TU WIEN Informatics

On the Complexity of Model Checking for Temporal Equilibrium Logic

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Logic and Computation

eingereicht von

Oscar Barreca
Matrikelnummer 12143764

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Mitwirkung: MSc Davide Soldà

Wien, 08.10.2025

Oscar Barreca | Thomas Eiter

TU WIEN Informatics

On the Complexity of Model Checking for Temporal Equilibrium Logic

MASTER THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Logic and Computation

by

Oscar Barreca
Registration Number 12143764

to the Faculty of Informatics
at the TU Wien
Advisor: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Assistance: MSc Davide Soldà

Vienna, 08.10.2025

Oscar Barreca Thomas Eiter

Declaration of Authorship

Oscar Barreca

I hereby declare that I have written this thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Wien, 08.10.2025

Oscar Barreca

Danksagung

Als ich zum ersten Mal von Professor Thomas Eiter eingeladen wurde, um die Vorbereitungen für diese Arbeit zu besprechen, hatte ich keine genaue Vorstellung davon, mit wem ich es zu tun hatte. Ich wusste zwar, dass er für eine beträchtliche Anzahl von Lehrveranstaltungen verantwortlich war, und ich ahnte vage, dass er durch seine eigene Forschungstätigkeit und seine Verwaltungsaufgaben bereits ausgelastet sein musste. Trotzdem hatten er und ich, noch bevor ich ernsthaft mit meiner Arbeit beginnen konnte, einen regen E-Mail-Austausch und vereinbarten nicht weniger als drei persönliche Treffen. Wie konnte jemand, den ich für so außerordentlich beschäftigt hielt, mir so großzügig seine Zeit zur Verfügung stellen? Bis heute ist mir die Antwort auf diese Frage nicht klar, aber ich bin ihm dankbar für all die Aufmerksamkeit, die er meiner Arbeit gewidmet hat. Vielen Dank für Ihre Zeit und Geduld.

Derjenige, der wahrscheinlich den größten Einfluss auf diese Arbeit hatte, war mein Co-Betreuer, Davide Soldà. Ich habe wahrscheinlich aufgehört zu zählen, wie viele E-Mails wir ausgetauscht haben[1] und wie viele Besprechungen wir abgehalten haben, um die Details dieser Arbeit zu besprechen. Er hat zu fast jeder geschriebenen Zeile Feedback gegeben, und wenn ihm etwas nicht gepasst hat, hat er nicht gezögert, darauf hinzuweisen. Vielen Dank, dass du ein gnadenloser Kritiker warst. Es ist ehrliches und direktes Feedback, und keine selbstgefällige Akzeptanz, das wir in der Wissenschaft brauchen.

Vielen Dank an Pedro Cabalar, David Pearce und Laura Bozzelli für die Beantwortung meiner E-Mails, und an Michael Gelfond für die Bereitstellung einer wertvollen Bibliografie-Datei, die mir viele Stunden manueller Arbeit erspart hat.

Vielen Dank an meine Familie für ihre finanzielle Unterstützung, die natürlich nicht nur für die Dauer dieser Arbeit, sondern während meines gesamten Studiums gewährt wurde. Ohne ihre Großzügigkeit wäre nichts von dem, was hier geschrieben steht, jemals entstanden.

Diese Arbeit wurde in Typst[2] geschrieben, einer neuen Satzsprache, und deshalb möchte ich auch dem Typst-Team dafür danken, dass es die Art und Weise, wie wir im 21. Jahrhundert digital setzen, neu überdacht hat. Typst ist eine schöne, übersichtliche Satzsprache, die das Verfassen wissenschaftlicher Arbeiten zu einem Vergnügen macht. Ich konnte das Verhalten dieses Dokuments mit klassischen Programmiersprachenkonstrukten (wie `if`- und `for`-Anweisungen) festlegen, was mir sowohl Spaß gemacht als auch meine Produktivität gesteigert hat. Ich hoffe, dass Typst in den kommenden Jahren weiter wachsen wird und dass es eines Tages zur gängigen akademischen Praxis wird, damit verfasste Artikel bei etablierten Fachzeitschriften einzureichen.

[1] Mein E-Mail-Client hat jedoch nicht. Bis heute haben ich und Davide insgesamt 270 E-Mails ausgetauscht. Nehmen wir einmal an, dass nur 85% davon mit der Abschlussarbeit zu tun hatten. Das ergibt immer noch insgesamt $\lfloor 270 \cdot \frac{85}{100} \rfloor = 229$ E-Mails.

[2] https://typst.app/.

Acknowledgements

When I was first invited by professor Thomas Eiter to discuss the preliminaries of this work, I had no exact idea of who I was in. I knew, as a matter of fact, that he was responsible to teach a considerable number of courses, and I vaguely sensed he must be held already occupied by his own research activity and administrative duties. Despite that, before I could even seriously begin with my work I and him had a thick exchange of e-mails, and arranged no less than three in-person meetings. How was, then, someone I regarded as so remarkably busy able to make such a generous offer of his time? To this day, the answer to this question remains unanswered to me, but I am grateful to him for all the attention he provided to my work. Thank you for all of your time and patience.

The one who probably had the most tangible impact on this work was my co-supervisor, Davide Soldà. I must have lost count of the e-mails we exchanged[3] and the meetings we held to discuss the nitty-gritty of this work. He provided a feedback to almost every written line, and when something was not right to him, he did not hesitate to point it out. Thanks a lot for being a merciless reviewer. It is a honest and direct feedback, and not a complacent acceptance, that we need in science.

Many thanks to Pedro Cabalar, David Pearce and Laura Bozzelli for replying to my e-mails, and to Michael Gelfond for sharing a precious bibliography file that saved me hours of manual labor.

Many thanks to my family for their financial support which, it goes without saying, was extended not just for the duration of this work but throughout the length of my studies. Without their generosity none of what has been written here would have ever seen the light of day.

This work was written in Typst[4], an emerging typesetting language, and because of that I would also like to thank the Typst team for rethinking the way we do digital typesetting in the twenty-first century. Typst is a beautiful, neat typesetting language that makes composing a scientific work an enjoyable experience. I was able to specify the behavior of this document using classical programming language constructs (like `if` and `for` statements) and doing so has been both a source of fun and a boost to my productivity. I hope to continue see Typst grow in the coming years, and that one day submitting papers written in it to established journals will have become a standard academic practice.

[3] My e-mail client, however, has not. To this day, I and Davide exchanged a grand total of 270 e-mails. Let us suppose, for the sake of the argument, that only 85% of these were related to the thesis work. This still yields a total of $\lfloor 270 \cdot \frac{85}{100} \rfloor = 229$ e-mails.

[4] https://typst.app/.

Kurzfassung

Bei einer LTL-Formel φ und einer Spur T besteht das Problem des Model Checkings für *Temporal Equilibrium Logic* (TEL) darin, zu bestimmen, ob T ein *Temporal Equilibrium Model* (TEM) von φ ist. Das Problem des Model Checkings, das bereits für mehrere andere prominente klassische Logiken wie *Linear Temporal Logic* (LTL) und *Computation Tree Logic* (CTL) untersucht wurde, stellt aufgrund seiner Anwendungen in der Software- und Hardwareverifikation sowohl ein interessantes theoretisches als auch ein praktisch relevantes Problem dar.

Diese Arbeit leistet einen Beitrag zur Untersuchung der rechnerischen Komplexität des Model Checking Problems für TEL. Zunächst wird die Komplexität von ELMC, dem Model Checking Problem der atemporalen Version EL von TEL, bestimmt und dann als Grundlage für die Bestimmung der Komplexität von TELMC$_f$, dem Problem des Model Checkings für TEL über Spuren endlicher Länge, verwendet. Anschließend wird die Komplexität von TELMC, der Model Checking Problem für TEL über Spuren unendlicher Länge, untersucht. Es wird gezeigt, dass die Komplexität von ELMC und TELMC$_f$ coNP-vollständig ist, während die von TELMC PSPACE-vollständig ist.

Motiviert durch die Unlösbarkeit des TELMC-Problems werden einige seiner Fragmente analysiert, sowohl um rechnerisch einfachere Probleme zu entdecken als auch um festzustellen, wann die Unlösbarkeit beginnt. Es wird gezeigt, dass die Entscheidung über TELMC$_1$ (das Fragment von TELMC, bei dem die sogenannte *temporale Höhe* durch 1 begrenzt ist) und TELMC$(\bigcirc, \mathbb{U})$ (das Fragment von TELMC, bei dem nur die Operatoren $\bigcirc$ und $\mathbb{U}$ zulässig sind) coNP-vollständig ist, während die Entscheidung über TELMC$(\mathbb{U}, \mathbb{R})$ (das TELMC-Fragment, in dem nur die Operatoren $\mathbb{U}$ und $\mathbb{R}$ zulässig sind) und TELMC$(\bigcirc, \mathbb{R})$ (das TELMC-Fragment, in dem nur die Operatoren $\bigcirc$ und $\mathbb{R}$ zulässig sind) PSPACE-vollständig ist. Als einfache Folge davon ist die Entscheidung über TELMC$^1_2$, TELMC$^1_2(\mathbb{U}, \mathbb{R})$ und TELMC$^1_2(\bigcirc, \mathbb{R})$ ebenfalls PSPACE-vollständig.

Die Komplexität von TELTS, einer Erweiterung des Model Checking Problems für TEL über sogenannte *Transition Systems*, wird ebenfalls untersucht. TELTS ist in zwei Varianten unterteilt: TELTS$_\forall$, die universelle Variante von TELTS, und TELTS$_\exists$, die existenzielle Variante. Die Entscheidung über TELTS$_\forall$ ist nachweislich PSPACE-vollständig, während die Entscheidung über TELTS$_\exists$ EXPSPACE-vollständig ist. Die Entscheidung von TELTS$_\forall(\mathbb{R})$ (dem Fragment des TELTS$_\forall$-Problems, bei dem nur der *Release* Operator $\mathbb{R}$ zulässig ist) ist weiterhin PSPACE-vollständig, und ebenso ist die Entscheidung von TELTS$_\exists(\mathbb{R})$ (dem Fragment des TELTS$_\exists$-Problems, bei dem nur der *Release* Operator $\mathbb{R}$ zulässig ist) weiterhin EXPSPACE-vollständig.

Abstract

Given an LTL formula φ and a trace T, the problem of model checking for *temporal equilibrium logic* (TEL) is the problem of determining whether T is a *temporal equilibrium model* (TEM) of φ. The problem of model checking, already studied for several other prominent classical logics, like linear temporal logic (LTL) and computation tree logic (CTL), represents both an interesting theoretical problem and a practically relevant one, due to its applications in software and hardware verification.

This work contributes to the study of the computational complexity of the model checking problem for TEL. The complexity of ELMC, the model checking problem of the atemporal version EL of TEL, is first determined and then used as a basis to determine the complexity of TELMC_f, the problem of model checking for TEL over traces of finite length. The complexity of TELMC, model checking for TEL over traces of infinite length, is subsequently studied. The complexity of ELMC and TELMC_f is shown to be coNP–complete, whereas the one of TELMC PSPACE–complete.

Motivated by the intractability of the TELMC problem, some of its fragments are analyzed, in the attempt to discover computationally easier problems and to determine when intractability starts to emerge. Deciding TELMC_1 (the fragment of TELMC where the so-called *temporal height* is bounded by 1) and $\text{TELMC}(\bigcirc, \mathbb{U})$ (the fragment of TELMC where only the $\bigcirc$ and $\mathbb{U}$ operators are allowed) is shown to be coNP–complete, while deciding $\text{TELMC}(\mathbb{U}, \mathbb{R})$ (the TELMC fragment where only the $\mathbb{U}$ and $\mathbb{R}$ operators are allowed) and $\text{TELMC}(\bigcirc, \mathbb{R})$ (the TELMC fragment where only the $\bigcirc$ and $\mathbb{R}$ operators are allowed) PSPACE–complete. As a simple corollary, deciding TELMC^1_2, $\text{TELMC}^1_2(\mathbb{U}, \mathbb{R})$ and $\text{TELMC}^1_2(\bigcirc, \mathbb{R})$ is also PSPACE–complete.

The complexity of TELTS, an extension of model checking for TEL over transition systems, is also investigated. TELTS is divided into two variants, $\text{TELTS}_\forall$, the universal variant of TELTS, and $\text{TELTS}_\exists$, the existential one. Deciding $\text{TELTS}_\forall$ is shown to be PSPACE–complete, whereas deciding $\text{TELTS}_\exists$ EXPSPACE–complete. Deciding $\text{TELTS}_\forall(\mathbb{R})$ (the fragment of the $\text{TELTS}_\forall$ problem where only the release operator $\mathbb{R}$ is allowed) is still PSPACE–complete, and similarly deciding $\text{TELTS}_\exists(\mathbb{R})$ (the fragment of the $\text{TELTS}_\exists$ problem where only the release operator $\mathbb{R}$ is allowed) EXPSPACE–complete.

