterministic case. With this respect, we started working [CHM$^+$16] by combining the disjunctions of the DTNU framework with conditionals in the CTN formalism, obtaining the Conditional Disjunctive Temporal Problem with Uncertainty (CDTNU). We extended the DTNU-to-TGA approach for solving the dynamic controllability to this case using the timestrict semantics.

Another direction that may be pursued is optimization. All the approaches we presented are concerned in finding a solution for the controllability problems, in the from of a consistent schedule, a strong schedule or a strategy: no distinction is imposed between two solutions. An important direction to explore is the optimization of solutions in order to find solutions that maximize some desired property. This could be very useful in practice: for example, one may look for the strategy that minimizes the time-span while respecting all the constraints, reducing the operational costs.

Finally, since all the problem we presented are at least NP-hard in complexity there is the never-ending quest for efficiency. We tried to exploit as much as possible the SMT solving framework to take advantage of the continuous improvement in heuristics and techniques in this field. The presented techniques would immediately benefit from any major advancement in SMT.

Part III

Strong Temporal Planning with Uncontrollable Durations

Introduction

In this part, we move from the realm of scheduling to the one of planning. As discussed in chapter 5, the landscape of planning techniques is narrower than the scheduling one. We discuss planning under temporal uncertainty, focusing on strong temporal planning with uncontrollable durations.

Planning in real world domains often involves modeling and reasoning about the duration of actions. For example, the activities of an exploratory rover, such as navigation or data transmission, require non-negligible time to be completed. Moreover, their successful execution is usually subject to timing constraints. For example, a transmission action must match precise time windows of availability for the communication with an orbiting satellite. In some practical applications, however, the duration of actions may be beyond the decision capabilities of the planner. For example, a navigation task to a given location may take more or less time, depending on external factors, such as bad weather conditions. Any plan to solve the problem, may specify when to start an action, but the actual duration is up to the environment. This is similar to the TNU framework we discussed in the previous part, but here we have a formal model of the system and a goal to achieve instead of a set of pre-specified time points to schedule.

In this context, we first focus on action-based languages (such as PDDL or ANML) and we introduce the problem of Strong Temporal Planning with Uncontrollable Durations (STPUD) corresponding to the "Duration-only Plant, Time-Strong Executor" cell of table 3.2. This is the planning problem where no discrete uncertainty is present, but the action durations may be uncontrollable. In particular, we develop a formal language to express the problem and define its semantics, then we proceed by introducing a portfolio of planning techniques to solve the problem. We devise a direct technique to solve a sub-class of the general problem by extending a state-of-the-art approach for temporal planning without uncertainty, then we tackle the general problem showing a sound and complete compilation technique that reduces any planning problem with uncontrollable duration to a plain temporal planning problem preserving the strong plans.

Second, we restate the problem in the context of timeline planning. We give a formalization of the problem also in this context and we present a theoretical first-order encoding of the problem with bounded horizon. This encoding is mainly theoretical, but can be solved by an SMT solver.

Structure of this part. This part is structured as follows. In chapter 12, we discuss action based languages and the STPUD problem, while in chapter 13 we move to the realm of timeline planning. We conclude the part in section 13.3, by summarizing the contributions and the future works.

Chapter 12

Action-Based STPUD

In this chapter, we introduce the problem of *Strong Temporal Planning with Uncontrollable Durations* (STPUD) for a general action-based planning language. In this setting, the planner is only allowed to choose the start of the actions, while their duration is chosen at run time, within known bounds, by the environment. A solution plan is required to be temporally strong, i.e. robust with respect to the uncontrollable action duration, and to achieve the goal on all possible executions, despite the run-time choices of the environment.

We explore two complementary approaches to solve STPUD. First, we discuss a dedicated planning method, that generalizes the forward state-space temporal planning (FSSTP, introduced in section 5.1.3) framework in to deal with uncontrollable durations. Intuitively, FSSTP applies classical planning over an abstraction of the temporal domain, where temporal precedences over events are taken into account at a qualitative level, to enumerate candidate plans. The quantitative aspects are then taken into account by solving the induced Simple Temporal Problem (STN) [DMP91b]. In order to deal with temporal uncertainty, we retain the main loop of the FSSTP approach, dealing with the quantitative aspects by means of Temporal Networks with Uncertainty (TNUs). We exploit the techniques for strong controllability we presented in chapter 9. We remark that this approach is far from trivial. In fact, we show how simply replacing the STN in with the corresponding STNU may result in an unsound technique.

Second, we present a radically different, compilation-based planning method, that reduces any STPUD problem to a "classical" temporal planning problem, where actions have controllable durations. The reduction eliminates uncontrollable durations by introducing *intermediate effects.* Because many available planners (all the PDDL 2.1 planners, in fact) do not support these, we propose an effective way to compile away intermediate effects.

We also investigate a domain transformation technique that is able to eliminate some of the uncontrollable durations by reasoning in terms of worst case execution. This technique preserves the space of plans and can be combined as a simplifying pre-processor both to the translation-based planner and to the direct planner.

The approaches described above have been implemented and experimentally analyzed. We considered a large number of instances obtained by extending with uncontrollable durations the temporal planning domains available in the literature [CCO$^+$12]. Our results demonstrate the practical applicability of our approaches, and provide interesting insights. First, the planning approaches often exhibit complementary behaviors. Thus, further efficiency can be achieved by combining them in a portfolio approach. Second, the simplification technique has negligible costs, but it greatly pays off, in selected cases, for both planning approaches. Finally, the proposed encoding for the elimination of intermediate effects yields good performance.

12.1 The STPUD Problem

In this section, we formally define the syntax and semantics of the Strong Temporal Planning with Uncontrollable Durations (STPUD) and we discuss the issues that arise in comparison to temporal planning without uncertainty.

12.1.1 Syntax

We propose a rich language for temporal planning with duration uncertainty that includes timed-initial-literals, and multi-valued variables. In addition, we extend the language to allow conditions expressed over sub-intervals of actions, and effects at arbitrary time points during an action. These features turn out to be particularly useful for encoding many problems of interest, and for encoding our translation[1].

In order to define the abstract syntax of our language we need to consider four kinds of intervals, namely closed, left-open, right-open and open. We will use these intervals to express durative conditions.

Definition 57. *Given two numeric expressions a and b, we define the four possible intervals having extremes a (lower bound) and b (upper bound) as:*

- $[a, b]$ *closed interval*
- $(a, b]$ *left-open interval*
- $[a, b)$ *right-open interval*
- (a, b) *open interval*

We write $\mathbb{I}(a, b)$ to indicate an instance of the above possibilities without distinguishing its type.

[1]To simplify the presentation, we exclude some features that are orthogonal to our approach of handling temporal uncertainty, such as *numeric variables* and *domain axioms*. Our techniques will work whether or not those features are included.

We can now define a STPUD planning problem P as a tuple $\langle V, I, T, A, G\rangle$ where:

- $V = \{f_1, \cdots f_n\}$ is a finite set of variables, each having a domain $Dom(f_i)$.
- I is the initial state: a complete assignment of values to each variable in V: for each variable $f \in V$, $I(f) \in Dom(f)$.
- T is a set of timed-initial-literals, each of the form $\langle [t]\ f := v\rangle$ with $f \in V$, $v \in Dom(f)$ and $t \in \mathbb{R}^+$. The real number t is the wall-clock time at which f will be assigned the value v.
- A is a set of durative actions each of the form $a \doteq \langle [l, u], C, E\rangle$ where:
  - $l, u \in \mathbb{R}^+$, with $l \leq u$ being the action duration bounds. Let st_a and et_a be the start and end times of action a, then the duration of action a is an element of $[l, u]$. We write $bounds(a)$ to indicate the interval $[l, u]$.
  - C is the set of conditions; each element $c \in C$ is of the form $\langle \mathbb{I}(st_c, et_c)\ \bigvee_{i=1}^{n} f_i = v_i\rangle$ where each $f_i \in V$ and $v_i \in Dom(f_i)$. The expressions st_c and et_c indicate the start and end time points of the condition c and are restricted to be equal to $st_a + \delta$ or $et_a - \delta$ with $0 \leq \delta \leq u$.
  - E is a set of instantaneous effects, each $e \in E$ is of the form $\langle [t_e]\ f := v\rangle$ where $f \in V$, $v \in Dom(f)$ and $t_e \doteq st_a + \delta$ or $t_e \doteq et_a - \delta$ with $0 \leq \delta \leq u$.
- G is the set of disjunctive goal conditions, each of the form $\bigvee_{i=1}^{n} f_i = v_i$, where each $f_i \in V$ and $v_i \in Dom(f_i)$.

Moreover, we assume that the set of actions A is partitioned in two sets A_c and A_u of controllable and uncontrollable actions. This is needed to distinguish between actions that can be terminated by the agent and those that have uncontrollable durations. In both cases, the bounds on the duration need to be satisfied, but analogously to TNU contingent links, if an action is uncontrollable, its duration is assumed to take a value in the specified bound.

The solution to a STPUD problem P is a plan π that assigns the starting of all the actions and specifies the duration only for controllable actions.

Definition 58. *A plan π of P is a finite set of tuples $\langle t, a, d \rangle$, in which actions $a \in A$, $t \in \mathbb{R}$, $d \in \mathbb{R}$ if $a \in A_c$ and $d = \bot$ if $a \in A_u$.*

Intuitively, an element $\langle t, a, d \rangle$ prescribes to start an action a at time t, with duration d if a is a controllable action. This reflects the intuitive notion of action uncontrollable duration: since the duration is not under the control of the plan executor, the resulting plan cannot specify it.

12.1.2 Semantics

We give the semantics of the planning language by defining the validity of a plan π for any given STPUD problem P. As usual, P admits a solution if there exists a valid plan, otherwise the problem is said to be unsolvable.

We start by defining the projection of a STPUD. Intuitively, in a projected problem, we consider each action to be controllable and all the other constraints are kept as in the original problem.

Definition 59 (Projected Problem)**.** *Given a STPUD problem P with durative actions $A \doteq A_c \cup A_u$, the projected problem without uncertainty is a STPUD ctrl(P) that is identical to P except for the set of actions that is $A' \doteq A'_c \cup A'_u$ with $A'_c \doteq A_c \cup A_u$ and $A'_u \doteq \emptyset$.*

The basic element of our semantics is a chronicle, that is used to assign a value to each variable in V for each time instant $x \geq 0 \in \mathbb{R}$.

Definition 60 (Chronicle)**.** *A chronicle τ for a STPUD problem instance $P \doteq \langle V, I, T, A, G \rangle$ is a set of functions $\tau_f : \mathbb{R}^+ \rightarrow Dom(f)$, one for each $f \in V$.*

Given a plan, we can now define the chronicle induced by it. In fact, in our language we have three components that contribute to change the state of a variable, namely the initial state, the TILs and action effects. Apart for these events, each variable is assumed to maintain its value in the other time instants. To formalize this concept we start by collecting the set of change events in the execution of the plan.

Definition 61 (Set of Changes CH)**.** *Given a projected planning problem $ctrl(P) \doteq \langle V, I, T, A, G \rangle$ and a plan $\pi \doteq \{\langle t_i, a_i, d_i \rangle \mid i \in [1, n]\}$, the set of changes induced by π is a set $CH(ctrl(P), \pi)$ defined as follows.*

- *for each $f \in V$, $\langle 0, f, I(f) \rangle \in CH(ctrl(P), \pi)$*
- *for each $\langle [t]\, f := v \rangle \in T$, $\langle t, f, v \rangle \in CH(ctrl(P), \pi)$*
- *for each $\langle t, a, d \rangle \in \pi$ with $a \doteq \langle [l, u], C, E \rangle$, for each $\langle [st_a + \delta]\, f := v \rangle \in E$, $\langle t + \delta, f, v \rangle) \in CH(ctrl(P), \pi)$ and for each $\langle [et_a - \delta]\, f := v \rangle \in E$, $\langle t - \delta, f, v \rangle) \in CH(ctrl(P), \pi)$.*

Now, we can define the chronicle induced by a plan by imposing that at each change point the chronicle changes its value and between any two changes, the chronicle maintains the "older" value.

Definition 62 (Induced Chronicle)**.** *Given a projected planning problem $ctrl(P) \doteq \langle V, I, T, A, G \rangle$ and a plan $\pi \doteq \{\langle t_i, a_i, d_i \rangle \mid i \in [1, n]\}$, the chronicle τ^π induced by π is defined as follows.*

For each $x \geq 0 \in \mathbb{R}$ and each $f \in V$, $\tau^\pi_f(x) = v$ with $\langle \hat{t}, f, v \rangle \in CH(ctrl(P), \pi)$, $\hat{t} \doteq x + min(\{t - x \mid \langle t, f, v \rangle \in CH(ctrl(P), \pi), t > x\})$.

Note the strict inequality in the definition: we impose the value of an effect immediately after the change is scheduled to happen. For example, consider an effect $\langle [et_a]\ f := v\rangle$ and suppose the action a terminates at time 10. The value of f is not changed at time 10 but immediately after. This means, that a condition requiring f to have value v is not satisfied a ti time 10, but a positive amount of time is required to pass. This view is compatible with the PDDL 2.1 specification (even though PDDL requires a known minimal time quantum ϵ to pass) and also with the continuous time version of the ANML language.

Given the induced chronicle, we can now define the validity of a plan for a projected problem: it suffices to check that no pair of (different) changes are applied in the same instant, that the durations specified in the plan for each action are valid with respect to the action specification, that each action condition holds in any time instant contained in the specified intervals, and, finally, that each goal is also satisfied.

Definition 63 (Projected Problem Plan Validity). *Given a projected problem $ctrl(P) \doteq \langle V, I, T, A, G\rangle$, a plan $\pi \doteq \{\langle t_i, a_i, d_i\rangle \mid i \in [1, n]\}$ is valid, if the following conditions hold:*

1. *for each $t \in \mathbb{R}$ and each $f \in V$,*
   $|\{\langle t, f, v\rangle \mid \langle t, f, v\rangle \in CH(ctrl(P), \pi)\}| \leq 1$;

2. *for each $\langle t, a, d\rangle \in \pi$ with $a \doteq \langle [l, u], C, E\rangle$, the following holds.*

   - $d \in [l, u]$;
   - *for each $\langle [st_a + \delta_s, st_a + \delta_e]\ \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}$.* $((t + \delta_s \leq x \leq t + \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
   - *for each $\langle (st_a + \delta_s, st_a + \delta_e]\ \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}$.* $((t + \delta_s < x \leq t + \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*

- *for each* $\langle [st_a + \delta_s, st_a + \delta_e) \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + \delta_s \leq x < t + \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle (st_a + \delta_s, st_a + \delta_e) \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + \delta_s < x < t + \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle [st_a + \delta_s, et_a - \delta_e] \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + \delta_s \leq x \leq t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle (st_a + \delta_s, et_a - \delta_e] \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + \delta_s < x \leq t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle [st_a + \delta_s, et_a - \delta_e) \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + \delta_s \leq x < t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle (st_a + \delta_s, et_a - \delta_e) \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + \delta_s < x < t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle [et_a - \delta_s, et_a - \delta_e] \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + d - \delta_s \leq x \leq t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle (et_a - \delta_s, et_a - \delta_e] \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + d - \delta_s < x \leq t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle [et_a - \delta_s, et_a - \delta_e) \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + d - \delta_s \leq x < t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*
- *for each* $\langle (et_a - \delta_s, et_a - \delta_e) \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, $\forall x \in \mathbb{R}.$ $((t + d - \delta_s < x < t + d - \delta_e) \rightarrow \bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i)$ *holds;*

3. *for each* $(\bigvee_{i=1}^{n} f_i = v_i) \in G$, $\bigvee_{i=1}^{n} \tau_{f_i}^{\pi}(x) = v_i$ *holds, for any* $x \in [t_{max}, t_{max} + \epsilon]$ *with* $t_{max} \doteq max(\{t + d \mid \langle t, a, d\rangle \in \pi\})$ *and a sufficiently small* $\epsilon \in \mathbb{R}$.

Intuitively, condition 1 imposes that there are no two changes on the same variable at the same time (preventing race conditions in the semantics); condition 2 checks that all the conditions of all the actions used in the plan are satisfied; and condition 3 requires that all the goals are reached immediately after the end of the last action in the plan.

Finally, we can define the semantics of any STPUD problem P by imposing that each plan obtained by specifying a valid duration for each uncontrollable action is valid for the projected problem of P. This captures the intuitive notion of strong plan: regardless of the actual duration of each uncontrollable action specified in the plan, the execution is valid and all the goals are satisfied.

Definition 64 (STPUD Plan Validity)**.** *Given a STPUD problem P, a plan $\pi \doteq \{\langle t_i, a_i, d_i\rangle \mid i \in [1, n]\}$ is valid, if all the plans $\pi' \in \{\langle t, a, d\rangle \mid \langle t, a, d\rangle \in \pi, a \in A_c\} \cup \{\langle t, a, t{+}k \mid \langle t, a, d\rangle \in \pi, a \in A_u, a \doteq \langle [l, u], C, E\rangle, k \in [l, u]\rangle\}$ are valid for $ctrl(P)$.*

In the following, we call "strong plan" a valid plan for a given STPUD.

12.1.3 Example

We give a small example problem that will be used throughout the chapter.

A rover, initially at location l_1, needs to transmit some science data from location l_2 to an orbiter that is only visible in the time window $[14, 30]$. The rover can move from l_1 to l_2 using an action *move* that has uncontrollable duration between 10 and 15 time units. The data transmission action *transmit* (abbreviated *trans*) takes between 5 and 8 time units to complete. The goal of the rover is to transmit the data to the orbiter. Because of the harsh daytime temperatures at location l_2 the rover cannot arrive at l_2 until the sun goes behind the mountains at time 15. Figure 12.1 illustrates this scenario, which we encode as:

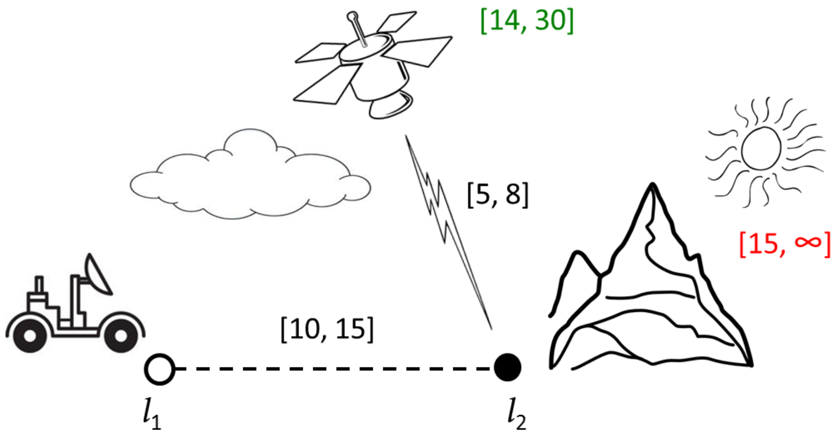


Figure 12.1: A graphical representation of the running example situation. The rover, initially in l_1 can move to l_2 where it can transmit data to a satellite. Action durations are indicated in black, the satellite visibility window is in green and the interval where the temperature in l_2 is favorable is indicated in red.

$$
\begin{aligned}
V &\doteq \{pos : \{l_1, l_2\}, visible : \{\mathtt{T}, \mathtt{F}\}, hot : \{\mathtt{T}, \mathtt{F}\}, sent : \{\mathtt{T}, \mathtt{F}\}\} \\
I &\doteq \{pos = l_1, visible = \mathtt{F}, sent = \mathtt{F}, hot = \mathtt{T}\} \\
T &\doteq \{\langle [14]\ visible := \mathtt{T}\rangle, \langle [30]\ visible := \mathtt{F}\rangle, \langle [15]\ hot := \mathtt{F}\rangle\} \\
G &\doteq \{(sent = \mathtt{T})\} \\
A_c &\doteq \emptyset \\
A_u &\doteq \{\langle [10, 15], C_{move}, E_{move}\rangle, \langle [5, 8], C_{trans}, E_{trans}\rangle\} \\
C_{move} &\doteq \{\langle [st_{move}, st_{move}]\ pos = l_1\rangle, \langle [et_{move}, et_{move}]\ hot = \mathtt{F}\rangle\} \\
C_{trans} &\doteq \{\langle [st_{trans}, et_{trans}]\ pos = l_2\rangle, \langle [st_{trans}, et_{trans}]\ visible = \mathtt{T}\rangle\} \\
E_{move} &\doteq \{\langle [et_{move}]\ pos := l_2\rangle\} \\
E_{trans} &\doteq \{\langle [et_{trans}]\ sent := \mathtt{T}\rangle\}
\end{aligned}
$$

Figure 12.2 graphically shows a valid strong plan:

$$\pi_{ex} \doteq \{\langle 6, move, \bot\rangle, \langle 22, trans, \bot\rangle\}$$

Note that all the actions in π_{ex} have uncontrollable duration; hence, the strong plan does not specify their duration.

12.1.4 Discussion

In general, finding a strong plan for a problem with uncontrollable durations is different from simply considering the maximum or the minimum duration for each action. Consider our rover example and its strong plan shown in figure 12.2. The *move* action must terminate before the transmit action can start and, at the same time, *move* cannot terminate before time 15 due to the temperature constraint. If we only consider the lower-bound on the duration of *move* (i.e., planning with a fixed duration of 10 for *move*) then one valid plan is: $\pi_{lb} \doteq \{\langle 11, move\rangle, \langle 22, trans\rangle\}$. However, because of the uncertainty in the actual execution duration of *move*, it may actually take 14 time units to arrive at l_2. Thus, the rover would start transmitting at time 22 before it actually arrives at l_2 at time $11 + 14 = 25$. Thus, plan π_{lb} is invalid. Similarly, if we consider only the maximal duration (i.e., planning with a fixed duration of 15), then one possible plan would be: $\pi_{ub} \doteq \{\langle 1, move\rangle, \langle 22, trans\rangle\}$. However, during the actual execution of *move*, it may again take only 11 time units (and not the planned maximum of 15 time units) to arrive at l_2. This would violate the constraint that the rover should arrive at l_2 after $t = 15$ to avoid the sun, so π_{ub} is also not a valid plan. In section 12.5, we present a simplification technique aimed at identifying the cases in which it is possible to soundly consider only the maximal (or minimal) duration for an action. However, we remark that this special-case optimization is not applicable in the general case: this chapter is devoted to the development of dedicated techniques for solving this problem.

Disjunctive Conditions. In contrast to ordinary temporal planning, given an STPUD it is not possible to compile away disjunctive conditions using the action duplication technique [GK97]. This is because the set of satisfied disjuncts in the presence of uncertainty can depend on the contingent

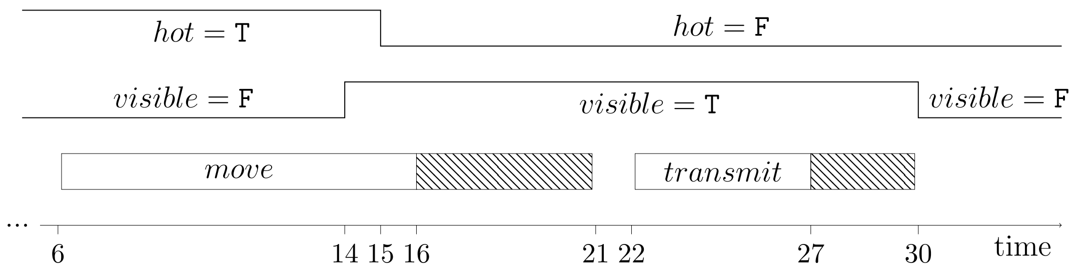


Figure 12.2: Graphical execution of π_{ex}. Striped regions represent the uncertainty in the action duration.

execution. For example, consider an action a starting at time t where two Boolean variables p_1 and p_2 are $\texttt{F}$. The action a has uncontrollable duration in $[l, u]$, a starting effect $e_1 \doteq \langle [st_a]\, p_1 := \texttt{T}\rangle$ and two ending effects $e_2 \doteq \langle [et_a]\, p_1 := \texttt{F}\rangle$ and $e_3 \doteq \langle [et_a]\, p_2 := \texttt{T}\rangle$. An at-start condition $p_1 \vee p_2$ of another action b is satisfied anywhere between the start of the action a and the next deletion of p_2. Thus, b can start anytime within $d \doteq (t + l, t + u]$. However, if we compile away this disjunctive condition by replacing b with two actions b_1 and b_2: one with an at-start condition p_1 and the other with an at-start condition p_2, then b_1 is not executable within d because there is no time point in d in which we can guarantee that $p_1 = \texttt{T}$ (because a may take the minimum duration l and thus the at-end effect e_2 will occur at $t + l$ to set $p_1 = \texttt{F}$). Similarly, we cannot start b_2 within d because we also cannot guarantee that $p_2 = \texttt{T}$ at anytime point within d (because a may take the maximum duration u and thus e_3 that set $p_2 = \texttt{T}$ will not happen until $t + u$). Thus, compiling away disjunctive conditions leads to incompleteness when there are uncontrollable durative actions. For this reason it is important to explicitly model disjunctive conditions in our language.

12.2 Overview of the Proposed Approaches

The planning problem formalism we described includes several features that guarantee a significant expressiveness. In particular, we focus on two

		Action Duration	
		Controllable Only	**Uncontrollable**
Condition/Effect Timing	**Extremes + Overall**	CTRL-EXTR (PDDL 2.1)	UNC-EXTR
	Arbitrary Intervals	CTRL-ARBIT (ANML)	UNC-ARBIT

Table 12.1: Classification of relevant problem sub-classes. For each case, we indicated the abbreviation name used in this thesis and a planning language representative in written in parenthesis where available.

features, namely the presence of uncertainty in the action durations, that constitutes the main objective of this work, and the presence of "intermediate" effects and conditions, that is the possibility of having action effects and to impose conditions in sub-intervals of the action execution. This latter feature is not new, languages such as ANML support it natively, but it is not natively supported in other languages. For example, PDDL 2.1 does not allow for intermediate effects nor for conditions at times different from the start the end or the whole action duration.

If we classify according to the presence or absence of these features, we obtain the landscape of planning problem sub-classes depicted in table 12.1. The table shows four classes of problems. Clearly, every class with arbitrary intervals subsumes the class with extreme intervals having the same action controllability. Similarly, each class having action uncertainty is strictly more general than the class without it. Standing these subsumption rules, the only two incomparable classes are UNC-EXTR and CTRL-ARBIT: both subsume CTRL-EXTR, but no obvious relation is present between the two. UNC-ARBIT is the most general class that subsumes every other case, moreover it coincides with the language we discussed in the previous section. The aim of this work is to define techniques to address problems in this class.

