Università degli Studi di Ferrara

Corso di Laurea in Informatica

Implementation of a Tableau-Based Satisfiability Checker for the Logic HS_3

Relatore:

Prof. Guido Sciavicco

Laureando:

Ionel Eduard Stan

Anno Accademico 2016 – 2017

Abstract

*Nelle logiche temporali ad intervalli (*interval temporal logics*), gli intervalli di tempo, al posto di soli punti, sono presi come principali entità ontologiche. In informatica le logiche temporali intervalli di tipo proposizionale giocano un ruolo significante e possono fornire un* framework *naturale per rappresentare e ragionare su proprietà temporali in vari domini di applicazioni. Tali logiche possono essere applicate a molti campi, come ad esempio sistemi hardware e sistemi di verifica in tempo reale (*real-time*), per il processamento di linguaggi, soddisfazione di vincoli e pianificazione, insieme ad altri. Ancor di più, grazie al fatto che le logiche temporali sono considerate come la base naturale per le estensioni temporali delle logiche descrittive (*description logics*), molti tentativi sono stati fatti per disegnare estensioni basate su intervalli di tali formalismi. Le logiche temporali ad intervalli possono essere pensate anche come la controparte temporale di TSQL, cioè, l'estensione temporale del linguaggio SQL per le basi di dati, incluso nello standard SQL:2011. La logica modale sulle relazioni intervalli di Allen chiamata* HS*, dagli autori Halpern e Shoham, potrebbe essere considerata la logica temporale intervallare più rappresentativa. La sua eleganza e il suo potere espressivo hanno chiamato l'attenzione della comunità delle logiche modali e temporali; tuttavia, applicazioni promettenti sono state ostacolate dal fatto, già scoperto quando la logica è stata introdotta, che il problema della soddisfacibilità di* HS *è altamente non decidibile.*

*L'algebra intervallare di Allen (*IA*) è la spina dorsale di* HS*: operatori modali*

nel repositorio di HS possono essere mappati uno-ad-uno sopra le relazioni ad intervalli di Allen. L'idea di Golumbic e Shamir è di considerare relazioni ad intervalli che descrivono relazioni topologiche tra intervalli in maniera meno precisa riducendo l'insieme di relazioni binarie dell'algebra di Allen e definendo relazioni più "grezze", ciascuna corrispondente alla disgiunzione logica di alcune delle relazioni di Allen. Questo approccio genera due algebre naturali grezze chiamate IA_7 e IA_3; queste algebre grezze ispirano le logiche HS_7 e HS_3, il cui problema della soddisfacibilià è stato già studiato. Questi linguaggi seguono idee simili dallo standard SQL:2011, dove relazioni ad intervalli non sono necessariamente quelle di Allen (per esempio, la relazione later *è interpretata come una disgiunzione delle relazioni di Allen* meets *e* later*); pertanto, non possono essere applicate a sole aree classiche dell'Intelligenza Artificiale, ma anche a basi di dati temporali. HS_7 è indecidibile su ogni classe interessante di insiemi linearmente ordinati, mentre HS_3 è* PSPACE-COMPLETE *nel caso finito/discreto, ed è* PSPACE-HARD *quando è interpretato su qualunque classe interessante di insiemi linearmente ordinati.*

In questa tesi, inizialmente studiamo ed analizziamo alcuni dei linguaggi formali che sono (attivamente) usati nel ambito dell'Intelligenza Artificiale con lo scopo di mettere in evidenza il loro potere espressivo, cercando di trovare il giusto compromesso con le proprietà computazionali. Alcuni dei risultati presentati sono ben noti nella letteratura mentre altri sono recenti. Tale studio viene fatto per motivare e, soprattutto, per convincerci della convenienza della scelta (o la non scelta) di un logica rispetto ad un'altra. Facendo particolare enfasi su tale compromesso, presentiamo un esempio pratico di un sistema reale modellato attraverso la logica HS_3, mostrandone l'utilità e l'eleganza della logica stessa.

*È stato recentemente dimostrato che il problema della soddisfacibilità di formule espresse in HS_3 può essere ridotto dando un limite superiore ad ogni modello finito; in particolare, grazie ad un teorema di modello piccolo (*small model theorem*). Oltre a questo risultato, tale dimostrazione da un algoritmo non deterministico di complessità* PSPACE. *Una volta presentato l'esempio reale modellato attraverso HS_3, risolviamo il problema di implementare in maniera efficiente tale algoritmo teorico. Prendendo spunto dalla logica classica, abbiamo scelto di implementare un sistema a tableau motivato dal fatto che il tableau è più intuitivo ed orientato a migliori ottimizzazioni. Un sistema deduttivo automatico corretto,* *completo e terminante che risolva il problema della soddisfacibilità di una certa logica (nel nostro caso di* HS_3*) è condizione essenziale per avere un sistema di Intelligenza Artificiale che è basato su un certo formalismo. Rendere l'algoritmo teorico non deterministico in un algoritmo pratico è stato difficile come ci aspettavamo. Data l'implementazione, abbiamo testato le capacità computazionali su formule di grandezza crescente, costruite in maniera sistematica. I test ci hanno permesso di analizzare i risultati dando luogo a future idee ed ottimizzazioni per le due categorie principali di formule; in particolare, per le formule finitamente soddisfacibili e non.*

Acknowledgements

My most sincere gratitude goes to my supervisor *Guido Sciavicco*. Thank you *Guido* from the bottom of my heart for being much more than an ordinary professor to me. Thanks for your far-sightness in letting me working with you, for being patient and for sharing your knowledge with me.

To all my friends. Particularly, thanks to my best friend *Damiano*. Thank you for always being at my side, for your kindness and for spurring me in going to Erasmus.

To my beloved girlfriend, *Andrea*. Thank you for your sweetness, for your support and for filling my days with joy and happiness.

Thanks to my entire family. Most specially, to my parents without whom this would be only a dream. Thank you for offering me a shelter, for your support during these years and because you always believed in me that the sun will come on your way too.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In *Interval Temporal Logics* (ITLs), time intervals, rather than points, are taken as primitive ontological entities. Propositional ITLs play a significant role in computer science and can provide a natural framework for representing and reasoning about temporal properties in a number of application domains [GMS04]. ITLs can be applied in many fields, such as hardware and real-time system verification, language processing, constraint satisfaction and planning, among other [All83, CH04, Mos83, PH05]. Moreover, due to the fact that temporal logics are considered as the natural basis for temporal extensions of Description Logics [ARKZ15], several attempts have been made to design interval-based extensions of such formalisms, as in [Sch90, Bet97, AF98, ABM$^+$14]. ITLs can be also considered as the temporal counterpart of TSQL, that is, the temporal extension to the language SQL for databases, included in the standard SQL:2011 [Zem12]. *Halpern and Shoham's Modal Logic of Allen's Relations* (HS) may very well be the most prominent ITL [HS91]. Its elegance and expressive power have attracted the attention of the temporal and modal logic communities; however, promising applications have been hampered by the fact, already discovered when the logic was first introduced, that HS is highly undecidable.

Allen's Interval Algebra (IA) [All83] is the backbone of HS: modal operators in the HS repository can be mapped one-by-one over Allen's interval relations.

Golumbic and Shamir's [GS93] idea is to consider interval relations that describe a less precise topological relationship between intervals reducing the set of binary relations of Allen's Interval Algebra [All83] and defining coarser relations, each corresponding to the logical disjunction of some Allen's relations. This approach generates two natural coarser algebras, namely IA_7 and IA_3; these coarser algebras inspire the logics HS_7 and HS_3, whose satisfiability problem has been studied in [MPSS15]. These languages follow similar ideas to the standard SQL:2011 [Kla14], where interval relations are not necessarily Allen's ones (for example, *later* is interpreted as the disjunction of Allen's *meets* and *later*); therefore, they cannot only be applied to classical areas of Artificial Intelligence, but also to temporal databases. HS_7 is undecidable over every interesting class of linearly ordered sets, while HS_3 is PSPACE-COMPLETE in the finite/discrete case, and it is PSPACE-HARD when interpreted in any interesting class of linearly ordered sets.

The result from [MPSS15] reduces the satisfiability problem of formulæ of HS_3 given an upper bound limit to the length of every finite model and gives a non deterministic algorithm with PSPACE complexity. In this thesis, we solve the problem of implementing in an efficient way the algorithm, and testing its computational power on formulæ of increasing length, generated in a systematic manner.

The thesis is organized as follows. We first give the necessary the right focus on the formalization of reasoning in Artificial Intelligence, and in Chapter 3 we give an example of applied Artificial Intelligence via language formalization. In Chapter 4 we describe an efficient implementation of a satisfiability checker for HS_3. Chapter 5 is devoted to the experiments made giving the results that we have achieved. Finally, we discuss the conclusions on our work pointing our some remarks along with the further work ideas.

Chapter 2

(Temporal) Logics in Artificial Intelligence

Every formal science (e.g. mathematics, computer science, physics) needs a language to be able for expressing results and communicate information with others in a comprehensible way. Logic's purpose is to study this formal representation without ambiguity in terms of a formal language. It is obvious that does not exist an universal logical language to represent every science; this explains why nowadays the literature is flourishing of different types of logic.

The logic is that part of science that provides to humans those crucial mathematical mechanisms to securely control reasonings. Artificial Intelligence uses logic as its formal language for knowledge representation. The natural and artificial languages are rich in *expressive power* and for this reason we need a language to describe such naturally inherent expressive conjectures.

In this chapter, our purpose is the study of the *satisfiability problem*, often (also) called SAT problem [Sis12], in order to build a program for it, namely SAT checker. The target is to present a journey through a possible set of formal languages (actively) used in Artificial Intelligence. The analysis of these logics is aimed to give a strong impact on what, in general, the logic communities have to deal with, when time to make decisions arrives. As in our case, a decision could be the choice of a formal language against another (to give a representation of a problem) or the selection of an appropriate solving method.

We present several of these logics studying their behavior under certain criteria with the aim of introducing an efficient way of satisfiability check in (mathematical) logic, that is the *tableau*.

2.1 Preliminaries

Formulæ are denoted with $\varphi, \psi, \tau, \ldots, \varphi_1, \varphi_2, \ldots, \psi_1, \psi_2, \ldots, \tau_1, \tau_2, \ldots$. We denote the truth value *true* with $\top$ or with the concrete (boolean) value 1, and the truth value *false* with $\bot$ or with the value 0. A *tree* is a graph $\mathcal{G} = (V, E, r)$, where V is a non-empty set of *vertices* (or *nodes*), $E \subseteq V \times V$ is the set of *edges* between each two nodes and $r \in V$ is the *root* of the tree. A *finite* tree is a directed connected graph where each node, apart for the root, has at most one entering edge. In this abstract representation, a *parent* node $v \in V$ is a vertex that has at least one *child* $v' \in V$, that is an edge from v to v'; sometimes the child is also called *successor*. A node without children is called *leaf*. A finite path π is a sequence of vertices $v_1, \ldots, v_n$ such that, $\forall i = 1, \ldots, n-1$, v_{i+1} is a successor of v_i. A branch $\mathcal{B}$ is a path from the root to one of the leaves. Conventionally we represent formulæ as trees with at most two children. A tableau can also be represented as a tree reusing the same notations when needed.

2.2 Deduction

The goal is to provide a method that proofs the calculus behind *deduction*. Deduction is the hearth of human rationality and it is a set of steps (a calculus) that leads to a conclusion (e.g., an arithmentic calculus is a demonstration). A proof consists on the application of various rules that are simple enough to be easily recognized as valid and small enough in number to be easily remembered.

We choose "one of the one thousand" deductive methods (e.g. see [AC08] for natural deduction, axiomatic systems, sequent calculus, ...) of satisfiability check for the SAT problem, namely (a tree based structure called) tableau. The chosen version of the tableau is the *semantic* tableau (or truth tree) because the principle behind it is simple: search for a *model* (satisfying *interpretation*) by decomposing the formula into sets of *atoms* and negations of atoms. The original formula is divided into indecomposable atoms until a model or a *contradiction* is found. A contradiction is an interpretation over the tableau where a branch $\mathcal{B}$ contains an atom together with its negation, therefore, this particular branch is not a model.

2.3 First Order Logic

Predicate Logic or, more commonly, *First Order Logic* (FOL) is an excellent departure for our survey. In a matter of fact, its important structural properties gives us enough formal "ingredients" to express classes of problems. Our intention is to give a representative (not crossable, for the sake of this thesis) upper bound to expressiveness. As for any language, in its most general form, we need to specify its syntax, through a grammar, of the formal language without associating any meaning to symbols, and to define its semantic by applying a meaning to the interpreted (syntactically correct) phrases. In other words, we need an "artifice" able to generate formulæ, that are phrases, recognized by the language and we have to give an interpretation over these well formed formulæ.

The alphabet Σ over FOL is formed by:

- logical *connectives*: $\bot, \neg, \wedge, \vee, \rightarrow$,
- *quantifiers*: $\forall, \exists$,
- a set $\mathcal{C}$ of *constant* symbols: $a, b, c, d, \ldots$,
- an infinite set VAR of *variable* symbols: $u, v, w, x, y, \ldots$,
- a set $\mathcal{F}$ of *function* symbols: $f, g, h, \ldots$,
- a set $\mathcal{P}$ of *predicate* symbols: $P, Q, R, \ldots$.

The universal quantifier $\forall$ is read as "*for all, every*" and the existential quantifer $\exists$ is read as "*exists, for some*". Each predicate symbol and each function symbol comes with an *arity*, the number of arguments that expects. In fact, constants can be thought of as 0-arity functions which does not take any argument, therefore constants live in the set $\mathcal{F}$ together with the true functions which do take arguments [HR04]; 0-arity functions are also called *nullary* functions.

In computer science, the notion of term is widely used in an implicit way; for instance, an arithmetic expression is a term obtained applying (arithmetical) operators to some operands. Therefore, the set of *terms* T of the FOL language is defined inductively as:

$$t ::= x \mid c \mid f(t_1, \dots, t_n),$$

where x ranges over the set of variables VAR, c over nullary function symbols in $\mathcal{F}$, and f over those elements of $\mathcal{F}$ with arity $n > 0$. Terms are dependent on the set $\mathcal{F}$ because the first elements of them are constants and variables, and whereupon more complex are built from function symbols.