Table of Contents

Chapter 1

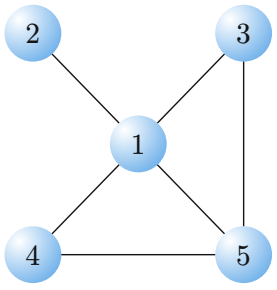
Introduction

I also defend the underlying methodology of equilibrium logic which one might summarise via the following thesis: intermediate and multi-valued logics provide a useful basis on which to construct nonmonotonic logics for programming and knowledge representation tasks.

— *D. Pearce [1]; Equilibrium Logic (2006).*

Temporal equilibrium logic is a temporal and nonmonotonic logic. It is *temporal*, because it features temporal operators, and it is *nonmonotonic*, because it features nonmonotonicity: Whenever new facts are added to a set of beliefs, a fact that was previously established might need to be retracted. This is in contrast to many prominent classical logics, like propositional, predicate, or linear temporal logic (LTL), where the addition of new information always translates to the increment of facts that can be derived.

Temporal equilibrium logic (henceforth abbreviated as TEL) has its roots in answer set programming (ASP) (V. Lifschitz [2], T. Eiter, G. Ianni, and T. Krennwallner [3], C. Baral [4]) a nonmonotonic formalism related to the logic programming language Prolog, used, among other things, for the resolution of constraint satisfaction problems (CSPs), in planning and in knowledge representation (KR). ASP is a versatile tool, and as shown in Table 1 (E. Erdem, M. Gelfond, and N. Leone [5]) the variety of its applications remarkably vast. Two simple examples of ASP in practice are given in Figure 1 and in Figure 2.



(a) A simple graph G.

vertex(1). vertex(2). vertex(3).
vertex(4). vertex(5).
edge(1, 2). edge(1, 3). edge(1, 4).
edge(1, 5). edge(3, 5). edge(4, 5).
colour(red). colour(blue). colour(green).

$$\text{coloured}(V, C) \leftarrow \text{vertex}(V) \wedge \text{colour}(C) \wedge \text{colour}(C_1) \wedge \text{colour}(C_2)$$
$$\wedge\, C \neq C_1 \wedge C \neq C_2 \wedge C_1 \neq C_2$$
$$\wedge\, \text{not coloured}(V, C_1) \wedge \text{not coloured}(V, C_2).$$
$$\leftarrow \text{edge}(V_1, V_2) \wedge \text{coloured}(V_1, C) \wedge \text{coloured}(V_2, C) \wedge V_1 \neq V_2.$$

(b) An ASP program solving the 3-colorability problem for G.

Figure 1: Solving the 3-colorability problem with ASP.

	2							5
					4			
								1
					8			
		4						3

(a) A Sudoku instance.

$$\text{cell}(0,1,2).\ \text{cell}(0,8,5).\ \text{cell}(1,5,4).$$
$$\text{cell}(2,8,1).\ \text{cell}(5,5,8).\ \text{cell}(8,2,4).$$
$$\text{cell}(8,8,3).$$

$$\text{row}(0).\ \cdots\ \text{row}(8).$$
$$\text{col}(0).\ \cdots\ \text{col}(8).$$
$$\text{value}(0).\ \cdots\ \text{value}(8).$$

$$\begin{aligned}
&\text{cell}(R,C,V) \leftarrow \text{row}(R) \wedge \text{col}(C) \wedge \text{value}(V)\\
&\wedge \text{value}(V_2) \wedge \cdots \wedge \text{value}(V_8)\\
&\wedge V \neq V_2 \wedge \cdots \wedge V \neq V_8\\
&\wedge V_2 \neq V_3 \wedge \cdots \wedge V_2 \neq V_8\\
&\wedge \vdots\\
&\wedge V_7 \neq V_8\\
&\wedge \text{not cell}(R,C,V_2) \wedge \cdots \wedge \text{not cell}(R,C,V_8).\\
&\leftarrow \text{cell}(R,C_1,V) \wedge \text{cell}(R,C_2,V).\\
&\leftarrow \text{cell}(R_1,C,V) \wedge \text{cell}(R_2,C,V).\\
&\leftarrow \text{cell}(R_1,C_1,V) \wedge \text{cell}(R_2,C_2,V) \wedge \text{sameblock}(R_1,C_1,R_2,C_2).
\end{aligned}$$

(b) An ASP program for the Sudoku instance.

Figure 2: Solving a Sudoku puzzle with ASP.

Area	References
Planning	V. Lifschitz [6]; J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang [7]; T. C. Son, E. Pontelli, and C. Sakama [8]; M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub [9]
Multi-agent systems	M. D. Vos, T. Crick, J. Padget, M. Brain, and O. C. a. J. Needham [10]; T. C. Son, E. Pontelli, and C. Sakama [8]; T. C. Son, E. Pontelli, and C. Sakama [8]
Natural language processing and understanding	E. Erdem, O. Erdem, and F. Ture [11]; R. Schwitter [12]; R. Schwitter [13]
Learning	C. Sakama [14]
Theory update/revision	K. Inoue and C. Sakama [15]
Preferences	T. Schaub and K. Wang [16]; G. Brewka [17]
Diagnosis	T. Eiter, W. Faber, N. Leone, and G. Pfeifer [18]; M. Balduccini and M. Gelfond [19]; G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni [20]
Description logics and semantic web	D. Calvanese, T. Eiter, and M. Ortiz [21]; T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits [22]
Probabilistic reasoning	C. Baral, M. Gelfond, and J. N. Rushton [23]

Data integration and query answering	N. Leone *et al.* [24]; M. Alviano, W. Faber, and N. Leone [25]
Multi-context systems	G. Brewka, T. Eiter, and M. Truszczynski [26]; T. C. Son, E. Pontelli, N.-H. Nguyen, and C. Sakama [27]
Argumentation	F. Toni and M. Sergot [28]; T. C. Son, E. Pontelli, N.-H. Nguyen, and C. Sakama [27]
Scheduling	M. Balduccini [29]
Wire routing	E. Erdem, V. Lifschitz, and M. F. Wong [30]; D. East and M. Truszczynski [31]; E. Erdem and M. D. F. Wong [32]
Decision support systems	M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry [33]
Bounded model checking	K. Heljanko and I. Niemelä [34]
Game theory	D. V. Marina and D. Vermeir [35]; M. D. Vos and D. Vermeir [36]
Games and puzzles	R. Finkel, V. W. Marek, and M. Truszczynski [37]; M. Celik, H. Erdogan, F. Tahaoglu, T. Uras, and E. Erdem [38]
Phylogenetics	E. Erdem, V. Lifschitz, and D. Ringe [39]; D. R. Brooks, E. Erdem, S. T. Erdogan, and J. W. M. and D. Ringe [40]; E. Erdem [41]; E. Erdem [42]
Combinatorial auctions	C. Baral and C. Uyan [43]
Systems biology	N. Tran and C. Baral [44]; T. Schaub and S. Thiele [45]; M. Gebser *et al.* [46]; M. Gebser, T. Schaub, S. Thiele, and P. Veber [47]
Automatic music composition	G. Boenn, M. Brain, M. D. Vos, and J. Fitch [48]

Table 1: Some applications of ASP.

One fundamental difference between TEL and ASP is that TEL is a logic, whereas ASP is not. In fact, even though ASP is characterized by its own syntax and semantics, the way the logical operators (or what could correspond to logical operators) of an ASP program can be combined is constrained to a very specific form. In its most general formulation, ASP programs consist of ASP rules, which all take the form

$$h_1(z_1) \vee \cdots \vee h_r(z_r) \leftarrow p_1(x_1) \wedge \cdots \wedge p_n(x_n) \wedge \text{not } p_{n+1}(x_{n+1}) \wedge \cdots \wedge \text{not } p_m(x_m).,$$

where $\vee, \wedge, \text{not}$ and $\leftarrow$ are the disjunction, conjunction, default negation and implication operators of ASP.

A reformulation of ASP that allows the free intermixing of these operators is equilibrium logic (EL) (D. Pearce [49]). EL is a nonmonotonic logic extending the syntax of ASP in a way that the standard stable model semantics (M. Gelfond and V. Lifschitz [50]) of ASP is preserved. All the $\vee, \wedge, \text{not}$ and $\leftarrow$ operators traditionally used in ASP can be freely mixed together in EL, and the behavior they yield is the same as that of a standard ASP program. Semantically, there is no difference between talking about ASP or talking about EL.

The presence of temporal operators is the second major difference between TEL and ASP. Even though it is possible to simulate a time-like behavior in ASP, by introducing a

discrete time variable t, this change does not bring the introduction of temporal operators (like the "always" $\Box$ or the "eventually" $\Diamond$) with it. To suit the need to specify temporal properties, EL is extended with the syntax of LTL, and the result of that union is TEL. In a way, TEL could be said to be expressed by the evocative equation

$$\text{TEL} \stackrel{\text{def}}{=} \text{EL} + \text{LTL},$$

but the real picture of what TEL actually is is a bit more nuanced, and requires a few more words. TEL is not, in fact, the direct union of EL and LTL, but of EL and temporal here-and-there (THT), a monotonic temporal logic which, in turn, is the union of here-and-there (HT) (A. Heyting [51]), a three-valued, atemporal monotonic logic, and LTL. EL, in turn, can be seen as the nonmonotonic extension of HT. For ease of understanding, Figure 3 puts all these logics together into one single picture.

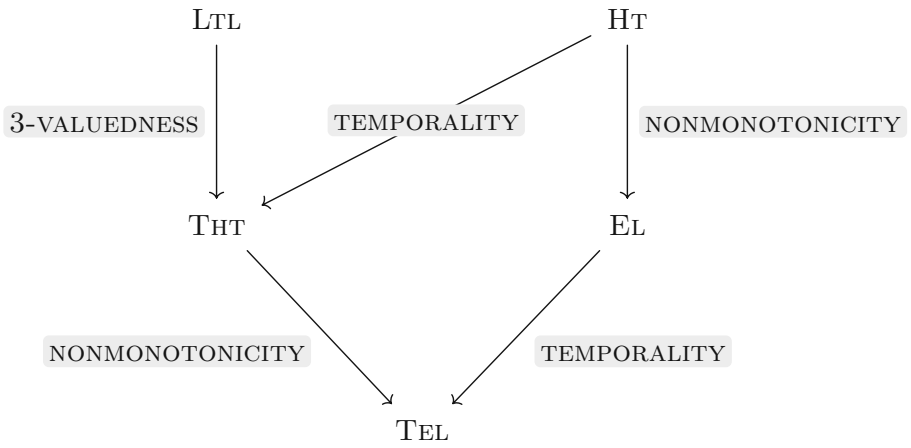


Figure 3: Schematic overview of TEL.

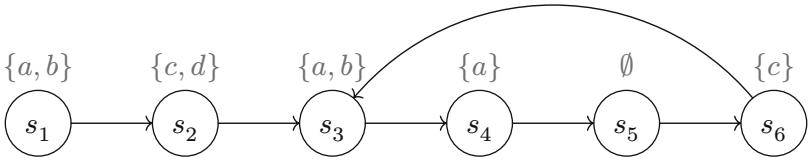
One distinguishing feature of the HT and THT logics to more ordinary ones is that their intended semantics is three-valued: Each atomic and composite formula can be either true, false, or *assumed*[5]. As a consequence, an interpretation for THT is not a single trace T, like for LTL, but a pair of traces (H, T), subject to the condition that for all i, $H_i \subseteq T_i$. Even an interpretation for TEL is described by a pair of two traces, but the intended models (H, T) of a TEL formula φ, referred to as temporal equilibrium models (TEMs), are those which are total, that is for which $H = T$, and for which there does not exist an $H' < H$ such that (H', T) satisfies φ.