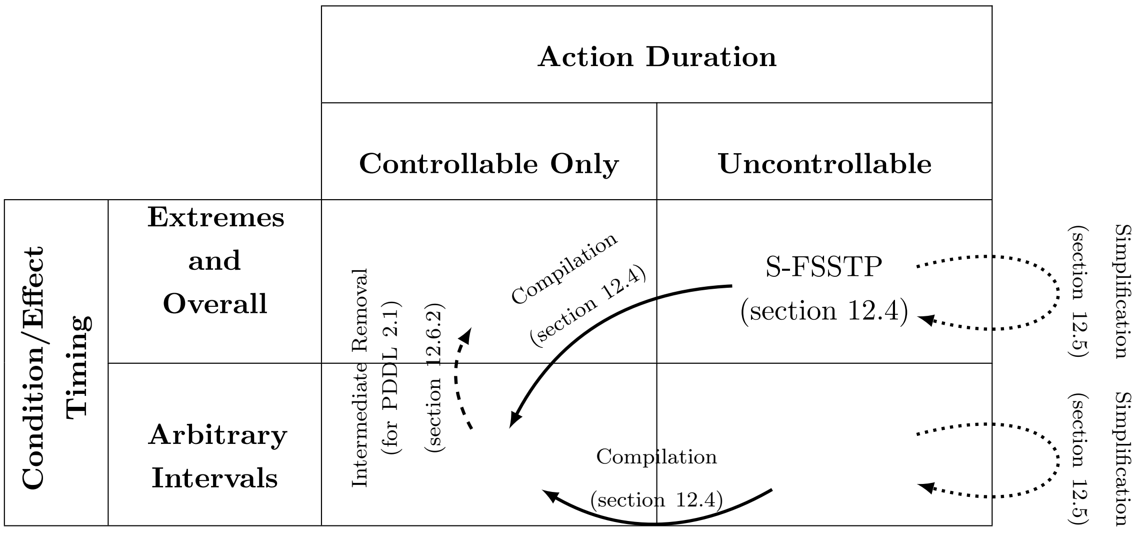
Some of the reported problem classes are supported in the literature by dedicated planning tools. In particular, Ctrl-Extr is the temporal planning problem addressed by all the planners supporting the PDDL 2.1 language. Ctrl-Arbit is a sub-case of the features provided by the ANML language, hence tools such as FAPE [DBMIG14] can natively reason on instances of this class. Moreover, reduction techniques have been presented for transforming an instance of the Ctrl-Arbit class to a problem in Ctrl-Extr [FLH04].

Standing this classification, we now give an outline of the techniques included in this chapter that have been developed to address the most general Unc-Arbit class.

First, we focus on the Unc-Extr class and in section 12.3 we present a dedicated solving technique that extends the Forward State-Space Temporal Planning (FSSTP) framework for dealing with uncontrollable durations in the context of STPUD. We call this technique S-FSSTP (for Strong FSSTP). We propose different variants of the technique one of which is proven to be sound and complete.

Second, devise a compilation technique that transforms any instance of the Unc-Arbit class into an instance of the Ctrl-Arbit class, effectively removing the temporal uncertainty from the problem. The compilation is such that any plan on the controllable instance admits a plan if and only if the original Unc-Arbit planning problem has a valid strong plan. This compilation, discussed in section 12.4, makes use of arbitrary-time conditions and effects; hence, even if applied on a Unc-Extr instance, it produces an equivalent Ctrl-Arbit instance.

We also present a simplification technique that is able to reduce (in some cases, to remove) the temporal uncertainty in a planning instance. This technique does not make use of intermediate effects nor conditions,



		Action Duration	
		Controllable Only	**Uncontrollable**
Condition/Effect Timing	**Extremes and Overall**		S-FSSTP (section 12.4)
	Arbitrary Intervals		

Intermediate Removal (for PDDL 2.1) (section 12.6.2)
Compilation (section 12.4)
Compilation (section 12.4)
Simplification (section 12.5)
Simplification (section 12.5)

Table 12.2: Overview of the proposed techniques in the Strong Temporal Planning with Uncontrollable Durations problem sub-classes landscape. Arrows stand for compilation techniques, S-FSSTP indicates a dedicated technique for its cell. For each technique, the section where it is discussed is reported.

hence it can be applied on both the Unc-Arbit and Unc-Extr classes without ending up in a different problem class.

Finally, in the experimental evaluation section, we recall existing techniques for removing intermediate conditions and effects in the context of PDDL 2.1 (section 12.6.2). We discuss and extend existing techniques to obtain an efficient compilation for the removal of intermediate effects and conditions when no actions having uncontrollable duration are present.

Table 12.2 gives an overview of the aforementioned techniques and reduction in the problem classes landscape.

12.3 STPUD via Forward State-Space Search

In this section, we focus on the Unc-Extr problem class and we propose a dedicated approach for solving STPUD, based on an extension of the Forward State-Space Temporal Planning (FSSTP) framework. We first recall and formalize the FSSTP temporal planning approach originally introduced in chapter 5, then we generalize it to handle temporal uncertainty. The resulting technique, that handles the UNC-EXTR problem class, is referred to as S-FSSTP .

12.3.1 FSSTP

The idea behind the FSSTP approach is to create an interplay between a state-based forward search planner to generate a plan sketch and a temporal reasoner to check its temporal feasibility [CFH+09, CCFL12]. Intuitively, a temporal plan is a sequence of events in time, each with an associated time-stamp. FSSTP builds a planning problem to generate sequences of events (encoded as classical planning actions) that are sound from the propositional point of view, but might violate some temporal constraint. For this reason, a scheduler is employed to check temporal feasibility and to associate a time-stamp to each event. If the scheduling succeeds a valid plan is constructed, otherwise the sequence of events is refused an another one has to be found.

The original description of FSSTP is given within the PDDL 2.1 language, that only allows effects at the start (`at start`) and at the end (`at end`) of an action; instantaneous conditions are allowed to be specified at the start or at the end of the action, while durative conditions are possible only on the whole interval (st_a, et_a) (`over all`).

In FSSTP, each durative action a is expanded in a pair of classical planning actions called *snap actions*: a_{st_a} encoding the starting event of a, and a_{et_a} corresponding to the ending of a. The action a_{st_a} has the starting conditions and effects of a as preconditions and effects, and, similarly, a_{et_a} has the ending conditions and effects of a as preconditions and effects. We force the planner to instantiate snap actions in pairs (each start is coupled with exactly one end) and forbid any action threatening the overall condition of a between the snap actions for A [CFH+09].

Algorithm 18 The FSSTP Framework

```
 1: procedure FSSTP($\mathcal{P}$)
 2:     for all partial $\chi$ generated while solving $abs(\mathcal{P})$ do
 3:         $D := \textsc{Durations}(\chi, \mathcal{P})$
 4:         $P := \textsc{Precedences}(\chi, \mathcal{P})$
 5:         if $\mu := \text{TN}.\textsc{Solve}((\chi, D \cup P))$ then
 6:             if $\textsc{IsComplete}(\chi)$ then
 7:                 return $\textsc{BuildTemporalPlan}(\mu, \chi)$
 8:             else
 9:                 $\textsc{continue}(\ )$
10:             end if
11:         else
12:             $\textsc{reject}(\chi)$
13:         end if
14:     end for
15:     return $\bot$
16: end procedure
```

Similarly, also timed initial literals are treated as instantaneous actions that can be instantiated without preconditions and have the effect of the TIL. For a TIL t, we indicate with $TA(t)$ such instantaneous action.

Putting all together, we can now define the Domain Abstraction as a classical planning problem derived from the PDDL 2.1 temporal planning instance.

Definition 65 (Domain Abstraction)**.** *Given a temporal planning problem $\mathcal{P} = \langle V, I, G, T, A \rangle$ restricted to the* Ctrl-Extr *problem class, the abstraction of $\mathcal{P}$ (written abs($\mathcal{P}$)) is a classical planning problem defined as $\langle V, I, G, \bigcup_{a \in A}\{a_{st_a}, a_{et_a}\} \cup \{TA(t) \mid t \in T\}\rangle$.*

Given the abstract domain, we can enumerate the “discrete” plans that are solution of the abstract problem. The overall idea of the FSSTP framework is to plug a scheduling phase on top of such a plan enumeration, to reject temporally-inconsistent plans.

The pseudo-code of a forward state-space temporal planner (FSSTP) is shown in algorithm 18. A classical planner implemented as a forward state-space search solves the abstract problem $abs(\mathcal{P})$. The planner keeps a totally-ordered partial plan χ composed of abstract actions we call steps. We indicate a step corresponding to an instance of action a as s^a. Each time an action is added to the partial plan, the scheduling check is invoked to assess the temporal consistency of the added action. The scheduling check builds a TN[2] that has the steps of χ as time points and has a set of constraints composed of duration constraints D and precedence constraints P. Duration constraints (created by the DURATIONS function) are used to bind pairs of snap actions instances $(s_i^{a_{sta}}, s_j^{a_{eta}})$, forcing the duration of each action to obey the domain specification $(a_{et_a} - a_{st_a} \in bounds(a))$. In addition, each TIL $t \doteq \langle [k] f := v \rangle$ is forced to happen at the predefined time by imposing a temporal constraint[3] $TA(t) = k$. Precedence constraints (created by the PRECEDENCES function) are used to maintain causality in the plan. If a step s_i is needed to achieve a precondition for another step s_j, we must impose a precedence among the two steps $(s_j - s_i > 0)$, in order to inform the scheduler of such causal constraint[4]. Similarly, precedence constraints are used to impose that the $overall(a)$ conditions for an action a are maintained.

When the TN is found to be consistent, two situations can occur. If χ was a plan achieving the goal in $abs(\mathcal{P})$, each $s_i^{a_{sta}}$ is followed by a corresponding $s_j^{a_{eta}}$ and all TILs appear in the plan (ISCOMPLETE returns true), we can terminate the procedure, otherwise we continue the search in the abstract domain (CONTINUE). To terminate, we build a temporal

[2]In this work, we only consider purely temporal planning. Other works cope with numeric fluents and continuous effects by using linear programs instead of temporal problems [CCFL12].

[3]In practice, we introduce a reference time point z marking the beginning of time and we impose a proper TN constraint $TA(t) - z \in [k, k]$

[4]In the PDDL 2.1 semantics we must impose the constraint $b - a \geq \epsilon$.

plan π from a consistent schedule μ of the TN[5]: each pair of snap actions steps $s_i^{a_{sta}}$, $s_j^{a_{eta}}$ in χ is a step in π, the time for the step is $\mu(s_i^{a_{sta}})$ and the duration is $\mu(s_j^{a_{eta}}) - \mu(s_i^{a_{sta}})$. If the TN is not consistent, the classical planner backtracks, as χ is not temporally sound and cannot be further extended.

Each step s_i of χ is an instance of a classical planning action. As such, it has a set of effects, denoted as $effects(s_i)$, where each effect is the form $\{\langle f := v\rangle$. Moreover, we write *conditions*(s_i) to indicate the set of preconditions of s_i, each in the form $\langle f = v\rangle$.

12.3.2 Handling Uncontrollable Durations

The general idea we pursue is to substitute the scheduling steps of algorithm 18 to solve a strong controllability problem for a TNU instead of consistency for a TN. The new general framework is shown in algorithm 19.

As in the purely controllable case, we first consider the abstract domain enumerating the "discrete" plans that are solution of the abstract problem. The overall idea of the FSSTP framework is to plug a scheduling phase on top of such a plan enumeration, to reject temporally-inconsistent plans. The steps of the abstract plan χ are time points for the temporal problem. The TNU is built by first separating the controllable steps in χ from the uncontrollable ones.

The technique we propose is limited to the Unc-Extr class of problems: for each action a, effects can only be specified at times st_a and et_a, and conditions are limited to timings $[st_a, st_a]$, $[et_a, et_a]$ or (st_a, et_a). Thanks to this limitation, we can consider all the time points corresponding to the action startings as controllable (the planner decides if and when an action should be started), while ending time points are controllable if the corresponding action is controllable, otherwise they are uncontrollable.

[5]We write $\mu(x)$ to indicate the value assigned to x by μ.

Algorithm 19 FSSTP for STPUD

```
1: procedure FSSTP($\mathcal{P}$)
2:     for all partial $\chi$ generated while solving $abstract(\mathcal{P})$ do
3:         $X_u \coloneqq$ UncontrollableSteps($\chi$, $\mathcal{P}$)
4:         $D_c \coloneqq$ ControllableDurations($\chi$, $\mathcal{P}$)
5:         $D_u \coloneqq$ UncontrollableDurations($\chi$, $\mathcal{P}$)
6:         $P \coloneqq$ Precedences($\chi$, $\mathcal{P}$)
7:         if $\mu \coloneqq$ TNU.Solve($(\chi - X_u, X_u, D_u, D_c \cup P)$) then
8:             if IsComplete($\chi$) then
9:                 return BuildStrongPlan($\mu$, $\chi$)
10:            else
11:                continue( )
12:            end if
13:        else
14:            reject($\chi$)
15:        end if
16:    end for
17:    return $\bot$
18: end procedure
```

The duration constraints are built analogously to the plain temporal case, but are divided in two sets: D_c are the duration constraints for controllable actions, D_u are the ones for uncontrollable actions.

Building a strong plan σ from a strong schedule for the TNU is analogous to the plain temporal planning case: each pair of neighboring $\langle s_i^{a_{sta}}, s_j^{a_{eta}} \rangle \in \chi$ is a step of σ, the time for the step is $\mu(s_i^{a_{sta}})$ and, if a is controllable, the duration is $\mu(s_j^{a_{eta}}) - \mu(s_j^{a_{sta}})$. We do not set the duration for uncontrollable durative actions.

The encoding of the precedence constraints P is crucial, because in presence of uncontrollability not all the techniques presented in the temporal planning literature for the controllable case [CCFL12, CCFL10] are complete. In the following, we consider two different encodings proposed in the temporal planning literature and we show that they are incomplete for solving the STPUD problem. Then, we borrow the idea of reordering from [Bäc98] and we derive the first sound and complete approach for the STPUD.

Total Order Encoding

A simple way of building the ordering constraints P, is to maintain the total order (TO) of the partial plan χ. Forcing this total order, clearly maintains the causal soundness but, as noted in [CCFL10], is heavily dependent on the order of actions chosen by the classical planner. Nevertheless, this encoding is complete for plain temporal planning and is adopted in the COLIN and CRIKEY 3 planners [CCFL12]. We call P_{TO} the set of precedence constraints for a given totally ordered plan $\chi = (s_1, \ldots, s_n)$, and we define it as follows:

$$P_{TO} \doteq \{s_i < s_{i+1} \mid i \in [1, n-1]\}\ .$$

We highlight that no disjunction is created, hence the encoding results in an STNU. Despite its simplicity, this encoding is incomplete in presence of temporal uncertainty. Indeed, at each step of algorithm 19, it might be the case that no total order produces a strongly controllable TNU, even if there exists a strong plan for the given problem. Thus, the planner can explore the complete search space and declare the problem unsolvable even if there exists a solution. Nonetheless, the approach is sound: if a solution is returned, it is a valid plan.

As an example, let us consider the situation depicted in figure 12.3. Let us suppose that both actions a and b must be started at the same time[6]. Action a is uncontrollable and b must end between the earliest and the latest possible ends of a. Literals p and q are initially true and no

[6]We just need that the end of b is forced to overlap with the interval in which a can uncontrollably end.

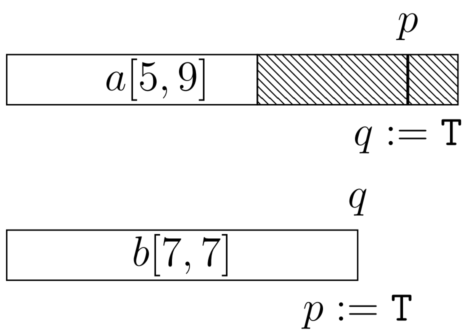


Figure 12.3: Example problem for which the TO and LAD encodings cannot find a plan, while a strong plan exists.

action falsify them. Let us focus on the relative order of $s_i^{a_{eta}}$ and $s_j^{b_{et_b}}$. If $\chi = (\cdots, s_i^{a_{eta}}, \cdots, s_j^{b_{et_b}}, \cdots)$, then we (transitively) impose the constraint $s_i^{a_{eta}} < s_j^{b_{et_b}}$, but this makes the STNU not strongly controllable, because, if a takes longer than 7 time units, a_{et_a} can happen after b_{et_b} violating the constraint. If $\chi = (\cdots, s_j^{b_{et_b}}, \cdots, s_i^{a_{eta}}, \cdots)$, then the situation is reversed and again the STNU is not strongly controllable. Therefore, in both cases χ is rejected and the planner returns $\perp$. This is incomplete, because there exists a simple strong plan for the problem: start both actions at time 0 (the two actions are non-interfering and all the conditions are satisfied as p and q are never falsified).

In our running example, reported in section 12.1.3, the TO approach can terminate yielding the strong plan π_{ex} when the following abstract plan is generated by the classical planner and the relative total order is considered.

$$\chi_{ex} \doteq \langle s_1^{move_{stmove}}, s_2^{visible:=\mathtt{T}}, s_3^{hot:=\mathtt{F}}, s_4^{move_{etmove}}, s_5^{trans_{st_{trans}}}, s_6^{trans_{et_{trans}}}, s_7^{visible:=\mathtt{F}} \rangle$$

This very same example of abstract plan that works for finding the plan π_{ex} also works in the following approaches.

Last Achiever Deordering Encoding

Another encoding that has been proposed, is to lift totally ordered plans to partially ordered plans [CCFL10]. The underlying idea is to use the greedy algorithm proposed in [VPC90] to reconstruct the causal links as precedence links. For each action in the plan requiring a literal l as precondition, the algorithm searches for the last achiever of that literal in the totally ordered plan, and imposes a precedence link between the two actions. In this way, it builds a partial order plan as a deordering [Bäc98] of χ and possibly reduces the commit on the specific input ordering. Also this encoding never introduces disjunctions, hence the resulting TNU is a STNU.

We now define the Last Achiever Deordering (LAD). Using a common trick in partial order planning, we consider two fictitious steps, s_0 and s_{n+1}, representing the initial state and the goal condition, respectively. Step s_0 has no preconditions and has the initial state I as effect. Step s_{n+1} has the goal as precondition and no effect.

Given a variable f and a value v, we denote with $achievers(f, v)$ the set $\{s_i \in \chi | \langle f := v \rangle \in effects(s_i)\}$ of steps that achieve l and with $deleters(f, v)$ the set $\bigcup_{v' \in Dom(f), v' \neq v} achievers(f, v')$. The intuition is that the $achievers(f, v)$ set gives the subset of steps in χ that set the variable f to value v. Conversely, $deleters(f, v)$ is the set of actions that set f to a value different from v.

Given a step s_i, we denote with $last(f, v, s_i)$ the step s_j such that $s_j \in achievers(f, v)$ and j is the maximum index strictly lower than i. Intuitively, $last(f, v, s_i)$ is the last action that sets f to value v before s_i in χ.

In the following, we define the precedence constraints P_{LAD} built using this technique.

Definition 66. *Given $\chi = (s_0, \cdots, s_{n+1})$, P_{LAD} is as follows.*

1. $\{(s_0 < s_i), (s_i < s_{n+1}) \mid i \in [1, n]\} \subseteq P_{LAD}$.

For each $s_i \in \chi$ and for each $\langle f = v \rangle \in pre(a)$,

2. $(last(f, v, s_i) < s_i) \in P_{LAD}$.

For each $s_i \in \chi$ and for each overall condition $\langle (st_a, et_a)\ f = v \rangle$ of the action a,

3. $\{(s_j < s_{st_a}) \mid s_j \in deleters(f, v), j < i\} \subseteq P_{LAD}$
4. $\{(s_{et_a} < s_j) \mid s_j \in deleters(f, v), i < j\} \subseteq P_{LAD}$.

This encoding is able to find a plan in many situations even in presence of uncertainty, but it is not complete in general. For example, it fails on the problem of figure 12.3: the encoding greedily assumes that the last achiever is the one that must be preserved in the form of a causal link; in reality there may be other achievers that could be used instead. Just as in the previous case, if $\chi = (\cdots, s_i^{a_{eta}}, \cdots, s_j^{b_{et_b}}, \cdots)$, then we impose the constraint $s_i^{a_{eta} < b_{et_b}}$ because $s_i^{a_{eta}}$ is the last achiever of p (required by $s_j^{b_{et_b}}$). Instead, if $\chi = (\cdots, s_j^{b_{et_b}}, \cdots, s_i^{a_{eta}}, \cdots)$, we impose the constraint $s_j^{b_{et_b}} < s_i^{a_{eta}}$ because $s_j^{b_{et_b}}$ is the last achiever of q (required by $s_i^{a_{eta}}$).

Disjunctive Reordering Encoding

In order to obtain a sound and complete reasoning, we need to relax the total order produced by the state-space search, retaining the precedence constraints needed to ensure plan validity. However, we must be careful in not over-constraining the TNU, otherwise we may discard valid plans. A solution is to consider all the reorderings [Bäc98] of the given plan that are causally sound: we build a set of (disjunctive) precedence constraints in such a way that all the orderings fulfilling the constraints are causally sound. We call *Disjunctive Reordering* (written DR) the approach using the precedence constraints P_{DR} defined as follows. We show that, given a partial plan χ, using DR to construct the precedences in algorithm 19, yields a complete technique for the STPUD.

Given a variable f, a value v and a pair of actions a and r of χ, we define the disjunctive temporal constraint $\rho(f, v, a, r)$ as follows.

$$\rho(f, v, a, r) \doteq (a < r \wedge \bigwedge_{s_i \in achievers(f,v) \setminus \{a,r\}} (s_i < a \vee s_i > r))$$

Intuitively, for a condition $c \doteq \langle f = v \rangle$, if a is an achiever of c and r is an action having c as precondition, $\rho(f, v, a, r)$ holds if a was the last achiever of l before r. We now define the precedence constraints induced by the DR approach, indicated as P_{DR}. As before, s_0 and s_{n+1}, represent the initial state the goal condition, respectively.

Definition 67. *Given $\chi = (s_0, \cdots, s_{n+1})$, P_{DR} is as follows.*

1. $\{(s_0 < s_i), (s_i < s_{n+1}) \mid i \in [1, n]\} \subseteq P_{DR}$.

For each $a \in \chi$ and for each $\langle f = v \rangle \in pre(a)$, the following constraints belong to P_{DR}:

2. $\bigvee_{s_j \in achievers(f,v) \setminus \{a\}} \rho(f, v, s_j, a)$;

3. $\bigwedge_{s_j \in achievers(f,v)} (\rho(f, v, s_j, a) \rightarrow \bigwedge_{s_t \in deleters(f,v) \setminus \{a\}} (s_t < s_j \vee s_t > a))$.

For each $s_i \in \chi$ and for each overall condition $\langle (st_a, et_a)\ f = v \rangle$ of the action a, the following constraint is in P_{DR}:

4. $\bigwedge_{s_j \in deleters(f,v)} ((s_j < s_{st_a}) \vee (s_j > s_{et_a})$.

Intuitively, constraint 2 says that at least one action s_j having as effect the precondition l of action a, should occur before a. Constraint 3 says that, if s_j is the last achiever for the precondition l of action a, between s_j

and a there must be no action falsifying $\langle f = v \rangle$. Conditions are cannot be canceled between their extremal time points markers (constraint 4). The following theorem states that DR is sound and complete. We give the full proof of the theorem in appendix B.1.

Theorem 12.1 (DR Completeness)**.** *Given a STPUD admitting a valid strong plan σ, if DR is used, algorithm 19 terminates with a valid strong plan.*

The intuition is that in DR the disjunctions encode all reorderings that are causally sound in the form of a DTNU, allowing the scheduler to re-arrange the actions independently of the total ordering of χ.

We highlight that the set of precedence constraints generated by the DR approach, conjoined with the duration constraints, yields a TNU that is not formally a DTNU. This is because of the use of strict inequalities and negations that are not expressible in the DTNU framework. However, if we take the PDDL 2.1. semantics, this is not a problem because such semantics prescribes that there is always a minimal time quantum (called ϵ) that is required to separate two events: each constraint $s_i > s_j$ can then be rewritten as $s_i - s_j \geq \epsilon$, and negations can be handled by reversing the inequalities (e.g. $\neg(s_i > s_j)$ is equivalent to $(s_i \leq s_j)$). Therefore, we can encode these constraints as a proper DTNU. Moreover, we remark that the strong controllability techniques for the DTNU problem class we presented in chapter 9 are applicable even in presence of strict inequalities (we only exploit the absence of strict inequalities in the static quantification for TCSNU, but never in DTNU).

The strong controllability of a DTNU is a NP-Hard problem [PVYS07] (and the same is true for the generalized DTNU with strict inequalities), thus the use of this encoding is quite costly; however, DR is important as it overcomes the incompleteness limitation of the other encodings.

12.4 Compiling STPUD in Temporal Planning

In this section, we present our compilation technique, which can be used to reduce any planning instance P having duration uncertainty into a temporal planning instance P' in which all actions have controllable durations. The translation guarantees that P is solvable if and only if P' is solvable, and it fully supports the Unc-Arbit problem class. Moreover, given any plan for P' we can derive a plan for P. This approach comes at the cost of duplicating some of the variables in the domain, but allows for the use of off-the-shelf temporal planners.

12.4.1 Formal Compilation

The overall intuition behind the translation is the following. Consider the *transmit* (i.e., *trans*) action in our example, and suppose it is scheduled to start at time k. Let v be the value of *sent* at time $k+5$; since *transmit* has an at-end effect $\langle [et_{trans}]\, sent := \mathtt{T}\rangle$, we know that the value of the variable *sent* during the interval $(k+5, k+8]$ will be either v or $\mathtt{T}$ depending on the duration of the action. After time $k+8$ we are sure that the effect took place, and we are sure of the value of *sent* until another effect is applied[7]. Since we are not allowed to observe anything at run-time in strong planning, we need to consider this uncertainty in the value of *sent* and produce a plan that works regardless. Since *sent* could appear as a condition of another action (or as a goal condition, as in our example) we must rewrite such conditions to be true only if both $\mathtt{T}$ and v are values that satisfy the condition.

To achieve this, we create an additional variable $sent_\sigma$ (called the *shadow variable of sent*). This secondary variable stores the alternative value of

[7]Note that there cannot be another concurrent action in the plan having an effect on *sent* during the interval $[k+5, k+8]$ because this would allow for the possibility of two concurrent effects on the same variable, that is forbidden in our semantics.

sent during uncertainty periods. When there is no uncertainty in the value of *sent*, both *sent* and $sent_\sigma$ will have the same value. In this way, all the conditions involving *sent* can be rewritten in terms of *sent* and $sent_\sigma$ to ensure they are satisfied by both the values.

In general, our translation rewrites a STPUD problem $P \doteq \langle V, I, T, G, A \rangle$ into a new planning instance $P' \doteq \langle V', I', T', G', A' \rangle$ that does not contain actions with uncontrollable duration.

Uncertain Variables

The first step is to identify the set of variables $L \subseteq V$ that appear as effects of uncontrollable actions and are executed at a time depending on the end of the action.

$$L \doteq \{f \mid a \in A_u, \langle [t]\, f := v \rangle \in E_a, t = et_a - \delta\}$$

Intuitively, this is the set of variables that can possibly have uncertain value during plan execution. A variable that is modified only at times linked to the start of actions or by timed initial literals, cannot be uncertain as neither the starting time of actions nor the timed initial literals can be uncertain in our model. In our running example, the set L becomes $\{sent, pos\}$.

We now define the set V' as the original variables V plus a shadow variable for each variable appearing in L.

$$V' \doteq V \cup \{f_\sigma \mid f \in L\}$$

We use the pair of variables f and f_σ to represent uncertainty: if $f = f_\sigma$ we know that there is no uncertainty in the value of f, while if $f \neq f_\sigma$ we know that the actual value of f in the original problem is either f or f_σ.