Well formed formulæ over FOL are generated by the following grammar:

$$\varphi ::= P(t_1, \dots, t_n) \mid (\neg\varphi) \mid (\varphi_1 \vee \varphi_2) \mid (\exists x(\varphi)),$$

where $P \in \mathcal{P}$ is a predicate symbol of arity $n \geq 1$, t_i are terms over $\mathcal{F}$ and $x \in \mathsf{VAR}$ is a variable symbol. The remaining connectives are derived from the others as:

- $\bot \equiv \neg(\neg\varphi \vee \varphi)$,
- $\top \equiv \neg\bot$,
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$,
- $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$,
- $\forall x(\varphi) \equiv \neg\exists x(\neg\varphi)$.

In absence of parentheses, we retain binding (precedence) priorities: (i) $\neg, \forall, \exists$ bind most tightly, then (ii) $\wedge, \vee$ and then (iii) $\to$ which is right-associative. Sometimes is helpful to distinct between *free* and *bounded* variables. A variable x is called free if from node x in the parsing tree (see Figure 2.1) there is no upwards path from node x to any node of type $\forall x$ or $\exists x$.

Predicate logic is much more expressive than other logics studied in this chapter. In a matter of fact, in order to define its semantics, *structures* are build with

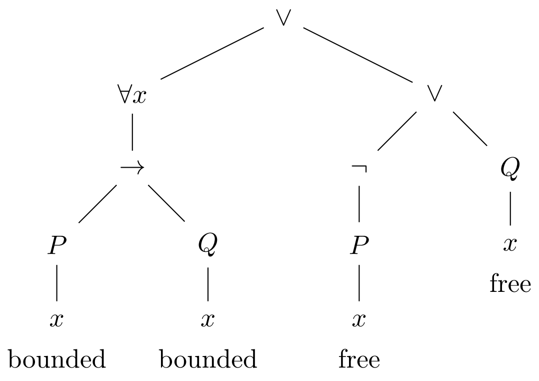


Figure 2.1: Parse tree for $\varphi \equiv (\forall x(P(x) \rightarrow Q(x))) \vee (\neg P(x) \vee Q(x))$.

a set D (the *domain*), together with a collection of functions and *relations* over the domain. A structure for a language is called interpretation. Formally, an interpretation I of the pair $(\mathcal{F}, \mathcal{P})$ consists in the following data set:

- a non-empty set D representing the domain of concrete values,
- for each $c \in \mathcal{C}$ a concrete element c^I of D,
- for each $f \in \mathcal{F}$ with arity $n > 0$ a concrete function $f^I : D^n \rightarrow D$,
- for each $P \in \mathcal{P}$ with arity $n > 0$ a subset $P^I \subseteq D^n$,

where c, f and P are just constant, function and predicate symbols, and c^I, f^I and P^I are concrete constant, function and relation, respectively.

For each term $t \in T$, we need a function $\sigma : T \rightarrow D$, called *state*, that assigns, inductively, to each term t a value $\sigma(t) \in D$, defined as follows:

- if t is a variable x, then $\sigma(t) = \sigma(x)$,
- if t is a constant c, then $\sigma(c) = c^I$,
- if t is a term like $f(t_1, \ldots, t_n)$, then $\sigma(t) = f^I(\sigma(t_1), \ldots, \sigma(t_n))$.

Given an interpretation I along with its state σ, we are able to define semantics of FOL's language as follows:

- $I, \sigma \models P(t_1, \ldots, t_n)$ if and only if $(\sigma(t_1), \ldots, \sigma(t_n)) \in P^I$ meaning that the concrete values $(d_1, \ldots, d_n) \in D^n$ are computed by replacing all terms with their values according to σ,
- $I, \sigma \models \neg\varphi$ if and only if $I, \sigma \not\models \varphi$,
- $I, \sigma \models \varphi_1 \vee \varphi_2$ if and only if $I, \sigma \models \varphi_1$ or $I, \sigma \models \varphi_2$,
- $I, \sigma \models \exists x(\varphi)$ if and only if $I, \sigma \models \varphi$ for some $d \in D$ such that $\sigma(x) = d$.

Formulæ like $P(t_1, \ldots, t_n)$ require extra attention because there is more going on when simple syntactical modifications are made. Given a property P that states "*is bigger than*", consider $P(x, y)$ as "*x is bigger than y*" to express a relation between x and y. Following the definition given for an interpretation I, we can give a sense for "bigger" within a domain of application, but this it is not enough. The expressiveness in FOL gives us some "law" to govern the value assignments of x and y in the domain; for example, if the domain is the set of natural numbers $\mathbb{N}$ and if x and y are, respectively, the numbers 5 and 2 then the predicate is true. We cannot assign truth values x and y without knowing the meaning of these terms. We use terms, and not variables, because $P(f(x, y), y)$ is still a relation over two terms but $f(x, y)$ makes no sense without a meaning. For example, if $f(x, y)$ represents the function that adds x to y then the predicate would be true following the (same) previous assignments for x and y; $P(f(5, 2), 2) = P(5 + 2, 2) = \top$. Still, consider $\exists x \exists y(P(x, y))$ translated as "*there exists some x and some y such that the relation P occurs between x and y*" that, in this case, makes perfect sense because x and y are bounded.

We say that a formula φ is *satisfiable* if there exists a true interpretation I along with its state σ; in this case I is called model $\mathcal{M}$ and we write $I, \sigma \models \varphi$ or $\mathcal{M}, \sigma \models \varphi$. If $\mathcal{M}, \sigma \models \varphi$ holds, we say that φ computes to $\top$ in the model $\mathcal{M}$ with respect to its state σ. A formula φ is *unsatisfiable*, denoted with $\not\models \varphi$, if there is no way to give a true interpretation over it. A formula φ is a *tautology* (or *validity*) if every interpretation is a model or if $\neg\varphi$ is unsatisfiable. Moreover, two formulæ φ_1 and φ_2 are said to be *semantically equivalent*, denoted with $\varphi_1 \equiv \varphi_2$, if and only if for all interpretations I it turns out that $I(\varphi_1) = I(\varphi_2)$.

In order to take a closer look to the inherent difficulty of FOL, we choose the SAT problem as emblematic. It turns out that the SAT problem gives us a good measure of the complexity (of a given problem). There are bad news and good news: the SAT problem for the FOL language is not decidable (see [Coo71, Sis12]). In a matter of fact, we cannot build a terminating algorithm (a program) for it that can output an answer "*yes*" or "*no*" to that decision problem and, therefore, we cannot build a SAT checker. On the other hand, the good news are that several problems can be transformed into SAT problems. For instance, the deduction problem, that is given the premises $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ the target is to conclude the consequent φ can be transformed into SAT verifying if that $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi \in \mathsf{SAT}$ and if it does, clearly, we cannot conclude φ.

2.4 Propositional Logic

The problem with FOL, as presented in the previous section, is that thanks to its expressive power we can capture a too large class of problems. Because of the undecidable nature, we are forced to limit its expressiveness in order to obtain languages that are useful from the practical point of view.

The backbone of all logics is *Propositional Logic* (PL), although is the most basic of them, presents interesting properties. First, its SAT problem is is known to be decidable and it is NP-COMPLETE [Coo71]; thus, we can build terminating programs for it. Second, it is strictly less expressive than FOL, denoted here as PL $\prec$ FOL. Our approach is distinct from those found in the literature (e.g. see [AC08, HR04]) in which, normally, the journey starts with PL, studying its syntactic and semantic structures ending with its expressive limits, and, after these limits are given, later FOL is the subject of the discourse. Limiting the expressive power of an undecidable logic, such as FOL, gives us better computational behaviors, this is the case, for example, of PL.

Formally, the alphabet Σ of the PL language is formed by:

- logical connectives: $\neg, \vee$,
- a set $\mathcal{AP}$ of *atomic propositions*: $p, q, r, t, \ldots$.

Well formed formulæ are generated by the following grammar:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi_1 \vee \varphi_2).$$

$$\frac{\varphi_1 \vee \varphi_2}{\varphi_1 \mid \varphi_2}\ \textsf{(OR)} \qquad \frac{\varphi_1 \wedge \varphi_2}{\begin{matrix}\varphi_1 \\ \varphi_2\end{matrix}}\ \textsf{(AND)}$$

$$\frac{\neg\neg\varphi}{\varphi}\ \textsf{(NEGNEG)} \qquad \frac{\varphi_1 \rightarrow \varphi_2}{\neg\varphi_1 \mid \varphi_2}\ \textsf{(IMPL)}$$

Table 2.1: Expansion rules for PL.

The remaining connectives are derived from the others as:

- $\bot \equiv \neg(\neg\varphi \vee \varphi)$,
- $\top \equiv \neg\bot$,
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$,
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$.

An example is ”$p \rightarrow q$” that could be interpreted as *"if it rains, then I get the umbrella"*. We take this "bare boned" formula as the representative example to point out how other languages can model it through the expressiveness of the language itself.

The investigation of PL is focused on propositions, that is a concept of truth value. These propositions can be atoms or *composition* of atoms. An atom (or atomic formula) are the most simple well formed formulæ of the logic with no deeper structure, meanwhile the more complex well formed formulæ are generated from simple atoms; for instance, atoms in FOL are formulæ of the form $P(t_1, \ldots, t_n)$ for P a predicate and the $t_1, \ldots, t_n$ terms, and composition of formulæ are those obtained, recursively, applying logical connectives and/or quantifiers to atoms.

Formally, semantics are defined as:

- $\mathcal{M} \models p$ if and only if $p \in \mathcal{M}$,
- $\mathcal{M} \models \neg\varphi$ if and only if $\mathcal{M} \not\models \varphi$,
- $\mathcal{M} \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{M} \models \varphi_1$ or $\mathcal{M} \models \varphi_2$.

In this formal language, an interpretation is a function $\nu : \mathcal{AP} \rightarrow \{\top, \bot\}$, that is a truth value assignment to each atomic formula. For instance, if $\mathcal{AP} = \{p, q\}$

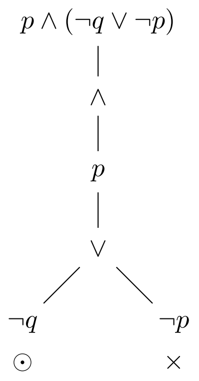


Figure 2.2: Check the satisfiability of $\varphi \equiv p \wedge (\neg q \vee \neg p)$.

then there are 2^2 different possible interpretations: (i) $\nu(p) = \top$, (ii) $\nu(p) = \bot$, (iii) $\nu(q) = \top$ and (iv) $\nu(q) = \bot$. Moreover, if an interpretation ν is a model $\mathcal{M}$ for a formula φ then we write $\mathcal{M} \models \varphi$. The idea is to assign truth values to these propositions and to extend them through the meaning of the connectives that generate composition of atoms, that are phrases.

It clearly emerges a set of limitations on PL (with respect to FOL):

1. the domain is made only by a point,
2. there are no free variables, and
3. there is no state σ.

In PL's case, it turns out that we can easily build a tableau-based deduction method. Such method is described by the *expansion rules* figured in Table 2.1. In this thesis, our study is concerned on propositional logics (whom use PL as backbone) and in Figure 2.2 we give a simple example of tableau satisfiability check for $\varphi \equiv p \wedge (\neg q \vee \neg p)$. We represent a model $\mathcal{M}$ with an *open* branch, denoted with $\odot$, and a *closed* branch, namely contradiction, with $\times$.

2.5 Modal Logic

Modal Logic (ML) is used in computer science, for example, to formalize reasoning about the way programs behave and to express dynamical properties of transition between states. The first study dates back to Aristotle who concerned his attention on *modes* of truth, namely *necessary* and *possible* truths.

The basic propositional ML syntax is based on the language:

- logical connectives: $\neg, \vee$,
- an unary modal operator: $\Diamond$,
- a set $\mathcal{AP}$ of atomic propositions: $p, q, \ldots$,

where $\Diamond$ is the existential *modality* that expresses possibility. Well formed formulæ are like:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi_1 \vee \varphi_2) \mid (\Diamond\varphi).$$

The remaining connectives are derived from the others as:

- $\bot \equiv \neg(\neg\varphi \vee \varphi)$,
- $\top \equiv \neg\bot$,
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$,
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$,
- $\Box\varphi \equiv \neg\Diamond\neg\varphi$,

where $\Box\varphi \equiv \neg\Diamond\neg\varphi$ stands for "*what is necessarily true is not possibly not true*". Moreover, $\Box\varphi$ and $\Diamond\varphi$ in terms of computer programs could say that "φ *will be true after every execution of the program*" and "φ *will be true after some execution of the program*", respectively.

Now, consider the example "$p \rightarrow q$" from Section 2.4. A more sophisticated (i.e. expressive) example for ML is "$p \rightarrow \Diamond q$" that could be interpreted as "*if it rains, then it is possible that I get the umbrella*". We say that ML is strictly less expressive than FOL, but strictly more expressive than PL; in this case, we write PL $\prec$ ML $\prec$ FOL. Now things should be clearer on what we mean by saying that a language is less/more expressive than another.

The semantics structures for ML are defined in terms of *Kripke frames* and *Kripke models* [Kri63]. Formally, a Kripke frame is a pair $\langle W, R\rangle$ where W is

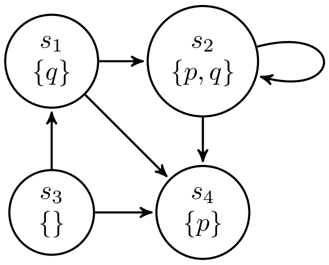


Figure 2.3: Kripke model where $V(p) = \{s_2, s_4\}$ and $V(q) = \{s_1, s_2\}$.

a non-empty set of possible worlds and $R \subseteq W \times W$ is an accessibility relation between possible worlds. A Kripke model over a frame T is a pair $\langle T, V\rangle$ where $V : \mathcal{AP} \to \mathcal{P}(W)$ is a valuation assigning to every atomic proposition the set of possible worlds where it is true. $\mathcal{P}(W)$ is the power set of W. In this thesis, we concern (only) on the logic K of all Kripke models whose SAT problem is known to be PSPACE-COMPLETE [Lad77, Gol03]. The truth of a formula φ at a possible world u in a Kripke model $\mathcal{M} = \langle W, R, V\rangle$, denoted as $\mathcal{M}, u \models \varphi$, is defined as follows:

- $\mathcal{M}, u \models p$ if and only if $u \in V(p) \quad \forall p \in \mathcal{AP}$,
- $\mathcal{M}, u \models \neg\varphi$ if and only if $\mathcal{M}, u \not\models \varphi$,
- $\mathcal{M}, u \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{M}, u \models \varphi_1$ or $\mathcal{M}, u \models \varphi_2$,
- $\mathcal{M}, u \models \Diamond\varphi$ if and only if $\mathcal{M}, w \models \varphi$ for some $w \in W$ such that $(u, w) \in R$,

and, respectively, $\mathcal{M}, u \models \Box\varphi$ if and only if $\mathcal{M}, w \models \varphi$ for every $w \in W$ such that $(u, w) \in R$.