1.1. Problem statement

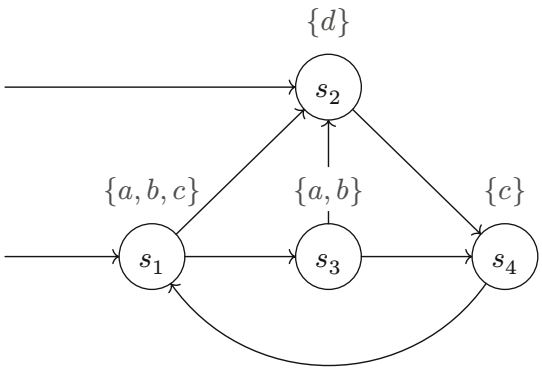
Given a THT formula φ and an interpretation I for φ, is I a TEM of φ? This is the model checking problem for TEL, and understanding the computational resources needed to solve it the aim of this work. In its most basic setting, the interpretation I for φ is a linear trace T, a sequence of states marked by the set of atomic propositions holding on it or, alternatively, a sequence of sets of atomic propositions. Alternative interpretations are, however, also possible. Chapter 5 takes an interpretation I for a formula φ to be a transition system, a graph-like structure from which different traces can be obtained[6]. These two types of interpretations are put in contrast in Figure 4.

[5] Sometimes, *unknown* is also said.

[6] It is indeed possible to see linear traces as an edge case of transition systems.



(a) An interpretation for a TEL formula can be a linear trace T,



(b) or a transition system τ.

Figure 4: Two different types of interpretations for TEL.

1.2. Related work

P. Cabalar and S. Demri [52] are arguably the first to investigate the complexity of reasoning problems over TEL, showing, among other things, that THT satisfiability (deciding whether a THT theory admits a model) is PSPACE–complete and that TEL satisfiability is in EXPSPACE and PSPACE–hard. L. Bozzelli and D. Pearce [53] extend these results by showing that TEL satisfiability is indeed EXPSPACE–hard, and not just PSPACE–hard, closing the gap left open by P. Cabalar and S. Demri [52] and showing TEL satisfiability to be EXPSPACE–complete. In light of these negative results, the satisfiability problem is investigated for various fragments of TEL, but intractability results, summarised in Table 2, are obtained again.

Language ($m \geq 1, k \geq 1$)	**Complexity**
THT, $\mathrm{THT}^{1}_{m+1}(\Diamond, \Box)$, $\mathrm{THT}^{k+1}_{m+1}(\Box)$, $\mathrm{THT}^{k+1}_{m+1}(\mathbb{U})$	Expspace–complete
$\mathrm{THT}(\bigcirc)$, $\mathrm{THT}(\Diamond)$, $\mathrm{THT}(\bigcirc, \Diamond)$	Σ_2–complete
THT_1, $\mathrm{THT}^{k+1}_{1}(\Diamond, \Box)$, $\mathrm{THT}^{k+1}_{1}(\mathbb{U})$, $\mathrm{THT}^{k+1}_{1}(\mathbb{R})$	Nexptime–complete
$\mathrm{THT}_1(\bigcirc, \Box)$	Σ_2–complete
$\mathrm{THT}^{1}(\mathbb{R})$, $\mathrm{THT}^{1}(\bigcirc, \mathbb{R})$, $\mathrm{THT}^{1}(\mathbb{U})$, $\mathrm{THT}^{1}(\bigcirc, \mathbb{U})$	Pspace–complete
THT^{1}_{1}	NP–complete
THT^{0}	• Pspace–hard • $\in$ Expspace
$\mathrm{THT}_0 = \mathrm{HT}$	Σ_2–complete

Table 2: Complexity of the TEL satisfiability (*consistency*) problem (L. Bozzelli and D. Pearce [53]). The symbols surrounded by parentheses ($\bigcirc, \Diamond, \Box, \mathbb{U}$ and $\mathbb{R}$) represent the set of temporal operators restricted to a given TEL fragment. The subscripts and superscripts, as in $\mathrm{THT}^{k+1}_{m+1}(\Box)$, refer to the so-called *temporal height* and *implication height* parameters, respectively.

M. Cadoli [54] studies the problem of model checking for circumscribed first-order formulas, formulas where a kind of minimization criterion is involved, showing that for most of these theories model checking is coNP–complete. Chapter 3 makes use of these results to derive the coNP–hardness of the Elmc problem, which is later used as a basis to determine the complexity of the $\textsc{Telmc}_f$ problem and some fragments of the Telmc one.

A. P. Sistla and E. M. Clarke [55] investigate the problem of satisfiability, validity and satisfaction for LTL over a Kripke struture. In particular, the satisfaction problem in a Kripke structure, asking whether a given Kripke structure admits a path satisfying a certain LTL formula, is shown to be Pspace–complete for LTL and several of its fragments. This work represents the basis for some of the results of L. Bozzelli and D. Pearce [53] and finds direct application even within this work, in particular in Chapter 5.

Motivated by the negative results of A. P. Sistla and E. M. Clarke [55], S. Demri and P. Schnoebelen [56] investigate the complexity of the model checking and satisfiability problem for LTL under restricted fragments, obtaining in certain cases tractability results. Parts of this work are used in Chapter 4, specifically Section 4.2, where the model checking problem for TEL under restricted theories is studied.

Marginally relevant to the present work are the contributions of N. Leone *et al.* [57] and T. Eiter *et al.* [58]. N. Leone *et al.* [57] present the DLV system for disjunctive logic programming (DLP), an extension of classical ASP that allows disjunctions in the head of rules, and present some computational results for the problem of answer set checking in fragments of the DLV language where so-called *weak constraints* are allowed. T. Eiter

et al. [58] extend these results by taking even the case of non-ground programs into account, showing that in most syntactic fragments (like bounded predicate arity ones) the complexity of answer set checking for non-ground programs lies within the polynomial hierarchy, and does not experience the exponential blow up that occurs in answer set existence, brave reasoning or cautious reasoning.

1.3. Contributions

This work contributes to the study of the complexity of the model checking problem for TEL. Other reasoning problems, like the satisfiability one, have already received attention in the past (P. Cabalar and S. Demri [52], L. Bozzelli and D. Pearce [53]), but no evidence for the study of the model checking problem seems to be available.

The first result of this work determines the complexity of ELMC, the model checking problem for the atemporal EL logic. The model checking problems for TEL over finite traces, TELMC_f, and infinite traces, TELMC, are studied next. The complexity of the ELMC problem is shown to be coNP–complete (Theorem 3.13) and similarly, the complexity of the TELMC_f problem is also shown to be coNP–complete (Theorem 3.17). In contrast, the one of the TELMC problem is shown PSPACE–complete (Theorem 3.23). TELMC_f and TELMC display a considerable complexity gap, even though the only difference between the two problems is that one considers finite traces, whereas the other infinite ones. The PSPACE–completeness of TELMC is a cornerstone result, so much so that it alone determines the presence of Chapter 4, and to corroborate it an additional, alternative proof based on domino tiling problems (P. v. E. Boas [59], F. Schwarzentruber [60]) is given in Theorem 4.52 for the $\text{TELMC}(\mathbb{U}, \mathbb{R})$ fragment, also shown to be PSPACE–complete.

Chapter 4 studies the complexity of different fragments of the TELMC problem, showing some of these to be easier and others just as hard the TELMC problem. Deciding the TELMC problem remains PSPACE–complete even for TELMC_2^1 (Corollary 4.34), the TELMC fragment where the so-called temporal height is limited to a value no greater than 2 and the implication height to a value no greater than 1. Deciding TELMC_1 and $\text{TELMC}(\bigcirc, \mathbb{U})$, however, is shown to be coNP–complete (Theorem 4.42 and Theorem 4.44). Deciding $\text{TELMC}(\mathbb{U}, \mathbb{R})$ and $\text{TELMC}(\bigcirc, \mathbb{R})$ is still PSPACE–complete (Theorem 4.52 and Theorem 4.55) and remains so even for the $\text{TELMC}_2^1(\mathbb{U}, \mathbb{R})$ and $\text{TELMC}_2^1(\bigcirc, \mathbb{U})$ fragments (Corollary 4.54 and Corollary 4.58).

Chapter 5 extends the setting of Chapter 3 in the study of model checking for TEL, replacing the linear structures of Chapter 3 with transition systems, graph-like structures that allow a ramification behavior and under which multiple traces are simultaneously considered. The two problems considered in Chapter 5 are $\text{TELTS}_\forall$, the universal variant of the model checking problem for TEL over transition systems, and $\text{TELTS}_\exists$, the existential one. Deciding $\text{TELTS}_\forall$ is shown to be PSPACE–complete (Theorem 5.67), while deciding $\text{TELTS}_\exists$ EXPSPACE–complete (Theorem 5.68).

Chapter 6 finally shows that deciding $\text{TELTS}_\forall(\mathbb{R})$, the $\text{TELTS}_\forall$ fragment where only the release operator $\mathbb{R}$ is allowed, remains PSPACE–complete (Theorem 6.81), and similarly that deciding $\text{TELTS}_\exists(\mathbb{R})$, the $\text{TELTS}_\exists$ fragment where only the $\mathbb{R}$ operator is allowed, also remains EXPSPACE–complete (Theorem 6.87).

A summary of the results of the work is given in Table 3.

	Problem	Complexity
Chapter 3	ELMC	coNP–complete (Theorem 3.13)
	ELMC_f	coNP–complete (Theorem 3.17)
	ELMC	PSPACE–complete (Theorem 3.23)
Chapter 4	ELMC_2^1	PSPACE–complete (Corollary 4.34)
	ELMC_1	coNP–complete (Theorem 4.42)
	$\text{ELMC}(\bigcirc, \mathbb{U})$	coNP–complete (Theorem 4.44)
	$\text{ELMC}(\bigcirc)$	coNP–complete (Corollary 4.47)
	$\text{ELMC}(\mathbb{U})$	coNP–complete (Corollary 4.48)
	$\text{ELMC}(\mathbb{U}, \mathbb{R})$	PSPACE–complete (Theorem 4.52)
	$\text{ELMC}_2^1(\mathbb{U}, \mathbb{R})$	PSPACE–complete (Corollary 4.54)
	$\text{ELMC}(\bigcirc, \mathbb{R})$	PSPACE–complete (Theorem 4.55)
	$\text{ELMC}_2^1(\bigcirc, \mathbb{R})$	PSPACE–complete (Corollary 4.58)
Chapter 5	$\text{ELTS}_\forall$	PSPACE–complete (Theorem 5.67)
	$\text{ELTS}_\exists$	EXPSPACE–complete (Theorem 5.68)
Chapter 6	$\text{ELTS}_\forall(\mathbb{R})$	PSPACE–complete (Theorem 6.81)
	$\text{ELTS}_\forall(\bigcirc, \mathbb{R})$	PSPACE–complete (Corollary 6.82)
	$\text{ELTS}_\forall(\mathbb{U}, \mathbb{R})$	PSPACE–complete (Corollary 6.83)
	$\text{ELTS}_\exists(\mathbb{R})$	EXPSPACE–complete (Theorem 6.87)
	$\text{ELTS}_\exists(\bigcirc, \mathbb{R})$	EXPSPACE–complete (Corollary 6.90)
	$\text{ELTS}_\exists(\mathbb{U}, \mathbb{R})$	EXPSPACE–complete (Corollary 6.91)

Table 3: Contributions of the work.

Even though the complexity of the different fragments displays some discrepancies between the satisfiability problem and the model checking one, it is interesting to extend Table 2, summarising the complexity results of the satisfiability problem, with an additional column in which the complexity of the model checking problem for the respective fragments is also shown. This comparison is displayed in Table 4.

Language	Satisfiability	Model checking
THT	EXPSPACE–complete	PSPACE–complete
$\mathrm{THT}^{k+1}_{m+1}(\mathbb{U})$	EXPSPACE–complete	coNP–complete
$\mathrm{THT}(\bigcirc)$, $\mathrm{THT}(\Diamond)$, $\mathrm{THT}(\bigcirc, \Diamond)$	Σ_2–complete	coNP–complete
THT_1, $\mathrm{THT}^{k+1}_1(\Diamond, \Box)$, $\mathrm{THT}^{k+1}_1(\mathbb{U})$, $\mathrm{THT}^{k+1}_1(\mathbb{R})$	NEXPTIME–complete	coNP–complete
$\mathrm{THT}_1(\bigcirc, \Box)$	Σ_2–complete	coNP–complete
$\mathrm{THT}^1(\bigcirc, \mathbb{R})$	PSPACE–complete	PSPACE–complete
$\mathrm{THT}^1(\mathbb{U})$, $\mathrm{THT}^1(\bigcirc, \mathbb{U})$	PSPACE–complete	coNP–complete
THT^1_1	NP–complete	coNP–complete
THT^0	• PSPACE–hard • $\in$ EXPSPACE	$\in$ PSPACE
$\mathrm{THT}_0 = \mathrm{HT}$	Σ_2–complete	coNP–complete

Table 4: Comparison of the complexity of the satisfiability and the model checking problem for TEL.

Chapter 2

Background

To bar an easy access to newcomers every scientific domain has introduced its own terminology and notation.

— J. van Leeuwen, Ed. [61]; Handbook of Theoretical Computer Science (1991).