Disjunctive Conditions

At the end of section 12.1, we outlined the reason why existing approaches of compiling away disjunctive conditions will not work with uncontrollable action durations. In order to rewrite a durative, disjunctive condition $c \doteq \langle \mathbb{I}(st_c, et_c) \bigvee_{i=1}^{n} f_i = v_i \rangle$ we need to ensure that the result is satisfied if and only if both the values of f and f_σ for each $f \in L$ are satisfying values for c. For this reason, we define an auxiliary function $\gamma(\psi)$ that takes a single disjunctive condition without timing information and returns a set of untimed disjunctive conditions.

$$\gamma(\psi) \doteq \begin{cases} \{\langle f = v \rangle\} & \text{if } \psi \doteq \langle f = v \rangle, f \notin L \\ \{\langle f = v \rangle, \langle f_\sigma = v \rangle\} & \text{if } \psi \doteq \langle f = v \rangle, f \in L \\ \{r \vee s \mid r \in \gamma(\psi_1), s \in \gamma(\psi_2)\} & \text{if } \psi \doteq \psi_1 \vee \psi_2 \end{cases}$$

For example, the condition of the *trans* action, $pos = l_2$, is translated as the two conditions $pos = l_2$ and $pos_\sigma = l_2$. Analogously, assuming that both f and g are in L, a given condition $(f = \texttt{T}) \vee (g = \texttt{F})$ in P is translated by function γ as the set of conditions $\{(f = \texttt{T}) \vee (g = \texttt{F}), (f_\sigma = \texttt{T}) \vee (g = \texttt{F}), (f = \texttt{T}) \vee (g_\sigma = \texttt{F}), (f_\sigma = \texttt{T}) \vee (g_\sigma = \texttt{F})\}$ in P'.

Uncertain Temporal Intervals

We also need to identify the temporal interval in which the value of a given variable can be uncertain. Given an action a with uncertain duration d_a in $[l, u]$, let $\lambda(t)$ and $\nu(t)$ be the earliest and latest possible times at which an at-end effect at $t \doteq et_{a'} - \delta$ may happen. Thus: $\lambda(t) \doteq st_{a'} + l - \delta$ and $\nu(t) \doteq st_{a'} + u - \delta$. Both functions are equal to $st_{a'} + \delta$ if $t \doteq st_a + \delta$. For example, consider the effect $e_1 \doteq \langle [et_{trans}]\ sent := \texttt{T} \rangle$ of action *trans*. We know that the duration of transmit is uncertain in $[5, 8]$, therefore the effect can be applied between $\lambda(et_{trans}) \doteq st_{trans'} + 5$ and $\nu(et_{trans}) \doteq st_{trans'} + 8$ and the *sent* variable has an uncertain value within that interval.

Uncontrollable Actions

For each uncontrollable action $a \doteq \langle [l, u], C_a, E_a \rangle)$ in A_u in the original model we create a new action $a' \doteq \langle [u, u], C_{a'}, E_{a'} \rangle$ in A'_c. Specifically, we first fix the maximal duration u as the only allowed duration for a' and then insert appropriate effects and conditions *during* the action to capture the uncertainty.

The effects $E_{a'}$ are partitioned in two sets $E^l_{a'}$ and $E^u_{a'}$ to capture possible values within the uncertain action execution duration. The conditions $C_{a'}$ are also composed of two elements: the rewritten conditions $C^R_{a'}$ and the conditions added to protect the new effects $C^E_{a'}$ (thus $C' \doteq C^R_{a'} \cup C^E_{a'}$).

Rewritten conditions $C^R_{a'}$. Controllable conditions are compiled by rewriting existing action conditions by means of the γ function. The intervals specifying the duration of the conditions are preserved; since the action duration is set to its maximum, the intervals of the conditions are “stretched” to match their maximal duration.

$$C^R_{a'} \doteq \{\langle \mathbb{I}(\lambda(t_1), \nu(t_2))\ \alpha \rangle \mid \alpha \in \gamma(\psi), \langle \mathbb{I}(t_1, t_2)\ \psi \rangle \in C_a\}$$

Here, we keep the interval type (e.g. a $[t_1, t_2)$ interval gets translated in $[\lambda(t_1), \nu(t_2)]$), but we “stretch” the bounds to match the earliest start and latest possible end.

For example, the set $C^R_{trans'}$ for the *trans* action is: $\{\langle [st_{trans'}, st_{trans'} + 8]\ pos = l_2 \rangle$, $\langle [st_{trans'}, st_{trans'} + 8]\ pos_\sigma = l_2 \rangle$, $\langle [st_{trans'}, st_{trans'} + 8]\ visible = \mathtt{T} \rangle\}$. This requires variables *visible*, *pos* and pos_σ to be true throughout the execution of $trans'$.

Compiling action effects. The effects of the original action are duplicated: both the affected variable f and its shadow f_σ are modified, but at different times. We first identify the earliest and latest possible times at which an effect can happen due to the duration uncertainty (see earlier discussion on $\lambda(t)$ and $\nu(t)$). We then apply the effect on f_σ at the earliest time point $\lambda(t)$, and at the latest time point $\nu(t)$ we re-align f and f_σ by also applying the effect on f:

$$E^l_{a'} \doteq \{\langle [\lambda(t)]\, f_\sigma := v\rangle \mid \langle [t]\, f := v\rangle \in E_a\}$$

$$E^u_{a'} \doteq \{\langle [\nu(t)]\, f := v\rangle \mid \langle [t]\, f := v\rangle \in E_a\}$$

For example, the *trans* action has $E^l_{trans'} \doteq \{\langle [st_{trans'} + 5]\, sent_\sigma := \mathtt{T}\rangle\}$ and $E^u_{trans'} \doteq \{\langle [st_{trans'} + 8]\, sent := \mathtt{T}\rangle\}$.

Additional conditions $C^E_{a'}$. Let $t \doteq et_a - \delta$ be the time of an at-end effect that affects the value of f. In order to prevent other actions from changing the value of f during the interval $(\lambda(t), \nu(t)]$ where the value of f is uncertain, we add a condition in $C^E_{a'}$ to maintain the value of f_σ throughout the uncertain duration $(\lambda(t), \nu(t)]$.

$$C^E_{a'} \doteq \{\langle (\lambda(t), \nu(t)]\, f_\sigma = v\rangle, \mid \langle [t]\, f := v\rangle \in E_a\}$$

Since the effect on f_σ (belonging to $E^l_{a'}$) is applied at time $\lambda(t)$, the condition is satisfied immediately after the effect and we want to avoid concurrent modifications of either f or f_σ until the uncertainty interval ends at $\nu(t)$.

For example, the *trans* action has $C^E_{trans'} \doteq \{\langle (st_{trans}+5, st_{trans}+8] sent_\sigma = \mathtt{T}\rangle\}$. Compilation of the *trans* action is depicted in figure 12.4.

Controllable Actions

Controllable actions are much simpler. For each $a \doteq \langle [l, u], C_a, E_a\rangle \in A_c$ we introduce a replacement action $a' \doteq \langle [l, u], C_{a'}, E_{a'}\rangle \in A'_c$, in which: (1) each

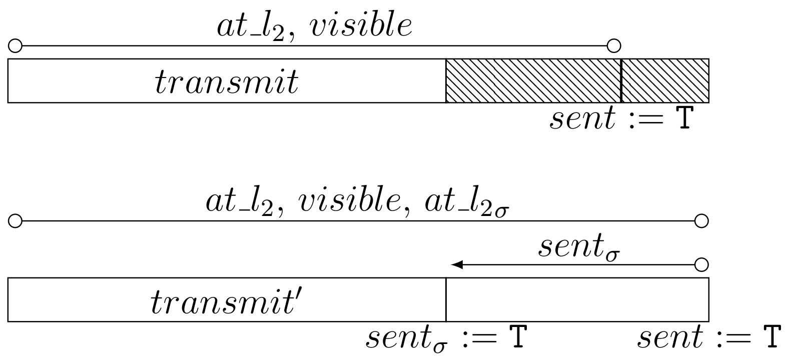


Figure 12.4: Graphical view of the original *transmit* action (top) and its compilation (bottom). Durative conditions are represented over the actions, while effects are reported under each action. We indicate closed interval extremes with circles and open ends with arrows. For example, the condition $sent_\sigma$ is open to the left and closed to the right.

condition in C is rewritten to check the values of both the variables and their shadows, and (2) each effect is applied to a variable and its shadow, if any.

$$C_{a'} \doteq \{\langle \mathbb{I}(\lambda(t_1), \nu(t_2))\ \alpha\rangle \mid \alpha \in \gamma(\psi), \langle \mathbb{I}(t_1, t_2)\ \psi\rangle \in C_a\}$$

$$E_{a'} \doteq E_a \cup \{\langle [t]\ f_\sigma := v\rangle \mid f \in L, \langle [t]\ f := v\rangle \in E_a\}$$

Initial State I

The initial state is handled by initializing variables and their corresponding shadow variables in the same way as in the original problem.

$$I'(x) \doteq \begin{cases} I(x) & \text{if } x \in V \\ I(f) & \text{if } x = f_\sigma \end{cases}$$

For example, the initial state of our running problem is the original initial state plus $\{sent_\sigma = \mathtt{F}, pos_\sigma = l_1\}$.

Timed Initial Literals

Timed Initial Literals T' are set similarly to the effects.

$$T' \doteq T \cup \{\langle [t]\ f_\sigma := v\rangle \mid f \in L, \langle [t]\ f := v\rangle \in T\}$$

$$
\begin{aligned}
V' &\doteq V \cup \{pos_\sigma : \{l_1, l_2\}, sent_\sigma : \{\mathtt{T}, \mathtt{F}\}\} \\
I' &\doteq I \cup \{pos_\sigma = l_1, sent_\sigma = \mathtt{F}\} \\
T' &\doteq \{\langle [14]\ visible := \mathtt{T}\rangle, \langle [30]\ visible := \mathtt{F}\rangle, \langle [15]\ hot := \mathtt{F}\rangle\} \\
G' &\doteq \{\langle [et_\pi, et_\pi]\ sent = \mathtt{T}\rangle, \langle [et_\pi, et_\pi]\ sent_\sigma = \mathtt{T}\rangle\} \\
A'_c &\doteq \{\langle [15, 15], C_{move'}, E_{move'}\rangle, \langle [8, 8], C_{trans'}, E_{trans'}\rangle\} \\
C_{move'} &\doteq \{\langle [st_{move}, st_{move}]\ pos = l_1\rangle, \langle [st_{move}, st_{move}]\ pos_\sigma = l_1\rangle, \\
&\qquad \langle [et_{move}, et_{move}]\ hot = \mathtt{F}\rangle, \langle (st_{move} + 10, st_{move} + 15]\ pos_\sigma = l_2\rangle, \\
C_{trans'} &\doteq \{\langle [st_{trans}, et_{trans}]\ pos = l_2\rangle, \langle [st_{trans}, et_{trans}]\ pos_\sigma = l_2\rangle, \\
&\qquad \langle [st_{trans}, et_{trans}]\ visible = \mathtt{T}\rangle, \langle (st_{trans} + 5, st_{trans} + 8]\ sent_\sigma = \mathtt{T}\rangle, \\
E_{move'} &\doteq \{\langle [st_{move} + 10]\ pos_\sigma := l_2\rangle, \langle [st_{move} + 15]\ pos := l_2\rangle\} \\
E_{trans'} &\doteq \{\langle [st_{trans} + 5]\ sent_\sigma := \mathtt{T}\rangle, \langle [st_{trans} + 8]\ sent := \mathtt{T}\rangle\}
\end{aligned}
$$

Figure 12.5: The compiled STPUD obtained by applying the compilation approach to the rover example.

In our example, we do not have timed initial literals operating on uncertain variables, thus $T \doteq T'$.

Goal Conditions

The goal conditions G are augmented to consider both the original and shadow variables, without modifying the application times, since they are fixed and cannot be uncertain.

$$G' \doteq \bigcup_{g \in G} \gamma(g)$$

In our example, the set G' becomes $\{(sent = \mathtt{T}), (sent_\sigma = \mathtt{T})\}$.

12.4.2 Example

The full compilation for our example problem is reported in figure 12.5.

12.4.3 Discussion

This compilation is sound and complete. Theorem 12.2 states that the original problem is solvable if and only if the resulting problem is solvable and any plan for the rewritten temporal planning problem is automatically a strong plan for the original problem (with the obvious mapping from the rewritten to the original actions). We report the complete proof of this theorem in appendix B.2.

Theorem 12.2 (Soundness and Completeness)**.** *Let $P \doteq \langle V, I, T, G, A \rangle$ be a planning instance and $R \doteq \langle V', I', T', G', A' \rangle$ be its translation. P has a strong plan π if and only if R has a temporal plan σ.*

The compilation produces a problem that has: (i) at most twice the number of variables of the original problem, (ii) at most twice the initial and timed assignments and (iii) exactly the same number of actions. The only point in which the compilation might produce exponentially large formulae is in the application of the γ function, which is exponential in the number of disjuncts constraining variables appearing in L. Since this only happens for disjunctive conditions, and the number of disjuncts is typically small, this is normally not a serious issue.

Finally, we remark that any technique can be used to solve the compiled temporal planning problem, and any valid plan corresponds to a strong plan. Therefore, this technique allows for the mix of controllability and flexibility we claimed in section 6.1: if we employ a planner that is able to produce flexible solutions (i.e. an STN of possible solutions), all those solutions will be valid strong plans. This is an example of flexible-strong planning.

12.5 Simplification

As we discussed in section 12.1.4, it is in general impossible to solve the STPUD problem by considering uncontrollable actions as taking either the maximal or minimal duration in their duration interval: this motivates the development of the S-FSSTP and the compilation approaches we presented. Nonetheless, under some conditions, it is possible to soundly remove uncertainty simply by fixing the maximal or the minimal duration of an action. In this section, we report a set of sufficient conditions that can be statically checked on a planning instance to simplify the uncertainty of an uncontrollable action. For the sake of simplicity, we consider the UNC-EXTR problem class, but the same reasoning can be extended also to the general UNC-ARBIT class.

12.5.1 Maximal-Duration Simplification

In order to soundly lengthen an uncontrollable action $a \doteq \langle [l, u], C, E \rangle$ to its maximum duration u without changing its conditions or its effects, we need to make sure that there is no other action condition that could occur within the time window $[l, u]$ that would be fooled by encoding the end conditions and end effects considering only the maximal duration. Furthermore, we need to ensure that no other action can have an effect that can occur within the time window $[l, u]$ that would conflict with the overall conditions, end conditions or end effects of a.

To keep the execution safe, we require the following.

1. For each end effect $\langle [et_a]\ f := v \rangle$, no other action b can have an effect on f that could occur within $[l, u]$.

   This prohibition is necessary because such an effect might happen at the same time as $\langle [et_a]\ f := v \rangle$ if a has duration shorter than u

(simultaneous effects on the same variable are prohibited). Note that if a has an overall condition on the variable s this will prevent any effect on f during $[l, u]$ so the condition is automatically satisfied in this case. Thus, in practice, we only need to consider the variables affected by end effects that are not protected by an overall condition.

2. For each end effect $\langle [et_a]\ f := v \rangle$, no other action b can have an incompatible condition on f that could occur within $[l, u]$.

   This prohibition is necessary because such a condition might happen at the same time as $\langle [et_a]\ f := v \rangle$ if a has duration shorter than u. Note that if a has an overall condition on the variable f this will prevent any inconsistent condition.

3. For each end condition $\langle [et_a, et_a]\ \phi \rangle$, no other action b can have an effect incompatible with ϕ that can occur within $[l, u]$.

   This prohibition ensures that b does not change any of the variables in ϕ making the condition false during the uncertain interval. Note that if f is also affected by and end effect, then this is already covered by (1).

4. For each end condition $\langle [et_a, et_a]\ \phi \rangle$, no other action b can have a condition inconsistent with ϕ that can occur within $[l, u]$.

   Since a could end earlier, this condition ensures that b does not have a conflicting condition with ϕ during $[l, u]$.

It might seem like we also need the condition that no other action b can depend on an overall condition of a within $[l, u]$. However, this prohibition is not necessary. If the variables in the overall condition are affected by an end effect, then this is already covered by prohibition (2). If not, then the overall condition prevents any effect on these variables during (l, u), which means that if a ends early, the condition will necessarily persist until u.

As a result, having an action b with a condition on threatening an overall condition that occurs during $[l, u]$ does not cause any problem as long as conditions (1)-(4) are satisfied.

12.5.2 Minimal-Duration Simplification

In order to soundly shorten an uncontrollable action $a \doteq \langle [l, u], C, E\rangle$ to its minimum duration l, we need to make sure that there is no other action or condition that could occur within the time window $[l, u]$ that would be fooled by encoding the end conditions and the end effects at l, or by the premature ending of the overall conditions. Furthermore, we need to ensure that no other action can have an effect that can occur within the time window $[l, u]$ that would conflict with the overall conditions, end conditions or the end effects.

The following conditions are sufficient to simplify an uncontrollable action fixing its minimal duration.

1. For each end effect $\langle [et_a]\, f := v\rangle$, no other action b can have an effect on f that could occur within $[l, u]$.

   This prohibition is necessary because such an effect might happen at the same time as $\langle [et_a]\, f := v\rangle$ if a takes longer than l, and simultaneous effects on the same variable are prohibited.

2. For each end effect $\langle [et_a] f := v\rangle$, no other action b can have a condition implied by $\langle [et_a]\, f := v\rangle$ that could occur within $[l, u]$

   This condition assures that b does not depend on the end condition of a prematurely, since a might not really end until as late as u.

3. No other action b can have an effect or condition inconsistent with either an overall or end condition that can occur within $[l, u]$

This condition and effect prohibition is necessary because the overall and end conditions might last or happen until u, creating and inconsistency with b.

Clearly the requirements will be satisfied if the prohibited conditions and effects do not exist at all. However, there are some cases where it is possible to infer something stronger. In particular, for every action b with a condition or effect that might violate one of the three conditions above, we can show that the troublesome condition or effect cannot occur in $[l, u]$ if the following three conditions are satisfied:

1. The offending effect or condition for b is at the end of b.

   This condition is necessary because if b had an offending condition or effect at the start, or offending condition overall there would be no way of guaranteeing that b would not start within $[l, u]$.

2. b has minimum duration greater than $u - l$.

   This condition is necessary to guarantee that if b starts after l its end conditions and effects will end after u and therefore won't cause a problem.

3. Either:

   - b cannot overlap with a (a and b are mutually exclusive)
   - or b cannot start during a and b has max duration less than the min duration of a.

   This condition is necessary to assure that b cannot start during or before a but end during $[l, u]$. This will be satisfied if b and a cannot overlap at all or if b cannot start during a, and is short enough that if it starts before a it will end before l.

Note that this set of conditions is correct, but not complete. There may be other cases where it is possible to prove that an action b with an offending condition or effect cannot occur within $[l, u]$. However, this proof is likely to be more complex and involve some sort of reachability argument.

12.5.3 Discussion

The conditions we listed above are sufficient to soundly simplify away uncontrollable durations fixing either the maximal or minimal duration.

The lengthening conditions are much easier to satisfy than the conditions for shortening an action to its minimum duration. The reason for this is that the lengthening process extends the overall condition of the action a, which tends to prevent bad things from happening during $[l, u]$. On the contrary, the condition for shortening an action are much harder to satisfy because they impose some outside-of-action requirements that are rarely met in practice.

Considering the maximal-duration simplification, there is a simpler, sufficient condition to ensure all the rules we listed: for every threatening action b (an action that might violate at least one rule), a and b are mutually exclusive. Mutual exclusion is not a new idea and there are techniques to check it even for temporal planning problems [BS11].

12.6 Experimental Evaluation

We now empirically evaluate the approaches and simplifications we presented. First we discuss the experimental setting, then we explain how to use PDDL 2.1 planners for dealing with the compilation output, and finally we discuss the results.

12.6.1 Experimental Set-Up

We implemented the direct approaches for the Unc-Extr problem class (described in section 12.3) as an extension of the Colin [CCFL12] planner. Colin can handle temporal domains expressed in PDDL+ [FL06] (an extension of PDDL 2.1), and its temporal reasoning component is modular with respect to the rest of the code base: it has a clear notion of scheduler for checking temporal consistency.

To enable for the specification of durative actions with uncontrollable durations, we extended the PDDL 2.1 grammar with the addition of an `uncontrollable-durative-action` specification: the construct is analogous to the `durative-action` construct, but marks the action as uncontrollable. We refer to this extension of the language as PDDL-U. The parser and the internal structures of Colin have been modified to support the processing of this extension.

We added three new schedulers to Colin, each implementing one of the defined encodings. We write TO to refer to the encoding based on total ordering, LAD for the Last Achiever Deordering and DR for the Disjunctive Reordering. The heuristic for the forward search planner was left unchanged. The temporal solvers for strong controllability were implemented in C++, following the approach presented in chapter 9, using the MathSAT [CGSS13] SMT solver as workhorse.

The compilation described in section 12.4 has been implemented as a Java translator in two versions. The first takes in input a PDDL-U specification and produces a plain PDDL 2.1 temporal planning problem, the second works in the context of ANML and takes in input an ANML problem with uncontrollable durations and emits a fully controllable ANML problem. In this experimental evaluation, we limit ourselves to PDDL 2.1 problems leaving a thorough ANML evaluation for future work.

We highlight how the formal translation is designed to generate a Ctrl-Arbit problem even when starting from an Unc-Extr instance. For this reason, the first version of translator is equipped with a technique to remove intermediate effects and conditions in the PDDL setting. We discuss in detail this technique in section 12.6.2. Since the direct approaches have been implemented using a modification of the Colin planner, we used Colin to solve the temporal planning instances produced by the compilation.

Finally, the maximal-duration simplification has been implemented as a Java re-writer that takes in input a PDDL-U instance and produces a (possibly simplified) PDDL-U specification. We highlight that in some cases the simplifier is able to remove all the temporal uncertainty from the problem and, thanks to the way PDDL-U is defined, a plain PDDL 2.1 planning problem is produced by the simplifier. In this experiments we did not use the minimal-duration simplification.

We considered the temporal planning domains from the temporal track of the International Planning Competition 2011 [CCO+12]: these domains are written in the PDDL 2.1 language. We modified them by declaring some actions uncontrollable and by enlarging the duration intervals of actions and by creating several versions of each domain. Our resulting benchmark set is composed of a total of 901 planning instances. Since both the compilation and the simplification techniques manipulate the domain specifications and automated planners are usually very sensitive to the input, we implemented a tool that randomly "scrambles" PDDL input problems. The tool always generates a problem instance that is absolutely equivalent to the input one, but it changes the order of actions, conditions and effects in the concrete specification. Using this tool, we executed each experiment 5 times (with different random seeds to obtain different "scramblings"), considering each run as a separate experiment. Thus, we are left with a virtual benchmark set of 4505 planning instances.

All the experiments were executed on a Scientific Linux 64 bit, 12 cores Intel Xeon at 2.67GHz, with 96GB RAM. We used a timeout of 10 minutes, and a memory limit of 8GB.

All the tools and the benchmark set can be downloaded as indicated in section 1.2.

12.6.2 Intermediate Effects

Given a PDDL-U planning instance, the theoretical compilation approach (described in section 12.4) produces an instance having no uncontrollable durations, but with intermediate effects and conditions. In particular, for the UNC-EXTR case that is expressible by PDDL-U, the algorithm introduces an intermediate effect and a durative condition for each uncontrollable action.

We now discuss how to get rid of these intermediate effects and conditions assuming a PDDL 2.1 semantics. In fact, it is in general not possible to compile away intermediate effects in a language having a real-time semantics, but PDDL 2.1 semantics assumes a minimum time quantum called ϵ forcing each pair of time points in the plan to be separated by at least ϵ time. We can exploit this semantics to compile away intermediate effects.

The work in [FLH04] presents some ideas on how to encode intermediate events in PDDL 2.1. It assumes actions having a fixed duration and proposes some standard constructs to encode several features as polynomial transformation of the PDDL domain. In particular, the clip-action construct forces two or more time points (either the start or the end of actions) to happen simultaneously or to be separated by exactly ϵ. The construction uses one additional action with duration 2ϵ (or 3ϵ in case an ϵ separation is required). The clip action defined as follows.

$$clip \doteq \langle [2\epsilon, 2\epsilon], \{\langle (st_{clip}, et_{clip})\, f_s = \top \rangle\}, \{\langle [st_{clip}]\, f_s := \top \rangle, \langle [et_{clip}]\, f_s := \bot \rangle\}\rangle$$

Intuitively, the clip is an action that sets a fresh, dummy variable f_s to true when it starts and resets it upon termination. We can now clip time points to impose the condition $f_s = \top$ exactly at that time points. Since in PDDL 2.1 no two events can happen with distance lower than ϵ, the clip guaranteed simultaneous execution. If the time points being clipped have mutually-exclusive effects, we need a clip of duration 3ϵ to achieve an ϵ separation of the two time points.

This construction can be used to encode intermediate effects and conditions by splitting an action duration in pieces, one for each sub-interval of the action duration delimited by an intermediate effect or condition bound. This technique works perfectly for fixed duration actions, but also when the duration is controllable. Note that this is enough to handle the outcome of the uncertainty compilation, because each uncontrollable action is transformed in a fixed-duration action and no other intermediate effects or conditions are artificially added. Figure 12.6 shows an example of the construction for the action $trans'$ derived from the running example action $trans$ by means of the uncertainty compilation.

The clip action is not the only construction that can be used to encode intermediate effects and conditions. A second relevant technique [Smi03] uses a container action that spans the whole duration of the original action that exactly contains a number of sub-actions, one for each sub intervals.