Consider the Kripke model $\mathcal{M}$ in the Figure 2.3. It is easy to verify if

$$\mathcal{M}, s_1 1 \overset{?}{\models} q \wedge \Box p.$$

It is true that q holds in s_1 and from all worlds accessed from s_1 it is true that p holds, therefore, it leads us to a conclusion: the formula $\varphi \equiv q \wedge \Box p$ is satisfiable within the model $\mathcal{M}$ whose initial world is s_1, therefore $\mathcal{M}, s_1 \models \varphi$.

We are interested in studying, in general, the expressiveness of logics. ML is backboned by PL but formulæ from ML can be translated into FOL language according to the *standard translation* (ST). Let L be a FOL language with R a 2-arity predicate (representing, in this case, the accessibility relation between worlds), individual variables $\mathsf{VAR} = \{x, y, \ldots\}$ and a set of 1-arity predicates $P_0, P_1, \ldots$ corresponding to the set $\mathcal{AP}$ of atomic propositions $p_0, p_1, \ldots$. The formulæ of ML are translated into L by the following standard translation ST, where the individual variable x (with respect to FOL, $\sigma(x) = u$ where $u \in W$ is the possible world of the Kripke model) is a parameter:

$$\begin{aligned}
ST_x(p_i) &\equiv P_i(x) \quad \forall p_i \in \mathcal{AP},\\
ST_x(\bot) &\equiv \bot,\\
ST_x(\neg\varphi) &\equiv \neg ST_x(\varphi),\\
ST_x(\varphi_1 \vee \varphi_2) &\equiv ST_x(\varphi_1) \vee ST_x(\varphi_2),\\
ST_x(\Diamond\varphi) &\equiv \exists y(xRy \wedge ST_y(\varphi)),
\end{aligned}$$

where $ST_y(\theta)$ is obtained by exchanging all free occurrences of x with y (a new fresh variable) in θ. Examples are:

$$\begin{aligned}
ST_x(\Box p_1 \to p_2) &\equiv \forall y(xRy \to P_1(y)) \to P_2(x),\\
ST_x(\Box\Diamond p_3) &\equiv \forall y(xRy \to \exists z(yRz \wedge P_3(z))),
\end{aligned}$$

As probably noted, each quantifier (from FOL) is managed in a different way from the other; more specifically, the formulæ are said to be *guarded.* A formula is said to be guarded:

- if it is bounded by $\forall x$ then its translation is $\forall x(xRy \to \varphi(x))$, or
- if it is bounded by $\exists x$ then its translation is $\exists x(xRy \wedge \varphi(x))$,

where $\varphi(x)$ stands for an arbitrary well formed formula.

Summarizing, the differences (thus, the limitations) between ML and FOL are:

1. the domain is any directed graph,
2. there is only one free variable,
3. there are only 1-arity predicates, and
4. the quantifiers are limited as guarded.

We mentioned that we are interested only on logic K of all Kripke models, although other modal logics (e.g. T, D, B, K4, S4, S5) can be defined semantically by restricting the class of Kripke frames in which modal formulæ are interpreted.

2.6 Temporal Logic

The typical approach of classical logic adopts a static fashion reasoning within a single fixed world. For example, a proposition such that "it is Sunday" must be either true or false. *Temporal Logic* is used as formalism to define the semantics of temporal expressions in natural language, as a language for encoding temporal knowledge in Artificial Intelligence [GG15]. Most systems are dynamic in their domain, therefore we need the granular properties of time.

Time can be thought as a set of paths (a sequence of time instances) or as a (rooted) tree. An emblematic example of temporal logic based on the former interpretation is *Linear Temporal Logic* (LTL). Similarly the most prominent example of temporal logic following the latter structure is *Computation Tree Logic* (CTL). In this section, we do not present CTL because is not interesting for the purpose of this thesis, but we do present a particular type of LTL, namely $\mathsf{LTL}_{F,P}$ with *future* and *past* whose SAT problem is NP-COMPLETE [Eme90]. For simplicity, in the entire section we denote it with LTL.

LTL is characterized by featuring two modalities that can be seen as two specific interpretations of the generic modalities $\Diamond$, $\blacklozenge$ and are usually symbolized with the symbols F and P representing the concepts of future and past, respectively.

The basic propositional LTL syntax is based on:

- logical connectives: $\neg, \vee$,
- unary modal operators: F and P,
- a set $\mathcal{AP}$ of atomic propositions: $p, q, \ldots$.

Well formed formulæ are generated by the following grammar:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi_1 \vee \varphi_2) \mid (F\varphi) \mid (P\varphi).$$

The remaining connectives are derived from the others as:

- $\bot \equiv \neg(\neg\varphi \vee \varphi)$,
- $\top \equiv \neg\bot$,
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$,
- $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$,
- $G\varphi \equiv \neg F \neg\varphi$,
- $H\varphi \equiv \neg P \neg\varphi$,

where $G\varphi$ stands for "*it will always (in the future) be the case that* φ" and $H\varphi$ stands for "*it has always (in the past) been the case that* φ".

With this new syntax we can model furthermore our "bare boned" example from Section 2.4. For instance, take "$p \to Fq$" that could be interpreted as "*if it rains, then in some future instant I will get the umbrella*"; therefore, in spite of defining the expressive power of LTL, we can say that PL $\prec$ ML $\prec$ LTL $\prec$ FOL.

In order to define semantics, we consider the same notation as for ML of Kripke frames and Kripke models (see Section 2.5). The primitive temporal entities are time instants, therefore the basic relationships between them are equality and successor, denoted with the function $succ()$; the operators F and P are based on the transitive closure of $succ()$, normally denoted with $<$. Meanwhile, the domain W is isomorphic to the set of natural numbers $\mathbb{N}$. Thus, the temporal frame, expressed in terms of Kripke frames, for LTL is the pair $\langle \mathbb{N}, < \rangle$ and the temporal model, expressed in terms of Kripke models, is the triple $\langle \mathbb{N}, <, V \rangle$ where the valuation V assigns to every $p \in \mathcal{AP}$ a set of time instants $V(p) \subseteq \mathbb{N}$ at which p is declared true. The truth of a formula at an instant $t \in \mathbb{N}$ in a model $\mathcal{M} = \langle \mathbb{N}, <, V \rangle$, as in ML, is defined as:

- $\mathcal{M}, t \models p$ if and only if $t \in V(p) \quad \forall p \in \mathcal{AP}$,
- $\mathcal{M}, t \models \neg\varphi$ if and only if $\mathcal{M}, t \not\models \varphi$,
- $\mathcal{M}, t \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{M}, t \models \varphi_1$ or $\mathcal{M}, t \models \varphi_2$,
- $\mathcal{M}, t \models F\varphi$ if and only if $\mathcal{M}, s \models \varphi$ for some $s \in \mathbb{N}$ such that $t < s$,

- $\mathcal{M}, t \models P\varphi$ if and only if $\mathcal{M}, s \models \varphi$ for some $s \in \mathbb{N}$ such that $s < t$.

Other satisfying interpretations are defined as in ML (see Section 2.5).

The main differences between LTL and FOL can be summarized as follows:

1. the domain is limited (*discrete*, isomorphic to the set of natural numbers $\mathbb{N}$ and totally ordered),
2. there is only one free variable,
3. there are only 1-arity predicates, and
4. the quantifiers are limited as guarded.

Note that $2-4$ are the same as for ML (see Section 2.5). Formulæ from LTL to FOL can be translated according to the following standard translation (ST) [GG15]:

$$\begin{aligned}
ST_x(p_i) &\equiv P_i(x) \quad \forall p_i \in \mathcal{AP},\\
ST_x(\bot) &\equiv \bot,\\
ST_x(\neg\varphi) &\equiv \neg ST_x(\varphi),\\
ST_x(\varphi_1 \vee \varphi_2) &\equiv ST_x(\varphi_1) \vee ST_x(\varphi_2),\\
ST_x(F\varphi) &\equiv \exists y(xRy \wedge ST_y(\varphi)),\\
ST_x(P\varphi) &\equiv \exists y(yRx \wedge ST_y(\varphi)),
\end{aligned}$$

where $ST_y(\theta)$ is obtained by exchanging all free occurrences of x with y (a new fresh variable) in θ. An example is:

$$ST(p_1 \rightarrow GPp_1) \equiv P_1(x) \rightarrow \forall y(xRy \rightarrow \exists z(zRy \wedge P_1(z)))$$

that could be interpreted as "*if it is certain that I have some knowledge then it will always have been the case that I had some knowledge*".

2.7 Interval Temporal Logic

In the previous section we discussed about temporal reasoning based on time instants, namely point-based temporal logic. However, time points cannot properly reason about events that have an intrinsic duration. Thus, in this section we present a different approach concerning the logical communities.

Interval Temporal Logic (ITL) takes as the primitive ontological entities *time intervals (periods)*, rather than points. The context of interval based logics is still "young", meaning that is still a very current topic of research. For this reason, we do not have a proper representative of such class of logics, as we have for other logics (previously presented in this chapter). However, Halpern and Shoham in [HS91] presented the Halpern-Shoham interval modal logic, hereafter HS for short, among with its undecidable result over many classes of linearly ordered sets. The distinguishing elegance and expressive power of HS have attracted the attention of modal and temporal logic communities, and, therefore, we take HS as representative element of such class of logics.

Given a strict (i.e., irreflexive) linearly ordered set $\mathbb{D} = \langle D, < \rangle$, an interval over $\mathbb{D}$ is an ordered pair $[x, y]$ such that $x, y \in D$ and $x \leq y$. If $x = y$ then the interval is called *point interval*, otherwise is called a *proper (strict) interval*. In the recent literature, the *strict semantics*, where only strict intervals are considered, is usually adopted [BMM+14]. An *interval structure* is a pair $\langle \mathbb{D}, \mathbb{I}(\mathbb{D}) \rangle$, where $\mathbb{I}(\mathbb{D})$ is the set of all strict intervals.

HS has its roots in Allen's Interval Algebra (IA) [All83]: modal operators in HS can be mapped one-by-one over Allen's Interval Relations, often called Allen's Relations. Figure 2.4 (middle column) represents a sketch of such binary relations over strict semantics: R_L (*later than*), R_A (*meets/after than*), R_O (*overlaps*), R_E (*ends/finishes*), R_D (*during*), R_B (*begins/starts*) and their inverses, that are, $R_{\overline{L}}$ (*after*), $R_{\overline{A}}$ (*met by*), $R_{\overline{O}}$ (*overlapped by*), $R_{\overline{E}}$ (*ended by*), $R_{\overline{D}}$ (*contains*), $R_{\overline{B}}$ (*begun by*). These 13 relations (including the *equality*) are mutually exclusive and jointly exhaustive, meaning that exactly one Allen's relation holds between any given pair of strict intervals [MGMS11].

The basic propositional HS syntax is based on the language:

- logical connectives: $\neg, \vee$,
- unary modal operators: $\langle X \rangle$, $\langle \overline{X} \rangle$,
- a set $\mathcal{AP}$ of atomic propositions: $p, q, \ldots$,

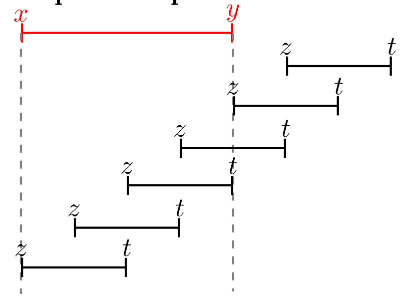
HS	Allen's Relations	Graphical representation
$\langle L\rangle$	$[x,y]R_L[z,t] \Leftrightarrow y < z$	
$\langle A\rangle$	$[x,y]R_A[z,t] \Leftrightarrow y = z$	
$\langle O\rangle$	$[x,y]R_O[z,t] \Leftrightarrow x < z < y < t$	
$\langle E\rangle$	$[x,y]R_E[z,t] \Leftrightarrow y = t, x < z$	
$\langle D\rangle$	$[x,y]R_D[z,t] \Leftrightarrow x < z, t < y$	
$\langle B\rangle$	$[x,y]R_B[z,t] \Leftrightarrow x = z, t < y$	

Figure 2.4: HS modalities and Allen's Relations.

where, for each $X \in \{L, A, O, E, D, B\}$, $\langle X\rangle$ is an (existential) interval modality according to those in Figure 2.4 and its dual is defined as $[X] \equiv \neg\langle X\rangle\neg$. Formulæ are defined by the following grammar:

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi_1 \vee \varphi_2) \mid \langle X\rangle\varphi \mid \langle\overline{X}\rangle\varphi.$$

The remaining connectives are derived from the others as:

- $\bot \equiv \neg(\neg\varphi \vee \varphi)$,
- $\top \equiv \neg\bot$,
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$,
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$,
- $[X]\varphi \equiv \neg\langle X\rangle\neg\varphi$,
- $[\overline{X}]\varphi \equiv \neg\langle\overline{X}\rangle\neg\varphi$,

Formulæ are interpreted on *interval models* $\mathcal{M} = \langle \mathbb{I}(\mathbb{D}), V\rangle$, where $V : \mathcal{AP} \rightarrow 2^{\mathbb{I}(\mathbb{D})}$ is a valuation function that associates every proposition letter $p \in \mathcal{AP}$ the set of intervals $V(p)$ on which p holds. Moreover, to be more accurate, following the same notation from ML (see Section 2.5), we should define models as $\mathcal{M} = \langle \mathbb{I}(\mathbb{D}), R_L, R_{\overline{L}}, R_A, R_{\overline{A}}, \ldots, V\rangle$, but the relations R_X and $R_{\overline{X}}$, where $X \in \{L, A, O, E, D, B\}$, of HS are implied by the relation $<$. The satisfiability relation is defined as:

- $\mathcal{M}, [x,y] \models p$ if and only if $[x,y] \in V(p) \quad \forall p \in \mathcal{AP}$,

- $\mathcal{M}, [x,y] \models \neg\varphi$ if and only if $\mathcal{M}, [x,y] \not\models \varphi$,
- $\mathcal{M}, [x,y] \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{M}, [x,y] \models \varphi_1$ or $\mathcal{M}, [x,y] \models \varphi_2$,
- $\mathcal{M}, [x,y] \models \langle X \rangle \varphi$ if and only if exists an interval $[z,t]$ such that $[x,y] R_X [z,t]$ and $\mathcal{M}, [z,t] \models \varphi$,
- $\mathcal{M}, [x,y] \models \langle \overline{X} \rangle \varphi$ if and only if exists an interval $[z,t]$ such that $[x,y] R_{\overline{X}} [z,t]$ and $\mathcal{M}, [z,t] \models \varphi$,

where $X \in \{L, A, O, E, D, B\}$ according to those in Figure 2.4. To give an example, we can define the semantics of the relation during as: $\mathcal{M}, [x,y] \models \langle D \rangle \varphi$ if and only if $\mathcal{M}, [z,t] \models \varphi$ for some $[z,t]$ such that $x < z < t < y$.