This initial chapter introduces some of the ideas that will be used in the remainder of the work that are not specific to any section in particular. The first two sections establish some common notation and the basic ideas of computational complexity theory. The third one provides a summary of formal languages and automata theory, including Büchi automata. The fourth one offers a technical overview of ASP, whereas the fourth one describes EL, a logical formalization of ASP. TEL is introduced in the last section.

2.1. Basic concepts and notation

Given a logic L and an L-formula φ, $|\varphi|$ denotes the number of symbols used to write φ, whereas $\text{sub}(\varphi)$ the set of subformulas of φ. For instance, if L is taken to be propositional logic and $\varphi \stackrel{\text{def}}{=} a \wedge ((b \wedge c) \rightarrow d)$, then $|\varphi| = 11$, whereas

$$\text{sub}(\varphi) \stackrel{\text{def}}{=} \{a \wedge ((b \wedge c) \rightarrow d), a, (b \wedge c) \rightarrow d, b \wedge c, d, b, c\}.$$

Sequence variables, of finite or infinite length, will be made use of throughout the whole document. If x is a sequence of n elements, its individual elements are represented as $x_1, ..., x_n$ and x in its expanded form is expressed as $x = (x_1, ..., x_n)$. Discrete intervals of the form $\{a, a+1, ..., b-1, b\}$, where $a \leq b$, are abbreviated as $[a..b]$. $[a..b)$ denotes $[a..b] \setminus \{b\}$, and similarly $(a..b]$ denotes $[a..b] \setminus \{a\}$.

2.1.1. Asymptotic notation

The Landau notation, also known as "big O" notation, is a notation widely used in many areas of computer science. This notation is expressed with reference to a function and, informally, corresponds to the class of all functions that grow within certain bounds of the initially given function. More precisely, let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a function from the naturals to the naturals themselves. Then it is said that $g \in O(f)$ (read as "g is a big-O of f") if $\exists c > 0$ and $n_0 \geq 0$ such that $\forall n \geq n_0$

$$g(n) \leq cf(n).$$

This corresponds to saying that after an initial threshold n_0, $g(n)$ is bounded by $f(n)$ times a constant c. Similarly, $g(n) \in \Omega(f(n))$ if $\exists c > 0$ and $n_0 \geq 0$ such that $\forall n \geq n_0$

$$cf(n) \leq g(n).$$

In fact, this is equivalent to claiming that $f(n) \in O(g(n))$. Finally, $g(n) \in \Theta(f(n))$ if $g(n) \in O(f(n))$ and $f(n) \in O(g(n))$.

2.2. Complexity theory

Computational complexity theory or, more simply, complexity theory, is one of the major branches of theoretical computer science that studies the classification of computational problems in terms of the resources needed to solve them. The notion of computational problem, formalized in a few paragraphs, can be informally seen as a question susceptible to an algorithmic solution and a set of possible solutions; if the problem is posed in terms of a yes-no question, then it is referred to as a decision problem. The two resources most typically taken under consideration are time and space. If the Turing machine (TM) is taken as a model of computation, the time required by a certain TM on a given input is determined by the number of steps or transitions that the machine performs on a given input, and the space by the number of cells used in all of its work tapes. This requirement is typically expressed in terms of the size of the input, and not of its values. For instance, when determining the behavior of an algorithm taking as input a list of integers, it is the number of elements of the list that matters, and not its individual values.

Asymptotic notation is used to classify computational problems. For example, if an algorithm running in time $f(n) = 6n^4 + 1.5n^2 + \log_2(n)$ is given, then the problem solved by this algorithm is said to belong to the class of problems solvable in $O(f(n))$ time. With some elementary calculation, this turns out to be $O(n^4)$. Therefore, not only are constant terms around expressions and initial values disregarded, but even lesser growing terms, such as $1.5n^2$ or $\log_2(n)$.

In this work, unless when explicitly noted otherwise, it is decision problems that will be taken under consideration. At least two main equivalent formulations of decision problem are given in the literature, a question- and a language-based one. In the first formulation, a decision problem is seen as a question together with a set of instances, or yes-instances, that constitute the set of acceptable answers to the problem. In the other formulation, a decision problem is taken to be a language, a finite or infinite set of words built over a finite alphabet A. In this work, the formulation of decision problem in terms of a language will be favored. The reason of this choice is that working with Karp reductions (introduced in Subsection 2.2.1) will be easier and technically more precise.

Definition 2.1 *(Decision problem.)* Given a language L, a decision problem is a subset P of L. □

Elements of P will be also called positive instances, while elements of $L \setminus P$ negative instances. Given a decision problem P, its host language, consisting of all of its positive and negative instances, will be denoted by Σ_P, while $\overline{P} = \Sigma_P \setminus P$ will denote the complement of P.

P is said to be decidable if there exists an algorithm A such that for all $x \in \Sigma_P$, A determines in finite time whether $x \in P$ or not. In this case P is also said to be decided by A. If A has time complexity $f(n)$, P is also said to be decided by A in time $f(n)$. (Similarly for space complexity or other notions of complexity.)

Example 2.2 The satisfiability (SAT) problem is the problem of determining whether a well-formed formula in propositional logic admits a satisfying assignment. Following the notation just introduced

$$\Sigma_{\text{SAT}} = \{\varphi \mid \varphi \text{ is a well-formed propositional formula}\},$$
$$\text{SAT} = \{\varphi \in \Sigma_{\text{SAT}} \mid \varphi \text{ is satisfiable}\},$$
$$\overline{\text{SAT}} = \text{UNSAT} = \{\varphi \in \Sigma_{\text{SAT}} \mid \varphi \text{ is not satisfiable}\}.$$

When framed in terms of a question-instance pair, this problem can also be described by the following table:

INSTANCE	A well-formed propositional formula φ.
QUESTION	Does φ admit a satisfying variable assignment?

□

2.2.1. Reductions

There are different ways to establish the computational complexity of a given problem, and of those exhibiting a concrete algorithm is arguably one of the simplest ones. If an algorithm solving a certain problem is given, then the complexity of this problem certainly is not higher than the complexity of the algorithm, for at worst the algorithm itself can be used to decide the problem. But what steps can be taken when wanting to show that a problem does *not* admit an algorithm with a given complexity? This is when reductions and reasonable computational complexity assumptions come in.

DEFINITION 2.3 *(Karp reduction.)* Given a decision problem P_1 and P_2, a *Karp reduction* (also said *polynomial-time many-one reduction*) is a function $f : \Sigma_{P_1} \to \Sigma_{P_2}$ such that:

1. $x \in P_1$ iff $f(x) \in P_2$;
2. f is computable in polynomial time with respect to the size $|x|$ of $x \in \Sigma_{P_1}$.

□

If a Karp reduction f from a problem P_1 to problem P_2 exists, then P_1 is said to "reduce to" P_2. In symbols, this is expressed as $P_1 \leq P_2$.

Karp reductions come in handy when a decision problem is related to another one. Suppose, in fact, P_1 to be a guaranteed "hard" problem (its hardness has in some way been established as an unmovable fact), which means that the resources required to decide it are prohibitive for sufficiently large input sizes. Also suppose that $P_1 \leq P_2$. Will P_2 then be an easy or a hard problem? The answer to that question is that P_2 will certainly not be an easy problem, for if it was even P_1 would become easy. In fact, if anything, P_2 is at least as hard as P_1. Unless no other fact disproves it, P_2 could even be much "harder" than P_1.

2.3. Formal languages and automata

This section recalls basic notions of finite regular languages and automata, and extends them to the case of ω-regular languages and Büchi automata. A more detailed explanation suitable for a first study can be found in C. Baier and J.-P. Katoen [62].

2.3.1. Regular languages

Given a finite set of symbols Σ, also called alphabet, a word σ on Σ is a finite sequence of symbols from Σ. The number of symbols that constitute σ, or its length, is denoted by $|\sigma|$. For instance, if $\Sigma = \{a, b, c\}$, then $\sigma = \text{ccbaaa}$ is a word from Σ of length $|\sigma| = 6$. For any alphabet Σ, the empty word, consisting of zero symbols, is denoted by ε.

Given two words σ and τ from Σ, their concatenation is the word formed by the symbols of σ followed by the symbols of τ, and is denoted by $\sigma \cdot \tau$ or simply by $\sigma\tau$. The length of $\sigma\tau$ is given by $|\sigma\tau| = |\sigma| + |\tau|$.

Given an alphabet Σ, a language is a finite or infinite set of words of finite length from Σ. Given two languages A and B, their concatenation $A \cdot B$, or AB, is defined as $AB \stackrel{\text{def}}{=} \{w_1 w_2 \mid w_1 \in A \wedge w_2 \in B\}$. For an arbitrary language A and $n \geq 1$, $A^0 \stackrel{\text{def}}{=} \{\varepsilon\}$ and $A^n \stackrel{\text{def}}{=} AA^{n-1}$. In other words, A^n is the language formed by the concatenation of n words from A. Thus for a given alphabet Σ, Σ^n is the language of words of length n from symbols in Σ. Finally, for a given alphabet Σ, the Kleene star Σ^* is defined as $\Sigma^* \stackrel{\text{def}}{=} \bigcup_{n \geq 0} \Sigma^n$, and corresponds to the set of words of arbitrary length from symbols in Σ. For convenience, $\Sigma^+ \stackrel{\text{def}}{=} \Sigma^* \setminus \{\varepsilon\}$ is also defined.

A language can be systematically constructed from a set of rules known as *regular expressions*.

Definition 2.4 Given an alphabet Σ, a *restricted regular expression* (RRE), or, more simply, just regular expression, is a syntactic expression inductively defined as:

- The expression $\emptyset$;
- The expression $\{a\}$, also written as a, where $a \in \Sigma$;
- The concatenation $e_1 \cdot e_2$, also denoted by $e_1 e_2$, of two regular expressions e_1 and e_2;
- The union $e_1 \cup e_2$, also denoted by $e_1 \mid e_2$, of two regular expressions e_1 and e_2;
- The Kleene star e^* for any regular expression e.

□

For example, given the alphabet $\Sigma = \{a, b, c, d\}$, abc $(c \mid d)b^*$ is a regular expression, where abc is short for $a \cdot b \cdot c$ and $(c \mid d)$ is an alternate form of $(c \cup d)$. Parentheses are part of the syntax and are used to specify precedence among neighboring regular expressions.

Restricted regular expressions are extended by *general regular expressions* (GRE), which add the intersection $e_1 \cap e_2$ of two GREs e_1 and e_2 and the complementation (with respect to Σ^*) $\neg e$ of any GRE e to the syntax of RGEs. *Star-free regular expressions* (SFRE) are defined as the subset of GREs deprived of the Kleene star e^*. For succinctness of exposition, the following paragraphs will focus only on RREs.

The semantics $L(e)$ of a (restricted) regular expression e is a set of words defined inductively as follows:

- $L(\emptyset) \stackrel{\text{def}}{=} \{\varepsilon\}$;
- $L(a) \stackrel{\text{def}}{=} \{a\}$;
- $L(e_1 e_2) \stackrel{\text{def}}{=} \{w_1 w_2 \mid w_1 \in L(e_1) \wedge w_2 \in L(e_2)\}$;
- $L(e_1 \mid e_2) \stackrel{\text{def}}{=} \{w \mid w \in L(e_1) \vee w \in L(e_2)\}$;
- $L(e^*) \stackrel{\text{def}}{=} \bigcup_{i \geq 0} e^i$, where $e^i = \underbrace{e \cdot e \cdot \cdots \cdot e}_{i}$ denotes the concatenation of e with itself i times.

A language L such that $L = L(e)$ for some regular expression e is said to be *regular*. Regular languages are closely related to nondeterministic and deterministic finite automata, described in the following paragraphs.

2.3.2. Nondeterministic and deterministic finite automata

Finite automata are discrete structures specified by a set of *states* and a set of *transitions* or *relations* among these states. Their intended behavior is to act as acceptors or recognizers of languages. Finite automata come in two variants, a nondeterministic and a deterministic one.

Definition 2.5 A *nondeterministic finite automaton* (NFA) is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where

- Q is a finite set of states;
- Σ is an alphabet;
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function;
- $Q_0 \subseteq Q$ is the set of initial states, and
- $F \subseteq Q$ is the set of accept or final states.