Both these techniques suffer from a significant overhead: to remove an intermediate effect or condition, a single action is substituted with three actions, increasing the plans length, and some additional variables are added to the problem description, widening the search space. However, we can exploit a characteristic common to both these compilation techniques to

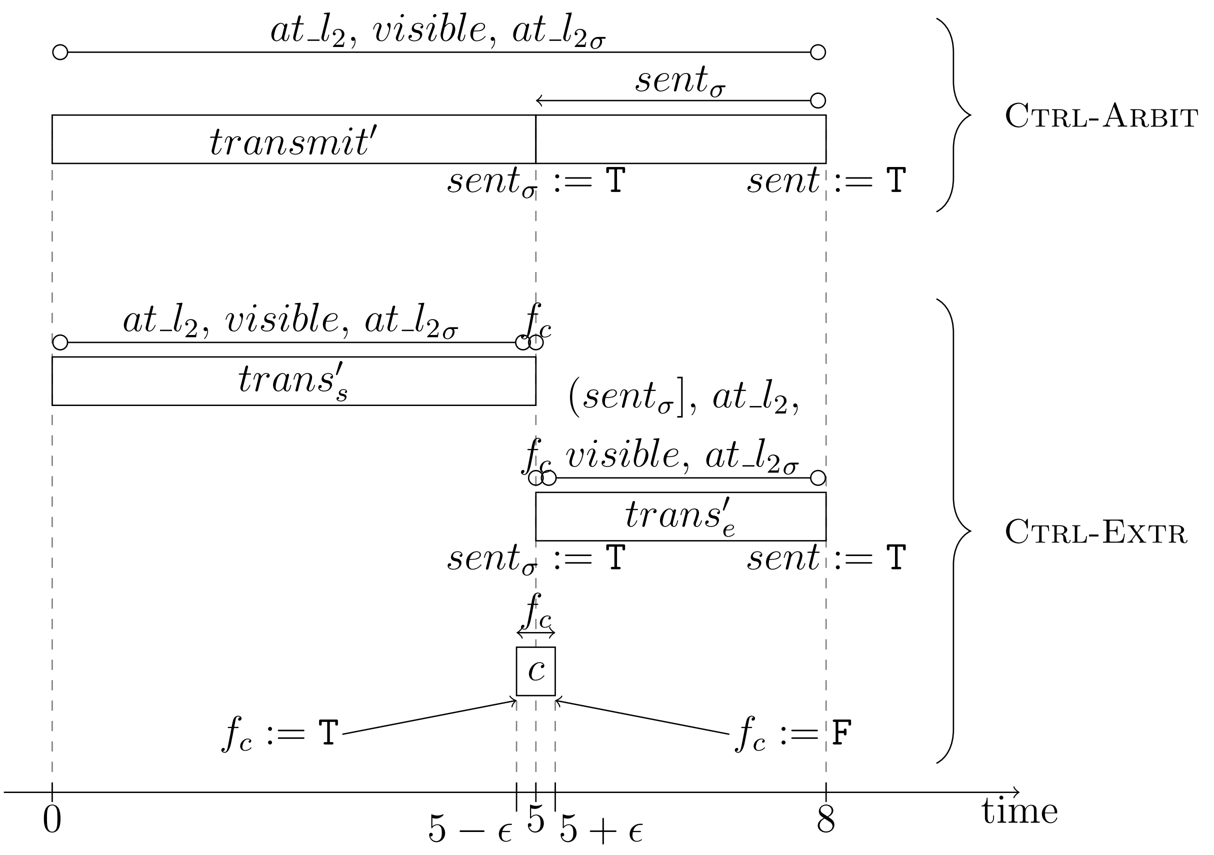

at_l_2, $visible$, $at\_l_{2\sigma}$
$sent_\sigma$
$transmit'$
$sent_\sigma := \mathtt{T}$
$sent := \mathtt{T}$
CTRL-ARBIT
at_l_2, $visible$, $at\_l_{2\sigma}$
f_c
$trans'_s$
$(sent_\sigma]$, at_l_2,
f_c $visible$, $at\_l_{2\sigma}$
$trans'_e$
$sent_\sigma := \mathtt{T}$
$sent := \mathtt{T}$
CTRL-EXTR
f_c
c
$f_c := \mathtt{T}$
$f_c := \mathtt{F}$
0
$5-\epsilon$
5
$5+\epsilon$
8
time

Figure 12.6: Clip action construction example: the action *trans′* derived from the running example action *trans* by means of the compilation is rewritten as an equivalent triplet of actions. The condition f_c that is the base of the construction is imposed at time 5 at the end of the action $trans'_s$ and at the begin of the action $trans'_e$; moreover it is an overall condition of the clip action c. For space problems, we indicated the condition $\langle (5, 8]\ sent_\sigma = \mathtt{T}\rangle$ together with the other conditions in the interval $[5, 8]$.

guide the planners and limit this overhead. Both these encodings are designed in such a way that when a container action is started, or the first piece of the decomposed action is started, the others follow in a pre-ordered fashion with no possibility of choice for the planner. However, even if the search takes this decision, nothing is telling the planner to avoid useless search on branches where the structure has not been correctly instantiated. Surely, we can devise a planner that recognizes a construction such as the clip and expands it accordingly, but we can also exploit the PDDL 2.1

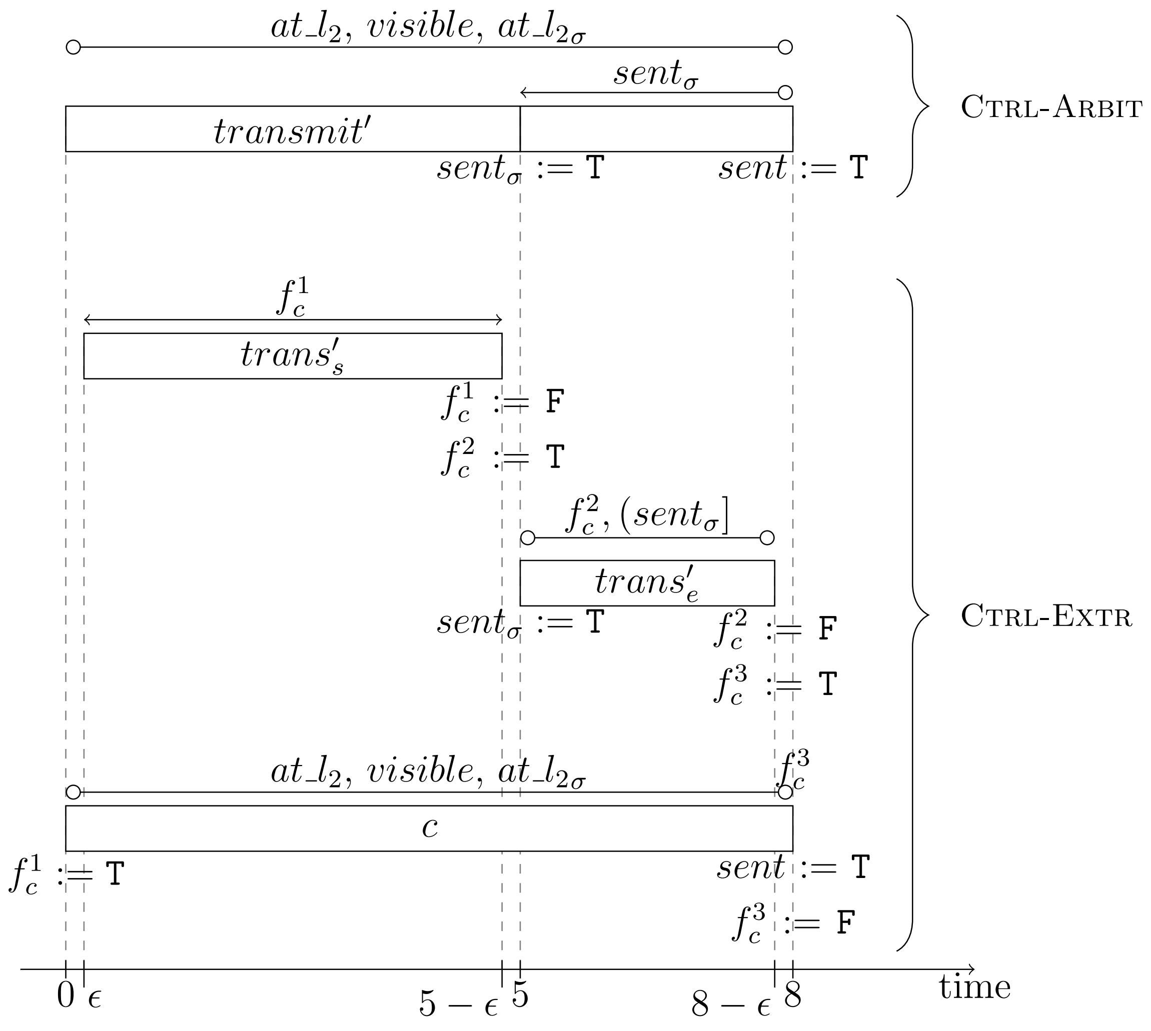


Figure 12.7: Container action construction example: the action *trans′* derived from the running example action *trans* by means of the compilation is rewritten as an equivalent triplet of actions.

features to force the planner into this choice. What we want to ensure is that the planner has no choice but to expand the clip just before the action terminates and to start the next piece before terminating the clip. This can be achieved by means of "exclusion literals". We introduce a fresh Boolean variable for each clip action, that is initially true and is falsified during the clip action. This literal is added as condition for starting and terminating every action in the problem except for the clip and the two clipped actions extremes. In this way, the planner has no choice but to exactly instantiate the two actions during the clip, greatly pruning the search space. The same idea can be applied also to the container construction by imposing mutual exclusion on the "holes" within the container action to force the planner to immediately expand the next action in the sequence. All these techniques have been implemented as a post-processing of the compilation result and are evaluated in the following sections.

12.6.3 Overall Results

Figure 12.8 gives an overview of the performance of the presented techniques in our experiments. We indicate with COLIN-Container the compilation approach solved with the Colin planner where we used the container technique with effect removal. COLIN-Container-Exclusion is again the compilation approach solved with the Colin planner where we used the container technique with effect removal, but adding the mutual exclusion literal to guide the solver. Similarly, COLIN-Clip and COLIN-Clip-Exclusion are the compilation technique solved by Colin using the clip construction to remove intermediate effects. The S-FSSTP techniques are referred to as TO-approach, LAD-approach and DR-approach, for the TO, LAD and DR techniques, respectively. First, we consider the direct approach obtained extending the FSSTP planning framework. We note how the LAD approach performs much better than TO approach, that in turn performs better than the DR approach. In fact, LAD it is able to solve almost twice as many instances as DR. However, we remark that the LAD and TO techniques are not complete and in some cases could spuriously detect that no plan exists. However, in our experiments this scenario never occurred and both the techniques always returned valid plans.

Concerning the compilation technique, we note how the performance dramatically depends on the technique used to compile away intermediate effects and conditions. The clip construction proved to be more effective

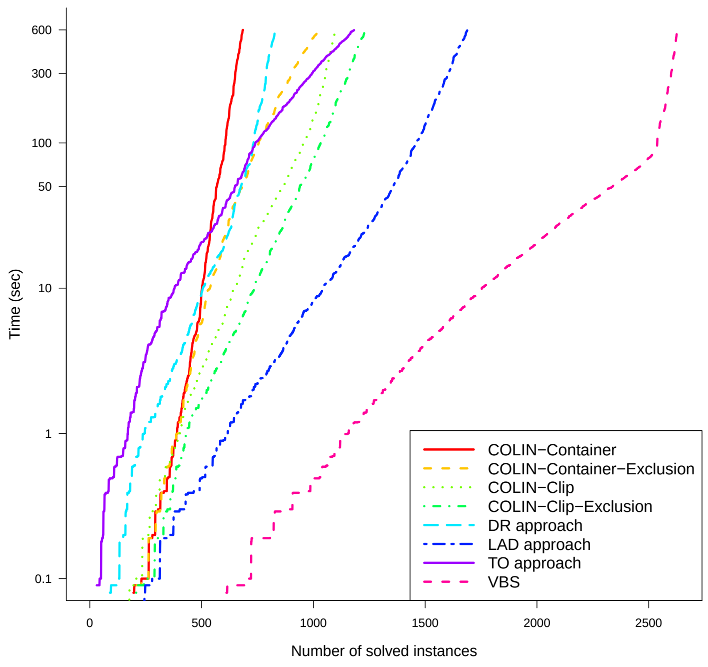


Figure 12.8: Log-scale cactus plot showing the performance of the solvers. The “VBS” line is the Virtual Best Solver that is computed taking for each instance the performance of the fastest solver among all the others.

than the container construction. Moreover, both the techniques benefit from the mutual exclusion improvement that prunes the search.

In order to compare the compilation technique with the direct approach, we reported in figure 12.9 the result of the best-performing compilation technique with the two native approaches. The situation is evidently split with no clear winner. This means that the two techniques exhibit a complementary behavior: on some instances the direct approach is vastly superior, on others it is beaten by the compilation. This is confirmed by the performance of the Virtual Best Solver (VBS) in figure 12.8: the performance of the VBS is obtained by taking the results of the fastest solver for each instance. The VBS line in the plot solves far more instances than any actual solver; hence, the various solvers that contribute to the VBS are

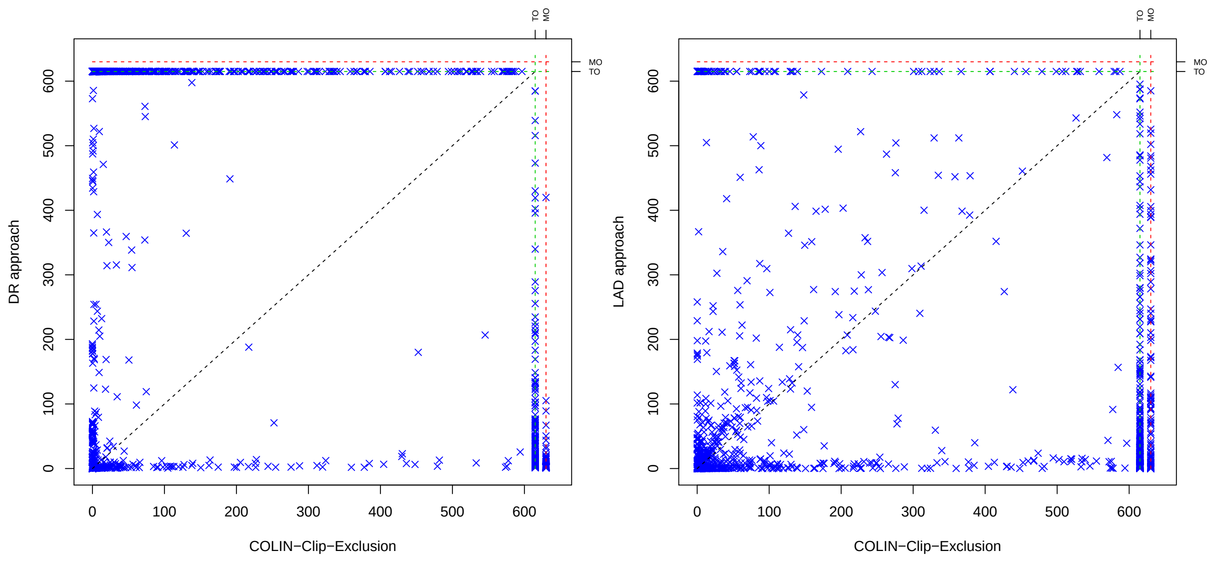


Figure 12.9: Scatter plot comparing the best technique for solving the compilation approach with the complete DR method (a) and the incomplete LAD method (b).

able to solve different sets of instances.

We analyzed several features of the instances without finding a clear correlation with the better solver. We leave the use of automated data mining techniques for the best solver prediction as future work.

12.6.4 Impact of Simplifications

We implemented the simplifier following the maximal-duration simplification described in section 12.5. Since the simplification requires a mutual exclusion generator to operate, we provided a simple generator based on syntactical rules. Moreover, to strengthen the exclusion predicated we manually checked for all the domains that all the uncontrollable actions were mutually exclusive with themselves, and we forced this information in the mutual-exclusion generator.

The results are reported in figure 12.10. We differentiate a technique from its version in which simplification is applied as pre-processing by

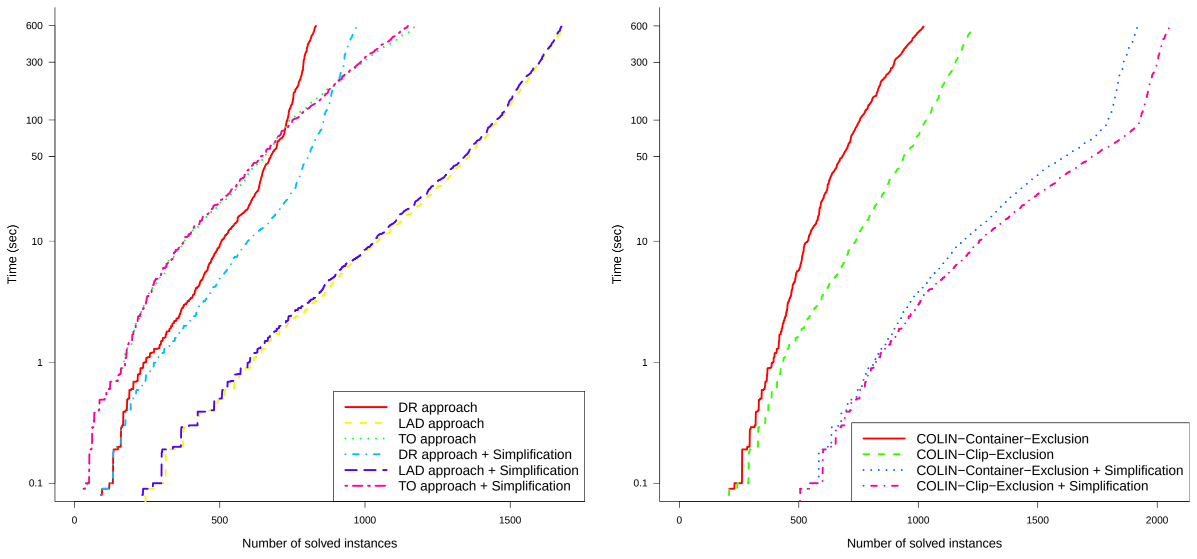


Figure 12.10: Log-scale cactus plot showing the impact of simplification in our experiments.

adding a "+ Simplification" to the technique name.

The simplification pre-processing turns out to be useful in all the cases except for the LAD and TO approaches where it is not beneficial nor detrimental. This is because there is little overhead in solving a STNU with respect to a DTNU and the simplification technique, removing some of the uncertainty, only changes some time points from uncontrollable to controllable by keeping the same constraint structure.

We also analyzed in detail the performance of the best-performing solvers of compilation and direct approach and we report the scatter plots in figure 12.11. Also in this case, there is an evident complementary behavior of the two techniques.

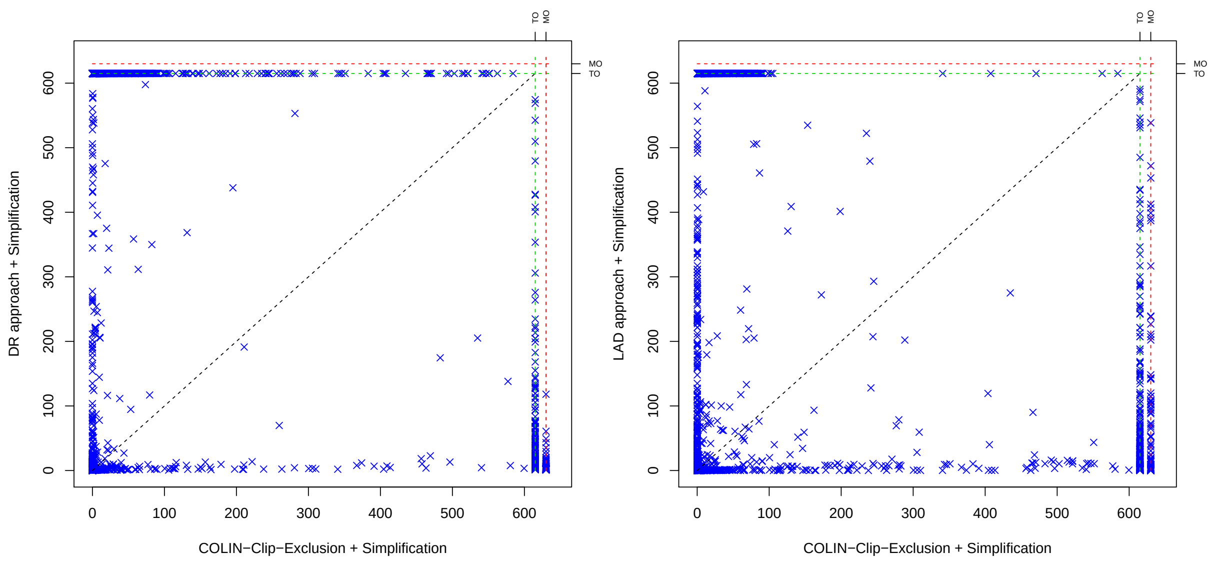


Figure 12.11: Scatter plots comparing the two best complete techniques.

Chapter 13

Timelines with temporal uncertainty

Differently from action-based languages, timeline-based formalisms represent the planning problem as a sort of sequential constraint problem. Many practical planners use this representation, because it is very powerful and compositional.

To the best of our knowledge, no work addresses the problem of uncontrollable durations in timeline-based formalisms: existing planners are either employed using re-planning when the contingencies gets unaligned with the model, or rely on flexible solutions.

We propose a principled, dedicated formalism that includes support for uncontrollable durations. We first define the general planning problem. Then, similarly to SATPlan approaches, we formalize the bounded planning problem given a time horizon and we give a theoretical, sound and complete encoding in First Order Logic modulo the $\mathcal{LRA}$ theory. The encoding is intended to clarify the semantics and to give a baseline for future approaches, but it is solvable by any SMT solver supporting the (quantified) $\mathcal{LRA}$ theory.

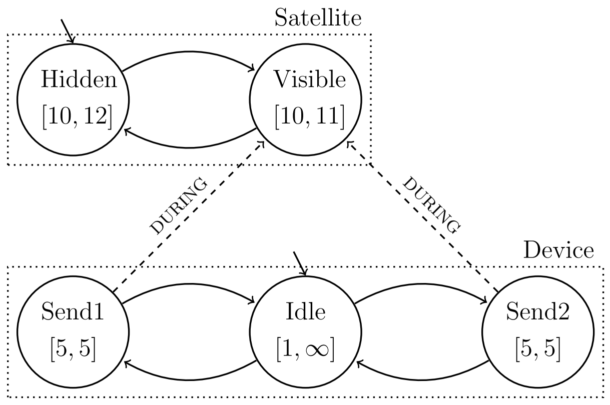


Figure 13.1: Timeline Running Example. The system is modeled as two generators: each state is a value with duration specified in brackets. Synchronizations are shown as dashed arrows labeled with Allen relations.

13.1 Problem Definition

In this section, we define the abstract syntax and semantics of a timeline-based language supporting temporal uncertainty in the interval durations. The language is similar to the APSI [CCF$^+$09] and NDDL [BWMB$^+$05] formalisms, but here we focus on the temporal uncertainty, disregarding advanced features such as resources or token parameters.

As an example, consider a communication device that can send data packets of two different types to a satellite during the time period in which the satellite is visible. The visibility window of the satellite is not controllable by the communication device and it ranges between 10 and 11 hours, while the satellite remains hidden in the following 10-12 hours (also uncontrollably). The device needs 5 hours to send each packet of data and a transmission has to happen *during* the visibility window. Notice that both the satellite and the device can be in each state more than once. The satellite is initially hidden, and the device is idle. The goal is to send one data packet per type. The situation is depicted in figure 13.1.

Syntax. We introduce an abstract notation for timeline-based domain descriptions. We retain all the features of the concrete languages used in timelines applications. Intuitively, the timeline framework can be thought of as a "sequential version" of Allen's algebra, where the same activity can be instantiated multiple times. The instantiations are obtained by means of generators.

Definition 68. *A* generator *G is a tuple $\langle V, T, \delta\rangle$ such that V is a finite set of values, $T \subseteq V \times V$ is a transition relation, δ is a temporal labeling function associating to each value $v \in V$ an interval of possible durations $[l, u]$.*

A generator represents a *state variable* over values V in a timeline framework[1]. The transition relation T is used to logically describe the evolution of the generator. If $T(v_i, v_j)$, then the end of (an instance of) activity v_i can be followed by the start of (an instance of) activity v_j. In our satellite example, the system is composed of two generators: the Satellite and the Communicator. The Satellite generator has values $\{Visible, Hidden\}$, the transition relation imposes the alternation of $Visible$ and $Hidden$ values, and the δ function imposes the minimal and maximal duration of each value ($Visible \rightarrow [10, 11)$, $Hidden \rightarrow [10, 12)$). The Communicator generator is three-valued ($Idle, Send1, Send2$), the transition relation imposes the automaton shape depicted in figure 13.1 and the duration constraints are $Idle \rightarrow [1, \infty)$, $Send1 \rightarrow [5, 5)$ and $Send2 \rightarrow [5, 5)$.

In order to express the constraints between different generators, we introduce the notion of synchronization.

Definition 69. *Let $G_i \doteq \langle V_i, T_i, \delta_i\rangle$ be generators, with $i \in \{0, \ldots, n\}$. An n-ary* synchronization *σ is a triple $\langle\langle G_0, v_0\rangle, \{\langle G_1, v_1\rangle, \ldots, \langle G_n, v_n\rangle\}, C\rangle$, such that, for all $i \in \{0, \ldots, n\}$, $v_i \in V_i$, and C is a set of Allen constraints*

[1]Without loss of generality, we disregard the parametrization used in some timeline languages.

in the form $v_h \bowtie v_k$*, with* $h, k \in \{0, \dots, n\}$*, with* $\bowtie$ *being a metric Allen operator.*

The synchronizations are based on Allen's temporal operators applied to generator values. The interpretation, however, is quite different from [All83]. For example, "*Send*1 *during* $[0, \infty)$ *Visible*" means that *every* instance of *Send*1 occurs during *some* instance of *Visible*; and, similarly, "*Visible during*$^{-1}$ $[0, \infty)$ *Send*1" means that during *every* visibility window *some* *Send*1 occurs. Therefore, the algebraic properties of [All83] are not retained here. In figure 13.1, we indicated two synchronizations using dashed arrows. These synchronizations are used to require that the packet of data is sent *during* the visibility window.

A set of generators and a set of synchronizations are sufficient to define a planning domain. For what concerns the planning problem, we do not distinguish between facts and goals: we just require an execution that exhibits a set of (temporally-extended and temporally constrained) *facts*.

Definition 70. *Let* $G \doteq \langle V, T, \delta \rangle$ *be a generator. A* unary fact *is a tuple* $\langle G, v, I_s, I_e \rangle$*, where* $v \in V$ *and* I_s, I_e *are two closed intervals. Let* f_1 *and* f_2 *be two unary facts and let* $\bowtie$ *be a metric Allen operator. A* binary fact *is a constraint in the form* $f_1 \bowtie f_2$*.*

A unary fact prescribes the existence of a value v in the execution of G, that starts during I_s and ends during I_e. A binary fact is useful to impose constraints (e.g. precedence, containment) between the intervals in the corresponding unary facts. In our satellite example, we use two unary facts to force the initial condition of the system: $f_1 \doteq \langle \mathit{Satellite}, \mathit{Hidden}, [0, 0], [0, \infty) \rangle$ forces the satellite to be in *Hidden* state at 0. Similarly, $f_2 \doteq \langle \mathit{Communicator}, \mathit{Idle}, [0, 0], [0, \infty) \rangle$ constrains the initial state of the communicator to be idle.

Similarly, to express the goals we introduce g_1 and g_2 defined as follows.

$$g_1 \doteq \langle Communicator, Send1, [0, \infty), [0, \infty)\rangle$$

$$g_2 \doteq \langle Communicator, Send1, [0, \infty), [0, \infty)\rangle$$

The goals require the communicator to be eventually in $Send1$ state and in $Send2$ state. If we need to order the goals, prescribing that the packet 1 must be sent before packet 2, we can impose a binary fact $(g1\ before[0, \infty)\ g2)$. Notice that the goals are temporally extended, i.e. they do not simply require to reach a final condition, but allow constraints throughout the execution.

The above definitions characterize timelines in the classical sense. In order to deal with temporal uncertainty, we now introduce an annotation to distinguish controllable and uncontrollable elements. This is similar to the subdivision in controllable and uncontrollable durative actions we presented in the previous chapter, but here we also allow uncontrollable starting points, hence we end up with four possible cases.

Definition 71. *A* CU-annotation for a set of generators $\mathcal{G} \doteq \{\langle V_i, T_i, \delta_i\rangle \mid i \in [1, n]\}$ *is a function* $\beta : \mathcal{G} \times \bigcup_i V_i \to \{\textsc{c}, \textsc{u}\} \times \{\textsc{c}, \textsc{u}\}$. *A* CU-annotation for a set of synchronizations S *is a function* $\beta : S \to \{\textsc{c}, \textsc{u}\}$.