Like FOL, the SAT problem for HS is undecidable. A difference between FOL and HS is that it is natural to investigate on reasonable auto-contained fragments with better computational features from the practical point of view. It turns out that HS is more expressive than LTL, therefore PL $\prec$ ML $\prec$ LTL $\prec$ HS $\prec$ FOL. In order to obtain different fragments of HS constraints have been added. These constraints can be summarized as follows:

- limiting the set of modalities that are included in the language [BMM+14],
- interpreting the language over semantically incomplete linear structures [MSV02],
- limiting the nesting of temporal modalities [BDMS14],
- restricting the applicability of boolean operators and/or relaxing the semantics of the modal operators [BMS16].

In [GS93] Golumbic and Shamir proposed to describe a less precise topological relationship between intervals. Following their idea, two new coarser algebras emerge, namely IA_3 and IA_7 . They reduce the set of binary relations over IA by defining coarser relations. IA_7 encompasses seven relations preserving the original relations (i) before, (ii) after, (iii) equal to, (iv) by joining meets and overlaps into a single relation (and similarly for (v) their inverse ones), and (vi) by joining during, starts and finishes into a single relations (and similarly for (vii) their inverse ones). IA_3 involves only three relations conserving the original relations (i) before,

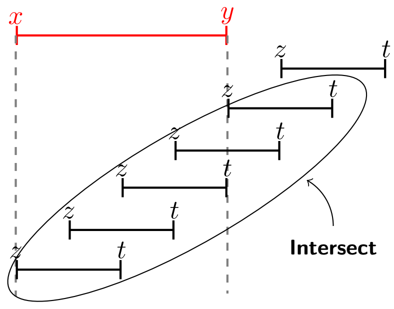


Figure 2.5: Graphical representation of HS_3's relations.

(*ii*) after and (*iii*) a new one (intersect) that can be viewed as the disjunction of all the remaining ones (and therefore is the inverse of itself and includes equality). These algebras inspire the logics HS_3 and HS_7. Hereafter we concern our attention on the former (see [MM14, MPSS15]).

Following the same terminology from HS, in order to define the semantics of HS_3 we need to introduce a new relation, namely intersect. In general, given any subset $S \subseteq \{X, \overline{X} \mid X \in \{L, A, O, E, D, B\}\}$, one can define the relation:

$$R_S = \bigvee_{X \in S} R_X \vee \bigvee_{\overline{X} \in S} R_{\overline{X}}.$$

Therefore, the relation intersect is defined as $I = A \vee \overline{A} \vee O \vee \overline{O} \vee E \vee \overline{E} \vee D \vee \overline{D} \vee B \vee \overline{B}$, or by just simply juxtaposing the original symbols to obtain a string so that the new relation can be introduced as shorthand $I = A\overline{A}O\overline{O}E\overline{E}D\overline{D}B\overline{B}$. In Figure 2.5 we give the graphical representation of this new operator. Well formed HS_3 formulæ can be recursively derived from the same HS grammar when $X \in \{L, I\}$. The semantics for HS_3 are defined similarly to those of HS with respect to the relation intersect.

It is worth to observe that HS_3 is expressive enough to simulate the universal operator, and this depends essentially from its modalities being jointly exhaustive:

$$[G]\varphi = \varphi \wedge \bigwedge_{X \in \{L, I\}} ([X]\varphi \wedge [\overline{X}]\varphi).$$

This has a very specific purpose: not all modal logics (e.g. K from Section 2.5) can simulate this kind of operator. Now, with this new grammar and universal

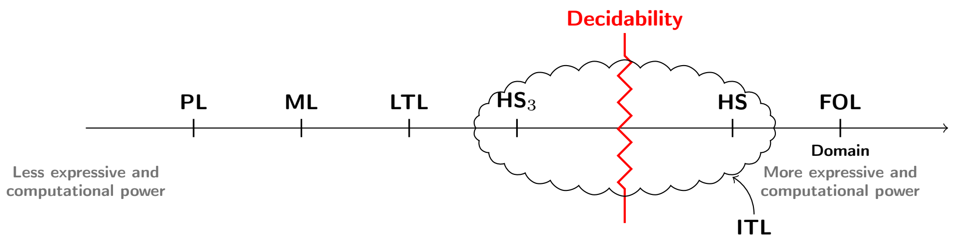

Decidability
PL
ML
LTL
HS$_3$
HS
FOL
Domain
Less expressive and computational power
More expressive and computational power
ITL

Figure 2.6: Representation of the studied logics.

operator, the formula $[G](p \rightarrow (\langle \overline{L} \rangle q \wedge [D]r))$ could be interpreted as "*it is always true that if it rains then in some moment before the rain I took the umbrella from my home and during the rain I use it*".

Two important results have been proven in [MPSS15]. The first one is: given a finitely satisfiable formula φ in HS$_3$ then it is satisfied on a model $\mathcal{M} = \langle \mathbb{I}(\mathbb{D}), V \rangle$ whose domain $\mathbb{D}$ has cardinality less or equal to:

$$|\varphi| \cdot 2^{|\varphi| \cdot (\log(4 \cdot |\varphi| + 1) + \log(|\varphi|) + |\varphi|)}.$$

Second, the SAT problem for HS$_3$ in the finite case, in the case of natural numbers, in the case of the integers, and in the case of discrete linear orders is PSPACE-COMPLETE. These results gives us the foundations for the implementation of the satisfiability checker of Chapter 4. Before leaving this chapter, we add the last piece to the "puzzle": PL $\prec$ ML $\prec$ LTL $\prec$ HS$_3$ $\prec$ HS $\prec$ FOL. In Figure 2.6 we give the complete picture of all logics studied here.

Chapter 3

Coarser Interval Relations at Work

In this chapter we give an example of an Intelligent System that it is naturally expressed in HS_3 pointing out its expressiveness from the practical point of view. FOL is not a suitable language for Artificial Intelligence because it is too expressive, therefore we prefer HS_3, choosing its domain's characteristics, whose computational behavior is better than FOL's.

3.1 Bike sharing

This example is inspired by Divvy, the City of Chicago's bike sharing system. The city is the nest of multiple docking stations. The magnitude of stations form a widespread network throughout the city. A fleet of bikes, locked into the latter, are simultaneously managed by the system. Chicagoans, provided with a member card or a 24-hour pass user, can rent bikes. Bikes can be rented from and returned to any station in the city, creating a network of trips with many possible combinations of starting (indicating the stations *from*) and ending (indicating the stations *to*) points. Anonymous trip data are stored in a *temporal database* (a database

Trips					
trip_id	starttime	stoptime	bikeid	from_st_id	to_st_id
4118	2013-06-27 12:11	2013-06-27 12:16	316	85	28
4275	2013-06-27 14:44	2013-06-27 14:45	64	32	32
4291	2013-06-27 14:58	2013-06-27 15:05	433	32	19
4316	2013-06-27 15:06	2013-06-27 15:09	123	19	19
4342	2013-06-27 15:13	2013-06-27 15:27	852	19	55
4480	2013-06-27 19:40	2013-06-27 22:28	27	340	46

Table 3.1: Trips table of bike sharing system database.

Maintenance					
repair_id	starttime	stoptime	bikeid	from_st_id	to_st_id
5327	2013-06-28 09:05	2013-06-28 10:15	594	27	1
5335	2013-06-28 09:14	2013-06-28 10:41	227	26	1
5346	2013-06-28 09:26	2013-06-28 14:25	118	74	1
5353	2013-06-28 09:35	2013-06-28 09:50	226	24	1

Table 3.2: Maintenance table of bike sharing system database.

containing temporal knowledge, that is information about time) and openly available through the Divvy Data Challenge program. The database consists of two tables:

- *Trips*, that stores the data of the user rentals
- *Maintenance*, that stores the data on the repairs and other maintenance activities of the bikes

In Tables 3.1 and 3.2 we give an example of such tables.

3.2 Description of the Problem

We assume that repairs and other types of maintenance always take place at a special station with identifier 1. Since the inherent temporal nature of the context, tables are equipped, along other attributes, with temporal attributes. A *starttime* and an *endtime* is stored for each *tuple* (roughly said, a row of the table); these two values indicate the validity of the rest of the values in the tuple.

Interval temporal logics can be used to reason on the database in at least two different ways: as *query language*, to extract information from an actual instance of the data querying the database, and as *specification language*, to express functional dependencies, integrity constraints and other requirements that every instance must respect; in this example, we follow the latter approach and use HS_3 as specification language. Once the requirements are defined and formalized by the domain experts, one of the first problems that one must solve is the *consistency problem*. An instance is inconsistent with another if a contradiction may occur between both; the problem of checking whether they can be implemented by an actual instance of the database or not.

3.3 Formalization

Formalization is the governance of a system by rules and procedures; each system should have its own formalization in order to be less error-prone. Therefore, we proceed with the formal description of the problem (specification language). Formulæ expressing the requirements are built from the following propositional letters:

- $\mathit{trip}(i, j, k)$, to represent that the bike i performed a trip from station j to station k, and
- $\mathit{repair}(i, j, k)$, to represent that the bike i was collected at station j and brought to station k to be repaired.

HS_3 is not nearly expressive as full HS, although, some requirements can be written in this language. Focusing our attention on Table 3.1, one can easily deduce that one bike can belong to at most one trip the start and the end point. This fact can be formalized, for every bike i, no pair of intervals satisfying $\mathit{trip}(i, j, k)$ and $\mathit{trip}(i, l, m)$ can share any point (in particular, a trip cannot start at the same time when the previous trip ended). We denote, whenever necessary, $\mathit{trip}\,(i)$ as short-hand for $\bigvee_{j,k} \mathit{trip}\,(i, j, k)$, and $\mathit{repair}\,(i)$ for $\bigvee_{j,k} \mathit{repair}\,(i, j, k)$.

$$\bigwedge_{i,j,k} [G](\mathit{trip}\,(i, j, k) \rightarrow \bigwedge_{(j,k)\neq(l,m)} \neg \mathit{trip}\,(i, l, m)),$$

$$\bigwedge_{i}[G](\mathit{trip}\,(i) \rightarrow \neg\langle I\rangle\mathit{trip}\,(i)).$$

Similarly, we can ensure that two repairs for the same bike do not overlap:

$$\bigwedge_{i,j,k}[G](\mathit{repair}\,(i,j,k) \rightarrow \bigwedge_{(j,k)\neq(l,m)} \neg\mathit{repair}\,(i,l,m)),$$

$$\bigwedge_{i}[G](\mathit{repair}\,(i) \rightarrow (\neg\langle I\rangle\mathit{repair}\,(i)))$$

During maintenance a bike cannot do any trip (and vice-versa):

$$\bigwedge_{i}[G](\mathit{repair}\,(i) \rightarrow (\neg\mathit{trip}\,(i) \wedge \neg\langle I\rangle\mathit{trip}\,(i))),$$

$$\bigwedge_{i}[G](\mathit{trip}\,(i) \rightarrow (\neg\mathit{repair}\,(i) \wedge \neg\langle I\rangle\mathit{repair}\,(i))).$$

Then, we guarantee that a repair finishes in station 1 and starts from a station different from 1:

$$\bigwedge_{i,j}[G](\neg\mathit{repair}\,(i,1,j) \wedge \bigwedge_{k\neq 1}\neg\mathit{repair}\,(i,j,k)).$$

Finally, we guarantee that maintenance is performed regularly:

$$\bigwedge_{i}[G]\langle L\rangle\mathit{repair}\,(i).$$

Chapter 4

Description of the Tableau

Based on the results in [MPSS15], in this chapter we present a relatively efficient satisfiability checker for HS_3 in the finite/discrete case. We describe the implementation strategies and difficulties that we have encountered in building such a decider [Sis12].

We decided to implement the checker in C++ (i.e. [Str13]). The standard library provides very powerful data structures and optimized implementations of the most known algorithms (e.g. sorting, listing, hashing). Our implementation does not need external libraries.

The chapter is organized as follows. First, we describe the flow chart of the tableau pointing out the satisfiability checking. Second, we focus our attention to analyzing the data structures motivating their use. Then we describe the most interesting auxiliary procedures used in our implementation. Finally, we illustrate how we structured the input formulæ before checking its satisfiability.

4.1 Flow Diagram

Tableau-based satisfiability checkers are not very common for interval temporal logics. Among the few exceptions, the tableau system for Right Propositional

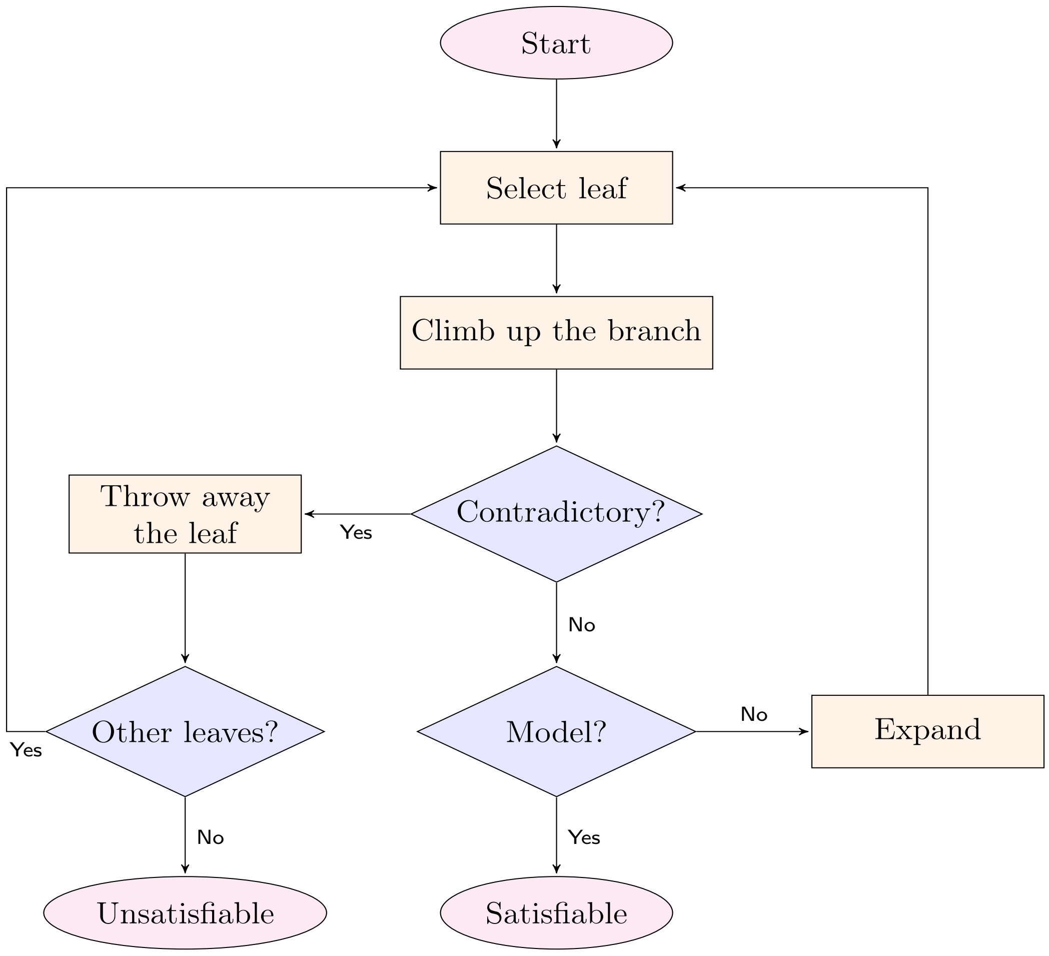


Figure 4.1: Flow diagram of the tableau.