□

An example NFA is given in Figure 5, where $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b, c, d\}$, $Q_0 = \{q_0\}$, $F = \{q_4\}$ and δ is the transition function defined as:

$$\begin{aligned}
&\delta(q_0, a) = \{q_1, q_1\},\\
&\delta(q_0, x) = \emptyset, x \in \{b, c, d\},\\
&\\
&\delta(q_1, a) = \emptyset,\\
&\delta(q_1, b) = \{q_1\},\\
&\delta(q_1, c) = \{q_4\},\\
&\delta(q_1, d) = \emptyset,\\
&\\
&\delta(q_2, d) = \{q_3\},\\
&\delta(q_2, x) = \emptyset, x \in \{a, b, c\},\\
&\\
&\delta(q_3, c) = \{q_4\},\\
&\delta(q_3, x) = \emptyset, x \in \{a, b, d\},\\
&\\
&\delta(q_4, x) = \emptyset, x \in \Sigma.
\end{aligned}$$

Initial states of an NFA are distinguished by an incoming arrow without predecessor node, while terminal states are distinguished by a doubly marked stroke around the name of the state.

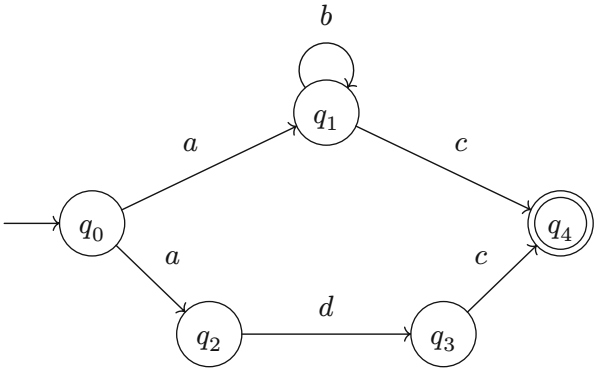


Figure 5: An example automaton.

The intended behavior of an NFA N is to receive an input word w built out of symbols from Σ, pick an initial state $q_0 \in Q_0$ and move from node to node according to the transition function δ. More specifically, given an initial state q_0 and the first character σ_0 of σ, a move is performed from state q_0 to state q if $q \in \delta(q_0, \sigma_0)$ or, in other words, if an edge labeled σ_0 from q_0 to q is present. This process is repeated for all the subsequent states

and characters σ_i from σ, until the word σ is read in its full length or a state in N is reached from which no more moves are possible. If σ is completely read, the outcome of the process is determined by whether the last state q in which N has come to a halt to belongs to F or not. In the first case, the word σ is said to belong to the language $L(N)$ recognized by N, while in the latter no statement about the membership of σ to $L(N)$ can be made, for another possible sequence of moves for N and σ could exist that leads to a state in F. This is also true for when a sequence of moves does not exhaust the symbols in σ but leads to a state from which no more moves are possible.

The transition function δ of an NFA N can be extended by the function $\hat{\delta} : Q \times 2^{\Sigma} \to 2^{Q}$, that generalizes the behavior of δ from single symbols in Σ to whole strings from 2^{Σ}. More precisely, if $\delta(q, s)$ denotes the set of states reachable from state q by reading the symbol $s \in \Sigma$, then $\hat{\delta}(q, \sigma)$ denotes the set of states reachable from q by successively reading the string $\sigma \in 2^{\Sigma}$. $\hat{\delta}$ can be defined as:

$$\hat{\delta}(q, \varepsilon) = \{q\},$$
$$\hat{\delta}(q, \sigma_0 \cdots \sigma_{n-1}\sigma_n) = \bigcup_{q' \in \hat{\delta}(q, \sigma_0 \cdots \sigma_{n-1})} \delta(q', \sigma_n).$$

The function $\hat{\delta}$ allows to define the language recognized by an NFA in a more precise way than in the previously stated one.

Definition 2.6 Given an NFA $N = (Q, \Sigma, \delta, Q_0, F)$, the language accepted, or recognized, by N is the set

$$L(N) \stackrel{\text{def}}{=} \left\{\sigma \in 2^{\Sigma} \mid \exists q_0 \in Q_0, \hat{\delta}(q_0, \sigma) \in F\right\}.$$

□

A variant of the NFA discussed so far is obtained by restricting the number of initial states Q_0 to at most one element, and require that $\forall q \in Q, s \in \Sigma, |\delta(q, s) \leq 1|$. An NFA for which these two conditions hold is known as *deterministic finite automata* (DFA).

Intuitively speaking, for each pair of transition state q and alphabet symbol s, a DFA D either allows a single transition from q to another state q', or brings the scanning process of the input word to a halt. The definition of DFA also admits the case of $|Q_0| = 0$, in which case D is understood to have no initial state. The language $L(D)$ recognized by one such DFA is consequently equal to $\emptyset$.

NFAs and DFAs are related by the following result.

Theorem 2.1 For any NFA N, there exists a DFA D such that $L(N) = L(D)$, and vice versa. □

There is a constructive proof of this statement, but the translation procedure from the given NFA to the output DFA that is part of it obtains a number of states for the output DFA that is exponential with respect to the number of states of the given NFA.

Another major result that relates NFAs to regular expressions is the following one.

Theorem 2.2 For any NFA N, there exists a restricted regular expression e such that $L(N) = L(e)$, and vice versa. □

2.3.3. ω-regular languages and Büchi automata

The preceding paragraphs have discussed finite automata and regular expressions, two formalisms used to describe languages of words of finite length, known as regular languages. The present work, however, will also necessitate a formalism to operate with words of infinite length. One of these formalisms, as will be seen, are ω-regular expressions, an

analogue of regular expressions to words of infinite length. Büchi automata will also be introduced as an analogue of NFAs to words of infinite length.

Let then a finite alphabet of symbols Σ be given. An *infinite* word, or *ω-word*, over Σ is a word $\sigma = \sigma_1\sigma_2\sigma_3\cdots$ of infinite length for which $\forall i, \sigma_i \in \Sigma$. For instance, for $\Sigma = \{a, b\}$, $\sigma = \text{ababab} \cdots$ is an ω-word. A language of infinite words is called *ω-language*. Σ^ω denotes the set of ω-words over Σ, and $\Sigma^\infty \stackrel{\text{def}}{=} \Sigma^* \cup \Sigma^\omega$.

For a language $L \subseteq \Sigma^*$, the ω-language generated by L is defined as

$$L^\omega \stackrel{\text{def}}{=} \{w_1w_2w_3\cdots \mid w_i \in L\}.$$

Definition 2.7 An *ω-regular expression* over an alphabet Σ is a syntactic expression

$$e_1f_1^\omega \mid \cdots \mid e_nf_n^\omega,$$

where e_i and f_i are restricted regular expressions over Σ and $\forall i \in \{1, ..., n\}, \varepsilon \notin L(f_i)$. □

The syntax of ω-regular expressions is very similar to that of regular expressions, except that it is now possible to construct expressions of the form e^ω, where e is a regular expression for which $\varepsilon \notin L(e)$.

The semantic of ω-regular expressions stays the same as in the case of regular expressions, with the difference that for a regular expression e with $\varepsilon \notin L(e)$

$$L(e^\omega) \stackrel{\text{def}}{=} \{w_1w_2w_3\cdots \mid w_i \in L(e)\}.$$

The requirement that $\varepsilon \notin L(e)$ is due to the fact that $L(e^\omega)$ is supposed to be an ω-language, something which it would fail to be if $\varepsilon \in L(e)$, for then ε, a word of finite length, would belong to $L(e^\omega)$.

An example of an ω-regular expression over the alphabet $\Sigma = \{a, b\}$ is $e = (A \mid B)^*B^\omega$, which describes the language $L(e)$ of infinite words over $\{a, b\}$ containing a finite number of occurrences of a. A language L is said to be ω-regular if there exists an ω-regular expression e for which $L = L(e)$. ω-regular languages are closed under union, intersection and complementation.

If ω-regular expressions are the counterpart of regular expressions for words of infinite length, then Büchi automata are the counterpart of finite automata for words of infinite length. As in the case of finite automata, there exists a nondeterministic and a deterministic variant of Büchi automata, and also another variant known as *generalized Büchi automata*. Nondeterministic Büchi automata are introduced first.

Definition 2.8 A *nondeterministic Büchi automaton* (NBA) is a tuple $B = (Q, \Sigma, \delta, Q_0, F)$, where

- Q is a finite set of states;
- Σ is an alphabet;
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function;
- $Q_0 \subseteq Q$ is the set of initial states, and
- $F \subseteq Q$ is the set of accept or final states.

□

NBAs are thus defined exactly as in the case of NFAs, and the difference between the two lies in their semantics. For a predicate $P(n)$ over a variable n, the abbreviation $\exists^\omega n\, P(n)$ is used to express that there exist infinitely many n for which $P(n)$. For an ω-word σ over an alphabet Σ, $\text{In}(\sigma) \stackrel{\text{def}}{=} \{a \in \Sigma \mid \exists^\omega i\, \sigma_i = a\}$. In other words, $\text{In}(\sigma)$ is the set of symbols from Σ that appear infinitely often in σ.

Given an NBA B, a computation of B for an ω-word σ over Σ is an ω-word ρ over the alphabet Q such that $\rho_0 \in Q_0$ and, for all $i \geq 1$, $\rho_i \in \delta(\rho_{i-1}, \sigma_{i-1})$. A computation ρ is said to be accepting if $\text{In}(\rho) \cap F \neq \emptyset$, and an ω-word σ over Σ is said to be accepted if there exists a corresponding accepting computation for it. Stated in words, this is equivalent to saying that σ is accepted by B if there exists a computation ρ for it that visits any of the accept states of B infinitely often. For instance, for the Büchi automaton B of Figure 6, the ω-word $\sigma_1 = \text{abbbca}(b^\omega)$ is accepted by the computation $\rho_1 = q_0q_1q_2q_2q_0q_1(q_2^\omega)$. Similarly, $\sigma_2 = (\text{abc})^\omega$ is also a word in $L(B)$, because the computation $\rho = (q_0q_1q_2)^\omega$ that corresponds to it visits q_2 infinitely often. However, any ω-word of the form ab^*dd^ω does not belong to $L(B)$, because for any computation corresponding to a word of this form the sole available accept state q_2 is visited only a finite number of times.

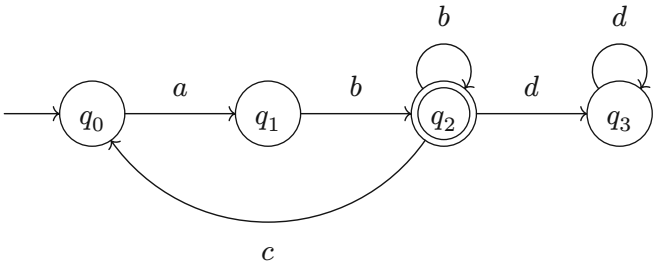


Figure 6: An example Büchi automaton.

In analogy to the case of regular languages and finite automata, the following result holds.

Theorem 2.3 For every ω-regular expression e there exists an NBA N such that $L(e) = L(N)$, and vice versa. □

As anticipated, two variants of NBAs, deterministic Büchi automata and generalized Büchi automata, are also given. These two types of automata are captured by the following definitions.

Definition 2.9 A *deterministic Büchi automaton* (DBA) is an NBA $B = (Q, \Sigma, \delta, Q_0, F)$ where $|Q_0| \leq 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and $\sigma \in \Sigma$. □

As in the case of finite automata, a NBA D for which $|Q_0| = 0$ corresponds to an automaton having no initial states, and whose language of accepted ω-words is consequently $L(D) = \emptyset$. Contrasting with the case of finite automata, however, DBAs are not as expressive as NBAs are.

Theorem 2.4 There does not exist a DBA D over the alphabet $\Sigma = \{a, b\}$ such that $L(D) = L((a \mid b)^*b^\omega)$. □

Proof See C. Baier and J.-P. Katoen [62, p. 190]. ■

Definition 2.10 A *generalized Büchi automaton* (GBA) is a tuple $B = (Q, \Sigma, \delta, Q_0, F)$, where Q, Σ, δ and Q_0 are as in the case of NBAs, and where $F \subseteq 2^Q$. □

A GBA is therefore an NBA where the set of acceptance states has been replaced by a set of sets of acceptance states. Consequently, the acceptance condition of a GBA shifts from requiring any of the accept states of F to be visited infinitely often to requiring each of the sets $E \in F$ to have any of their states visited infinitely often. Formally, a computation ρ over Q is said to be accepting for a GBA G if $\forall E \in F$, $\exists^\infty i \in \mathbb{N}$ such that $\rho_i \in E$, where $\exists^\infty i \in \mathbb{N}$ means that an infinite number of $i \in \mathbb{N}$ exist. In contrast to DBAs, a GBA G for

which $F = \emptyset$ is an automaton in which all computations are accepting, because the $\forall E \in F, \exists^{\infty} i \in \mathbb{N}$ condition is vacuously satisfied. For such a GBA, therefore, $L(G) = \Sigma^{\omega}$.