With a slight abuse of notation, we overload the β function. The U flag identifies an uncontrollable element, therefore the flagged time instant is not under the control of the agent. Instead, the C flag identifies controllable elements. Consider again the running example. If we flag both the states of the satellite with $(\textsc{u}, \textsc{u})$ and all the rest as controllable, we are modeling a situation in which the satellite visibility is not decidable by the communicator, the only possible assumption is the minimal and maximal durations. We now define what a planning problem is.

Definition 72. *Let $\mathcal{G}$ be a generator set, Σ a set of synchronizations over the generators in $\mathcal{G}$, $\mathcal{F}$ and $\mathcal{R}$ be sets of unary and binary facts, respectively. Let β be a CU-annotation. A* timeline controllability problem *P is a tuple $\langle \mathcal{G}, \Sigma, \mathcal{F}, \mathcal{R}, \beta \rangle$.*

In this work, possible solutions are time-triggered plans, defined as follows.

Definition 73. *A* time-triggered plan *is a (possibly infinite) sequence $\langle G_1, v_1, cmd_1, t_1 \rangle; \langle G_2, v_2, cmd_2, t_2 \rangle; \ldots$ where, for all $i \geq 1$, v_i is a value for G_i, $cmd_i \in \{\text{S}, \text{E}\}$, and $t_i \leq t_{i+1}$.*

Intuitively, at a specific time point, a time triggered plan may specify one or more start/end commands to be executed on a specific generator and value. This definition is syntactic; the executability of a time-triggered plan is defined at the semantic level. This plan formulation is completely analogous to the one in definition 58, we simply cast the same idea in the timeline framework. This definition allows for infinitely-long plans, but we will disregard infinite plans when we introduce the bounded planning problem.

Semantics. In the following we assume that a timeline description is given. We provide an interpretation of timelines by means of streams, i.e. possibly infinite sequences of time-labeled activity instances.

Definition 74. *Let $G \doteq \langle V, T, \delta \rangle$ be a generator. A* stream *S for G is a (possibly infinite) sequence $\langle v_1, d_1 \rangle; \langle v_2, d_2 \rangle; \ldots$ such that, for all $i \geq 1$, $v_i \in V$, $\langle v_i, v_{i+1} \rangle \in T$, $d_i \in \delta(v_i)$.*

Given a stream S, we use the following notation:

- $Value(S, i) \doteq v_i$;
- $StartTime(S, i) \doteq \sum_{j=1}^{i-1} d_j$;

- $EndTime(S,i) \doteq StartTime(S,i) + d_i$;
- $Interval(S,i) \doteq \langle StartTime(S,i), EndTime(S,i)\rangle$.

We can now define the compatibility of a stream with the problem constraints.

Definition 75. *Let $G_0, \ldots, G_n$ be generators, and let σ be a synchronization $\langle\langle G_0, v_0\rangle, \{\langle G_1, v_1\rangle, \ldots, \langle G_n, v_n\rangle\}, C\rangle$. For $0 \leq i \leq n$, let S_i be a stream for G_i. $\{S_0, \ldots, S_n\}$* fulfills *σ if and only if for all j_0 such that $(Value(S_0, j_0) = v_0)$, there exist $j_1, \ldots, j_n$ such that for every constraint $(v_h \bowtie v_k) \in C$, $Interval(S_h, j_h) \bowtie Interval(S_k, j_k)$ holds.*

Notice that, in general, n-ary synchronizations, cannot be expressed in terms of binary synchronizations only. This is true only in the case where each Allen constraint involves one value from G_0 and one from another G_i. In the case of constraints between G_i and G_j, with $i, j > 0$ a binding between the activities in G_i and G_j is introduced, but the binding is further constrained by G_0.

Definition 76. *Let G be a generator, and let S be a stream for G. S fulfills the unary fact $\langle G, v, I_s, I_e\rangle$ at i if and only if $Value(S,i) = v$, $StartTime(S,i) \in I_s$ and $EndTime(S,i) \in I_e$.*

Definition 77. *Let $f_1 \bowtie f_2$ be a binary fact, where $f_i \doteq (G_i, v_i, I_{s_i}, I_{e_i})$. Let S_1 and S_2 be streams for G_1 and G_2 respectively. S_1 and S_2 fulfill $f_1 \bowtie f_2$ if and only if S_1 fulfills f_1 at i_1, S_2 fulfills f_2 at i_2, and $Interval(S_1, i_1) \bowtie Interval(S_2, i_2)$.*

Definition 78. *A time-triggered plan $\langle G_1, v_1, cmd_1, t_1\rangle; \langle G_2, v_2, cmd_2, t_2\rangle; \ldots$ induces a stream S on $G \doteq \langle V, T, \delta\rangle$ if and only if for all $i \geq 1$, when $G = G_i$, there exists $j \geq 1$ such that :*

1. *if $cmd_i = \mathrm{S}$ then $StartTime(S,j) = t_i$;*

2. if $cmd_i = \textsc{e}$ then $EndTime(S, j) = t_i$.

Definition 79. *A time triggered plan $\langle G_1, v_1, cmd_1, t_1\rangle; \langle G_2, v_2, cmd_2, t_2\rangle; \ldots$ obeys a CU-annotation β if and only if for each $i \geq 1$:*

1. if $cmd_i = \textsc{s}$ then $\beta(G_i, v_i) \in \{\langle \textsc{c}, \textsc{c}\rangle, \langle \textsc{c}, \textsc{u}\rangle\}$;

2. if $cmd_i = \textsc{e}$ then $\beta(G_i, v_i) \in \{\langle \textsc{c}, \textsc{c}\rangle, \langle \textsc{u}, \textsc{c}\rangle\}$.

Intuitively, this means that each assigned time point is labeled as controllable.

Definition 80. *Let π be a time-triggered plan, β a CU-annotation and $\mathcal{G}$ the set of generators controlled by π. π is* complete *with respect to β if for each $G \in \mathcal{G}$ and for each stream $S \doteq \langle v_1, d_1\rangle; \langle v_2, d_2\rangle; \ldots$ of G induced by π and for each i:*

1. if $\beta(G, v_i) \in \{\langle \textsc{c}, \textsc{c}\rangle, \langle \textsc{c}, \textsc{u}\rangle\}$ then $\langle G, v_i, \textsc{s}, StartTime(S, i)\rangle \in \pi$;

2. if $\beta(G, v_i) \in \{\langle \textsc{c}, \textsc{c}\rangle, \langle \textsc{u}, \textsc{c}\rangle\}$ then $\langle G, v_i, \textsc{e}, EndTime(S, i)\rangle \in \pi$.

In other words, if π is complete, each controllable time point of an induced stream S is assigned by π.

Definition 81. *Given the CU-annotation β, a stream $\langle v_1, d_1\rangle; \langle v_2, d_2\rangle; \ldots$ for generator $G \doteq \langle V, T, \delta\rangle$ is said to* satisfy contingencies *of G if and only if for each $i \geq 1$, $v_i \in V$, $\langle v_i, v_{i+1}\rangle \in T$ and if $\beta(G_i, v_i) \in \{\langle \textsc{u}, \textsc{u}\rangle, \langle \textsc{u}, \textsc{c}\rangle\}$ then $d_i \in \delta(v_i)$.*

In other words, a stream satisfies the contingencies of a generator if it is compatible with the generator constraints on the uncontrollable values.

Definition 82 (Solution to strong controllability problem)**.** *A time-triggered plan π is a strong* solution *for $P \doteq \langle \mathcal{G}, \Sigma, \mathcal{F}, \mathcal{R}, \beta\rangle$ if and only if it obeys and is complete w.r.t β, and all the streams induced by π that are compatible with the consistencies of the generators in $\mathcal{G}$ and that fulfill the*

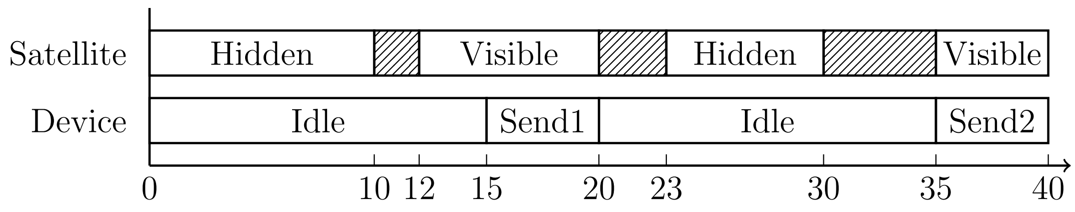


Figure 13.2: An execution of the satellite example that fulfills the problem constraints. The striped regions are uncertain: depending on the actual duration of the intervals the satellite can be either in Hidden or in Visible state.

synchronizations labeled as u, also fulfill each generator, the rest of Σ, $\mathcal{F}$ and $\mathcal{R}$.

Intuitively, we are searching for a plan that constrains the execution in such a way that for every possible evolution of the uncontrollable parts (fulfilling the assumed contingencies), all the problem constraints are satisfied.

In many practical problems, we are interested in finding solutions to a strong controllability problem within a given temporal horizon H.

Definition 83 (Bounded solution to strong controllability problem). *A finite time-triggered plan π is a* strong bounded solution *for $P \doteq \langle \mathcal{G}, \Sigma, \mathcal{F}, \mathcal{R}, \beta \rangle$ for a time horizon $H \in \mathbb{R}^+$ if and only if the following conditions hold:*

1. *π obeys and is complete with respect to β;*
2. *all the streams compatible with π finish after H;*
3. *each stream S that is compatible with the contingencies of the generators in $\mathcal{G}$ and that satisfies the synchronizations labeled as u, also satisfies the generator constraints, $\mathcal{F}$, $\mathcal{R}$, and the rest of Σ is satisfied for every interval of S that ends before H.*

Note that we chose to impose no constraint on intervals that end after the horizon, but other semantics are possible.

We highlight that searching for a time-triggered plan means searching for a fixed assignment of controllable decisions in time. For instance, in the satellite example it is possible to produce a time triggered plan for sending each packet once as shown in figure 13.2. However, it is not possible to send more packets, because the uncertainty in the satellite compresses the guaranteed visibility window. Consider again figure 13.2, the next guaranteed visibility window of the satellite would be $[58, 60)$ that is too short for sending another packet.

13.2 Bounded Encoding in FOL

We now reduce the problem of finding a solution for a bounded strong controllability problem to an SMT problem. Intuitively, we aim at finding a *finite* sequence of intervals, that completely covers the time-span between 0 and the horizon H, that fulfill all the problem constraints. Note that no synchronization constraints are imposed on intervals that end after the horizon bound.

We resort to an encoding that resembles a SAT-Plan [KS92] encoding of a planning problem: the underlying idea is to logically model a set of bounded streams and to impose the problem constraints on the streams. If the resulting formula is satisfiable, it means that a model for the formula codifies a stream that witnesses a solution for the original problem.

Let $H \in \mathbb{R}^+$ be the horizon, a generator $G \doteq \langle V, T, \delta \rangle$ is associated with a maximum number of intervals (assuming each $\delta(v) > 0$). A coarse upper bound M_G is given dividing H by the minimal duration associated with any value in V:

$$M_G \doteq \lceil \frac{H}{\min_{v \in V} start(\delta(v))} \rceil .$$

We use two set of variables for each generator G: $ValueOf^G(j)$ and $EndOf^G(j)$, whose interpretation defines the stream for G. $ValueOf^G(j)$

gives the value of the j-th interval, while $\mathit{EndOf}^G(j)$ encodes the end time point of the j-th interval. Thus, for each generator $G \doteq \langle V, T, \delta\rangle$, we can use M_G variables $ValueOf^G(j)$ ranging over the domain V, and M_G variables $\mathit{EndOf}^G(j)$ of type $\mathbb{R}^+$ to model a bounded stream that is guaranteed to cover the interval $[0, H]$.

$\mathit{EndOf}^G(j)$ defines time points in which the stream changes its value. Unfortunately, whether a time point is controllable or not cannot be detected statically in general. In fact, depending on the discrete path encoded in the assignments to $ValueOf^G(j)$, the j-th time point can be either controllable or uncontrollable. For this reason, we have to introduce M_G new variables, called $U^G(j)$, that model the uncertain values (analogous to $\mathit{EndOf}^G(j)$). In order to properly capture the strong controllability of the execution, we consider $\mathit{EndOf}^G(j-1)$ and $ValueOf^G(j)$ as existentially-quantified variables, and $U^G(j)$ as universally quantified variables. We indicate with U^G the set of all the $U^G(j)$ variables. In order to impose the proper constraints on either $\mathit{EndOf}^G(j)$ or $U^G(j)$ we have to condition the constraint on the controllability of the j-th interval that is decided at solving time. Therefore we introduce two macros $S^G(j, U^G)$ and $E^G(j, U^G)$ that encapsulate this conditioning and return the proper value that encodes the start or the end of the j-interval respectively. The first formula, $S^G(j, U^G)$, is defined as follows.

$$\begin{aligned} S^G(j, U^G) \doteq \mathit{ite}(j = 0, 0, \mathit{ite}(&\beta(G, ValueOf^G(j)) \in \{\langle \textsc{c}, \textsc{c}\rangle, \langle \textsc{c}, \textsc{u}\rangle\},\\ &\mathit{EndOf}^G(j-1),\\ &U^G(j-1))) \end{aligned}$$

Similarly, $E^G(j, U^G)$ is defined as

$$\mathit{ite}(\beta(G, ValueOf^G(j)) \in \{\langle \textsc{c}, \textsc{c}\rangle, \langle \textsc{u}, \textsc{c}\rangle\}, \mathit{EndOf}^G(j), U^G(j)) .$$

Let $Used^G(j)$ be the predicate defined as $E^G(j, U^G) \leq H$. The encoding is defined as follows. For each generator $G \doteq \langle V, T, \delta\rangle$, we define

$Value_G \doteq \bigwedge_{j=1}^{M_G} ValueOf^G(j) \in V$ to force the domain of $ValueOf^G(j)$ and $Trans_G \doteq \bigwedge_{j=1}^{M_G-1} T(ValueOf^G(j), ValueOf^G(j+1))$ to codify the transition relation of G.

We split the constraints encoding the interval durations in two distinct formulae as follows.

$$\begin{aligned}\Gamma_G(U^G) \doteq {} & \textstyle\bigwedge_{j=1}^{M_G}((\beta(G, ValueOf^G(j)) \in \{\langle \text{C}, \text{U}\rangle, \langle \text{U}, \text{U}\rangle\}) \rightarrow \\ & (E^G(j, U^G) - S^G(j, U^G) \in \delta(ValueOf^G(j)))) \\ \Psi_G(U^G) \doteq {} & \textstyle\bigwedge_{j=1}^{M_G}((\beta(G, ValueOf^G(j)) \in \{\langle \text{C}, \text{C}\rangle, \langle \text{U}, \text{C}\rangle\}) \rightarrow \\ & (E^G(j, U^G) - S^G(j, U^G) \in \delta(ValueOf^G(j))))\end{aligned}$$

For every uncontrollable synchronization

$$\sigma \doteq \langle\langle G_0, v_0\rangle, \{\langle G_1, v_1\rangle, \ldots, \langle G_n, v_n\rangle\}, C\rangle$$

with $(\beta(\sigma) = \text{U})$, we define $\Gamma_\sigma(U^{G_0}, \ldots, U^{G_n})$ as follows.

$$\begin{aligned}& \textstyle\bigwedge_{j_0=1}^{M_{G_0}}(ValueOf^{G_0}(j_0) = v_0 \wedge Used^{G_0}(j_0)) \rightarrow \\ & \quad (\textstyle\bigvee_{j_1=1}^{M_{G_1}} \left(ValueOf^{G_1}(j_1) = v_1 \wedge Used^{G_1}(j_1)\right) \wedge \ldots \\ & \quad\quad (\textstyle\bigvee_{j_n=1}^{M_{G_n}}(ValueOf^{G_n}(j_n) = v_n \wedge Used^{G_n}(j_n)) \wedge \\ & \quad\quad\quad \textstyle\bigwedge_{v_k \bowtie v_h \in C} \xi(\bowtie, S^{G_k}(j_k, U^{G_k}), E^{G_k}(j_k, U^{G_k}), \\ & \quad\quad\quad\quad S^{G_h}(j_h, U^{G_h}), E^{G_h}(j_h, U^{G_h}))) \ldots)\end{aligned}$$

Where $\xi(\bowtie, s_1, e_1, s_2, e_2)$ is the $\mathcal{LRA}$ encoding of the Allen constraint $I_1 \bowtie I_2$ with the interval I_i being $[s_i, e_i]$.

We also define $\Psi_\sigma(U^{G_0}, \ldots, U^{G_n})$ in the very same way for each controllable synchronization $(\beta(\sigma) = \text{C})$. The formula encoding unary facts is obtained by imposing the existence of a compatible interval in the considered stream. For each unary fact $f \doteq \langle G, v, I_s, I_e\rangle$ we define $\Psi_f(U^G)$ as:

$$\Psi_f(U^G) \doteq \bigvee_{j=1}^{M_G} Fact(U^G, j)$$

where $Fact(U^G, j)$ is defined as:

$$Used^G(j) \wedge (ValueOf^G(j) = v) \wedge S^G(j, U^G) \in I_s \wedge E^G(j, U^G) \in I_e\ .$$

For every binary fact requirement $r \doteq f_1 \bowtie f_2$, where $f_i \doteq \langle G_i, v_i, I_{s_i}, I_{e_i}\rangle$ we define $\Psi_r(U^{G_1}, U^{G_2})$ as $\bigvee_{j_1=1}^{M_{G_1}} \bigvee_{j_2=1}^{M_{G_2}}(Fact(U^{G_1}, j_1) \wedge Fact(U^{G_2}, j_2) \wedge \xi(\bowtie, S^{G_1}(j_1, U^{G_1}), E^{G_1}(j_1, U^{G_1}), S^{G_2}(j_2, U^{G_2}), E^{G_2}(j_2, U^{G_2})))$.

Finally, let Σ_u be the subset of Σ of the uncontrollable synchronizations and let Σ_c be Σ/Σ_u. The overall encoding for the problem is:

$$\begin{aligned}
&\textstyle\bigwedge_{G\in\mathcal{G}} Value_G \wedge \bigwedge_{G\in\mathcal{G}} Trans_G\ \wedge\ \forall U^{G_0}, \ldots, U^{G_n}.\\
&((\textstyle\bigwedge_{G\in\mathcal{G}} \Gamma_G(U^G) \wedge \bigwedge_{\sigma\in\Sigma_u} \Gamma_\sigma(U^{G_0}, \ldots, U^{G_n})) \rightarrow\\
&(\textstyle\bigwedge_{G\in\mathcal{G}} \Psi_G(U^G) \wedge \bigwedge_{\sigma\in\Sigma_c} \Psi_\sigma(U^{G_0}, \ldots, U^{G_n})\wedge\\
&\textstyle\bigwedge_{f=\langle G,v,I_s,I_e\rangle\in\mathcal{F}} \Psi_f(U^G)\wedge\\
&\textstyle\bigwedge_{r=\langle G_1,v_1,I_{s_1},I_{e_1}\rangle\bowtie\langle G_2,v_2,I_{s_2},I_{e_2}\rangle\in\mathcal{R}} \Psi_r(U^{G_1}, U^{G_2}))).
\end{aligned}$$

The universal quantification captures the "universality" of the solution: for each possible allocation of the uncontrollables, given by $U^{G_0}, \ldots, U^{G_n}$, we impose that the contingent part of the problem implies the requirements. The encoding admits a model if and only if there exist a bounded solution to the original problem and the model can be used to build a complete time-triggered plan for the original bounded strong controllability problem. This formula is a first-order quantification over a finite set of real variables. Therefore, it can be decided by a SMT($\mathcal{LRA}$) solver equipped with a quantifier elimination procedure.

13.3 Discussion

The first-order encoding we reported is sound and complete for the bounded-horizon problem. However, the approach is quite naive and preliminary experiments show it does not scale well even on very small problems. This is due to cost of quantifier elimination: the universal quantifier modeling the uncontrollable behaviors covers the entire encoding, requiring a huge monolithic elimination to be handled.

We remark that the value of the encoding is to give a base-line for future improvements and to clarify the semantics of the problem.

We think that two directions are promising for extending this work. First, the encoding is similar to SAT-Plan, hence we can try to adapt the different optimizations that have been proposed in classical planning to improve scalability [Rin12]. A second direction is to extend the plan-space search approaches that are commonly used in timeline-based planners to deal with uncertainty. This is similar but different to the work we did to define the S-FSSTP approaches, because the operators in the plan-space search framework are radically different. Nonetheless, we imagine that similar issues to the one we addressed in S-FSSTP will arise.

Strong Temporal Planning with Uncontrollable Durations Conclusions

In this part, we pushed the temporal planning techniques to the "Duration-only Plant, Time-Strong Executor" cell of table 3.2. We analyzed in detail the case of action-based formalisms and started extending the concepts to the timeline languages. The work primarily concentrated in developing techniques that exploit the experience gained when dealing with Temporal Networks with Uncertainty. In fact, the DR approach directly exploits the strong controllability encodings we developed, the compilation approach generalizes to real of planning the static elimination techniques for STNU and TCSNU.

This work can be extended in many different directions. First, we plan to continue the research line in timeline planning, proposing practical algorithms for dealing with the strong planning problem. With this respect a promising idea is to work on extending Plan-Space Search techniques analogously to what we did for FSSTP.

Second, resources are not considered by the current approach. As we explained in chapter 6, resources can also be a source of uncertainty much like the action duration and this is currently not supported by our approaches. Another direction that is very important is the support for continuous change (sometimes called hybrid planning).

An orthogonal future work is surely to push these techniques towards dynamic planning (The “Duration-only Plant, Time-Dynamic Executor” cell of table 3.2). This step is not just a matter of substituting the strong controllability solver with a dynamic controllability technique: in planning, if the executor is allowed to observe at runtime, it is also allowed to execute different sets of actions depending on an observation. Hence, the shape of plans is no longer linear, but is rather a strategy (analogously to contingent planning in the non-temporal case). Nonetheless a middle-ground between these techniques and dynamic planning with duration uncertainty can be found and the development of dynamic planning techniques is surely a very important open problem.

Chapter 14

Thesis Conclusion

In this thesis, we analyzed the problem of planning and scheduling in presence of temporal uncertainty from different perspectives. First, we rationalized the problem by proposing a novel classification schema based on the interplay of an abstract plan executor and a plant. The derived classification table (table 3.2) is used as a guide in the landscape of scheduling and planning problems.

We started by surveying the state-of-the-art concerning temporal networks scheduling and temporal planning, then we contributed in several directions.

1. We improved the state-of-the-art on the problem of strong controllability for DTNU using a set of novel encodings of the problems in the SMT framework.

2. We tackled the weak controllability problem for DTNU, proposing a number of algorithms for strategy synthesis that exploit SMT solvers for quantitative reasoning as well as encodings for deciding if a given network is weakly controllable.

3. We discussed the open problem of dynamic controllability for DTNU, showing a reduction from dynamic controllability problem of temporal

networks to reachability game in a Timed Game Automaton and also dedicated synthesis and validation algorithms.

4. In the planning context, we dealt with the problem of Strong Temporal Planning with Uncontrollable Durations in action-based languages, proposing a portfolio of techniques to deal with the landscape of sub-classes of the problem.

5. Finally, we defined the strong temporal planning problem for timeline-based planning, providing a novel formalization of the problem and a theoretical formalization of a bounded-horizon solution.

14.1 Future Work

There are several directions for extending this work.

First, it would be important to go beyond strong planning looking into dynamic planning (i.e. considering the next column of table 3.2). This is surely important for practical applications, but is non-trivial. In fact, if we allow the executor to observe the duration of actions at run-time, then different sets of actions may be chosen according to different observation. In this respect, a dynamic plan would be similar to Conditional Planning in Non-Deterministic domains [CPRT03]. Moreover, as we discussed in section 3.6.1, it would be interesting to explore the middle-ground between strong and dynamic controllability and between dynamic controllability and weak controllability (also in the planning case).

As a second direction, in both the scheduling and planning parts we disregarded resources (controllable or even uncontrollable), but resources are extremely important in practical applications and are widely studied in the scheduling community. Extending our scheduling techniques for dealing with resources is possible, but presents some challenges: in some parts of our reasoning we exploit the fact that temporal constraints are limited to a specific form in $\mathcal{QF\_RDL}$, but we often use very expressive SMT theories, such as $\mathcal{QF\_LRA}$, that can accommodate at least some classes of resources. Similarly, our planning approaches can be extended to deal with resources.

Third, in this thesis work we focused on the problems of schedulability check, strategy synthesis and plan generation, but other important problems are open in the context of planning and scheduling under temporal uncertainty.

Plan validation is the problem of checking the validity of a given plan against a planning instance. It is a fundamental problem because it allows the cross-checking of the synthesis tools correctness and the validity assessment of hand-written plans. The plan validation problem is relatively easy in the context of plain temporal planning: it suffices to simulate the domain controlled by the plan and ensure that no action condition is violated and that the goals are satisfied by the execution trace. When we add uncertainty to the domain, however, the problem becomes much harder as the validation must consider all the possible evolutions of the environment and ensure that the plan is valid for each of them. In this sense, the problem can be seen as a model-checking problem of a timed system.

As any other model-based technique, also planning results are as good as the model provided in input, for this reason any technique that goes in the direction of improving models is important for the development of the field. Domain validation is the problem of checking the planning instance specification against a set of desired and undesired behaviors to improve the confidence in the adherence of the specification with the modeled reality. This is traditionally treated as a model checking problem for the case without uncertainty: a set of formal properties are checked against the planning specification. When we add uncertainty, the issue becomes much harder as we are formally checking a set of properties on a two-party game formalism. Moreover, if we consider continuous resources in the problem specification, the validation problem enters the realm of Hybrid Automata verification that is largely an open field. We believe that these issues are largely under-estimated in the planning community and require a deeper investigation.

Fourth, the work in this thesis is focused on finding a solution (if any) to the problems we addressed, but, as in many other fields of computer science, when multiple solutions are possible the issue of finding the best solution according so some criteria arises. This is true for scheduling, where we might seek the optimal scheduling strategy, but also for planning if we look for an optimal plan (e.g. the shortest possible plan).

Fifth, the non-deterministic rows of table 3.2 are very important for the practical applications, but are currently uncovered by existing techniques.

Finally, there is the never-ending quest for performance: all the techniques we presented have worst-case exponential computational complexity (because the addressed problems are not polynomial), hence scalability is the main issue for the adoption of such techniques in practical applications. All the empirical tests we presented go in this direction: we always try to push the performance of our technique as much as possible.

Bibliography

[ACG99] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In *ECP*, pages 97–108, 1999.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AF92] David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8(1):295–313, 1992.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832–843, 1983.

[ATZ04] Douglas Aberdeen, Sylvie Thiébaux, and Lin Zhang. Decision-theoretic military operations planning. In *ICAPS*, pages 402–412, 2004.

[BA01] Fahiem Bacchus and Michael Ady. Planning with resources and concurrency: A forward chaining approach. In *IJCAI*, pages 417–424, 2001.