Neighborhood Logic (RPNL) over finite linear orders, presented in [BDMMS13], deals with Allen's relation *meets* only (the so-called fragment A). Therefore, it is natural to expect that a checker for the much more expressive HS_3 is more complex to implement. The entire reasoning process is depicted in Figure 4.1.

The process is very straightforward. Once a leaf has been selected, the reasoner climbs up the branch (with a bottom-up style) until the root has been encountered and, then, stops. At the end of the process a data structure containing the branch information is built. The next step consists of deciding if the branch is contradictory. If the answer is positive, the system simply throws away the leaf without giving it any other future chance to be selected. At this level, if there are other

struct FormulaNode
$$\begin{cases} OpCode \\ LeftChild* \\ RightChild* \end{cases}$$

Figure 4.2: Formula node's decoration.

leaves to be processed, then one of them is picked, otherwise, if no other leaves are still waiting, the formula is declared unsatisfiable/contradictory. Meanwhile, if the branch is not contradictory, the next phase is checking if that particular (not contradictory) branch is a model for the given formula. If it is then the formula is satisfiable.

4.2 Data Structures

From a mathematical point of view, both the formula and the tableau are represented as *rooted decorated trees* [GMS03]. The decoration for such trees can be seen as a function that assigns to each node the information carried by that node. When we represent formulæ, the decoration is a propositional letter or an operator. On the other hand, when we represent a semantic tableau, the decoration is the collection of all the information needed to expand or to close the tableau.

Formulæ are represented as *binary* rooted decorated trees. Each node of the tree contains an unique code that describes the operator (boolean or modal, distinguishing between existential and universal), of which a propositional letter is a special case seen as no-operator. As being a binary tree, every node has at most two successors (left and right). It might be the case that the right link is undefined (with the left one defined) because it is intended that nodes whose decoration is an unary (modal) operator have only the left child (as being the argument of the operator, for instance φ in $\langle L\rangle\varphi$). It is obvious that every leaf has both links undefined (because they have no successor). We give the representation of such data structure concerning each node of the formula in Figure 4.2.

A tableau is represented as a k-ary rooted decorated tree in form of Left-Child-Right-Sibling (LCRS) [CLRS09]. Formulæ like $\langle X\rangle\varphi$, where X is a HS_3 relation, means that φ is possibly true in some world from the set of all (branching) worlds,

struct TableauNode

$$\left\{\begin{array}{l} FormulaNode* \\ LeftChild* \\ RightSibling* \\ Parent* \\ Interval \\ Active \\ Domain \\ Leaf \\ PolicyData \end{array}\right.$$

Figure 4.3: Tableau node's decoration.

that is we do not know a priori in which of these; therefore, given its nice properties to manage this unbounded branching factor, it is natural to use such data structure. Each tableau node contains a pointer to the formula tree that represents the sub-formula under analysis, the interval over which it holds, an active/inactive flag if it is or not active and pointers to the left child, right sibling and parent. The complete decoration of each node of the tableau includes the domain. Domains are represented as a totally ordered sets of floating points, so that an interval is a pair of points (of the domain). The choice of using floating points has a very specific purpose: whenever a new point must be added in between a pair of already existing ones, it can be created by simply computing their average; nevertheless, a model is always finite by construction. Every possible model is represented by a branch of the tableau which is identified by a (temporary) leaf. Leaves also store domain information. Moreover, each leaf carries additional side information (e.g. domain cardinality) used for selecting the next leaf/branch to be processed. In Figure 4.3 we give the schematic of such decoration.

In order to build the reasoner, we need to define a data structure by which we can manipulate the (temporary) leaves. The leaves are stored into a linked list; we call such data structure $\mathcal{L}$. The leaves are not "physically" stored into such data structure, but only a link from each element of this structure to one of the real ones. The choice of this data structure is convenient because the (front/back) insertions and deletions take constant time. Moreover, linking two heterogeneous list between them (one at the end of another) also takes constant time. A non-

```
proc FINDLEAVES (current)
  let $S$ be a stack
  let $\mathcal{A}$ be a hash unordered set
  while $(done == 0)$
    if $(current\,! = Nil)$
      then
        $S.push(current)$
        $current = current.left$
      else
        if $(S.empty == 1)$
          then $done = 1$
          else
            $current = S.pop$
            if $(current.left == Nil)$
              then $\mathcal{A}.insert(current)$
            if $(current.rightsibling\,! = Nil)$
              then $current = current.rightsibling$
  return $\mathcal{A}$
```

Figure 4.4: Non-recursive algorithm for finding the leaves on a LCRS tree.

trivial adaptation of a generic tree visit algorithm has been implemented in order to correctly identify, given a tableau node, the set $\mathcal{A}$ of all and only leaves that belong to the sub-tree rooted at it; in Figure 4.4 we give the pseudocode of such algorithm. The returned data structure is an (hash) unordered set [Str13] with an efficient (constant time) lookup method.

In Figure 4.5 we give the complete picture of all the concepts described until this point. The figure represents a tableau (on the left hand side, lhs for short), among with the list of leaves representing the (temporary) leaves that are waiting to be selected, that it is abstractly communicating with a formula tree (on the right hand side, rhs for short) representing $\varphi \equiv p \wedge \neg p \vee \langle I \rangle (q \vee \neg q)$.

The algorithm from Figure 4.6 fully represents, from the highest level of implementation, the tableau-based satisfiability checking for HS_3.

The initial tableau, for a formula φ whose initial satisfiability must be checked, is a tableau tree composed by a single tableau node with the following decoration: its formula node pointer points to the root of the formula tree that represents φ, its interval is $[0, 1]$, its active and leaf flag are both true, its domain is $\{0 < 1\}$ and

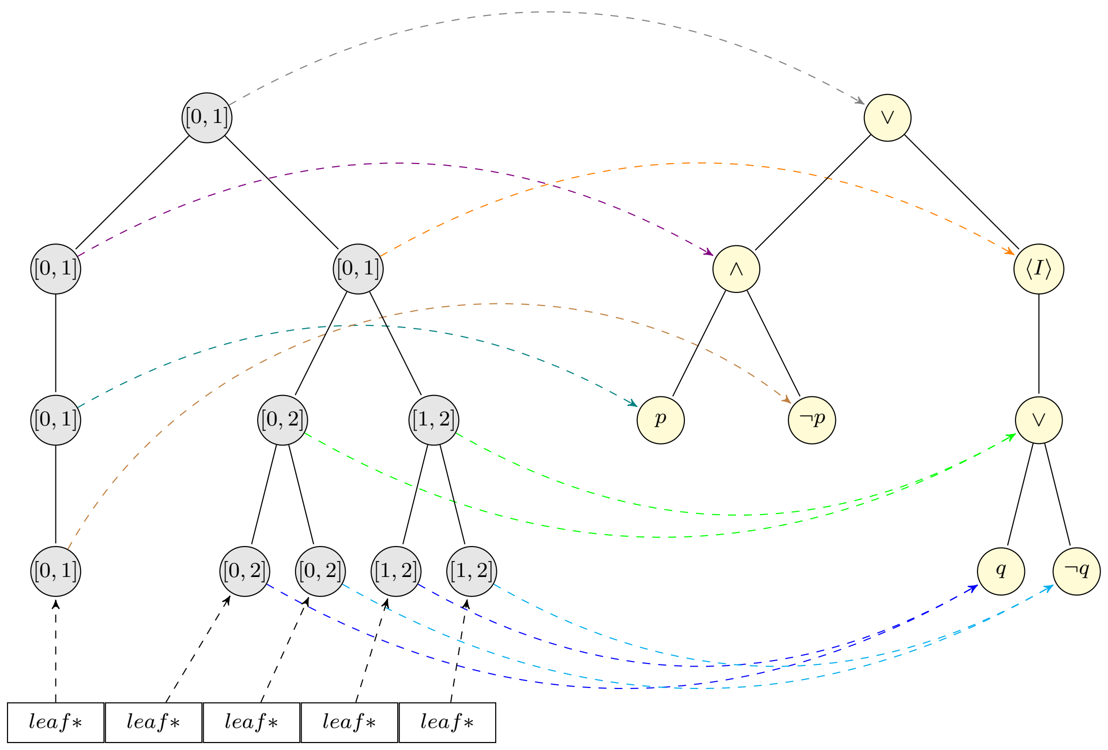


Figure 4.5: Interaction between the tableau (lhs) and the tree formula (rhs).

all other pointers are null.

4.3 Auxiliary Procedures

The most important operation realized by the checker is the expansion of a node; the rules are described in Table 4.1. Once a leaf l is selected from the list of leaves $\mathcal{L}$, a step-by-step bottom-up walk on the branch $\mathcal{B}$ of the tableau must be performed in order to store all that information needed for model checking and for closing checking; namely the knowledge $\mathcal{K}$ of the branch $\mathcal{B}$. Such knowledge has two hash unordered sets, that we respectively call $\mathcal{H}$ and $\mathcal{S}$. At each step of the computation, $\mathcal{H}$ is filled with the interval on which that particular tableau node holds on $\mathcal{B}$ among with its unique identification code (by simply returning its tableau node's address) in the case of operators or with its propositional letter in the case of atomic propositions. Meanwhile, $\mathcal{S}$ is build as support for the universal

```
proc $\mathbf{HS_3}$-SAT ($\mathcal{L}$)
let $l$ be a tableau node representing a concrete leaf
let $\mathcal{K}$ be the knowledge of the branch
let $\mathcal{M}$ be a model
while ($\mathcal{M} != SAT$ or $\mathcal{M} != UNSAT$)
    $l = GetNextLeaf(\mathcal{L})$
    $\mathcal{K} = GetBranchInformation(l)$
    if ($\mathcal{K}.contradictory == 1$)
      then
        $\mathcal{L}.remove(l)$
        if ($\mathcal{L}.empty == 1$)
          then $\mathcal{M} = UNSAT$
      else
        if ($IsModel(\mathcal{K}) == 1$)
          then $\mathcal{M} = SAT$
          else $Expand(\mathcal{K}, \mathcal{L})$
return $\mathcal{M}$
```

Figure 4.6: Tableau-based satisfiability checking for HS_3.

relations $[X]\varphi$ encountered on the path. Given the current inspected tableau node, in order to give its contribute to the tableau while its active flag it is 1, every time the procedure encounters an active universal operator it generates the sequence of all intervals on which φ should hold, based on the semantics of the operators itself, and updates $\mathcal{S}$ by inserting the entire sequence. As being a mono-valued unordered set [Str13], the insertions do not generate duplicates. If the current tableau node is a propositional letter of $\mathcal{AP}$ then we check if this particular node generates a contradiction, with the already partially seen part of the branch $\mathcal{B}$ on the way up, thanks to the set $\mathcal{H}$. The expansion algorithm, described in Figure 4.7, takes in input the knowledge $\mathcal{K}$ of the branch and the list of leaves $\mathcal{L}$. Once determined the next to-be-expanded node n we apply the algorithm from Figure 4.4 to find the set $\mathcal{A}$ of leaves that are rooted in n. For each leaf l that are both in $\mathcal{A}$ and $\mathcal{L}$ we then create new branches of the tableau based on the expansion rules. Boolean rules are standard, while the rules for modal operators are defined as follows. Let $\mathfrak{D}$ be the set of all finite domains, and let $\mathfrak{I}$ be the set of all intervals

$$\frac{\varphi_1 \vee \varphi_2, [x,y], \mathbb{D}}{\varphi_1, [x,y], \mathbb{D} \mid \varphi_2, [x,y], \mathbb{D}} \text{(OR)} \qquad \frac{\varphi_1 \wedge \varphi_2, [x,y], \mathbb{D}}{\begin{array}{c}\varphi_1, [x,y], \mathbb{D} \\ \varphi_2, [x,y], \mathbb{D}\end{array}} \text{(AND)}$$

$$\frac{\langle X \rangle \varphi, [x,y], \mathbb{D}}{\varphi, \mu^{e,X}_{i,I}([x,y],\mathbb{D}), \mu^{e,X}_{i,D}([x,y],\mathbb{D}) \mid \ldots \mid \varphi, \mu^{e,X}_{\mu^{e,X}([x,y],\mathbb{D}),I}([x,y],\mathbb{D}), \mu^{e,X}_{\mu^{e,X}([x,y],\mathbb{D}),D}([x,y],\mathbb{D})} \text{(EXIST)}$$

$$\frac{[X]\varphi, [x,y], \mathbb{D}}{\begin{array}{c}\varphi, \mu^{u,X}_1([x,y],\mathbb{D}), \mathbb{D} \\ \varphi, \mu^{u,X}_2([x,y],\mathbb{D}), \mathbb{D} \\ \ldots \\ \varphi, \mu^{u,X}_{\mu^{u,X}([x,y],\mathbb{D})}([x,y],\mathbb{D}), \mathbb{D}\end{array}} \text{(UNIV)}$$

Table 4.1: Expansion rules of the semantic tableau-based procedure.