2.4. Answer set programming

Answer set programming has been informally introduced in Chapter 1, but no conceptual overview of it has been given so far. This section shall serve that purpose. Other explanations can be found in V. Lifschitz [2], which gives a very short introduction, or in T. Eiter, G. Ianni, and T. Krennwallner [3], which provides an introduction suitable to study ASP for the first time. A textbook-length coverage is given by C. Baral [4].

ASP is a programming language paradigm suited, among other things, for the resolution of CSPs and KR tasks. The term "answer set programming" first appeared in 1999 in a collection of two papers (V. Lifschitz [2]), but uses of it before the term was coined were already given by T. Soininen and I. Niemelä [63] in 1998.

Although ASP is not, strictly speaking, a logic, it comes with its own syntax and semantics. The syntax of ASP depends on which of its extensions is considered. In its most basic form, *classical ASP*, the language of ASP is formed by expressions of the form

$$H(x_0) \leftarrow P_1(x_1), ..., P_n(x_n)., \tag{1}$$

where $H, P_1, ..., P_n$ are atomic predicate symbols and $x_0, x_1, ..., x_n$ list of constant or variable symbols. The reading of such an expression, called *classical rule*, is that if $P_1(x_1), ..., P_n(x_n)$ are accepted propositions, then $H(x_0)$ also has to be an accepted proposition. If $n = 0$, then $H(x_0) \leftarrow .$ is simply written as $H(x_0).$ and $H(x_0).$ is considered to be unconditionally asserted. A rule of this form is called *fact*.

Definition 2.11 A *classical logic program* (CLP) is a set of classical rules. □

Starting from classical ASP, Expression 1 can be extended in several ways. One first way of doing so is to add *default negation* to it, expressed by the operator not, in such a way that it now becomes

$$H(x_0) \leftarrow P_1(x_1), ..., P_n(x_n), \text{not } P_{n+1}(x_{n+1}), ..., \text{not } P_m(x_m)., \tag{2}$$

where, as before, $H, P_1, ..., P_n, ..., P_m$ are atomic predicate symbols and $x_0, x_1, ..., x_n, ..., x_m$ list of constant or variable symbols. A rule of this form, called *normal rule*, expresses the fact that if $P_1(x_1), ..., P_n(x_n)$ are all accepted propositions and there is no explicit evidence for the acceptance of $P_{n+1}(x_{n+1}), ..., P_m(x_m)$, then $H(x_0)$ should also be an accepted proposition.

Definition 2.12 A *normal logic program* (NLP) is a set of normal rules. □

Another way of extending ASP along the same direction is to add a second form of negation, *strong negation*, to NLPs. This form of negation is denoted by the operator ¬ and corresponds to the notion of negation from classical logic. By adding strong negation to NLPs, Expression 2 becomes

$$H(x_0) \leftarrow P_1(x_1), ..., P_n(x_n), \text{not } P_{n+1}(x_{n+1}), ..., \text{not } P_m(x_m)., \tag{3}$$

where, this time, $H, P_1, ..., P_n, ..., P_m$ are atomic or strongly negated atomic predicate symbols, referred to as literals for short, while $x_0, x_1, ..., x_n, ..., x_m$ continue to be list of constant or variable symbols. A rule enriched with strongly negated atoms is referred to as *extended rule*.

Definition 2.13 An *extended logic program* (ELP) is a set of extended rules. □

Before being able to evaluate the semantics of an ASP program, a process of variable elimination, termed *grounding*, has to be applied to it. Given a program Π, the process of grounding turns Π into a logically equivalent program Π' where all variable symbols have been replaced by constants terms from a so called *Herbrand universe*.

Definition 2.14 *(Herbrand universe.)* Given a set C of constant symbols and a set F of function symbols, the *Hebrand universe* is the smallest set HU for which

- $\forall c \in C, c \in \text{HU}$;
- If $f \in F$ is a function symbol of arity k and $c_1, ..., c_k \in \text{HU}$, then $f(c_1, ..., c_k) \in \text{HU}$.

□

If no Herbrand universe HU is specified before grounding a program Π, then HU is taken to be the Herbrand universe constructed from the constant and function symbols found in Π. The following example should help clarify these ideas.

Example 2.15 Consider the following CLP Π:

$$\begin{aligned} &\text{Eats}(X, Y) \leftarrow \text{Hungry}(X), \text{Food}(Y). \\ &\text{Hungry}(\text{Joe}). \\ &\text{Hungry}(\text{Barbara}). \\ &\text{Food}(\text{Banana}). \\ &\text{Food}(\text{Milk}). \\ &\text{Food}(\text{Meat}). \end{aligned} \tag{4}$$

The Herbrand universe HU determined by $C = \{\text{Joe}, \text{Barbara}, \text{Banana}, \text{Milk}, \text{Meat}\}$ and $F = \emptyset$ is given by

$$\text{HU} = C = \{\text{Joe}, \text{Barbara}, \text{Banana}, \text{Milk}, \text{Meat}\}. \tag{5}$$

When grounded over HU, Π is equivalent to

$$\begin{aligned} &\text{Eats}(\text{Joe}, \text{Banana}) \leftarrow \text{Hungry}(\text{Joe}), \text{Food}(\text{Banana}). \\ &\text{Eats}(\text{Joe}, \text{Milk}) \leftarrow \text{Hungry}(\text{Joe}), \text{Food}(\text{Milk}). \\ &\text{Eats}(\text{Joe}, \text{Meat}) \leftarrow \text{Hungry}(\text{Joe}), \text{Food}(\text{Meat}). \\ &\text{Eats}(\text{Barbara}, \text{Banana}) \leftarrow \text{Hungry}(\text{Barbara}), \text{Food}(\text{Banana}). \\ &\text{Eats}(\text{Barbara}, \text{Milk}) \leftarrow \text{Hungry}(\text{Barbara}), \text{Food}(\text{Milk}). \\ &\text{Eats}(\text{Barbara}, \text{Meat}) \leftarrow \text{Hungry}(\text{Barbara}), \text{Food}(\text{Meat}). \\ &\text{Hungry}(\text{Joe}). \\ &\text{Hungry}(\text{Barbara}). \\ &\text{Food}(\text{Banana}). \\ &\text{Food}(\text{Milk}). \\ &\text{Food}(\text{Meat}). \end{aligned} \tag{6}$$

□

A rule r with no variable terms is said to be a *ground rule*, and a program Π made of only ground rules is, similarly, said to be a *ground program*. All the rules of Expression 6 are ground, so the program Π' that they form is, consequently, a ground program.

When predicates are combined with constant terms from a Herbrand universe HU, then something going under the name of *Herbrand base* is obtained.

Definition 2.16 *(Herbrand base.)* Given a set T of predicate symbols and a Herbrand universe HU, the *Herbrand base* is the smallest set HB for which if $P \in T$ is a predicate symbol of arity k and $t_1, ..., t_k \in \text{HU}$, then $P(t_1, ..., t_k) \in \text{HB}$. □

Example 2.17 The Herbrand base determined from the Herbrand universe from Example 2.15 and the set of predicate symbols $T = \{\text{Hungry}, \text{Food}, \text{Eats}\}$ is given by

$$\begin{aligned}\text{HB} = \{&\text{Hungry(Joe)}, \text{Hungry(Barbara)}, \text{Food(Banana)}, \text{Food(Milk)},\\ &\text{Food(Meat)}, \text{Eats(Joe, Banana)}, \text{Eats(Joe, Milk)}, \text{Eats(Joe, Meat)},\\ &\text{Eats(Barbara, Banana)}, \text{Eats(Barbara, Milk)}, ...\}.\end{aligned} \tag{7}$$

□

The notion of semantics for ASP can, at this point, be introduced.

For a CLP Π, an interpretation I is just a subset $I \subseteq \text{HB}$ of the Herbrand base induced by the predicate, constant and function symbols found in Π.

I is a model of a ground rule $r = H(t_0) \leftarrow P_1(t_1), ..., P_n(t_n).$ if it is *closed* with respect to it, that is if $\{P_1(t_1), ..., P_n(t_n)\} \subseteq I$ implies $H(t_0) \in I$, and it is a model of a generic, not necessarily ground, rule $r = H(x_0) \leftarrow P_1(x_1), ..., P_n(x_n).$ if it is a model of all its ground instances over a previously specified Herbrand universe HU. Finally, I is a model of a program Π if it is a model of all of its rules, ground or otherwise.

In the case of CLPs, the interest lies in determining not just the set of models of a CLP Π, but in finding the subset-minimal ones. It turns out, in fact, that for a CLP Π, only one such model is given, referred to as the *least model* of Π.

For NLPs things are more complicated, for the introduction of weak negation leads to the possibility of a NLP Π to possess more than one minimal model, now referred to as *stable model*. To introduce the notion of stable model, the notion of Gelfond-Lifschitz reduct needs to be given first.

Definition 2.18 *(Gelfond-Lifschitz reduct.)* Given a ground NLP Π and a set of atoms Λ, the *Gelfond-Lifschitz reduct*, or simply *reduct*, of Π with respect to Λ is the program

$$\begin{aligned}\Pi^\Lambda \stackrel{\text{def}}{=} \{H \leftarrow P_1, ..., P_n \mid &H \leftarrow P_1, ..., P_n, \text{not } P_{n+1}, ..., \text{not } P_m \in \Pi,\\ &\{P_{n+1}, ..., P_m\} \cap \Lambda = \emptyset\}.\end{aligned} \tag{8}$$

□

Intuitively, given a ground normal logic program Π and a set of atoms Λ, the program Π^Λ is the program formed by discarding all rules r in Π that have at least one weakly negated atom present in Λ, and by removing all weakly negated atoms from all remaining rules. Π^Λ is a CLP, because no weak negation appears in it.

Definition 2.19 Given an NLP Π, a set of atoms Λ is called *stable model* of Π if it is the least model of Π^Λ. □

For an ELP, an interpretation I is no longer restricted to be a set of ground atoms from a given Herbrand base HB, but it can also include strongly negated literals. Minimal models for ELPs are now referred to as *answer sets*.

Definition 2.20 Given an ELP Π, a set of literals Λ is called *answer set* of Π if it is the least model of Π^Λ. □

One final extension of NLPs is also worth mentioning.

Definition 2.21 A *disjunctive logic program* (DLP) is an NLP where each rule is of the form

$$H_1(z_1) \vee \cdots \vee H_r(z_r) \leftarrow P_1(x_1), ..., P(x_n), \text{not } P(x_{n+1}), ..., \text{not } P(x_m)., \tag{9}$$

where each $H_1(z_1), ..., H_r(z_r), P_1(x_1), ..., P_n(x_n), ..., P_m(x_m)$ is an atom, and $z_1, ..., z_r, x_1, ..., x_n, ..., x_m$ list of variable or constant symbols. □

Simply put, DLPs are NLPs where the head term is now replaced by a disjunction of head terms. The Gelfond-Lifschitz reduct Π^Λ of a DLP Π is defined analogously as it is defined for NLPs. In particular, the disjunction of terms at the head of a rule is retained in the reduct Π^Λ.

Definition 2.22 Given a DLP Π, a set of atoms Λ is called *stable model* of Π if it is the least model of Π^Λ. □

Replacing the head term of a rule with a disjunction of head terms is an extension that can be also performed to ELPs.

Definition 2.23 An *extended disjunctive logic program* (EDLP) is an ELP where each rule is of the form

$$H_1(z_1) \vee \cdots \vee H_r(z_r) \leftarrow P_1(x_1), ..., P(x_n), \text{not } P(x_{n+1}), ..., \text{not } P(x_m)., \tag{10}$$

where each $H_1(z_1), ..., H_r(z_r), P_1(x_1), ..., P_n(x_n), ..., P_m(x_m)$ is an atom or strongly negated atom, and $z_1, ..., z_r, x_1, ..., x_n, ..., x_m$ list of variable or constant symbols. □

The notion of answer sets of EDLPs is analogous to that of DLPs, differing only by the fact that sets of literals, and not just sets of atoms, are considered this time.

Definition 2.24 Given an EDLP Π, a set of literals Λ is called *answer set* of Π if it is the least model of Π^Λ. □

DLPs will be considered in Chapter 3 and will be at the basis of some of the computational results derived there.

2.5. Equilibrium logic

Equilibrium logic (D. Pearce [49]) is a logical characterization of ASP based on the logic of HT (A. Heyting [51]), a restriction of Heyting intuitionistic logic H under the Kripke semantics (S. A. Kripke [64], D. Van Dalen [65, Section 6.3]) where only one frame, composed of just two worlds, the *here* world and the *there* world, is considered. EL shares the same syntax as HT[7], but it adds a model selection criterion based on the minimization of literals to the semantics of HT.