[Bäc98] Christer Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

[BCC$^+$03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[BCD$^+$07] Gerd Behrmann, Agns Cougnard, Alexandre David, Emmanuel Fleury, KimG. Larsen, and Didier Lime. Uppaal-Tiga: Time for playing games! In *CAV*, pages 121–125. 2007.

[BCF$^+$08] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In *CAV*, pages 299–303, 2008.

[BD11] James Boerkoel and Ed Durfee. Challenges in maintaining minimal, decomposable disjunctive temporal problems. In *IJCAI - AILog Workshop*, pages 494–502, 2011.

[BDM$^+$02] John L. Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David E. Smith, and Richard Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *UAI*, pages 77–84, 2002.

[BDM$^+$13] Clark Barrett, Morgan Deters, Leonardo Moura, Albert Oliveras, and Aaron Stump. 6 years of SMT-COMP. *Journal of Automated Reasoning*, 50:243–277, 2013.

[Ben02] Johan Bengtsson. *Clocks, DBM, and States in Timed Systems.* PhD thesis, Uppsala University, 2002.

[BF97] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[BL11] Bahareh Badban and Martin Lange. Exact incremental analysis of timed automata with an SMT-solver. In *Formal Modeling and Analysis of Timed Systems*, pages 177–192. 2011.

[BPST10] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In *TACAS*, pages 150–153, 2010.

[BS11] Sara Bernardini and David E. Smith. Automatic synthesis of temporal invariants. In *SARA*, 2011.

[BSST09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.

[BST^{+}10] Clark Barrett, Aaron Stump, Cesare Tinelli, Sascha Boehme, David Cok, David Deharbe, Bruno Dutertre, Pascal Fontaine, Vijay Ganesh, Alberto Griggio, Jim Grundy, Paul Jackson, Albert Oliveras, Sava Krstić, Michal Moskal, Leonardo De Moura, Roberto Sebastiani, To David Cok, and Jochen Hoenicke. The SMT-LIB standard: Version 2.0. Technical report, 2010.

[Bul12] Peter Bulychev. The uppaal pydbm library — http://people.cs.aau.dk/ adavid/udbm/python.html, 2012.

[BWMB^{+}05] Tania Bedrax-Weiss, Conor McGann, Andrew Bachmann, Will Edgington, and Michael Iatauro. Europa2: User and contributor guide. Technical report, NASA Ames Research Center, 2005.

[CCF^{+}09] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, Angelo Oddi, and Riccardo Rasconi. The APSI framework: a planning and scheduling software development environment. In *ICAPS - Application Showcase Program*, Thessaloniki, Greece, 2009.

[CCFL10] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *ICAPS*, pages 42–49, 2010.

[CCFL12] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, 44:1–96, 2012.

[CCO^{+}12] Amanda Jane Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez Celorrio, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1), 2012.

[CDF^{+}05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, pages 66–80, 2005.

[CFH^{+}09] Andrew Coles, Maria Fox, Keith Halsey, Derek Long, and Amanda Smith. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44, 2009.

[CFLS08] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. Planning with problems requiring temporal coordination. In *AAAI*, pages 892–897, 2008.

[CG67] Harold S. M. Coxeter and Samuel L. Greitzer. Collinearity and concurrence. In *Geometry Revisited*, pages 51–79. Mathematical Association of America Textbooks, 1967.

[CGSS13] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *TACAS*, 2013.

[CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[CHM$^+$14] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *TIME*, pages 27–36, 2014.

[CHM$^+$16] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Informatica*, page to appear, 2016.

[CHMR14] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, and Marco Roveri. Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In *AAAI*, pages 2242–2249, 2014.

[CKMW07] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. When is temporal planning really temporal? In *IJCAI*, pages 1852–1859, 2007.

[CM06] Scott Cotton and Oded Maler. Fast and flexible difference constraint propagation for dpll(t). In *SAT*, pages 170–183, 2006.

[CMR12a] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving temporal problems using SMT: strong controllability. In *CP*, pages 248–264, 2012.

[CMR12b] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving temporal problems using SMT: weak controllability. In *AAAI*, 2012.

[CMR13] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Timelines with temporal uncertainty. In *AAAI*, pages 195–201, 2013.

[CMR14] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving strong controllability of temporal problems with uncertainty using SMT. *Constraints*, 2014.

[CMR15a] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. An SMT-based approach to weak controllability for disjunctive temporal problems with uncertainty. *Artificial Intelligence*, 224:1–27, 2015.

[CMR15b] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Strong temporal planning with uncontrollable durations: A state-space approach. In *AAAI*, pages 3254–3260, 2015.

[CMR16] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *AAAI*, page to appear, 2016.

[CO96] Amedeo Cesta and Angelo Oddi. Gaining efficiency and flexibility in the simple temporal problem. In *TIME*, pages 45–50, 1996.

[CPRT03] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[CRK$^+$00] Steve Chien, Gregg Rabideau, Russell Knight, Robert Sherwood, Barbara Engelhardt, Darren Mutz, Tara Estlin, Benjamin Smith, Forest Fisher, et al. Aspen-automating space mission operations using automated planning and scheduling. In *SpaceOps*, 2000.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[CWwH06] Yixin Chen, Benjamin W. Wah, and Chih wei Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research*, 26:323–369, 2006.

[CYF$^+$15] Jing Cui, Peng Yu, Cheng Fang, Patrik Haslum, and Brian C. Williams. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *ICAPS*, pages 52–60, 2015.

[DBMIG14] Filip Dvorak, Arthur Bit-Monnot, Felix Ingrand, and Malik Ghallab. A flexible anml actor and planner in robotics. In *ICAPS - Planning and Robotics Workshop*, 2014.

[DdM06a] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94, 2006.

[DdM06b] Bruno Dutertre and Leonardo Mendonça de Moura. The Yices SMT solver. Tool paper at `http://yices.csl.sri.com/tool-paper.pdf`, 2006.

[DK03] Minh Binh Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, 2003.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of ACM*, 5(7):394–397, 1962.

[dlT90] Thierry de la Tour. Minimizing the number of clauses by renaming. In Mark Stickel, editor, *CADE*, volume 449 of *LNCS*, pages 558–572. Springer, 1990.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[DMP91a] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[DMP91b] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[EH04] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, 2004.

[FJ03] Jeremy Frank and Ari Jónsson. Constraint-based attribute and interval planning. *Constraints*, 8(4):339–364, 2003.

[FL03] Maria Fox and Derek Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[FL06] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006.

[FLH04] Maria Fox, Derek Long, and Keith Halsey. An investigation into the expressive power of PDDL2.1. In *ECAI*, pages 328–342, 2004.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(34):189–208, 1971.

[GFL02] Antonio Garrido, Maria Fox, and Derek Long. A temporal planning system for durative actions of PDDL2.1. In *ECAI*, pages 586–590, 2002.

[GHL$^+$09] Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[GK97] B. Cenk Gazen and Craig A. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Sam Steel and Rachid Alami, editors, *ECP - Recent Advances in AI Planning*. Springer-Verlag, New York, 1997.

[GL94] Malik Ghallab and Hervé Laruelle. Representation and control in ixtet, a temporal planner. In *AIPS*, pages 61–67, 1994.

[GM15] Marco Gario and Andrea Micheli. pySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *SAT - SMT Workshop*, 2015.

[GNT04] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[GSS03] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

[Gt12] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012. `http://gmplib.org/`.

[HA96] Robert B. Hughes and Michael R. Anderson. Simplexity of the cube. *Discrete Mathematics*, 158(13):99–150, 1996.

[HG01] Patrik Haslum and Héctor Geffner. Heuristic planning with time and resources. In *ECP*, 2001.

[HPC12] Luke Hunsberger, Roberto Posenato, and Carlo Combi. The dynamic controllability of conditional STNs with uncertainty. In *ICAPS - PlanEx Workshop*, pages 1–8, 2012.

[Hun09] Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *TIME*, pages 155–162, 2009.

[Hun10a] Luke Hunsberger. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *TIME*, pages 121–128, 2010.

[Hun10b] Luke Hunsberger. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *TIME*, pages 121–128, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[Hun13] Luke Hunsberger. A faster execution algorithm for dynamically controllable stnus. In *TIME*, pages 26–33, 2013.

[Hun14] Luke Hunsberger. A faster algorithm for checking the dynamic controllability of simple temporal networks with uncertainty. In *ICAART*, 2014.

[IG14] Felix Ingrand and Malik Ghallab. Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 2014.

[KD00] Jonas Kvarnström and Patrick Doherty. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):119–169, 2000.

[Kes96] Christoph W. Kessler. Parallel fourier-motzkin elimination. In *Euro-Par*, pages 66–71, 1996.

[KJN12] Roland Kindermann, Tommi Junttila, and Ilkka Niemelä. Beyond lassos: Complete SMT-based bounded model checking for timed automata. In *Formal Techniques for Distributed Systems*, pages 84–100. 2012.

[Kle67] Stephen C. Kleene. *Mathematical Logic*. J. Wiley & Sons, 1967.

[KS92] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

[KSJ09] Hyondeuk Kim, Fabio Somenzi, and HoonSang Jin. Efficient Term-ITE conversion for satisfiability modulo theories. In *SAT*, pages 195–208, 2009.

[LAT05] Iain Little, Douglas Aberdeen, and Sylvie Thiébaux. Prottle: A probabilistic temporal planner. In *AAAI*, pages 1181–1186, 2005.

[LF03] Derek Long and Maria Fox. Exploiting a graphplan framework in temporal planning. In *ICAPS*, pages 52–61, 2003.

[LP98] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theorey of Computation.* Prentice-Hall, Inc., 2 edition, 1998.

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *Computer Journal*, 36(5):450–462, 1993.

[McD00] Drew V. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[MDS15] Andrea Micheli, Minh Do, and David E. Smith. Compiling away uncertainty in strong temporal planning with uncontrollable durations. In *IJCAI*, 2015.

[MM05] Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *AAAI*, pages 1193–1198, 2005.

[MMV01] Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI*, pages 494–502, 2001.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *DAC*, pages 530–535, 2001.

[MNPW98] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[Mon08] David Monniaux. A quantifier elimination algorithm for linear real arithmetic. In *LPAR*, pages 243–257, 2008.

[Mor06] Paul Morris. A structural characterization of temporal dynamic controllability. In *CP*, pages 375–389, 2006.

[Mor14] Paul Morris. Dynamic controllability and dispatchability relationships. In Helmut Simonis, editor, *CPAIOR*, volume 8451 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2014.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS*, pages 229–242, 1995.

[MSCD91] Nicola Muscettola, Stephen F. Smith, Amedeo Cesta, and Daniela D'Aloisi. Coordinating space telescope operations in an integrated planning and scheduling architecture. In *ICRA*, 1991.

[MW08] Mausam and Daniel S. Weld. Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*, 31:33–82, 2008.

[PdWvdK12] Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of Artificial Intelligence Research*, 43:353–388, 2012.

[Ped89] Edwin P. D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *KR*, pages 324–332, 1989.

[PVYS07] Bart Peintner, Kristen B. Venable, and Neil Yorke-Smith. Strong controllability of disjunctive temporal problems with uncertainty. In *CP*, pages 856–863, 2007.

[PW94] J. Scott Penberthy and Daniel S. Weld. Temporal planning with continuous change. In *AAAI*, pages 1010–1015, 1994.

[Rin12] Jussi Rintanen. Engineering efficient planners with SAT. In *ECAI*, pages 684–689, 2012.

[Rin15a] Jussi Rintanen. Discretization of temporal models with application to planning with SMT. In *AAAI*, pages 3349–3355, 2015.

[Rin15b] Jussi Rintanen. Models of action concurrency in temporal planning. In *IJCAI*, pages 1659–1665, 2015.

[RLT06] Silvio Ranise, Loria, and Cesare Tinelli. The SMT-LIB standard: Version 1.2. Technical report, 2006.

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming.* J. Wiley & Sons, 1998.

[SD05] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artificial Intelligence*, 166(1-2):194–253, 2005.

[SFC08] David E. Smith, Jeremy Frank, and William Cushing. The ANML language. In *ICAPS - Poster session*, 2008.

[SK00] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[Smi03] David E. Smith. The case for durative actions: A commentary on PDDL2.1. *Journal of Artificial Intelligence Research*, 20:149–154, 2003.

[ST12] Roberto Sebastiani and Silvia Tomasi. Optimization in SMT with $\mathcal{LA}(\mathbb{Q})$ cost functions. In *IJCAR*, pages 484–498, 2012.

[SV98] Eddie Schwalb and Lluís Vila. Temporal constraints: A survey. *Constraints*, 3(2/3):129–149, 1998.

[SW99] David E. Smith and Daniel S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI*, pages 326–337, 1999.

[TP03] Ioannis Tsamardinos and Martha E Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1–2):43–89, 2003.

[TVP03] Iohannis Tsamardinos, Thierry Vidal, and Martha Pollack. Ctp: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003.

[VF99a] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[VF99b] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[VPC90] Manuela M. Veloso, M. Alicia Perez, and Jaime G. Carbonell. Nonlinear planning with parallel resource allocation. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 207–212. Morgan Kaufmann, 1990.

[VPL10] Gérard Verfaillie, Cédric Pralet, and Michel Lemaître. How to model planning and scheduling problems using constraint networks on timelines. *Knowledge Engineering Review*, 25(3):319–336, 2010.

[VVPYS10] Kristen Brent Venable, Michele Volpato, Bart Peintner, and Neil Yorke-Smith. Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *ICAPS - COPLAS Workshop*, pages 50–59, 2010.

[Wil93] Doran K. Wilde. A library for doing polyhedral operations. Technical report, 1993.

[XC03] Lin Xu and Berthe Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *TIME*, pages 212–222, 2003.

[YS03] Håkan L. S. Younes and Reid G. Simmons. VHPOP: versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.

[ZDDD93] Monte Zweben, Eugene Davis, Brian Daun, and Michael Deale. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1588–1596, 1993.

Appendix A

Scheduling Proofs

A.1 Consistency Proofs

Theorem 8.1 (Switch Correctness)**.** *The temporal network P is consistent if and only if equation* (8.3) *is satisfiable (and a consistent schedule can be derived from any model of equation* (8.3)*).*

Proof. We prove that equation (8.3) is equi-satisfiable to equation (8.2) and that the model of equation (8.3) is always an extension of a model of equation (8.2) from which a consistent schedule can be extracted.

First we prove that a model μ of equation (8.2) can be extended to a model μ' of equation (8.3). For each i, there exists $\bar{\jmath}$ such that $((x_{i,\bar{\jmath}} - y_{i,\bar{\jmath}}) \geq \ell_{i,\bar{\jmath}}) \wedge ((x_{i,\bar{\jmath}} - y_{i,\bar{\jmath}}) \leq u_{i,\bar{\jmath}})$. Let $\mu' \doteq \mu \cup \{s_{i,\bar{\jmath}} = \top | \forall i\} \cup \{s_{i,j} = \bot | \forall i, \forall j : j \neq \bar{\jmath}\}$. μ' is a model of equation (8.3) because (1) $\bigvee_{j=1}^{D_i} s_{i,j}$ is satisfied by $s_{i,\bar{\jmath}}$, and (2) each conjunct is satisfied because in μ', $s_{i,j}$ is false for all $j \neq \bar{\jmath}$ and thus the i-j-conjunct is satisfied and the i-$\bar{\jmath}$-conjunct is such that $x_{i,\bar{\jmath}}$ and $y_{i,\bar{\jmath}}$ fulfill $((x_{i,\bar{\jmath}} - y_{i,\bar{\jmath}}) \geq \ell_{i,\bar{\jmath}}) \wedge ((x_{i,\bar{\jmath}} - y_{i,\bar{\jmath}}) \leq u_{i,\bar{\jmath}})$.

We now prove that a model μ' of equation (8.3) can be reduced to a model μ of equation (8.2). Let μ be the restriction of μ' to the $x_{i,j}$ and $y_{i,j}$ variables only. μ' fulfills $\bigvee_{j=1}^{D_i} s_{i,j}$, therefore there is a $\bar{\jmath}$ such that $s_{i,\bar{\jmath}}$ is true in μ'. $x_{i,\bar{\jmath}}$ and $y_{i,\bar{\jmath}}$ are such that $((x_{i,\bar{\jmath}} - y_{i,\bar{\jmath}}) \geq \ell_{i,\bar{\jmath}}) \wedge ((x_{i,\bar{\jmath}} - y_{i,\bar{\jmath}}) \leq u_{i,\bar{\jmath}})$, therefore, also equation (8.2) is satisfied. □

Theorem 8.2 (Mutex Switch Correctness). *If P is a TCSN, P is consistent if and only if equation* (8.4) *is satisfiable (and a model of equation* (8.4) *yields a consistent schedule).*

Proof. We prove that for any TCSN, equation (8.4) is logically equivalent to equation (8.3).

We highlight that equation (8.4) is analogous to equation (8.3), but it adds a new constraint in the form $\bigwedge_{j=1}^{D_i} \bigwedge_{k=j+1}^{D_i} (\neg s_{i,j} \vee \neg s_{i,k})$.

Let μ be a model of equation (8.3). Since the network is a TCSN we know that in each constraint the intervals are disjoint. Therefore, for each constraint c_i, there exists exactly one $s_{i,\bar{\jmath}}$ that is true, while all the other $s_{i,j}$ are assigned to false. Clearly, μ is also a model for equation (8.4) because the added term $\bigwedge_{j=1}^{D_i} \bigwedge_{k=j+1}^{D_i} (\neg s_{i,j} \vee \neg s_{i,k})$ is satisfied: only $s_{i,\bar{\jmath}}$ is set to true, therefore in each conjunct at least one variable is set to false.

Let μ' be a model of equation (8.4). Because of the added term, there exists exactly one $s_{i,\bar{\jmath}}$ that is true, while all the other $s_{i,j}$ are assigned to false. For the same reasoning followed above, μ' is also a model of equation (8.3). □

Theorem 8.3 (Hole Encoding Correctness). *If P is a TCSN, P is consistent if and only if equation* (8.5) *is satisfiable (and a model of equation* (8.5) *yields a consistent schedule for P).*

Proof. Assuming that P is a TCSN, we prove that equation (8.5) is equivalent to equation (8.2).

We start from a the formula in equation (8.2), and we consider a single constraint in isolation: $\bigvee_{j=1}^{D_i}(((x_i - y_i) \geq \ell_{i,j}) \wedge ((x_i - y_i) \leq u_{i,j}))$. This sub-formula is in Disjunctive Normal Form, and can be transformed in an equivalent exponential-size CNF by applying the distributive rule. We obtain a formula with 2^{D_i} clauses of D_i disjuncts each. Each clause is a permutation obtained by picking a conjunct for each disjunct of the original formula.

One clause in this CNF formula is composed of all the upper bound constraints: $\bigvee_{k=1}^{D_i}((x_i - y_i) \leq u_{i,k})$, this clause can be trivially simplified to $((x_i - y_i) \leq u_{i,D_i})$ as u_{i,D_i} is bigger than any other upper bound. Similarly, the clause $\bigvee_{k=1}^{D_i}((x_i - y_i) \geq \ell_{i,k})$ becomes $((x_i - y_i) \geq \ell_{i,1})$.

The remaining clauses contain both upper and lower constraints. Each clause c can be reduced to a binary clause in the form $((x_i - y_i) \geq L_{i,c}) \vee ((x_i - y_i) \leq U_{i,c})$, where $L_{i,c}$ is the minimum of the lower bounds and $U_{i,c}$ is the maximum of the upper bounds. The obtained 2-CNF formula is exponential in the size of the original constraint. For each j, the clause $((x_i - y_i) \leq u_{i,j}) \vee ((x_i - y_i) \geq \ell_{i,(j+1)})$ subsumes all the binary clauses with bigger upper bound and smaller lower bound.

We can apply this reasoning to all the conjuncts of equation (8.2), and we obtain the formulation in equation (8.5). Since the two formulations are equivalent, they have the same models, a consistent schedule can be extracted from each model of equation (8.5) □

A.2 Strong Controllability Proofs

Theorem 9.1 (Offset Encoding Correctness)**.** *The TNU P is strongly controllable if and only if equation* (9.2) *is satisfiable (and a strong schedule can be extracted from any of its models).*

Proof. Equation (9.2) is obtained by rewriting equation (9.1) with the offsets, we prove that it is equivalent to equation (9.1). We show this property by refutation.

Suppose μ is a model of equation (9.1) but it is not a model of equation (9.2). Then, there exist a $\vec{Y}_u$ such that $\Gamma(\vec{Y}_u)$ holds but $\Psi(\mu, \vec{Y}_u)$ does not hold. Let $\vec{T}_u \doteq \langle \alpha e + y_e \mid y_e \in \vec{Y}_u \rangle$. By definition, $(\bigwedge_{c_i \in \mathcal{C}} [\![c_i]\!])(\vec{T}_c, \vec{T}_u)$

does not hold and $[\![\rho(\mathcal{L})]\!](\vec{T}_c, \vec{T}_u)$ holds. But this is absurd because this makes equation (9.1) unsatisfiable.

Similarly we can show that any model of equation (9.2) is also a model of equation (9.1). $\square$

Theorem 9.2 (Distributed Encoding Correctness)**.** *If the TNU P is consistent, it is strongly controllable if and only if equation* (9.3) *is satisfiable (and each model yields a strong schedule).*

Proof. We show that equation (9.3) is equivalent to equation (9.2).
We start from equation (9.2). and we rewrite it as follows.
$\forall \vec{Y}_u.(\Gamma(\vec{Y}_u) \to \Psi(\vec{T}_c, \vec{Y}_u))$
$\Leftrightarrow \forall \vec{Y}_u.(\neg\Gamma(\vec{Y}_u) \vee \Psi(\vec{T}_c, \vec{Y}_u))$

We can now replace $\Psi(\vec{T}_c, \vec{Y}_u)$ with its CNF formulation and distribute the disjunction over the big conjunction.
$\Leftrightarrow \forall \vec{Y}_u.(\neg\Gamma(\vec{Y}_u) \vee \bigwedge_{h=1}^{H} \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h}))$
$\Leftrightarrow \forall \vec{Y}_u. \bigwedge_{h=1}^{H}(\neg\Gamma(\vec{Y}_u) \vee \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h}))$

By distribution of $\forall$ over $\wedge$ we obtain the following formulation.
$\Leftrightarrow \bigwedge_{h=1}^{H} \forall \vec{Y}_u.(\neg\Gamma(\vec{Y}_u) \vee \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h}))$

Let $\vec{Y}_{u_k} \doteq \vec{Y}_u \setminus \vec{Y}_{u_h}$ be the variables of $\vec{Y}_u$ not appearing in the h-th clause. The clauses of $\Gamma(\vec{Y}_u)$ can be split in two parts depending on the offset variables they constrain, because every clause contains exactly one offset variable by construction.
$\Leftrightarrow \bigwedge_{h=1}^{H} \forall \vec{Y}_u.(\neg\Gamma(\vec{Y}_u)|_{Y_{u_h}} \vee \neg\Gamma(\vec{Y}_u)|_{Y_{u_k}} \vee \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h}))$

The universal quantification $\forall \vec{Y}_u$ can be split in $\forall \vec{Y}_{u_k}.\forall \vec{Y}_{u_h}$.
$\Leftrightarrow \bigwedge_{h=1}^{H} \forall \vec{Y}_{u_k}.(\neg\Gamma(\vec{Y}_u)|_{Y_{u_k}} \vee \forall \vec{Y}_{u_h}.(\neg\Gamma(\vec{Y}_u)|_{Y_{u_h}} \vee \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h})))$

Since $\Gamma(\vec{Y})$ is assumed to be non-contradictory, $\Gamma(\vec{Y_u})|_{Y_{u_k}}$ cannot be false for every $\vec{Y}_{u_k}$. Therefore, $\neg\Gamma(\vec{Y_u})|_{Y_{u_k}}$ reduces to $\bot$. We can then remove this disjunct and the relative quantification become useless.
$\Leftrightarrow \bigwedge_{h=1}^{H} \forall\vec{Y}_{u_k}.(\forall\vec{Y}_{u_h}.(\neg\Gamma(\vec{Y_u})|_{Y_{u_h}} \vee \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h})))$
$\Leftrightarrow \bigwedge_{h=1}^{H} \forall\vec{Y}_{u_h}.(\neg\Gamma(\vec{Y_u})|_{Y_{u_h}} \vee \psi_h(\vec{T}_{c_h}, \vec{Y}_{u_h}))$
This is exactly the formulation of equation (9.3). □

Theorem 9.3 (EFE Encoding Correctness). *The TNU P is strongly controllable if and only if equation* (9.3) *is satisfiable (and a strong schedule can be extracted from a model of equation* (9.3)*).*

Proof. Equation (9.4) derives from equation (9.3) by resolving the universal quantifier using a quantifier elimination procedure. Since the elimination procedure builds equivalent formulae, equations (9.3) and (9.4) are logically equivalent. □

A.3 Weak Controllability Proofs

Proposition 10.2 (Assumption Extraction Correctness). *Equation* (10.2) *and equation* (10.3) *are logically equivalent.*

Proof. We show how to convert equation (10.2) into equation (10.3), using logically equivalent rewritings.

$$\begin{aligned}
&\neg\exists\vec{T}_c.(\Gamma(\vec{Y_u}) \rightarrow \Psi(\vec{T}_c, \vec{Y_u}))\\
\Leftrightarrow\ &\neg\exists\vec{T}_c.(\neg\Gamma(\vec{Y_u}) \vee \Psi(\vec{T}_c, \vec{Y_u}))\\
\Leftrightarrow\ &\neg((\exists\vec{T}_c.\neg\Gamma(\vec{Y_u})) \vee (\exists\vec{T}_c.\Psi(\vec{T}_c, \vec{Y_u})))\\
\Leftrightarrow\ &\neg(\neg\Gamma(\vec{Y_u}) \vee (\exists\vec{T}_c.\Psi(\vec{T}_c, \vec{Y_u})))\\
\Leftrightarrow\ &\Gamma(\vec{Y_u}) \wedge \neg(\exists\vec{T}_c.\Psi(\vec{T}_c, \vec{Y_u}))
\end{aligned}$$

□

Theorem 10.1 (Weak Controllability Skolemization). *A TNU $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ is weakly controllable if and only if the formula*

$$\forall \vec{Y_u}. \Gamma(\vec{Y_u}) \rightarrow \Psi(f(\vec{Y_u}), \vec{Y_u}) \tag{10.4}$$

is satisfiable.

Proof. equation (10.4) is the result of applying the skolemization [Kle67] procedure to equation (10.1). Since skolemization produces an equi-satisfiable formula, equation (10.4) is equi-satisfiable to equation (10.1). Since equation (10.1) has no free variables nor has uninterpreted terms, satisfiability coincides with validity. Therefore, equation (10.1) is valid if and only if equation (10.4) is satisfiable, and proposition 10.1 states that the TNU is weakly controllable if and only if equation (10.1) is valid. □

Theorem 10.4 (Vertex Encoding Correctness). *Let $P \doteq \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be an STNU, $\langle \vec{T_c}, \vec{Y_u}, \Gamma(\vec{Y_u}), \Psi(\vec{T_c}, \vec{Y_u}) \rangle$ be its encoding and let $\bar{f} : \mathbb{R}^{|Y_u|} \rightarrow \mathbb{R}^{|X_c|}$ be a linear strategy. If $\bar{f}$ fulfills $\Psi(\vec{T_c}, \vec{Y_u})$ in all the vertexes $v_i \in V_\Gamma$, then $\bar{f}$ is a weak linear strategy for P.*

Proof. For the sake of contradiction, let us suppose that there exists a point $\bar{p}$ in the space of $\vec{Y_u}$ such that $\Gamma(\bar{p})$ holds and $\Psi(\bar{f}(\bar{p}), \bar{p})$ does not hold.