in any domain in $\mathfrak{D}$. For a given existential operator $\langle X \rangle$, we define a function:

$$\mu^{e,X} : \mathfrak{I} \times \mathfrak{D} \to \mathbb{N}$$

that, for a given pair $([x,y],\mathbb{D})$, returns the number of different intervals in the relation R_X with $[x,y]$ plus the number of new intervals that should be created in the relation R_X in order to explore all qualitatively distinct possibilities. For example, $\mu^{e,L}([0,1],\{0<1<2\})$ is 5: indeed, if $\langle L \rangle \varphi$ holds on $[0,1]$ and the current domain is $\{0<1<2\}$, then φ may hold on some interval $[1.25,1.75]$, $[1.5,2]$, $[2,2.5]$, $[3,4]$ or $[1.5,2.5]$; notice that, for example, $[1.25,1.75]$ is necessary possible as it represents a new interval completely between existing points; the same holds for $[3,4]$. In this way, given $\langle X \rangle \varphi$ holding on $[x,y]$ in the finite domain $\mathbb{D}$, the parametric function(s):

$$\begin{aligned}\mu^{e,X}_{i,I} &: \mathfrak{I} \times \mathfrak{D} \to \mathfrak{I}, \\ \mu^{e,X}_{i,D} &: \mathfrak{I} \times \mathfrak{D} \to \mathfrak{D}\end{aligned}$$

return, respectively, the i-th interval on which φ holds and the corresponding i-th domain (not necessarily different from $\mathbb{D}$) in no particular order; following up with the above example:

$$\begin{aligned}&\mu^{e,X}_{1,I}([0,1],\{0<1<2\}) = [1.25,1.75], \\ &\mu^{e,X}_{1,D}([0,1],\{0<1<2\}) = \{0<1<1.25<1.75<2\}.\end{aligned}$$

```
proc EXPANSION ($\mathcal{K}, \mathcal{L}$)
let $l'$ be a tableau node
let $\mathcal{N}$ be an array of pairs of type $([x, y], \mathbb{D})$
    $\mathcal{A} = FindLeaves(\mathcal{K}.n)$
    for all $(l \in \mathcal{L} \cap \mathcal{A})$
        if $(\mathcal{K}.n.opcode == \wedge$ or $\mathcal{K}.n.opcode == \vee)$
            then $\mathcal{N}[1] = (\mathcal{K}.n.interval, l.domain)$
            else if $(\mathcal{K}.n.opcode == \langle X \rangle_{X \in \mathsf{HS}_3})$
            then
                for $(i = 1$ to $\mu^{e,X}(\mathcal{K}.n.interval, l.domain))$
                $\mathcal{N}[i] = (\mu^{e,X}_{i,I}(\mathcal{K}.n.interval, l.domain), \mu^{e,X}_{i,D}(\mathcal{K}.n.interval, l.domain))$
            else if $(\mathcal{K}.n.opcode == [X]_{X \in \mathsf{HS}_3})$
            then
                for $(i = 1$ to $\mu^{u,X}(\mathcal{K}.n.interval, l.domain))$
                $\mathcal{N}[i] = (\mu^{u,X}_{i}(\mathcal{K}.n.interval, l.domain), l.domain)$
        $l' = CreateSubTree(\mathcal{N}, \mathcal{K})$
        $\mathcal{L}.insert(FindLeaves(l'))$
        $Substitute(l, l')$
        $\mathcal{L}.remove(l)$
```

Figure 4.7: Expansion algorithm.

Because we are restricting ourselves to initial satisfiability, these functions never return new points between 0 and 1, nor smaller than 0. In this way for each existential operator $\langle X \rangle \varphi$ we have an existential disjunctive rule that creates enough branches to search for every possible location for φ. Dually, for universal operators, we have functions $\mu^{u,X}$ and $\mu^{u,X}_i$ to return all intervals seen from $[x, y]$ via R_X and the i-th interval on which φ holds; in this case, domains do not change.

With the help of the versatile naturalness of the notation, in Figure 4.7 we present the expansion algorithm. There is still a remaining sub-routine to analyze, that is the creation of the sub-tree given the array $\mathcal{N}$ of pairs of type $([x, y], \mathbb{D})$ and the knowledge $\mathcal{K}$. The creation is made accordingly to the expansion rules, described in Table 4.1, whose root is a new fresh node l'. Before creating new tableau nodes of the sub-tree, the procedure checks if the decoration of that particular node (that is intended to be created) is already on the branch. Two particular situations may arise. The first one concerns the universal operators and the second the existential operators. If the decoration of the next to-be-expanded node n is

```
proc GetBranchInformation ($l$)
let $\mathcal{K}$ be the knowledge of the branch
  $current = l$
  $\mathcal{K}.activeOnlyUniversal = 1$
  while $(current != Nil)$
    if $(current.active == 1)$
      then
        if $(current.opcode \notin \mathcal{AP})$
          then
            if $(current.opcode == [X]_{X \in \mathsf{HS}_3})$
              then $\mathcal{K}.\mathcal{S} = UpdateSupportStructure(l, \mathcal{K})$
              else $\mathcal{K}.activeOnlyUniversal = 0$
            $\mathcal{K}.n = current$
            $\mathcal{K}.\mathcal{H}.insert(current)$
      else
        if $(current.opcode \in \mathcal{AP})$
          then
            if $(IsContradictory(current, \mathcal{K}))$
              then
                $\mathcal{K}.contradictory = 1$
                $\mathcal{K}.complement = NegateProposition(current)$
            $\mathcal{K}.\mathcal{H}.insert(current)$
    $current = current.parent$
  return $\mathcal{K}$
```

Figure 4.8: Getting branch information algorithm.

of type $[X]\varphi$ then it will create as many tableau nodes, with the decoration φ, as those that are not already present on the branch $\mathcal{B}$ satisfying the relation R_X. For instance, if the current decorations on $\mathcal{B}$ are $(\varphi, [0, 1]), (\varphi, [0, 2])$ and the relation R_X tries to generate $(\varphi, [0, 1]), (\varphi, [0, 2]), (\varphi, [1, 2])$ then it will only generate the sub-tree formed by the tableau node whose decoration is $(\varphi, [1, 2])$. On the other hand, in the case of existential operators of type $\langle X\rangle\varphi$ the procedure only checks if φ is already present on $\mathcal{B}$ without concerning about the interval. After the creation is done, the procedure returns the tableau node l' that identifies the sub-tree, whose leaves are inserted into $\mathcal{L}$, to be substituted in exchange of l; that is, previously l had its left link undefined (for being a leaf) and l becomes the root of the sub-tree, previously rooted in l', whose left link will be defined by having the left link's information of l'. Therefore, l is removed from $\mathcal{L}$ because it is no

```
proc IsModel ($\mathcal{K}$)
  if ($\mathcal{K}.activeOnlyUniversal\,!= 1$ or $\mathcal{K}.\mathcal{S} \not\subseteq \mathcal{K}.\mathcal{H}$)
    then return 0
  return 1
```

Figure 4.9: Checking if the branch is a model algorithm.

longer a leaf of the tableau.

Propositional contradictions are verified in the classic way: check if there exist two nodes on the branch $\mathcal{B}$ whose decorations are, respectively, $p, [x, y]$ and $\neg p, [x, y]$; if this is the case, then the $\mathcal{B}$ is said to be closed. A branch is also closed if a formula φ has a domain, stored in l, greater than [MPSS15]:

$$|\varphi| \cdot 2^{|\varphi| \cdot (\log(4 \cdot |\varphi| + 1) + \log(|\varphi|) + |\varphi|)}.$$

If $\mathcal{B}$ is contradictory then we store in $\mathcal{K}$ a flag that witnesses this fact and, secondly, we save the "complement" of that particular letter. For instance, if in $\mathcal{H}$ there is the information regarding a tableau node whose decoration, thanks to the formula node pointer, is a propositional letter p that hold on $[x, y]$ then its complement is $\neg p$ holding on the same interval. This has a very specific purpose: we adopted an heuristic that prunes the sub-tree that is rooted in the complement of the letter that generated the contradiction. This fact is common to all those leaves, found by the procedure in Figure 4.4, whose root is the complement. In Figure 4.8 we give the algorithm for such procedure.

The missing piece of the reasoner is described by the algorithm in Figure 4.9. The algorithm searches for a model represented by the knowledge $\mathcal{K}$ of the branch $\mathcal{B}$. The procedure is very straightforward. If on the branch $\mathcal{B}$ there are other active tableau nodes, ready to give their contribute to the tableau itself, that are not universal operators or if the structure $\mathcal{S}$ is not a sub-set of $\mathcal{H}$ then the inspected branch is not a model. The first case it is obvious that is not a model because it might be the case that, for example, an existential operator is still active on $\mathcal{B}$. The second case is non-trivial to see: if $\mathcal{S}$ is not a sub-set of $\mathcal{H}$ it means that there are pending universal operators on $\mathcal{B}$ that are not fully satisfied and, therefore, $\mathcal{B}$ it cannot be a model. In Figure 4.10 we show the tableau tree that corresponds

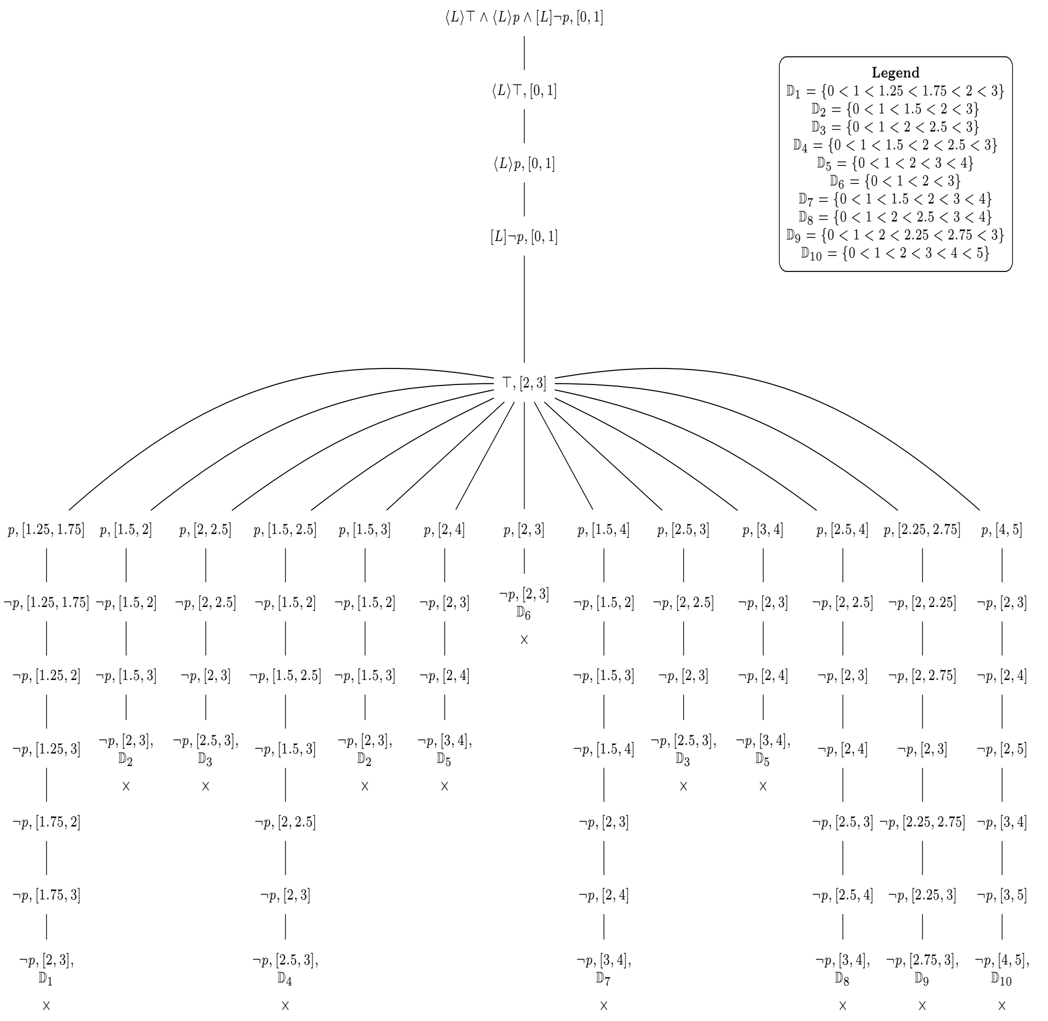

$\langle L\rangle\top \wedge \langle L\rangle p \wedge [L]\neg p, [0,1]$
$\langle L\rangle\top, [0,1]$
$\langle L\rangle p, [0,1]$
$[L]\neg p, [0,1]$
Legend
$\mathbb{D}_1 = \{0 < 1 < 1.25 < 1.75 < 2 < 3\}$
$\mathbb{D}_2 = \{0 < 1 < 1.5 < 2 < 3\}$
$\mathbb{D}_3 = \{0 < 1 < 2 < 2.5 < 3\}$
$\mathbb{D}_4 = \{0 < 1 < 1.5 < 2 < 2.5 < 3\}$
$\mathbb{D}_5 = \{0 < 1 < 2 < 3 < 4\}$
$\mathbb{D}_6 = \{0 < 1 < 2 < 3\}$
$\mathbb{D}_7 = \{0 < 1 < 1.5 < 2 < 3 < 4\}$
$\mathbb{D}_8 = \{0 < 1 < 2 < 2.5 < 3 < 4\}$
$\mathbb{D}_9 = \{0 < 1 < 2 < 2.25 < 2.75 < 3\}$
$\mathbb{D}_{10} = \{0 < 1 < 2 < 3 < 4 < 5\}$
$\top, [2,3]$
$p, [1.25, 1.75]$ $p, [1.5, 2]$ $p, [2, 2.5]$ $p, [1.5, 2.5]$ $p, [1.5, 3]$ $p, [2, 4]$ $p, [2, 3]$ $p, [1.5, 4]$ $p, [2.5, 3]$ $p, [3, 4]$ $p, [2.5, 4]$ $p, [2.25, 2.75]$ $p, [4, 5]$
$\neg p, [1.25, 1.75]$ $\neg p, [1.5, 2]$ $\neg p, [2, 2.5]$ $\neg p, [1.5, 2]$ $\neg p, [1.5, 2]$ $\neg p, [2, 3]$ $\neg p, [2, 3]$ $\mathbb{D}_6$ × $\neg p, [1.5, 2]$ $\neg p, [2, 2.5]$ $\neg p, [2, 3]$ $\neg p, [2, 2.5]$ $\neg p, [2, 2.25]$ $\neg p, [2, 3]$
$\neg p, [1.25, 2]$ $\neg p, [1.5, 3]$ $\neg p, [2, 3]$ $\neg p, [1.5, 2.5]$ $\neg p, [1.5, 3]$ $\neg p, [2, 4]$ $\neg p, [1.5, 3]$ $\neg p, [2, 3]$ $\neg p, [2, 4]$ $\neg p, [2, 3]$ $\neg p, [2, 2.75]$ $\neg p, [2, 4]$
$\neg p, [1.25, 3]$ $\neg p, [2, 3], \mathbb{D}_2$ × $\neg p, [2.5, 3], \mathbb{D}_3$ × $\neg p, [1.5, 3]$ $\neg p, [2, 3], \mathbb{D}_2$ × $\neg p, [3, 4], \mathbb{D}_5$ × $\neg p, [1.5, 4]$ $\neg p, [2.5, 3], \mathbb{D}_3$ × $\neg p, [3, 4], \mathbb{D}_5$ × $\neg p, [2, 4]$ $\neg p, [2, 3]$ $\neg p, [2, 5]$
$\neg p, [1.75, 2]$ $\neg p, [2, 2.5]$ $\neg p, [2, 3]$ $\neg p, [2.5, 3]$ $\neg p, [2.25, 2.75]$ $\neg p, [3, 4]$
$\neg p, [1.75, 3]$ $\neg p, [2, 3]$ $\neg p, [2, 4]$ $\neg p, [2.5, 4]$ $\neg p, [2.25, 3]$ $\neg p, [3, 5]$
$\neg p, [2, 3], \mathbb{D}_1$ × $\neg p, [2.5, 3], \mathbb{D}_4$ × $\neg p, [3, 4], \mathbb{D}_7$ × $\neg p, [3, 4], \mathbb{D}_8$ × $\neg p, [2.75, 3], \mathbb{D}_9$ × $\neg p, [4, 5], \mathbb{D}_{10}$ ×

Figure 4.10: Checking the satisfiability of $\varphi \equiv \langle L\rangle\top \wedge \langle L\rangle p \wedge [L]\neg p$.

to the formula $\varphi \equiv \langle L\rangle\top \wedge \langle L\rangle p \wedge [L]\neg p$.