The syntax of Heyting propositional intuitionistic logic H is based on a starting set A of atomic symbols and the operators $\vee, \wedge$ and $\rightarrow$.

Definition 2.25 Given a set of atomic symbols A, the language of H is the smallest set L for which:

1. $\bot \in L$;
2. If $a \in A$, then $a \in L$;
3. If $\varphi_1, \varphi_2 \in L$, then $\varphi_1 \odot \varphi_2 \in L$, where $\odot \in \{\vee, \wedge, \rightarrow\}$.

□

[7] In the formulation of D. Pearce [49], an additional *strong negation* operator is added to the syntax of HT. For reasons of simplicity, this type of negation is not considered in this work.

On the basis of the primitive operators, a few derived ones can be also defined. The $\neg$ operator, defined as $\neg a \stackrel{\text{def}}{=} a \rightarrow \bot$, is the *weak negation* operator, whereas $\top$, defined as $\top \stackrel{\text{def}}{=} \neg\bot$, is the *verum* constant.

For a formula φ of H, an interpretation under the Kripke semantics is a triple $I = (W, \leq, V)$ such that $F = (W, \leq)$ is a *frame* composed of two elements, a set of possible worlds W and a reflexive and transitive accessibility relation $\leq \subseteq W \times W$. V is a function $V : W \rightarrow 2^A$ assigning to each possible world $w \in W$ a set of atomic propositions from A and satisfying the *monotonic* or *persistence* property, asserting that if $w \leq w'$ and $a \in V(w)$, then also $a \in V(w')$. An atomic proposition $a \in A$ for which $a \in V(w)$ is said to be *true* at world w, and *false* otherwise. A satisfaction relation $\vDash$ is defined that extends the notion of truth of atomic propositions from A at a world w to composite formulas φ.

Definition 2.26 The satisfaction relation $\vDash$ between worlds $w \in W$ and H formulas φ is a relation such that:

1. $w \nvDash \bot$;
2. $w \vDash a$, for $a \in A$, if $a \in V(w)$;
3. $w \vDash \varphi_1 \vee \varphi_2$ if $w \vDash \varphi_1$ or $w \vDash \varphi_2$;
4. $w \vDash \varphi_1 \wedge \varphi_2$ if $w \vDash \varphi_1$ and $w \vDash \varphi_2$;
5. $w \vDash \varphi_1 \rightarrow \varphi_2$ if for all w' such that $w \leq w'$, if $w' \vDash \varphi_1$ then $w' \vDash \varphi_2$.

□

A formula φ for which $w \vDash \varphi$ is said to be *true* at world w, and it is said *false* otherwise, denoted by $w \nvDash \varphi$. "False" has to be taken in this case as lacking sufficient information to conclude φ at world w.

The semantics of the $\vee$ and $\wedge$ operators is analogous to that of classical logic, but that of $\rightarrow$, in the case of Kripke semantics, differs. Stated in words, an implication $\varphi_1 \rightarrow \varphi_2$ holds at a world w if, for each world w' reachable from w, if $w' \vDash \varphi_1$ then also $w' \vDash \varphi_2$. For the weak negation operator, it holds that $w \vDash \neg a$ if for all w', $w \leq w'$, $w' \nvDash a$, whereas for the $\top$ constant, $w \vDash \top$ always holds. The monotonicity of V that was previously supposed for atomic propositions carries over to composite formulas, in the sense that if $w \leq w'$ and $w \vDash \varphi$, then also $w' \vDash \varphi$. Finally, a formula φ is satisfied by an interpretation I, written $I \vDash \varphi$, if $\forall w \in W, w \vDash \varphi$.

The syntax and the semantics of HT is the same as that of H, with the sole exception that the only admissible frame is the frame $F = (\{h, t\}, \leq)$ where $h \leq t$ or, in more explicit terms

$$\leq = \{(h, h), (h, t), (t, t)\}.$$

An interpretation I of an HT formula is consequently a triple $I = (\{h, t\}, \leq, V)$, and φ is satisfied at I if $h \vDash \varphi$.

Definition 2.27 Given a set of atomic symbols A, the language of HT is the smallest set L for which:

1. $\bot \in A$;
2. If $a \in A$, then $a \in L$;
3. If $\varphi_1, \varphi_2 \in L$, then $\varphi_1 \odot \varphi_2 \in L$, where $\odot \in \{\vee, \wedge, \rightarrow\}$.

□

Proposition 2.5 An HT formula φ is satisfied at an interpretation I iff $h \vDash \varphi$. □

Proof If $I \vDash \varphi$, then $\forall w \in \{h,t\}, w \vDash \varphi$, including the special case of $h \vDash \varphi$. Conversely, if $h \vDash \varphi$, then given that $h \leq t$ and $\vDash$ is monotonic over HT formulas, $t \vDash \varphi$. Therefore $\forall w \in \{h,t\}, w \vDash \varphi$ and, by definition, $I \vDash \varphi$. ■

By virtue of its single frame assumption, an HT interpretation I can be also seen as a pair (H,T), where $H = \{a \in A \mid a \in V(h)\}$ is the set of atomic propositions holding on h, and $T = \{a \in A \mid a \in V(t)\}$ is the set of propositions holding on t. An HT interpretation $I = (H,T)$ is *total* if $H = T$, that is the set of atomic propositions holding on h is the same as the set of atomic propositions holding on t, and it is said to be *h-minimal* if for any other HT interpretation $I' = (H',T)$, $H \subseteq H'$.

The notions thus introduced allow the definition of equilibrium model to be given.

Definition 2.28 An HT interpretation $I = (H,T)$ is an *equilibrium model* (EM) of an HT formula φ if

1. $I \vDash \varphi$;
2. I is total;
3. No $H' \subsetneq T$ exists for which $(H',T) \vDash \varphi$.

□

Formulas in HT have a syntactical correspondence to the DLPs seen in Section 2.4. Given that HT has been defined only for the propositional case, this correspondence will be described with reference to ground rules only, where each ground predicate $P(t_1,...,t_r)$ is denoted by a single symbol P for simplicity.

Under this setting, a DLP rule

$$h_1 \vee \cdots \vee h_r \leftarrow P_1,...,P_n, \text{not } P_{n+1},...,\text{not } P_m.$$

where each h_i and each P_i is a positive atom, corresponds to an HT formula

$$P_1 \wedge \cdots \wedge P_n \wedge \neg P_{n+1} \wedge \cdots \wedge \neg P_m \rightarrow h_1 \vee \cdots \vee h_r.$$

The notion of equilibrium model is at this point used to capture the notion of answer set from a logical perspective.

Proposition 2.6 (D. Pearce [49, Proposition 10].) *Let* Π *be a logic program. A consistent set* H *of literals is an answer set of* Π *if and only if* H *is the set of literals true in some equilibrium model of* Π. □

This correspondence also holds for the case of non-ground logic programs and predicate HT formulas, but for succinctness of exposition the first-order definition of HT and the correspondence to non-ground programs have been omitted from this section[8].

2.6. Temporal equilibrium logic

TEL has already been briefly touched on in Chapter 1, but, as in the case of ASP, no technical overview of it has been given so far. This section will serve this purpose. Some knowledge of temporal logic, more precisely LTL, as found in C. Baier and J.-P. Katoen [62], is assumed throughout the section.

[8]The correspondence also holds for generalized class of logic programs, including *generalised disjunctive programs*, where rules take on the form $H_1 \vee \cdots \vee H_r \vee \text{not } H_{r+1} \vee \cdots \vee \text{not } H_s \leftarrow P_1,...,P_n, \text{not } P_{n+1},...,\text{not } P_m$, and logic programs with nested rules.

As already anticipated in the first few paragraphs of Chapter 1 and schematically depicted in Figure 3, TEL is a combination of the temporal here-and-there logic, THT, and EL, already described in Section 2.5. TEL and THT share indeed the same syntax, and the only difference between the two is that TEL is a nonmonotonic extension of THT, in the sense that TEL adds a model selection criterion to the semantics of THT, very similar, if not conceptually the same as, the one that EL adds to HT.

The syntax of THT enjoys different formulations. The one chosen here is an attempt to make use of as little temporal operators as possible, and express the remaining ones as a rewriting of the former. To keep the exposition succinct, only the propositional case of the syntax of THT will be presented.

Definition 2.29 *(THT syntax.)* Given a countable set A of atomic symbols, the language of THT is the smallest set L such that

1. $\bot \in L$;
2. If $a \in A$, then $a \in L$;
3. If $\varphi_1, \varphi_2 \in L$, then $\varphi_1 \odot \varphi_2 \in L$, where $\odot \in \{\vee, \wedge, \rightarrow\}$;
4. If $\varphi \in L$, then $\bigcirc\varphi \in L$;
5. If $\varphi_1, \varphi_2 \in L$, then $\varphi_1 \mathbb{U} \varphi_2 \in L$;
6. If $\varphi_1, \varphi_2 \in L$, then $\varphi_1 \mathbb{R} \varphi_2 \in L$.

□

In Definition 2.29, $\bigcirc$ is the *next* operator, and in the intended semantics a formula of type $\bigcirc\varphi$ can be intuitively interpreted as "next time, φ." $\mathbb{U}$, is the *until* operator, and $\varphi_1 \mathbb{U} \varphi_2$ means that φ_2 holds at a certain point, and that φ_1 holds until then. Finally, $\mathbb{R}$ is the *release* operator, and $\varphi_1 \mathbb{R} \varphi_2$ holds if φ_2 is true until the point where φ_1 first becomes true, or if φ_2 is true forever.

An LTL interpretation is a *linear trace*, or just *trace* for short, corresponding to a finite or infinite sequence T of sets of atomic symbols. In THT, instead, an interpretation is a pair of traces of the form $M = (H, T)$, such that $\forall i, H_i \subseteq T_i$ and $|H| = |T|$. The length of M, $|M|$, is defined as $|M| \stackrel{\text{def}}{=} |H| = |T|$. If $|M| \in \mathbb{N}$, then M is said to be finite, otherwise it is said infinite, in which case $|M| = \omega$ is written. By M_i the pair (H_i, T_i) is meant, representing a single HT interpretation.

Example 2.30 If

$$T_1 = \{a\} \cdot \emptyset \cdot \{a\} \cdot \emptyset,$$
$$T_2 = (\{a\} \cdot \emptyset)^\omega = \{a\} \cdot \emptyset \cdot \{a\} \cdot \emptyset \cdot \cdots,$$

then T_1 is a finite LTL trace of length $|T_1| = 4$, while T_2 is an infinite LTL trace. □

The expression $(\{a\} \cdot \emptyset)^\omega$ used in Example 2.30 to express T_2 is referred to as *ω-regular expression*.

Example 2.31 If

$$M = (H, T) = (\{a\} \cdot \emptyset \cdot \{b\}, \{a, b\} \cdot \{a\} \cdot \{b\}),$$

then M is a finite THT trace of length $|M| = 3$. □

Definition 2.32 *(THT semantics.)* A THT interpretation $M = (H, T)$ of length λ satisfies a THT formula φ at time i, written $M, i \vDash \varphi$, if the following rules are respected:

1. $M, i \not\models \bot$;
2. $M, i \models a$ if $a \in H_i$;
3. $M, i \models \varphi_1 \vee \varphi_2$ if $M, i \models \varphi_1$ or $M, i \models \varphi_2$;
4. $M, i \models \varphi_1 \wedge \varphi_2$ if $M, i \models \varphi_1$ and $M, i \models \varphi_2$;
5. $M, i \models \varphi_1 \rightarrow \varphi_2$ if:
   - $(H, T), i \models \varphi_1$ implies[9] $(H, T), i \models \varphi_2$, and
   - $(T, T), i \models \varphi_1$ implies $(T, T), i \models \varphi_2$;
6. $M, i \models \bigcirc\varphi$ if $i < \lambda$ and $M, i + 1 \models \varphi$;
7. $M, i \models \varphi_1 \mathbb{U} \varphi_2$ if
   - $\exists j \in [i..\lambda)$ such that $M, j \models \varphi_2$, and
   - $\forall j' \in [i..j), M, j' \models \varphi_1$;
8. $M, i \models \varphi_1 \mathbb{R} \varphi_2$ if
   - $\forall j \geq i, M, j \models \varphi_2$, or
   - $\exists k \in [i..\lambda)$ such that $M, k \models \varphi_1$ and $\forall k' \in [i..k], M, k' \models \varphi_2$.