Then, there must exist a free constraint c_k that is violated by $\bar{p}$. Since the problem is an STNU, each free constraint is geometrically either a half-space (for example $a - b \in [1, \infty)$) or the intersection of two half-spaces (for example, $a - b \in [10, 15]$ is the intersection of the half-space $a - b \leq 15$ with $a - b \geq 10$). Therefore, $\bar{p}$ does not belong to one of the half-spaces encoded by c_k. Let H be the violating half-space.

However, for each vertex $v_i \in V_\Gamma$, $\bar{f}(v_i)$ must belong to H because the free constraints are fulfilled in all the vertexes.

The point $\bar{f}(\bar{p})$ belongs to the convex hull of the points $\bar{f}(v_i)$, but then it must belong to the half-space H. Hence, we have a contradiction. □

Theorem 10.5 (Simplex Strategy Existence)**.** *Let P be an encoded weakly controllable STNU $\langle \vec{T_c}, \vec{Y_u}, \Gamma(\vec{Y_u}), \Psi(\vec{T_c}, \vec{Y_u})\rangle$. For each $|Y_u|$-simplex $\sigma(\vec{Y_u})$ such that $\sigma(\vec{Y_u}) \subseteq \Gamma(\vec{Y})$ there exists a valid weak linear strategy f such that $\forall \vec{Y_u}.((\sigma(\vec{Y_u}) \wedge \vec{T_c} = f(\vec{Y_u})) \rightarrow \Psi(\vec{T_c}, \vec{Y_u}))$ is valid.*

Proof. Let $m \doteq |\vec{Y_u}|$ and V be the set of $m+1$ vertexes of $\sigma(\vec{Y_u})$. By definition of simplex, $\sigma(\vec{Y_u})$ is the convex hull of the points in V. P is weakly controllable by assumption, therefore for each $v_i \in V$, there exists a point t_i that extends v_i in the space of $\vec{T_c} \cup \vec{Y_u}$ such that $t_i \in \Psi(\vec{T_c}, \vec{Y_u})$. Let T be $\{t_i | v_i \in V\}$.

Let f be a linear strategy $A \cdot \vec{Y_u} + \vec{c}$, such that for each controllable time point $b_i \in X_c$, $b_i = A_{i,1} y_{A_e} + \cdots + A_{i,m} y_m + c_i$ is the hyperplane passing through all the points $t_i \in T$. Such a hyperplane exists and is unique because it is the m-hyperplane containing the $m+1$ not-collinear [CG67] points $t_i \in T$ (points in T are not collinear as they are the results of an extension of the points in V that are not collinear because are the $m+1$ vertexes of a simplex).

We now prove that f is a strategy such that $\forall \vec{Y_u}.(\sigma(\vec{Y_u}) \wedge \vec{T_c} = f(\vec{Y_u})) \rightarrow \Psi(\vec{T_c}, \vec{Y_u})$ is valid by showing that the hyperplane $b_i = A_{i,1} y_{A_e} + \cdots + A_{i,m} y_m + c_i$ is contained in $\Psi(\vec{T_c}, \vec{Y_u})$ for each $\vec{Y_u} \models \sigma(\vec{Y_u})$. Since the hyperplane contains all the $t_i \in T$, for each point k in the convex hull of V, the hyperplane computed in k is contained in $\Psi(\vec{T_c}, \vec{Y_u})$ because of the convexity of $\Psi(\vec{T_c}, \vec{Y_u})$. This proves the thesis because $\sigma(\vec{Y_u})$ is the convex hull of the points in V. □

A.4 STNU Execution Semantics

This section introduces a novel formulation of the time-strict execution semantics for dynamic controllability of STNUs as a two-player game. This formalization and the relative proof has been developed by Luke Hunsberger in the context of a joint paper [CHM+16].

The game is played between Agnes (the agent) and Vera (the environment), where Agnes controls the execution of controllable time points and Vera controls the contingent durations. Agnes seeks an execution strategy that will ensure the satisfaction of all constraints in $\mathcal{C}$ no matter what durations Vera chooses; Vera seeks a strategy that will ensure that at least one constraint in $\mathcal{C}$ is unsatisfied no matter what Agnes does.

This formulation highlights an important asymmetry in the execution semantics: Agnes is not able to react instantaneously to observations of uncontrollable time points executing, but Vera is able to react instantaneously to executions of controllable time points.

A partial schedule represents the current state of the game (i.e., the set of time points that have executed so far)[1].

Definition 84 (Partial Schedule)**.** *A partial schedule for an STNU, $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, is a set, PS, of variable assignments to time points in $\mathcal{T}$. $\mathit{TPs}(PS) \subset \mathcal{T}$ denotes the set of time points appearing in PS; $\mathit{Vals}(PS) \subset \mathbb{R}$ denotes the set of values appearing in PS; for any $X \in \mathit{TPs}(PS)$, $PS(X)$ denotes the value assigned to X; and $\texttt{now}_{PS} = \max\{v \mid v \in \mathit{Vals}(PS)\}$ is the time of the latest execution event in PS. (If $PS = \emptyset$, let $\texttt{now}_{PS} = -\infty$.) time points in $\mathit{TPs}(PS)$ are said to be executed. A partial schedule is called respectful if its assignments do not violate the bounds on any contingent link.*

Given a partial schedule PS, Agnes must decide what to do next. She has two options: (1) wait for Vera to (eventually) do something; or (2) conditionally commit to executing a set of controllable time points at some time, $t_c > \texttt{now}_{PS}$. For example, given $PS = \{\langle A_2, 0\rangle, \langle X, 1\rangle\}$, for which $\texttt{now}_{PS} = 1$, Agnes could decide to wait until Vera eventually executes C_2.

[1]Definition 84 definition 85 are drawn from [Hun09].

Alternatively, she could decide that "if nothing happens before time 7, I shall execute A_1 at time 7."

The decisions available to Agnes are called real-time execution decisions (RTEDs).

Definition 85 (RTED, for Agnes). *Let PS be a respectful partial schedule. An RTED for Agnes has one of two forms:* `wait` *or* $\langle t_c, \chi_f \rangle$. *A* `wait` *decision is applicable if at least one uncontrollable time point, C, is active in PS (i.e., C's activation time point has already been executed, but C has not). A $\langle t_c, \chi_f \rangle$ decision (i.e., "If nothing happens before time t_c, execute the time points in χ_f at time t_c") is applicable if $t_c > \mathtt{now}_{PS}$ and χ_f is a non-empty subset of unexecuted controllable time points (i.e., $\chi_f \neq \emptyset$ and $\chi_f \cap TPs(PS) = \emptyset$).*

The kinds of decisions available to Vera are different in two important respects. First, Vera's version of an RTED (called an RTED*) allows a decision of the form, "if nothing happens before or at time t_u, then I shall execute the uncontrollable time points in the set $\chi_u \subseteq \mathcal{T}_u$ at time t_u." Note that when time t_u arrives, should Vera observe Agnes executing any time points at time t_u, Vera has the option of instantaneously changing her mind. Second, in such cases, Vera may instantaneously react by executing some other uncontrollable time points at time t_u. Such decisions are called instantaneous reactions. For example, suppose Vera had decided that "if nothing happens before or at time 7, then I shall execute C_2 at time 7", but when time 7 arrived, she observed Agnes executing some time point(s). Vera could withdraw her decision to execute C_2 and instantaneously react by deciding to execute some other uncontrollable time point(s) at time 7.

Definition 86 (RTED*, for Vera). *Let PS be a respectful partial schedule. A before-or-at RTED (RTED*) has one of two forms:* `wait` *or* $\langle t_u, \chi_u \rangle$. *A* `wait` *decision is only applicable if no uncontrollable time points are cur-*

rently active in PS. A $\langle t_u, \chi_u \rangle$ decision (i.e., "If nothing happens before-or-at time t_u, I shall execute the time points in χ_u at time t_u") is applicable only if $t_u > \texttt{now}_{PS}$, and χ_u is a non-empty subset of currently-activated uncontrollable time points each of whose execution window includes t_u; and all other uncontrollable time points that are unexecuted in PS are either unactivated in PS or have execution windows that extend beyond t_u.

Definition 87 (Instantaneous reaction, for Vera). *Let PS be a respectful partial schedule. Let χ° be the (probably empty) set of uncontrollable time points that are currently active in PS whose execution windows happen to terminate precisely at $\texttt{now}_{PS}$; and let $\chi^\star$ be any (possibly empty) subset of the uncontrollable time points that are currently active in PS whose execution windows include $\texttt{now}_{PS}$, but also extend beyond $\texttt{now}_{PS}$. An instantaneous reaction is a decision (by Vera) to execute the uncontrollable time points in the set $\chi^\circ \cup \chi^\star$ at the time $\texttt{now}_{PS}$.*

To accommodate Vera's ability to react instantaneously, the outcome for a pair of decisions (one by Agnes, one by Vera) is defined in two stages: partial and full.

Definition 88 (Partial Outcome). *Let PS be a respectful partial schedule; let Δ_f be an RTED for Agnes; and let Δ_u be an RTED* for Vera. The partial outcome, $\mathcal{O}_p(PS, \Delta_f, \Delta_u)$, is defined as follows*[2].

[2]Note that a `wait` decision cannot be simultaneously applicable for both Agnes and Vera.

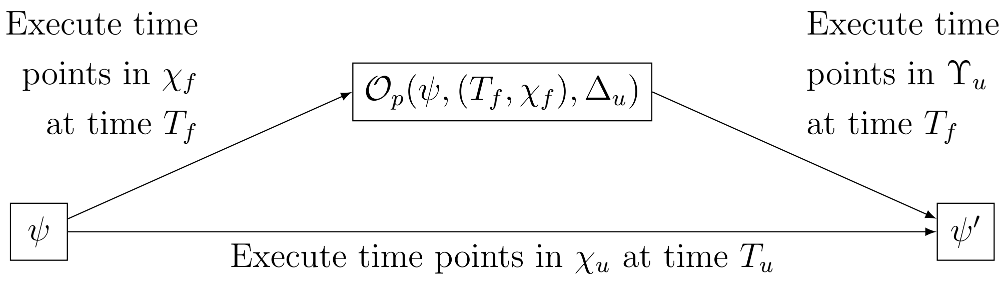

Execute time points in χ_f at time T_f
$\mathcal{O}_p(\psi, (T_f, \chi_f), \Delta_u)$
Execute time points in Υ_u at time T_f
ψ
ψ'
Execute time points in χ_u at time T_u

Figure A.1: Deriving the outcome PS' of decisions by Agnes and Vera from the partial schedule PS.

(1a) $\mathcal{O}_p(PS, \texttt{wait}, \langle t_u, \chi_u\rangle) \quad = PS \cup \{\langle C, t_u\rangle \mid C \in \chi_u\}$.

(1b) $\mathcal{O}_p(PS, \langle t_c, \chi_f\rangle, \langle t_u, \chi_u\rangle) = PS \cup \{\langle C, t_u\rangle \mid C \in \chi_u\}$, *if* $t_u < t_c$.

(2a) $\mathcal{O}_p(PS, \langle t_c, \chi_f\rangle, \texttt{wait}) \quad = PS \cup \{\langle X, t_c\rangle \mid X \in \chi_f\}$.

(2b) $\mathcal{O}_p(PS, \langle t_c, \chi_f\rangle, \langle t_u, \chi_u\rangle) = PS \cup \{\langle X, t_c\rangle \mid X \in \chi_f\}$, *if* $t_c \le t_u$.

Note that in cases (1a) and (1b), the partial outcome includes only the execution of the uncontrollable time points in χ_u at time t_u. Cases (2a) and (2b) are analogous, in that the partial outcome includes only the execution of the controllable time points in χ_f at time t_c, except that Vera is also able to instantaneously react by executing one or more uncontrollable time points, also at time t_c, as described below.

Definition 89 (Full Outcome)**.** *Let $PS_p = \mathcal{O}_p(PS, \Delta_f, \Delta_u)$ be a partial outcome, as described above; and let Υ_u be a set of uncontrollable time points that constitute an instantaneous reaction to PS_p. The full outcome, $\mathcal{O}(PS, \Delta_f, \Delta_u, \Upsilon_u)$, is the same as PS_p, except that in cases (2a) and (2b), the schedule is augmented to include the execution of the time points in Υ_u at time t_c.*

$$\begin{aligned} PS &= \{\langle A_2, 0\rangle, \langle X, 1\rangle\};\ \ \Delta_f = \langle 7, \{A_1\}\rangle;\ \ \Delta_u = \langle 6, \{C_2\}\rangle. \\ PS' &= \{\langle A_2, 0\rangle, \langle X, 1\rangle, \langle C_2, 6\rangle\};\ \ \Upsilon_u \text{ irrelevant.} \end{aligned}$$

$$\begin{aligned} PS &= \{\langle A_2, 0\rangle, \langle X, 1\rangle\};\ \ \Delta_f = \langle 7, \{A_1\}\rangle;\ \ \Delta_u = \langle 8, \{C_2\}\rangle. \\ PS' &= \{\langle A_2, 0\rangle, \langle X, 1\rangle, \langle A_1, 7\rangle, \langle C_2, 7\rangle\}, \text{ where } \Upsilon_u = \{C_2\}. \end{aligned}$$

$$\begin{aligned} PS &= \{\langle A_2, 0\rangle, \langle X, 1\rangle\};\ \ \Delta_f = \langle 7, \{A_1\}\rangle;\ \ \Delta_u = \langle 8, \{C_2\}\rangle. \\ PS' &= \{\langle A_2, 0\rangle, \langle X, 1\rangle, \langle A_1, 7\rangle\}, \text{ where } \Upsilon_u = \emptyset. \end{aligned}$$

Table A.1: The outcomes PS' for sample decisions by Agnes and Vera for the STNU from figure 11.1.

Figure A.1 illustrates the possible pathways from a partial schedule PS to the full outcome $PS' = \mathcal{O}(PS, \Delta_f, \Delta_u, \Upsilon_u)$. Note that $\texttt{now}_{PS'}$ is either t_c or t_u, depending on which pathway is taken. Note, too, that the full outcome, PS', is typically a partial schedule, except at the very end when all of the time points have been executed. Table A.1 shows the outcomes that result from sample decisions by Agnes and Vera in the case of the STNU from figure 11.1. In each case, $PS' = \mathcal{O}(PS, \Delta_f, \Delta_u, \Upsilon_u)$.

Definition 90 (Execution Strategies). *An RTED-based strategy (for Agnes) is a mapping from respectful partial schedules to RTEDs. An RTED*-based strategy (for Vera) is a pair of mappings, $\langle f_1, f_2\rangle$, where f_1 is a mapping from respectful partial schedules to RTED*s; and f_2 is a mapping from respectful partial schedules to instantaneous reactions.*

Definition 91 (Outcomes of Strategies). *Let PS be a respectful partial schedule; R an RTED-based strategy; and $R^\star = \langle f_1, f_2\rangle$ an RTED*-based strategy. The one-step outcome, $\mathcal{O}^1(PS, R, R^\star)$, is defined by:*

$$\mathcal{O}^1(PS, R, R^\star) \ = \ \mathcal{O}(PS, R(PS), f_1(PS), f_2(\mathcal{O}_p(PS, R(PS), f_1(PS))))$$

The terminal outcome, $\mathcal{O}^*(R, R^\star)$, is the complete schedule that results from the following recursive definition: $PS_0 = \emptyset$ and $PS_{i+1} = \mathcal{O}^1(PS_i, R, R^\star)$.

The constraints on the decisions generated by $R^\star$, namely, that Vera must observe the bounds on the contingent durations, ensure that each PS_i in the sequence will be respectful, given that $PS_0 = \emptyset$ is trivially respectful.

Given the above execution semantics for STNUs, the definition of dynamic controllability is straightforward.

Definition 92 (Dynamic Controllability). *An STNU, $\langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$, is dynamically controllable if there exists an RTED-based strategy R, such that for all RTED*-based strategies $R^\star$, the variable assignments in the complete schedule, $\mathcal{O}^*\langle PS_0, R, R^\star \rangle$, satisfy all of the constraints in $\mathcal{C}$.*

Theorem A.1. *Definition 92 is equivalent to the definition of dynamic controllability given in [Hun09].*

Proof. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be any STNU. First, suppose that $\mathcal{S}$ is dynamically controllable according to the RTED-based semantics. Then there exists an RTED-based execution strategy R such that for any situation ω, the full schedule that results from following R in ω satisfies all of the constraints in $\mathcal{C}$. Let that strategy R be the one chosen by Agnes in the two-player game semantics. Let $R^* = \langle f_1, f_2 \rangle$ be any strategy for Vera. It will be shown that the terminal outcome $\mathcal{O}^*(R, R^*)$ that results from Agnes and Vera playing these two strategies against each other necessarily satisfies the constraints in $\mathcal{C}$. In particular, it will be shown by induction that each (partial or full) schedule obtained at any point during the execution phase by following R and R^* according to the two-player game semantics can also be obtained by following R in some situation in the RTED-based semantics.

- **Base Case.** Let PS_0 be the empty partial schedule. This is the starting partial schedule in either semantics.
- **Recursive Case.** Let PS be any partial schedule obtained by following R and R^* in the two-player game semantics. There are three sub-cases to consider.
  1. $R(PS) = \texttt{wait}$; $R^*(PS) = \langle t_u, \chi_u \rangle$. In this case, the partial outcome involves the execution of the uncontrollable time points in χ_u at the time t_u. Since the applicability conditions for Vera's RTED* decision requires the execution times for uncontrollable time points to respect the lower and upper bounds on the corresponding contingent links, the resulting partial outcome is a partial schedule obtainable from any situation ω that is respected by PS and includes the durations specified by the uncontrollable time points in χ_u.
  2. $R(PS) = \langle t_c, \chi_f \rangle$; $R^*(PS) = \langle t_u, \chi_u \rangle$, where $t_u < t_c$. This case is essentially the same as the first case, since $t_u < t_c$.
  3. $R(PS) = \langle t_c, \chi_f \rangle$; $R^*(PS) = \texttt{wait}$. In this case, the partial outcome involves the execution of the executable time points in χ_f. Since Vera can only use the `wait` decision when the partial schedule PS does not contain any currently active contingent links, this must be the outcome in the RTED-based semantics, too. There can be no instantaneous reaction by Vera in this case.
  4. $R(PS) = \langle t_c, \chi_f \rangle$; $R^*(PS) = \langle t_u, \chi_u \rangle$, where $t_c \leq t_u$. This case is the same as the preceding case except that Vera may choose to react instantaneously (i.e., $f_2(PS)$ may not be empty). The applicability conditions of instantaneous reactions require the uncontrollable time points in $f_2(PS)$ to be currently active in PS, and such that their execution windows include the time $\texttt{now}_{PS}$. In

addition, any uncontrollable time points that happen to have their execution window terminate precisely at $\texttt{now}_{PS}$ must be included in $f_2(PS)$. Thus, the full outcome is the same as in the preceding case except that some uncontrollable time points may also execute at the time t_c. Again, this corresponds to any situation ω that is respected by PS, while also respecting the contingent durations determined by the execution of the uncontrollable time points in $f_2(PS)$.

For the other direction, suppose that $\mathcal{S}$ is not dynamically controllable according to the RTED-based semantics. In other words, for any RTED-based strategy R, there is a situation ω_R such that the outcome $\mathcal{O}^*(R, \omega_R)$ that results from following the strategy R in the situation ω_R does not satisfy the constraints in $\mathcal{C}$. (Any situation with this property will be said to thwart the strategy R.) It will be shown that there must be a strategy $R^* = \langle f_1, f_2 \rangle$ for Vera that will ensure that Agnes loses the two-player game. The proof is by induction. The proposition to prove is the following:

Let PS be any partial schedule that can be reached by following any RTED-based strategy R in any thwarting situation ω_R, according to the RTED-based semantics. Then there is an RTED* decision Δ_u (that depends only on PS, not on R) and an instantaneous reaction Υ_u for Vera such that the full outcome obtained from $R(PS), \Delta_u$ and Υ_u according to the two-player game semantics is a schedule that is identical to one obtained by following R in some thwarting situation ω_R.

Let PS be a partial schedule that can be reached by following some RTED-based execution strategy R in some thwarting situation ω_R, according to the RTED-based semantics. Now, if no uncontrollable time points are currently active in PS, then Agnes must choose a $\langle t_c, \chi_f \rangle$ decision, and Vera must choose the `wait` decision. But in that case, the outcome is fully determined: the time points in χ_f will be executed at time t_c. Furthermore, the outcome is the same whether using the RTED-based semantics or the two-player game semantics.

On the other hand, suppose that at least one uncontrollable time point is currently active in PS. Let Θ_{PS} be the set of RTED-based execution strategies for Agnes that can generate the partial schedule PS at some point during the execution of the network, if followed in some thwarting situation. For each $t > \texttt{now}_{PS}$, let $\Theta(t)$ be the subset of Θ_{PS} that contains all strategies θ whose decisions, $\theta(PS)$, specify execution times greater than t. Now, for any strategy $\theta \in \Theta(t)$, there must be a situation ω_θ that thwarts θ; however, that situation may involve the execution of uncontrollable time point(s) at some time before t (i.e., at some time ρ, where $\texttt{now} < \rho < t$). Of particular interest are the values of $t > \texttt{now}_{PS}$ for which all of the strategies in $\Theta(t)$ can be thwarted by situations that do not involve executing any uncontrollable time points before time t. In particular, let Γ be the set of real numbers $t > \texttt{now}_{PS}$ for which every strategy $\theta \in \Theta(t)$ can be thwarted by a situation that is consistent with no new contingent executions occurring before time t (i.e., at any time ρ such that $\texttt{now}_{PS} < \rho < t$).

Now, suppose that $\Gamma = \emptyset$. Let Agnes adopt the following strategy: wait until some uncontrollable time point happens to execute.

Let $t > \texttt{now}_{PS}$ be the time of that next contingent execution. Since $t \notin \Gamma$, there must be some strategy, $\theta \in \Theta(t)$, that could only be thwarted by situations that involve the execution of uncontrollable time points before time t. Since no uncontrollable time points executed before time t, that strategy is not thwarted by the current situation and, thus, is a winning strategy for Agnes, which is a contradiction. Thus, $\Gamma \neq \emptyset$.

Next, let $t_u = \inf\{t \mid t > \texttt{now}_{PS} \text{ and } t \notin \Gamma\}$. Now, t_u is well defined since Γ is non-empty and bounded below by $\texttt{now}_{PS}$. Consider the possibility that $t_u = \texttt{now}_{PS}$. This implies that for any time $t > \texttt{now}_{PS}$, there is a time $t' \in \langle \texttt{now}_{PS}, t\rangle$ such that $t' \notin \Gamma$. But then a similar argument as that used to show that Γ is not empty can be used to show that t_u cannot equal $\texttt{now}_{PS}$. In this case, given the time t of the next contingent execution, there must be a time $t' \in \langle \texttt{now}_{PS}, t\rangle$ such that $t' \notin \Gamma$ and, hence, some strategy $\theta \in \Theta(t')$ that could only be thwarted by contingent executions before time $t' < t$. Since no such executions occurred, that strategy could be followed by Agnes as a winning strategy, a contradiction. Thus, $t_u > \texttt{now}_{PS}$. It remains to be seen whether $t_u \in \Gamma$.

Next, let Γ^* be the subset of $(\texttt{now}_{PS}, t_u]$ such that for each $t \in \Gamma^*$, there exists a (possibly empty) set $\chi(t)$ of uncontrollable time points such that every strategy $\theta \in \Theta(t)$ can be thwarted by a situation that is consistent with (1) no new contingent executions before time t; and (2) the execution of all of the uncontrollable time points in $\chi(t)$ at time t. Now, suppose Γ^* were empty. Then let $t \in \langle \texttt{now}_{PS}, t_u\rangle \subseteq \Gamma$ be arbitrary; and consider the following strategy for Agnes: wait until the time t, or the execution of the next uncontrollable time point, whichever happens first. If no uncontrollable time points happen to execute before time t, then let $t' = t$; otherwise, let t' be the time at which the first uncontrollable time point executed. In either case, since $t' \in \Gamma$, but $t' \notin \Gamma^*$, there could not be a single set $\chi(t')$ as described earlier. Therefore, there would have to be at least two strategies, θ_1 and θ_2, in $\Theta(t')$ whose thwarting would require two different sets of uncontrollable time points executing at time t'. Agnes could then choose to follow whichever strategy, θ_1 or θ_2, was not thwarted by the execution events that occurred at time t'. Since that chosen strategy could only have been thwarted by execution events which did not occur, it must be a winning strategy, which is a contradiction. Therefore $\Gamma^* \neq \emptyset$.

Next, let $t_u^* = \inf\{t \mid t > \texttt{now}_{PS} \text{ and } t \notin \Gamma^*\}$. Consider the possibility that $t_u^* = \texttt{now}_{PS}$. Then for any $t > \texttt{now}_{PS}$, there exists a t' such that

$\texttt{now}_{PS} < t' < t$ and $t' \notin \Gamma^*$. Let Agnes wait until the time of the next contingent execution, say at time $t > \texttt{now}_{PS}$. Then there exists a time t' strictly between $\texttt{now}_{PS}$ and t such that $t' \notin \Gamma^*$. In that case, there exist strategies θ_1 and θ_2 in $\Theta(t')$ whose thwarting situations required different sets of contingent executions at time $t' < t$. Since no such contingent executions occurred, Agnes can simply choose whichever strategy has thereby become a winning strategy, yielding a contradiction. Therefore, $t_u^* > \texttt{now}_{PS}$.

There are now three cases to consider:

- **Case 1: $t_u^* = t_u$, but $t_u \notin \Gamma$.** Suppose that for all $t \in \langle \texttt{now}_{PS}, t_u \rangle$, $\chi(t) = \emptyset$. In other words, for each $t \in \langle \texttt{now}_{PS}, t_u \rangle$, every $\theta \in \Theta(t)$ can be thwarted by situations in which no uncontrollable time points execute at or before time t. But that implies that every $\theta \in \Theta(t_u)$ can be thwarted by situations in which no uncontrollable time points execute before time t_u and, hence, that $t_u \in \Gamma$, a contradiction. Therefore, it must be that for some $t^* \in \langle \texttt{now}_{PS}, t_u \rangle$, $\chi(t^*) \neq \emptyset$. Let Vera's RTED* decision be $\langle t^*, \chi(t^*) \rangle$.

- **Case 2: $t_u^* = t_u \in \Gamma$.** Suppose that $t_u^* \notin \Gamma^*$. Then there must be two strategies, θ_1 and θ_2, in $\Theta(t_u^*)$ that can only be thwarted by situations involving two different sets of uncontrollable time points at time t_u^*. But then Agnes could simply wait until time t_u^* to see which of the two strategies was not thwarted, to yield a winning strategy. But that is a contradiction. Therefore, $t_u^* \in \Gamma$.

  Now, suppose that $\chi(t_u^*) = \emptyset$. That is, every strategy in $\Theta(t_u^*)$ can be thwarted by situations that do not involve any new contingent executions at or before t_u^*. Let Agnes employ the following strategy: wait until the next contingent execution. Suppose it happens at some time $t > t_u^*$. By the definition of t_u and the fact that $t_u \in \Gamma$, it follows that there must be some t' strictly between t_u and t such that $t' \notin \Gamma$.