4.4 Description of the Input

In the previous sections, we naturally assumed that formulæ are represented as binary rooted decorated trees. In this section we are going to take a closer look, analyzing the key factors, on how we managed to achieve such transformation (from formulæ to trees).

Formulæ are stored in an input file were each line represents a concrete formula φ whose satisfiability is checked via the tableau-based procedure. To avoid ambiguity, when iterating the file by reading each specific formula, we adopted to represent fully parenthesized formulæ maintaining this strategy when necessary. For instance, given the formula:

$$\varphi_{\mathsf{init}} \equiv (\neg\neg p_1 \rightarrow p_2) \wedge (\neg\langle L\rangle(p_2 \rightarrow p_3))$$

its fully parentheses version is

$$\varphi_{\mathsf{par}} \equiv ((\neg(\neg(p_1))) \rightarrow (p_2)) \wedge (\neg(\langle L\rangle((p_2) \rightarrow (p_3)))),$$

where φ_{init} is semantically equivalent to φ_{par}.

Given the fully parentheses version of the initial formula φ_{par}, we store the entire formula into a linked list F whose elements are the symbols of the formula itself; for instance, if φ_{par} then (abusing of notation) $F = \{\{(\}, \{(\}, \{\neg\}, \{(\}, \{\neg\}, \{(\}, \{p_1\}, \{)\}, \{)\}, \{)\}, \{\rightarrow\}, \{(\}, \{p_2\}, \{)\}, \{)\}, \{\wedge\}, \{(\}, \{\neg\}, \{(\}, \{\langle L\rangle\}, \{(\}, \{(\}, \{p_2\}, \{)\}, \{\rightarrow\}, \{(\}, \{p_3\}, \{)\}, \{)\}, \{)\}, \{)\}\}$. We have implemented a recursive procedure that creates the raw binary rooted decorated tree representing φ_{par}, thanks to the non-ambiguity property that the parentheses provide. Firstly, the procedure checks the head element of the list to verify if it is an unary operator, and if this is the case, it creates a formula node whose decoration is the unary operator. After the new node is created, it recalls the procedure on the argument of the operator restricting the input, that originally was the linked list F, to a smaller portion that is closed into parentheses. For example, if the original formula is $\varphi_{\mathsf{ex}} \equiv \neg(p)$ whose linked list is $F_{\mathsf{ex}} = \{\{\neg\}, \{(\}, \{p\}, \{)\}\}$ the head of the list is $\neg$ and the portion is p. This makes perfect sense because we give precedence priority to unary operators before binary. Following the example, binary operators are processed in an alternative way: using a simple counter variable that counts the occurrences of the parentheses in an intelligent way. For every opened parenthesis the counter gets $+1$ and viceversa (for every closed parenthesis) gets -1. When the counter reaches the value 0, pointing to an element of the list, then it checks the one-ahead element of the list and verifies if it is a binary operator and, if this is the case, creates a node whose decoration is the binary operator. After the creation, it recalls twice the recursive procedure

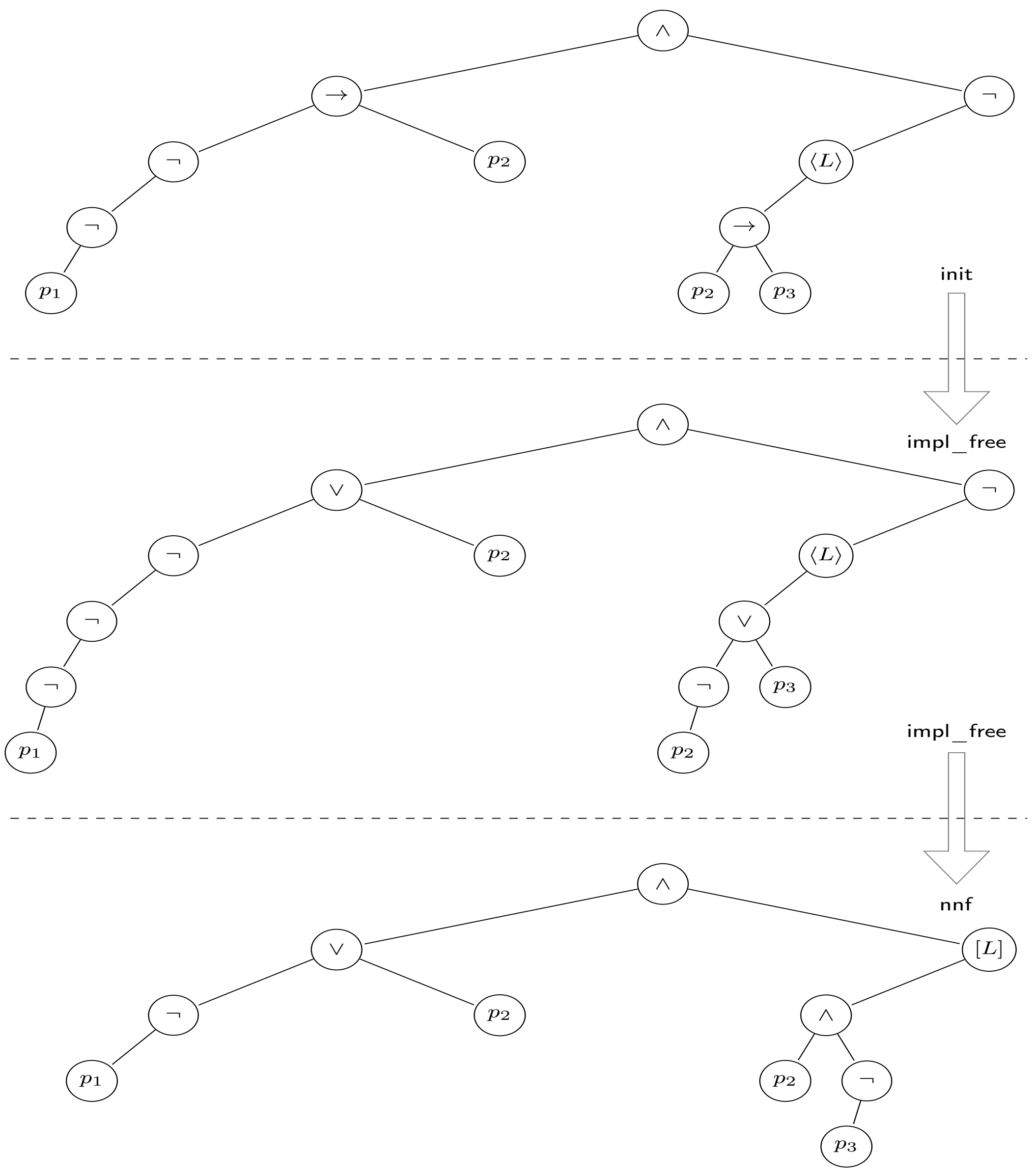


Figure 4.11: Transformation of $\varphi \equiv (\neg\neg p_1 \rightarrow p_2) \wedge (\neg\langle L\rangle(p_2 \rightarrow p_3))$ into negated normal form.

on the two operands of the binary relation, that is splitting the original input into two pieces and restricting them in a similar manner as for unary operators.

A formula with an implication, such as the example in Section 2.4, $\varphi \equiv p \rightarrow q$, possibly interpreted as "*if it rains then I get the umbrella*", is semantically equivalent to $\varphi_{\mathsf{impl\_free}} \equiv \neg p \vee q$, that could be interpreted as "*it is the case that it*

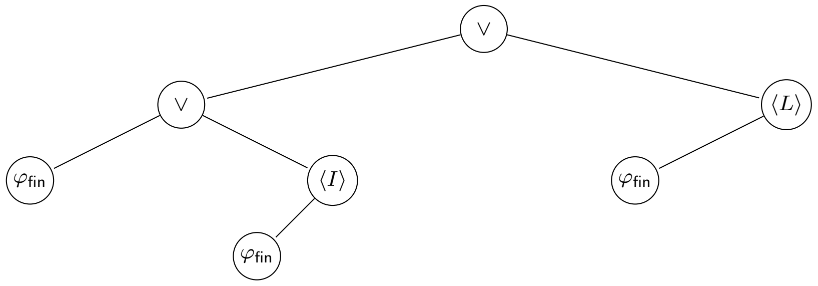


Figure 4.12: Formula tree for initial satisfiability.

does not rain or I get the umbrella"; therefore $\varphi \equiv \varphi_{\mathsf{impl\_free}}$. Following our initial example of this section:

$$\varphi_{\mathsf{par}} \equiv ((\neg(\neg(p_1))) \to (p_2)) \wedge (\neg(\langle L\rangle((p_2) \to (p_3))))$$

is semantically equivalent to

$$\varphi_{\mathsf{impl\_free}} \equiv ((\neg(\neg(\neg(p_1)))) \vee (p_2)) \wedge (\neg(\langle L\rangle((\neg(p_2)) \vee (p_3)))).$$

Consider now the natural language sentence "*it is not true that is possible that later it will rain and (in the same period) it will not rain*" that can be traduced, into our language, as $\varphi \equiv \neg\langle L\rangle(p \wedge \neg p)$. The sentence corresponds to "*it is necessary true that later it will not rain or it will rain*" translated as $\varphi \equiv [L](\neg p \vee p)$. We prefer the latter example because is in negated normal form. A formula φ is in negated normal form if only propositional atoms are negated. Therefore:

$$\varphi_{\mathsf{impl\_free}} \equiv ((\neg(\neg(\neg(p_1)))) \vee (p_2)) \wedge (\neg(\langle L\rangle((\neg(p_2)) \vee (p_3))))$$

is semantically equivalent to

$$\varphi_{\mathsf{nnf}} \equiv ((\neg(p_1)) \vee (p_2)) \wedge ([L]((p_2) \wedge (\neg(p_3))))$$

that is equivalent to

$$\varphi_{\mathsf{fin}} \equiv ((\neg p_1) \vee (p_2)) \wedge ([L]((p_2) \wedge (\neg p_3))).$$

Gathering all these concepts together, we can affirm that $\varphi_{\mathsf{init}} \equiv \varphi_{\mathsf{par}} \equiv \varphi_{\mathsf{impl\_free}} \equiv \varphi_{\mathsf{nnf}} \equiv \varphi_{\mathsf{fin}}$. In Figure 4.11 we give a schematic of the transformation from φ_{par} to φ_{nnf} where the parentheses do not appear because they are intrinsically represented by the branches. Note that a revisited In-Order-Tree-Walk algorithm [CLRS09] of the tree outputs the initial formula. The case for φ_{fin} is trivial because the nodes whose decoration represents a negation along with its child, the atomic proposition, are (abstractly) squeezed into a single node.

As mentioned, we are reducing the SAT problem to the initial satisfiability of the formula:

$$\varphi = \varphi_{\mathsf{fin}} \vee \langle I \rangle \varphi_{\mathsf{fin}} \vee \langle L \rangle \varphi_{\mathsf{fin}}$$

whose initial interval is $[0, 1]$ in order to find a model $\mathcal{M}$. Therefore, the input for the tableau procedure is φ whose initial satisfiability is about to be checked. In Figure 4.12 we give the resulting tree for the initial satisfiability.

Chapter 5

Experiments and Selection Policies

In the previous chapter we left a pending issue. We assumed that the next to-be-processed leaf is selected from the list of leaves in some manner, without effectively worrying too much on how this actually happens. The first part of this chapter is devoted to how we managed to solve the leaf selection. After giving the right focus on the selection policies, we move on the implemented experiments. Finally, we present the results of our experiments.

5.1 Selection Policies

As mentioned, our procedure is programmed fully object-oriented in C++ standard library. A selection policy is said to be fair if every branch of the tableau is eventually examined, that is if every leaf of the list of leaves $\mathcal{L}$ is sooner or later inspected. We are of course interested in fair selection policies only.

Inspired from the tableau-system presented in [BDMMS13] for RPNL, we have opted to take into consideration the following key aspects. Firstly, we have considered the domain cardinality of the branch, that is the number of points in the

Finitely satisfiable	
k	formula
φ_1	$p_0 \wedge [I]\neg p_2 \wedge [\overline{L}]\neg p_2 \wedge \langle L\rangle p_1 \wedge [L](p_1 \rightarrow ([I]\neg p_3 \wedge [L]\neg p_3 \langle \overline{L}\rangle p_2))$
φ_3	$p_0 \wedge [I]\neg p_2 \wedge [\overline{L}]\neg p_2 \wedge \langle L\rangle p_1 \wedge [L](p_1 \rightarrow ([I]\neg p_3 \wedge [L]\neg p_3 \langle \overline{L}\rangle \varphi_1[+2]))$
φ_5	$p_0 \wedge [I]\neg p_2 \wedge [\overline{L}]\neg p_2 \wedge \langle L\rangle p_1 \wedge [L](p_1 \rightarrow ([I]\neg p_3 \wedge [L]\neg p_3 \langle \overline{L}\rangle \varphi_3[+2]))$
...	...