But then there must be a strategy $\theta \in \Theta(t')$ whose thwarting requires the execution of a uncontrollable time point before time $t' < t$. Since no such execution occurred, Agnes can employ θ as a winning strategy, which is a contradiction. Thus, $\chi(t_u^*) \neq \emptyset$. Vera's RTED* decision can then be $\langle t_u^*, \chi(t_u^*)\rangle$.

- **Case 3: $t_u^* < t_u$.** As in Case 2, it follows here that $t_u^* \in \Gamma^*$. Now, let t be any time such that $t_u^* < t < t_u$. Let Agnes wait until the next contingent execution or the time t, whichever comes first. Let $t^\dagger$ be that time. By the definition of t_u^* as an infemum, and the fact that $t_u^* \in \Gamma^*$, it follows that there is some t' strictly between t_u^* and $t^\dagger$ such that $t' \notin \Gamma^*$, but $t' \in \Gamma$ (since $t' < t_u$). But then there exist strategies θ_1 and θ_2 in $\Theta(t')$ whose thwarting situations require different sets of uncontrollable time points to execute at time $t' < t^\dagger \leq t$. Since no such contingent executions occurred, Alice can simply choose whichever strategy has thereby become a winning strategy, yielding a contradiction. Therefore, it cannot be that $t_u^* < t_u$.

Only Cases 1 and 2 avoid a contradiction; and in each of those cases generates a decision for Vera of the form $\Delta_u = \langle t, \chi\rangle$, where χ is a set of uncontrollable time points that are to be executed at time t if Agnes does not execute any time points at or before t. It remains to show that all possible outcomes of the decisions of Agnes and Vera result in a schedule that can be obtained by following a strategy R in some thwarting situation ω_R.

First, suppose Agnes uses a `wait` decision. In that case, the uncontrollable time points in χ will be executed at time t. By the construction of the χ set (cf. the definition of Γ^*), it follows that all strategies in $\Theta(t)$, of which `wait` is one, can be thwarted by situations that are consistent with this outcome. Similar remarks apply to Agnes using a $\langle t_c, \chi_f\rangle$ decision where $t_c > t$.

Second, suppose Agnes uses a $\langle t_c, \chi_f \rangle$ decision where $t_c \leq t$. Then the partial outcome will involve the execution of the executable time points in χ_f at time $t_c \leq t$, but not the uncontrollable time points in χ. Now, since $t_c \leq t$, it follows that $t_c \leq t_u^*$. Thus, for each time $t' < t_c$, all strategies in $\Theta(t')$, of which, Agnes' $\langle t_c, \chi_f \rangle$ is one, must be thwartable by situations involving no new uncontrollable time points before time t'. But then, for any $t^\dagger < t_c$, there is some t' such that $t^\dagger < t' < t_c$, from which it follows that no uncontrollable time points need be executed at or before $t^\dagger$. Thus, no uncontrollable time points need be executed before time t_c. However, thwarting the strategies that involve the execution of the time points in χ_f at time t_c may require the execution of some uncontrollable time points at time t_c. A single set of such time points must be sufficient; otherwise, it would contradiction the thwartability of those strategies. That set of time points constitutes an instantaneous reaction by Vera.

Thus, in all cases, Vera has a decision $\langle t, \chi \rangle$ available (that only depends on PS, not on R) that, together with a possible instantaneous reaction, generates an outcome according to the two-player game semantics that is identical to an outcome that is obtained by following an RTED-based strategy in a thwarting situation. $\square$

The dynamic controllability problem for DTNUs is defined analogously to the STNU case [PVYS07, VVPYS10]. In fact, disjunctive free constraints simply give more freedom to the agent, while disjunctions in contingent links allow the environment to choose among a set of intervals, but this does not change the semantics of the dynamic controllability problem.

A.5 STNU to TGA Formal Correctness

Here we present the theoretical results that confirm the correctness of the encoding, and the correspondence between strategies for the STNU and its TGA counterpart. This proof has been developed by Luke Hunsberger in the context of a joint paper [CHM+16].

Theorem A.2. *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, \mathcal{L} \rangle$ be any STNU; and let Θ be the encoding of $\mathcal{S}$ as a TGA, as described in section 11.3.1. Then, Θ correctly captures the execution semantics for $\mathcal{S}$ in the sense that any sequence of partial schedules that can be generated for $\mathcal{S}$ according to the execution semantics for STNUs corresponds to a run for Θ that can be generated by following its transitions according to the TGA semantics.*

Proof. The following invariant is proved by induction. Each respectful partial schedule PS that can be generated for $\mathcal{S}$ corresponds to a state of Θ in which the location is $\texttt{env}$, $\overline{\delta} = 0$, $\texttt{now}_{PS} = \hat{\texttt{t}}$, for each executed time point X, $PS(X) = \hat{\texttt{t}} - \texttt{tX}$, and for each unexecuted time point Y, $PS(Y) = \hat{\texttt{t}}$. For the base case, the initial partial schedule, $PS_0 = \emptyset$, corresponds to the initial state of Θ in which the location is $\texttt{env}$, all clocks are at zero, and all time points are unexecuted. Note that PS_0 is trivially respectful.

Now, suppose that PS is a respectful partial schedule that can be generated according to the execution semantics for STNUs, and that satisfies the hypothesized invariant. Let θ be the corresponding state of the TGA. Since $\overline{\delta} = 0$, the only transitions that are immediately enabled are the loops whereby uncontrollable time points are executed. These transitions, if taken, correspond to the instantaneous reaction decisions for Vera, in which a set Υ_u of one or more uncontrollable time points can be executed simultaneously. However, suppose that Vera does not make any such transitions at $\overline{\delta} = 0$. Once $\overline{\delta} > 0$, both Agnes and Vera have transitions that they could make at any time. For example, Vera might decide to execute one or more uncontrollable time points when $\overline{\delta} = 3$. That would correspond to an RTED*-based decision, (T_u, χ_u), where $T_u = \texttt{now}_{PS} + 3$ and χ_u contains the time points to be executed. Since each transition by Vera resets $\overline{\delta}$ to 0, Agnes is unable to interrupt Vera's simultaneous execution of uncontrollable time points. The resulting outcomes are equivalent to the partial schedules that arise in Cases (1a) and (1b) of definition 88. The guards on Vera's transitions, which enforce the duration bounds for the contingent links, ensure that the resulting partial schedule is respectful. Also, when Vera's sequence of "simultaneous" transitions complete, $\hat{\texttt{t}}$ equals the time of the most recent execution (i.e., $\texttt{now}_{PS} + 3$). In addition, for each newly executed time point, C, the clock $\texttt{tC}$ is set to 0, ensuring that $\hat{\texttt{t}} - \texttt{tC}$ equals the execution time of C. Since both clocks will never again be reset, this difference remains fixed forever.

On the other hand, suppose that Agnes decided to execute the time points in χ_f at an earlier time, say, $\texttt{now}_{PS}+2$. This would correspond to her making the transition to the `ctrl` location and instantaneously executing the time points in χ_f at that time and, then, immediately returning to the `env` location. Since `ctrl` is an *urgent* state, the global clock equals $\texttt{now}_{PS} + 2$ when the return transition is made. This sequence of transitions corresponds to the *partial* outcomes in Cases (2a) and (2b) in definition 88, where Agnes' decision is $\langle T_f, \chi_f \rangle$, where $T_f = \texttt{now}_{PS} + 2$. Furthermore, if Vera chooses to instantaneously execute some uncontrollable time points at that same time, $\texttt{now}_{PS} + 2$, that will correspond to an instantaneous reaction, as specified in definition 87.

Finally, if at time $\texttt{now}_{PS}$, Agnes and Vera both decided to execute some time points at time $\texttt{now}_{PS} + 1$, then the STNU semantics ensures that Agnes' time points will be executed, and that Vera will be able to instantaneously react, if she chooses. This corresponds to Agnes' transition having *priority* over Vera's transition. Agnes transitions to the $\texttt{ctrl}$ state, executes her time points, and returns to the $\texttt{env}$ state, with the global clock ending up at $\texttt{now}_{PS} + 1$.

Since, in all cases, the resulting state of the TGA satisfies the desired invariant property, the result is proven. □

Theorem A.3. *Let $\mathcal{S}$ be any STNU; let Θ be the encoding of $\mathcal{S}$; and let σ be a winning TGA counter-strategy for Agnes. Then there is an equivalent RTED-based strategy for Agnes that will ensure the satisfaction of all constraints in $\mathcal{S}$ no matter how the contingent durations turn out.*

Proof. Let $\mathcal{S}, \Theta$ and σ be as described in the statement above. Therefore, $\sigma : L \times \mathbb{R}^{\mathcal{X}}_{>=0} \rightarrow Act_u \cup \{\lambda\}$, where Act_u is the set of uncontrollable actions (for Agnes).

Suppose the TGA has just entered the state, $\langle \texttt{env}, v\rangle$, where v represents the vector of clock values. As has already been noted, for any time point X and associated clock $\overline{x}$: (1) before X executes, $\overline{x} = \hat{\texttt{t}}$; and (2) after X executes, $\overline{x} < \hat{\texttt{t}}$ and the fixed difference, $\hat{\texttt{t}} - \overline{x}$, equals the time at which X executed. Thus, the vector of clock values specifies a partial schedule, PS.

Now, suppose that $\texttt{now}_{PS} < \hat{\texttt{t}}$ (i.e., that some positive time has elapsed since the last execution event in PS). The only way that could have happened is if the state $\langle \texttt{env}, v\rangle$ had been preceded by one or more useless loops (i.e., loops using only the $\texttt{gain}$ and $\texttt{pass}$ transitions to go back and forth between $\texttt{env}$ and $\texttt{ctrl}$ without executing any time points). Let $\langle \texttt{env}, v'\rangle$ be the state immediately preceding the first such useless loop. Then for some positive ϵ, $v = v' + \epsilon$ (i.e., the clock values in v are ϵ units larger than their corresponding values in v'). And by construction, $\texttt{now}_{PS} = v'(\hat{\texttt{t}})$.

Next, let D be the minimum time that can elapse from v before the strategy σ recommends a non-trivial transition to the $\texttt{ctrl}$ location. That is: $D = \min\{d \mid \sigma(\texttt{env}, v' + d) \neq \lambda,\ \sigma(\texttt{ctrl}, v' + d) \neq \texttt{pass}\}$. Let $v_0 = v' + D$. The unique sequence of *execution* transitions at the `ctrl` location is: $\tau_1 = \sigma(\texttt{ctrl}, v_0)$, $\tau_2 = \sigma(\texttt{ctrl}, v_1), \tau_3 = \sigma(\texttt{ctrl}, v_2), \ldots$, where each v_{i+1} is the same as v_i, except that the clock for the just-executed time point is 0 in v_{i+1}. This sequence must terminate, since there are only finitely many time points, and each can be executed only once. If τ_m is the last execution transition, it follows that $\texttt{pass} = \sigma(\texttt{ctrl}, v_m)$. That transition leads back to the state, $\langle \texttt{env}, v_m \rangle$, where v_m is the same as v', except that the clocks for the time points executed by the transitions, $\tau_1, \ldots, \tau_m$, are all zero in v_m.

Next, let $T_f = v_0(\hat{\texttt{t}})$ be the global time at which σ recommends its first non-trivial transition to `ctrl`; and let χ_f be the set of time points that correspond to the execution transitions, $\tau_1, \ldots, \tau_m$. Then $\langle T_f, \chi_f \rangle$ is an RTED for PS that corresponds to what the strategy σ recommends at $\langle \texttt{env}, v' \rangle$. Note that Vera may decide to instantaneously react by executing some uncontrollable time points also at time T_f, an outcome that is sanctioned by the execution semantics for STNUs. Finally, it may happen that Vera decides to intervene *before* time T_f arrives, by executing one or more uncontrollable time points and effectively generating a new partial schedule, PS^*. In that case, the same procedure could be applied to PS^* to generate an appropriate RTED. Since the guard on the transition from `env` to `ctrl` requires a positive time delay, that RTED is properly prohibited from any kind of instantaneous reaction (by Agnes).

This procedure provides a mapping from any $\langle \texttt{env}, v \rangle$ state that is reachable following the winning strategy σ. In addition, the sequences of partial schedules generated by following the RTEDs correspond to runs that can be produced by σ. Thus, the complete schedules generated by the RTEDs are guaranteed to satisfy all STNU constraints assuming Vera observes the bounds on all contingent links. $\square$

A.6 Dynamic Controllability Proofs

Theorem 11.1 (Sufficient Syntax)**.** *A DTNU is dynamically controllable if and only if it admits a solution strategy expressible as per definition 56.*

Proof. Following the correct TGA encoding in section 11.3, we know that a memory-less TGA strategy exists for every dynamically controllable problem. Our strategy can be seen as a representation of the computation tree of that TGA strategy. □

Theorem 11.2 (Correctness)**.** *With no pruning, the algorithm terminates and returns $\perp$ if and only if the TNU is not dynamically controllable.*

Proof. (Sketch) The search space explored by algorithm is a restriction of the one for the TGA in section 11.3 that captures the dynamic controllability problem semantics. We remove states whose winning set is empty as they cannot satisfy the free constraints: this is a sound and complete restriction. Algorithm 15 is derived from [CDF^{+}05] that correctly solves any TGA. □

Appendix B

Planning Proofs

B.1 Proof of DR Approach Completeness

In this section, we prove theorem 12.1.

Definition 93 (Abstract Plan χ)**.** *Given a time-triggered, strong plan $\sigma \doteq \{\langle t_1, a_1, d_1\rangle, \cdots, \langle t_n, a_n, d_n\rangle\}$, an abstract plan χ is the ordered sequence of steps obtained by exploding each action a_i of σ in its snap components and considering the order given by the time of the happenings: each $s_i^{a_{sta}}$ happens at time t if it corresponds to a $\langle t, a, d\rangle$, each $s_i^{a_{eta}}$ happens at time $t + d$ if it corresponds to a $\langle t, a, d\rangle$ with $d \neq \bot$, or at time $t + x$ with $x \in bounds(a)$ if it corresponds to a $\langle t, a, \bot\rangle$.*

Lemma B.1. *Given a strong plan σ and any abstract plan χ of σ, σ is valid for a STPUD if and only if the DTNU K created by algorithm 19 with DR is strongly controllable.*

Proof. Clearly, K is defined over all (and only over) the snap actions of the action appearing in σ.

First, we prove that if σ is valid, then K is strongly controllable. Let μ be the assignment to the controllable time points of K, defined as follows.

$$\mu(x) = \begin{cases} t(x) & \text{if } x = s_i^{a_{sta}} \text{ for some action } a \\ t(a_{st_a} + \delta(a)) & \text{if } x = s_i^{a_{eta}} \text{ for some controllable } a \end{cases}$$

We now prove that μ is a strong schedule for K. For the sake of contradiction, suppose it is not. Then, there exists a duration for the uncontrollable actions for which one of the free constraints in K is violated. It is impossible to violate a duration constraint, therefore one of the three constraints in definition 67 must be violated for some $\bar{a}$. This is impossible, because *sigma* is a valid plan and if we violate constraint 2 or constraint 3, it means that there the preconditions of the action in σ corresponding to $\bar{a}$ are unsatisfied, if we violate constraint 4, then the overall conditions of the action in σ corresponding to $\bar{a}$ are unsatisfied.

Now, we prove that if K is strongly controllable, σ is valid. Reversing the argument before, we assume to have a strong schedule μ for K, and we prove that setting each step s of sigma as follows, yields a valid strong plan.

- $t(s) = \mu(s_i^{a_{sta}})$
- $\delta(s) = \mu(s_j^{a_{eta}} - \mu(s_i^{a_{sta}})$, if s is controllable.

For the sake of contradiction, assume that σ defined as above is not a valid strong plan. Then there exists a temporal plan $\pi \in I_\sigma$ that is not a valid plan for the domain in which we removed temporal uncertainty as per definition 64. If π is invalid, it is either causally unsound (inapplicable in the initial state, not simulable, not leading to the goal state) or it violates some temporal constraint of the domain. But π cannot be causally unsound, because it fulfills all the constraints of definition 67; and it cannot violate a temporal constraints, because the only temporal constraints in the plan are the duration of actions that are encoded in K and fulfilled by μ. $\square$

The proof of theorem 12.1 descends from lemma B.1.

Theorem 12.1 (DR Completeness). *Given a STPUD admitting a valid strong plan σ, if DR is used, algorithm 19 terminates with a valid strong plan.*

Proof. We assume that the classical planner employed in algorithm 19 is sound and complete. Therefore, sooner or later it will produce an abstract plan χ of a valid strong plan σ as it is a plan achieving the goal. Then, by lemma B.1, we know that the DR approach yields a strongly controllable DTNU, and therefore the algorithm terminates with a valid strong plan. $\square$

B.2 STPUD Formal Compilation Proof

In this section we prove theorem 12.2.

B.2.1 Plan Mapping

Consider a plan $\sigma \doteq \langle\langle t_1, a_1, d_1\rangle, \cdots, \langle t_n, a_n, d_n\rangle\rangle$ for an STPUD R. We call π_σ the *regression plan* for P when it has actions a_i^π corresponding to the actions a_i in σ such that:

- a_i^π also starts at time t_i.
- a_i^π has duration d_i if a_i is controllable,
- otherwise the duration of a_i^π is unspecified.

Analogously, we call σ^π the *projection plan* for R of a strong plan π for P obtained by fixing the duration of each uncontrollable action in π to its maximum.

B.2.2 Plan Execution

Given a temporal plan, an *execution* ϵ^X of a temporal planning instance X, is a set of changes applied to the variables: $\epsilon \doteq \{(t_1, f_1, v_1), \cdots (t_n, f_n, v_n)\}$.

An element $\langle t_i, f_i, v_i\rangle \in \epsilon^X$ means that at time t_i the variable f_i takes value v_i. Following definition 62, we take the view that at time t_i the change is not yet visible: the value v_i is taken immediately after t_i. Such an element of ϵ^X can be caused by either: (1) an initial condition (i.e. $t_i = 0$); (2) by a Timed Initial Literal or (3) by an action effect. Therefore, for the rest of this proof we refer to *execution elements* as either initial conditions, timed initial literals or action effects. Moreover, $\mathtt{ex_X} ft$ represents the value v of f at time t during the execution ϵ^X.

Given a strong plan π for P, we have a set of possible executions, one for each possible duration of each uncontrollable action in π. For a given execution ϵ^P, then $\mathtt{ex_P} ft$ indicates the value of variable f at time t during this particular execution ϵ^P.

We now need to compare the executions of plans for P and R. The next theorem states that if σ is a valid plan for R and its corresponding regression plan for R is π_σ, then the variables in each pair of executions are aligned at each time in which f_σ is equal to f during the execution of R.

Lemma B.2. *Given a valid plan σ for R and its corresponding regression plan π_σ, for each execution ϵ^P of π_σ: if $\mathtt{ex_R} f_\sigma t = \mathtt{ex_R} ft$, then $\mathtt{ex_R} ft = \mathtt{ex_P} ft$, for each variable $f \in L$ and each $t \in \mathbb{R}$.*

Proof. For the sake of contradiction, let us focus on an execution ϵ^P in which there is $t \in \mathbb{R}$ such that $\mathtt{ex_R} f_\sigma t = \mathtt{ex_R} ft$ but $\mathtt{ex_R} ft \neq \mathtt{ex_P} ft$.

Let $E^R_{f_\sigma} \doteq \langle t^R_\sigma, f_\sigma, v^R_\sigma \rangle$, $E^R_f \doteq \langle t^R, f, v^R \rangle$ and $E^P_f \doteq \langle t^P, f, v^P \rangle$ be the latest execution elements involving f_σ and f in ϵ^R and ϵ^P. By our hypothesis, $v^R \neq v^P$. Moreover, due to the translation constraints, we know that $v^R_\sigma = v^R$ and $t^R_\sigma \leq t^R$. In fact, each time we change f in R we also change f_σ either at the same time or at the beginning of an uncertain interval d and we prevent any other change on f_σ until d is over.

We prove that this hypothesis is impossible by considering all possible cases.

1. E_f^R and $E4_f^P$ are both initial conditions: since all the initial conditions are copied from P to R, we must have $v^P = v^R$.

2. E_f^R and E_f^P are both timed initial literals (TIL): since all TILs are copied from P to R, either $v^P = v^R$ or there are two TILs at the same time on the variable f, which is not allowed.

3. E_f^R is either a TIL or an initial condition and E_f^P is the effect of action a:

   - If a is uncontrollable, there is a TIL $\langle t^R, f, v^R \rangle$ also in P corresponding to E_f^R. Given that E_f^P is an action effect, it must be the case that t_σ^R happens before t^P, but after t^R. Hence, either $v_\sigma^R \neq v^R$ or E_f^R is not the last effect changing f.
   - If a is controllable, then there should be an effect corresponding to E_f^P for R at time t^P. Therefore, either E_f^R or E_f^P is not the latest effect modifying f in the executions of R and P, respectively.

4. E_f^P is either a TIL or an initial condition and E_f^R is an action effect: since TIL are copied from P to R, there is a TIL analogous to E_f^P in R. Hence $t^R \geq t_\sigma^R > t^P$. However, since t^R and t_σ^R are the two extremes of an uncertainty interval, there must be another effect on f in P during this interval. Hence, E_f^P is not the last effect changing f in P.

5. E_f^P and E_f^R are both effects of two actions instances a and a': four following possible sub-cases.

   - Both E_f^P and E_f^R are effects of controllable actions (or effects of uncontrollable actions that happen at times independent of the duration): they must be the same effect since the controllable actions in P are kept in R without changing the duration nor the effect times.

- Both E^R_f and E^P_f are effects of uncontrollable actions: E^R_f and E^P_f must be the same effect of the same uncontrollable action, because of the condition imposed on f_σ. If this were not the case, then the condition preserving the value of f_σ for the whole uncertainty interval would have been violated in ϵ^R.
- E^R_f is an effect of a controllable action (or effect of an uncontrollable action happens at a time independent of the uncertain duration) while E^P_f comes from an uncontrollable action. Since E^P_f is an uncontrollable effect, there is a corresponding pair of effects on f and f_σ for the execution in R. If $t^R > t^P$ then there would be an effect on f equivalent to E^R_f that happens after t^P but before t. If E^P_f happens after E^R_f, then either $\texttt{ex}_\texttt{R} f_\sigma t \neq \texttt{ex}_\texttt{R} f t$ or E^P_f is not the latest effect on f.
- E^P_f is an effect of a controllable action (or effect of an uncontrollable action happens at a time independent of the uncertain duration) while E^R_f comes from an uncontrollable action. This is analogous to the previous case.

□

Lemma B.3. *Given a strong plan π for P and its corresponding projection plan σ^π for R, let ϵ^R be the execution of σ^π. For each variable $f \in L$, each $t \in \mathbb{R}$ and each execution of P: if $\texttt{ex}_\texttt{R} f_\sigma t = \texttt{ex}_\texttt{R} f t$ then $\texttt{ex}_\texttt{R} f t = \texttt{ex}_\texttt{P} f t$,*

Proof. We apply the same reasoning as in the proof of the previous lemma. Cases 1 and 2 are identical to the previous proof, the other cases are changed as follows.

3. E^R_f is either a TIL or an initial condition and E^P_f is the effect of an action a.

- If a is uncontrollable, there should be a TIL $\langle t^R, f, v^R \rangle$ in P corresponding to E_f^R. Given that E_f^P an action effect, there is an execution of P where t^P happens before t^R and another in which it happens after (no two effects of the same action on the same variable can happen at the same time). Hence, π is not a valid strong plan.
- If a is controllable, then there is an effect corresponding to E_f^P also for R at time t^P, therefore either E_f^R or E_f^P is not the latest effect modifying f in the executions of R and P.

4. E_f^P is either a TIL or an initial condition and E_f^R is an action effect. Since TILs are copied from P to R, there must be a TIL analogous to E_f^P in R. Hence $t^R \geq t_\sigma^R > t^P$. But there are two executions of P in which $t^R = t^P$ and $t_\sigma^R = t^P$ (t^R and t_σ^R the extremes of an uncertainty interval). Hence, E_f^P is not the last effect changing f in P.
5. E_f^P and E_f^R are both effects of two actions instances a and a'.

   Four sub-cases are possible:

   - Both E_f^P and E_f^R are effects of controllable actions (or effects of uncontrollable actions that happen at times independent of the uncertain action duration): they must be the same effect, as the controllable actions in P are all kept in R without changing the duration or the effect times.
   - Both E_f^R and E_f^P are effects of uncontrollable actions: E_f^R and E_f^P must be the same effect of the same uncontrollable action, because actions are started at the same time in the two executions and no pair of effects on the same variable are allowed if the plan is strong.
   - E_f^R is an effect of a controllable action (or happens at a time independent of the duration) while E_f^P comes from an uncontrollable

action. Since E_f^R is an uncontrollable effect, there is an original effect in P that is concurrent with E_f^P. If $t^R > t^P$ then there would be an effect on f equivalent to E_f^R after t^P and before t. If E_f^P happens after E_f^R, then either $\texttt{ex}_\texttt{R} f_\sigma t \neq \texttt{ex}_\texttt{R} f t$ or E_f^P is not the latest effect on f.

- E_f^P is an effect of a controllable action (or happens at a time independent of the duration) while E_f^R comes from an uncontrollable action. This is analogous to the previous case.

□

Theorem 12.2 (Soundness and Completeness)**.** *Let $P \doteq \langle V, I, T, G, A \rangle$ be a planning instance and $R \doteq \langle V', I', T', G', A' \rangle$ be its translation. P has a strong plan π if and only if R has a temporal plan σ.*

Proof. Let π be a strong plan for P. σ^π is a valid temporal plan for R because:

- It achieves the goal G' of R, because all the original goals in G are achieved by π and by σ in the same way, and the goals on the shadow variables must be achieved because π is a strong plan. In fact, π achieves the goals regardless of the concrete durations of the actions, therefore it achieves them outside of the uncertainty intervals, where the variables and the shadow variables are aligned because of lemma B.3.

- Each action a' is executable in R, because each $a \in \pi$ is executable in P regardless of the action durations. Thus the possible uncertainty introduced by the durations is irrelevant for the executability of a (all the conditions are satisfied and variables and the shadow variables are aligned because of lemma B.3). In the translated instance R, all the conditions are also satisfied because the conditions are imposed via the γ function that only checks that both the variable and its shadow fulfill the original condition.

- No conflicting effects are possible because of the conditions added in $C^E_{a'}$ that prevents any modification of the interested shadow variables during the uncertainty intervals.

Similarly, let σ be a plan for R. Then π_σ is a valid strong temporal plan for P because:

- It achieves the goal G, because σ achieves the goal G' that is a super-set of G, and the variables are aligned because of lemma B.2.
- Each action a is executable in P regardless of the action duration, because the variables are aligned because of lemma B.2, $a' \in \sigma$ is executable in R and the conditions in the translated actions are a super-set of the ones in the original action, because of the γ function.
- No conflicting effects are possible regardless of the uncertain duration, because each effect at time t can be uncertain only between $\lambda(t)$ and $\nu(t)$ and we guarantee no other effect is possible in that interval by means of $C^E_{a'}$.

□