Table 5.1: A benchmark for (finitely) satisfiable formulæ of HS_3.

domain. The cardinality is computed by simply returning the size of the domain itself. On the other hand, we have considered the branch sparseness. The sparseness degree allows us to estimate how many intervals, already existing in the domain, are actually used. To compute such estimation, we calculate the average of the positive propositional letters assigned to some interval, and define the sparseness of the branch as the variance of the (ideal) binomial probabilistic variable associated to assigning positive propositions to intervals. The variance is calculated as:

$$\sigma^2 = \frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \overline{y})^2$$

where n is the number of positive propositions, y_i is the i-th proposition's value and $\overline{y}$ is the mean. Finally, we have considered how would be better for leaves to be processed, that is a breadth-first search or a depth-first search [CLRS09]. Our test showed out that the latter gives a better elapsed time. Mixing the ideas, we have implemented the following selection policies:

1. branches with smaller domain and less sparse first (SBF),
2. branches with longer domain and more sparse first (LBF), and
3. branches in First-In-First-Out order, that is the tableau tree is explored depth-first (FIFO).

5.2 Experiments

We have decided to use the thread capabilities: instead of running an experiment with only one selection policy, we have decided to run various experiments in

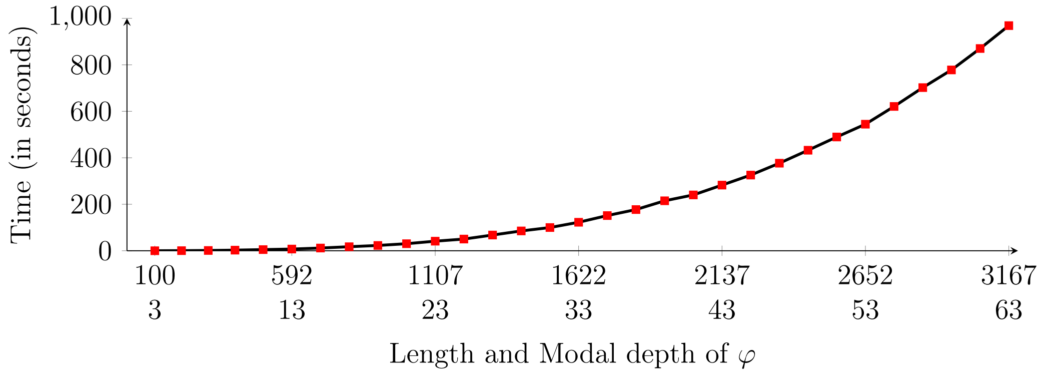


Figure 5.1: Elapsed time for satisfiable formulæ (by length and modal depth).

parallel creating as much threads as the policies, where each of them carries a specific policy. This particular approach tries to reflect the following scenario. Imagine that in a room there is a set of intelligent people whose experience could be distinguished from others. Given a formula φ, whose satisfiability must be checked, these people, having their own reasoning method, that abstractly could be associated to various selection policies, are trying to find a model. When one of these persons finds a model $\mathcal{M}$ (or proves that there is none) for φ could notice the solution to all others, stopping their reasoning, because $\mathcal{M} \models \varphi$ (or $\not\models \varphi$).

Since threads can be seen as lighter processes operating in the local environment, where each process acts only on its own variables (not directly accessible from other processes), they must be synchronized exchanging messages. This explains the part where a thread, having a solution for the particular formula, could notice the model to its "partners". Another advantage of using threads is that they all run in the same virtual space, therefore a context-switch between threads is more efficient than context-switching between processes that introduce more overhead.

We have designed a scalable experiment to generate a sequence of (arbitrary) finitely satisfiable formulæ shown in Table 5.1, that are systematically generated for $k = 1, 3, 5, \ldots$, so that their length can be put in relation with the time that our method takes to establish its satisfiability. Formulæ are generated inductively, and to generate φ_k, we use $\varphi_{k-2}[+2]$, that is φ_{k-2} where each propositional letter p_i has been replaced by p_{i+2}. In order to generate a similar benchmark for unsat-

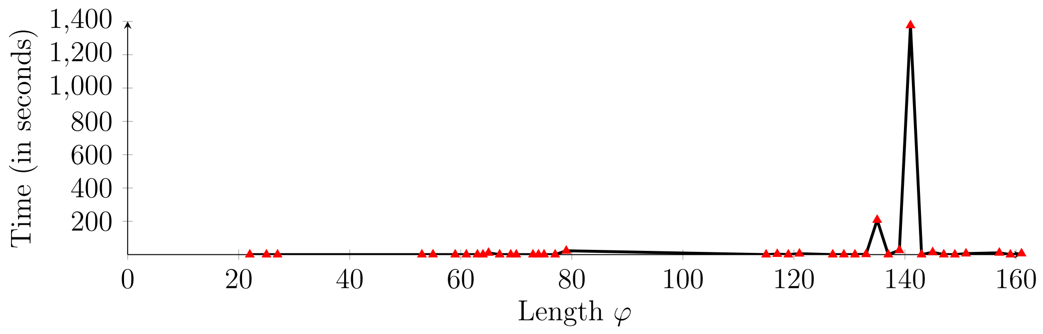


Figure 5.2: Elapsed time for finitely unsatisfiable formulæ (by length).

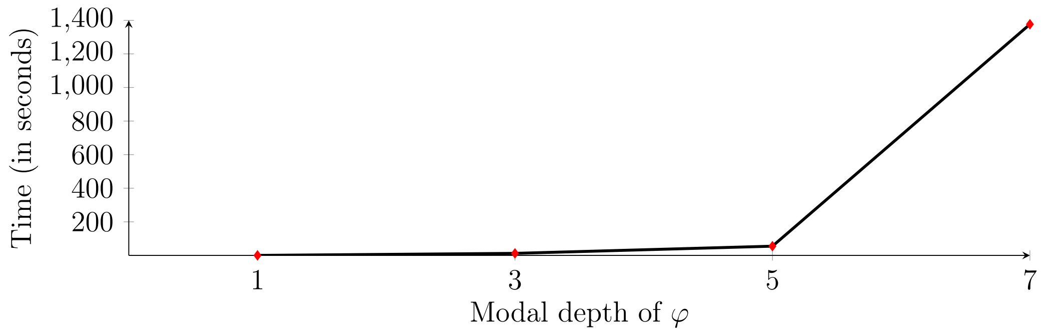


Figure 5.3: Elapsed time for finitely unsatisfiable formulæ (by modal depth).

isfiable formulæ we used a rather straightforward method: we considered a set of propositional and modal tautologies (of HS_3), and we systematically applied universal substitution to generate longer and longer tautologies; we then tested their negation for satisfiability. The universal substitution is mathematically solid for building a benchmark because it maintains the validity of a formula. For instance, consider $\varphi \equiv p \vee \neg p$, if we substitute each occurrence of p with φ, the formula still remains valid.

5.3 Results

All experiments have been carried out on Intel(R) Core(TM) i7-6700HQ, with a clock of 2.60GHz, four cores and 16GB RAM. It turns out that the elapsed time for testing a satisfiable formula grows uniformly with the length of the and the modal depth of formulæ while for unsatisfiable ones they differ; therefore there is a single plot in Figure 5.1 for the first case and two different plots in Figures 5.2 and 5.3 for the second case. Notice also that formula length and modal depth are two among the most important difficulty indicators for satisfiable/unsatisfiable formulæ in the modal case (see [BHS00]). To give a general overview of the performances, the length of unsatisfiable formulæ that we could check within a reasonable limit of 23 minutes is of at most 161 symbols, and with maximum modal depth of 7. On the other hand, satisfiable formulæ have been successfully checked up to 3167 symbols (and modal depth of 63) in less than 17 minutes.

Chapter 6

Conclusions

In this thesis we have presented an efficient implementation of a tableau-based deduction method for the interval temporal logic HS_3, whose finite SAT problem had been shown to be PSpace-complete in [MPSS15]. The results of our experiments showed that we are able to check the satisfiability of long formulæ with relatively high modal depth. As future work, we plan to improve our reasoner in several ways; here we present the most interesting of them.

The nature of reasoning is based on multiple reasonings simultaneously in order to find the right interpretation for the context. Since we are intended to build new selection features we could also try to give dynamicity to the system, that is trying to select the best selection policy on certain circumstances.

The multi-threaded approach provides us a playground for further (mathematical) studies over the selection policies. The results of our experiments provides us non-trivial ideas. For the case of satisfiable formulæ we could build a huge arsenal of selection policies for intelligently selecting the most promising branches to be inspected. Giving priority to those branches who might "look like" a possible model is the key aspect of the idea.

Unsatisfiable formulæ merely fail when trying to answer "*yes*". Therefore, we are strongly intended to optimize the reasoner for such class of formulæ. An aggressive option would be trying to identify specific portions of the tableau that can be surgically pruned in order to reduce the branching factor of the tree. This might be the case of the implementation of a new type of intelligent system whose rules are those for pruning at a specific height of the tableau by early detecting the contradictory branch via possible heuristics. In our opinion, reducing the size could be the right strategy for giving an answer in an affordable amount of time.

Exploring new heuristics for satisfiable and unsatisfiable formulæ, whose computational behavior might be as expected, could lead us to improve the system when treating with formulæ that have a high propositional component. Increasing this factor would outcome a system with a huge reasoning throughput. Two possible future works emerge from this aspect. The first one is the one above and the other is the implementation of a generator with these peculiarities.

Temporal logics are the kindergarten of model checking. Formulæ formalized via HS_3 semantics can be expressed into HS. Would be interesting to use the satisfiability checker of HS_3 developed in this thesis to find models that could be translated into the language of HS for model checking over compass structures [MPSS15].

We could formalize the problem like a Machine Learning [Mit97, WFH11] problem and design an intelligent system capable to do reasoning for us. We feel optimistic with this aspect and, therefore, we hope that for satisfiable formulæ the Machine Learning techniques could give us a boost by selecting the most promising branches in order to check satisfiability.

Bibliography

[ABM$^+$14] A. Artale, D. Bresolin, A. Montanari, V. Ryzhikov, and G. Sciavicco. DL-lite and interval temporal logics: A marriage proposal. In *Proc. of the 21st European Conference of Artificial Intelligence (ECAI)*, pages 957–958, 2014.

[AC08] Andrea Asperti and Agata Ciabattoni. *Logica ad Informatica.* McGraw-Hill, 2008.

[AF98] A. Artale and E. Franconi. A temporal Description Logic for reasoning about actions and plans. *Journal of Artificial Intelligence Reasoning*, 9:463–506, 1998.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[ARKZ15] A. Artale, V. Ryzhikov, R. Kontchakov, and M. Zakharyaschev. A cookbook for temporal conceptual data modelling with Description Logics. *ACM Transaction on Computational Logic*, 2015. To appear.

[BDMMS13] Davide Bresolin, Dario Della Monica, Angelo Montanari, and Guido Sciavicco. *A Tableau System for Right Propositional Neighborhood Logic over Finite Linear Orders: An Implementation*, pages 74–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[BDMS14] D. Bresolin, D. Della Monica, A. Montanari, and G. Sciavicco. The light side of interval temporal logic: the Bernays-Schönfinkel fragment of CDT. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):11–39, 2014.

[Bet97] C. Bettini. Time-dependent concepts: Representation and reasoning using temporal description logics. *Data Knowedge Engeneering*, 22(1):1–38, 1997.

[BHS00] P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *J. Autom. Reasoning*, 24(3):297–317, 2000.

[BMM$^+$14] D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theoretical Computer Science*, 560:269–291, 2014.

[BMS16] D. Bresolin, E. Muñoz-Velasco, and G. Sciavicco. On the complexity of fragments of Horn modal logics. In *Proc. of the 23rd International Symposium on Temporal Representation and Reasoning (TIME)*, pages 186–195, 2016.

[CH04] Z. Chaochen and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS: Monographs in Theoretical Computer Science. Springer, 2004.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[Eme90] E.A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B: formal models and semantics, pages 995–1072. Elsevier MIT Press, 1990.

[GG15] Valentin Goranko and Antony Galton. Temporal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2015 edition, 2015.

[GMS03] V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood temporal logics. *Journal of Universal Computer Science*, 9(9):1137–1167, 2003.

[GMS04] V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *Applied Non-classical Logics*, 14(1-2):9–54, 2004.

[Gol03] Robert Goldblatt. Mathematical modal logic: A view of its evolution. *Journal of Applied Logic*, 1(5-6):309–392, October 2003.

[GS93] M.C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *Journal of the ACM*, 40(5):1108–1133, 1993.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, 2th edition, 2004.

[HS91] J.Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38:279–292, 1991.

[Kla14] S. Klarman. Practical querying of temporal data via OWL 2 QL and SQL:2011. In *Proc. of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 26 of *EPiC Series*, pages 52–61. Center for Artificial Inteligence Research, 2014.

[Kri63] Saul A. Kripke. Semantical analysis of modal logic I. Normal propositional calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 9(56):67–96, 1963.

[Lad77] R.E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[MGMS11] Dario Della Monica, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. Interval temporal logics: A journey. *European Association for Theoretical Computer Science. Bulletin*, 105:73–99, 2011.

[Mit97] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[MM14] J. Marcinkowski and J. Michaliszyn. The undecidability of the logic of subintervals. *Fundamenta Informaticae*, 131(2):217–240, 2014.

[Mos83] B. Moszkowski. *Reasoning About Digital Circuits.* PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1983.

[MPSS15] E. Muñoz-Velasco, M. Pelegrín-García, P. Sala, and G. Sciavicco. On coarser interval temporal logics and their satisfiability problem. In *Proc. of the 16th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2015)*, pages 105–115, 2015.

[MSV02] A. Montanari, G. Sciavicco, and N. Vitacolonna. Decidability of interval temporal logics over split-frames via granularity. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *LNAI*, pages 259–270. Springer, 2002.

[PH05] I. Pratt-Hartmann. Temporal prepositions and their logic. *Artificial Intelligence*, 166(1–2):1–36, 2005.

[Sch90] A. Schmiedel. Temporal terminological logic. In *Proc. of the 8th National Conference on Artificial Intelligence (AAAI)*, pages 640–645. AAAI Press, 1990.

[Sis12] Michael Sisper. *Introduction to the Theory of Computation.* Cengage Learning, Inc, 3rd edition, 2012.

[Str13] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 4th edition, 2013.

[WFH11] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3 edition, 2011.

[Zem12] F. Zemke. What's new in SQL:2011. *SIGMOD Record*, 41(1):67–73, 2012.