□

On the basis of the operators from Definition 2.29, the following new operators can be defined:

$$\begin{aligned}
\neg\varphi &\stackrel{\text{def}}{\equiv} \varphi \rightarrow \bot; \\
\top &\stackrel{\text{def}}{\equiv} \neg\bot; \\
\varphi_1 \leftrightarrow \varphi_2 &\stackrel{\text{def}}{\equiv} (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1); \\
\Box\varphi &\stackrel{\text{def}}{\equiv} \bot \mathbb{R} \varphi; \\
\Diamond\varphi &\stackrel{\text{def}}{\equiv} \top \mathbb{U} \varphi; \\
\mathbb{F} &\stackrel{\text{def}}{\equiv} \neg\bigcirc\top; \\
\widehat{\bigcirc}\varphi &\stackrel{\text{def}}{\equiv} \bigcirc\varphi \vee \mathbb{F}.
\end{aligned}$$

$\Box$ and $\Diamond$ are the *always* and the *eventually* operators, in some cases also written as G and F, while $\mathbb{F}$ is the *final* operator, of arity 0, for which $M, i \models \mathbb{F}$ if $|M| \in \mathbb{N}$ and $i = |M|$, or in other words if M is of finite size and i is its last state. $\widehat{\bigcirc}$ is also know as the *weak next* operator. In infinite THT traces, $\widehat{\bigcirc}\varphi \equiv \bigcirc\varphi$.

Similarly to the case of HT interpretations, given two LTL interpretations M and N of length $\lambda \in \mathbb{N} \cup \{\omega\}$, M is said to be *less than or equal to* N, written $M \leq N$, if $\forall i \in [0..\lambda], M_i \subseteq N_i$. M is *strictly less than* N, written $M < N$, if $M \leq N$ and $\exists i \mid M_i \subsetneq N_i$. A THT interpretation $M = (H, T)$ is called *total* if $H = T$. These definitions allow the central notion of temporal equilibrium model to be introduced.

Definition 2.33 A THT interpretation M is a *temporal equilibrium model* (TEM) of a THT formula φ if:

1. $M, 0 \models \varphi$;
2. $M = (T, T)$ is total;
3. $\nexists H < T$ for which $(H, T), 0 \models \varphi$.

□

Given that a TEM $M = (T, T)$ is a total trace, M can be regarded just as the single LTL trace T, on which an additional minimization criterion is enforced. For total THT interpretations, the following holds.

[9] The metalogic under which *implies* is here understood is classical logic.

Proposition 2.7 Given an LTL trace T of length λ, a THT formula φ and a time point $k \in [0..\lambda)$

$$T, k \vDash_{\text{LTL}} \varphi \Leftrightarrow (T, T), k \vDash_{\text{THT}} \varphi.$$

□

A few examples might help highlight the differences between ordinary THT interpretations and TEMs.

Example 2.34 *(P. Cabalar and G. Pérez Vega [66].)* Given the THT formula $\varphi = \Diamond p$:

1. The LTL models of φ are the LTL traces H such that $p \in H_i$ for some $i \in [0..\lambda)$:
$$\begin{aligned} H_1 &= \emptyset^* \cdot \{p\} \cdot \emptyset^*, \\ H_2 &= \emptyset^* \cdot \{p\} \cdot \{p\} \cdot \emptyset^*, \\ H_3 &= \emptyset^* \cdot \{p\} \cdot \emptyset^* \cdot \emptyset^*, \\ &\vdots \end{aligned}$$
2. The THT models of φ are the THT traces $M = (H, T)$, where H is any of the LTL traces of point 1. and $H_i \subseteq T_i$;
3. The only TEMs of φ are the THT interpretations $M = (T, T)$ with $T = \emptyset^* \cdot \{p\} \cdot \emptyset^*$.

□

The following example shows how TEMs differ over finite and infinite traces, and that not all THT formulas admit a TEM.

Example 2.35 *(G. De Giacomo and M. Y. Vardi [67].)* Let $\varphi = \Box\Diamond a$, where $\Box\Diamond a$ can be read as "infinitely often a in the future." Given a total interpretation $M = (T, T)$ of φ:

1. If $|M| = \lambda \in \mathbb{N}$, then M is a TEM of φ iff $T, \lambda \vDash a$ and $T, k \nvDash a$ for $0 \leq k < \lambda$;
2. If $|M| = \omega$, then M can *never* be a TEM of φ, for infinitely many occurrences of a in T would allow to construct a trace H such that $|H| = \omega$ and $H < T$.

□

Chapter 3

Model checking over linear traces

"Magic's just science that we don't understand yet."

— Arthur C. Clarke.

Given a trace T, understood as a sequence of atomic propositions, and a temporal formula φ, the problem of model checking for TEL consists of determining whether T is a TEM of φ. This deceptively simple problem, so easy to formulate using just a few words, belies a computational complexity that can easily go beyond the initial expectations of a casual reader. In its finite-trace form, TELMC$_f$, the problem turns out to be "just" coNP–complete, while in the infinite-trace one, where the traces are supposed to have infinite length, the complexity of the problem raises up to PSPACE–completeness.

The chapter investigates the complexity of model checking for TEL by starting with model checking of the atemporal version of TEL, EL. After that, the complexity of TELMC$_f$ and TELMC, respectively, is determined. Table 5 presents a summary of the results of the chapter.

Problem	Complexity
ELMC	coNP–complete (Theorem 3.13)
TELMC$_f$	coNP–complete (Theorem 3.17)
TELMC	PSPACE–complete (Theorem 3.23)

Table 5: Summary of the results of the chapter.

3.1. Atemporal model checking

Given a set of atoms Λ and an EL formula φ, the ELMC problem is the problem of determining whether Λ is an equilibrium model of φ.

DEFINITION 3.36 *(ELMC)* The problem of model checking for EL is the language

$$\text{ELMC} \stackrel{\text{def}}{=} \{(\Lambda, \varphi) \mid \Lambda \text{ is an equilibrium model of } \varphi\},$$

where Λ is a set of atoms and φ an EL formula. □

The definition of ELMC reflects the conventions established in Section 2.2, particularly Definition 2.1. ELMC is therefore not just a language, but also a decision problem, subject to decision procedures and Karp reductions. Its host language, Σ_{ELMC}, is defined as

$$\Sigma_{\text{ELMC}} \stackrel{\text{def}}{=} \{(\Lambda, \varphi)\},$$

where Λ and φ are a set of atoms and an EL formula, irrespective of whether Λ is an equilibrium model of φ.

Another closely related problem is the problem of *stable model checking*, or Smc for short, asking whether a set of atoms is the stable model of a ground DLP.

Definition 3.37 *(Smc.)* The problem of *stable model checking* is the language

$$\text{Smc} \stackrel{\text{def}}{=} \{(\Lambda, \Pi) \mid \Lambda \text{ is a stable model of } \Pi\},$$

where Λ is a set of atoms and Π a ground DLP. □

A definition of what a DLP is was already given in Definition 2.21. Introducing Smc is useful because it allows studying the complexity of Elmc by studying its own complexity first, and then translate it to Elmc by means of the equivalence between DLPs and EL formulas established in D. Pearce [49]. For Smc, the following holds:

Theorem 3.8 Deciding Smc is coNP–complete. □

Proof Consequence of Proposition 3.9 and Proposition 3.11. ■

Showing the coNP–hardness of Smc involves a reduction from CNF, the problem of deciding whether a SAT formula φ in conjunctive normal form (CNF) is satisfiable. The syntax of this problem considered in this chapter is different than what is normally found in modern literature, and rather corresponds to what is found in T. J. Schaefer [68]. A brief exposition of it is found in M. Cadoli [54], and is reported here for the sake of completeness.

A logical relation R of arity k is any subset of the cartesian product $\{0,1\}^k$. Given a relation symbol R of arity k and k, not necessarily distinct, boolean variables $x_1, ..., x_k$, $R(x_1, ..., x_k)$ denotes a *clause*. A variable assignment to the set of variables V of a clause is a function $M : V \to \{0,1\}$, and M is said to satisfy $R(x_1, ..., x_k)$, written $M \vDash R$, if $(M(x_1), ..., M(x_k)) \in R$. Each variable assignment $M : V \to \{0,1\}$ induces a subset $L \subseteq V$ of V and vice versa, in such a way that $M(x_i) = 1$ iff $x_i \in L$.

Given a set $S = \{R_1, ..., R_n\}$ of logical relations, an S-formula φ on a set of variables V is any conjunction of clauses from S. When not specified beforehand, V is taken to be just the set of variables occurring in φ and is denoted by $V(\varphi)$.

This allows the definition of the SAT(S) problem to be introduced.

Definition 3.38 Given a set S of logical relations

$$\text{SAT}(S) \stackrel{\text{def}}{=} \{\varphi = R_1(x_1) \wedge \cdots \wedge R_m(x_m) \mid \exists L \subseteq V(\varphi), L \vDash \varphi\}.$$

□

A concrete example is given by the set $S = \{T_1, T_2\}$, where $T_1 = \{(0,0,0), (1,1,1)\}$ and $T_2 = \{(0,1,1), (1,0,1), (1,1,0)\}$. Under this setting, $\varphi = T_1(x,y,y) \wedge T_2(x,y,z)$ is an S-formula over the alphabet $V(\varphi) = \{x,y,z\}$, and it is satisfied by the variable assignment $L = \{x,y\}$.

SAT(R_1), where $R_1 = \{(1,0,0), (0,1,0), (0,0,1)\}$, is the set of satisfiable CNF formulas where exactly one variable per clause is required to be true and T. J. Schaefer [68, Lemma 3.5] shows it to be NP–complete. This result forms the basis of the following proposition.

Proposition 3.9 Deciding Smc is coNP–hard. □

Proof To show the coNP–hardness of Smc, the NP–hardness of $\overline{\text{Smc}}$ is shown. This fact, in turn, is shown by reducing the SAT(R_1) problem to $\overline{\text{Smc}}$.

To show that $\mathrm{SAT}(R_1) \leq \overline{\mathrm{SMC}}$, a Karp reduction $f : \Sigma_{\mathrm{SAT}(R_1)} \to \Sigma_{\overline{\mathrm{SMC}}}$ is defined such that $\varphi \in \mathrm{SAT}(R_1)$ iff $f(\varphi) = (\Lambda, \Pi) \in \overline{\mathrm{SMC}}$. Given a formula $\varphi \in \Sigma_{\mathrm{SAT}(R_1)}$ of the form

$$\varphi = R_1(x_1, y_1, z_1) \wedge \cdots \wedge R_1(x_m, y_m, z_m)$$

where x_i, y_i and z_i are not all necessarily distinct variables, and where, again

$$R_1 = \{(1,0,0), (0,1,0), (0,0,1)\},$$

$f(\varphi) = (\Lambda, \Pi)$ is a set of atoms Λ and a ground DLP program Π pair, characterized as follows. Π is defined over the Herbrand universe $\mathrm{HU} = \{x_1, y_1, z_1, ..., x_m, y_m, z_m\}$, and formed by the three subprograms Π_1, Π_2 and Π_3:

$$\Pi_1 = \begin{cases} x_1 \vee y_1 \vee z_1. \\ \quad \vdots \\ x_m \vee y_m \vee z_m., \end{cases}$$

$$\Pi_2 = \begin{cases} x_1 \leftarrow y_1, z_1. \\ y_1 \leftarrow x_1, z_1. \\ z_1 \leftarrow x_1, y_1. \\ \quad \vdots \\ x_m \leftarrow y_m, z_m. \\ y_m \leftarrow x_m, z_m. \\ z_m \leftarrow x_m, y_m., \end{cases}$$

$$\Pi_3 = \begin{cases} x_2 \leftarrow x_1, y_1, z_1. \\ y_2 \leftarrow x_1, y_1, z_1. \\ z_2 \leftarrow x_1, y_1, z_1. \\ \quad \vdots \\ x_1 \leftarrow x_m, y_m, z_m. \\ y_1 \leftarrow x_m, y_m, z_m. \\ z_1 \leftarrow x_m, y_m, z_m. \end{cases}$$

Even if a notation usually reserved for variables has been used above, Π is a ground program. The set of atoms Λ, instead, is set to $\{x_1, y_1, z_1, ..., x_m, y_m, z_m\}$.

These rules can be intuitively explained as follows. Π_1 associates the rule $r = x_1 \vee y_i \vee z_i.$ to each of the clauses $R_1(x_i, y_i, z_i)$ from φ, and says that r is satisfied if either one of x_i, y_i or z_i is "true."[10] Π_2 says that if two variables within the same triple are set to true, then even the third variable has to be set to true. Coupled with the rules of Π_1, this is equivalent to saying that each triple of variables x_i, y_i, z_i either has exactly one variable set to true or all of them set to true. Π_3 finally says that if any triple of variables x_i, y_i, z_i is set to true, then all of the other triples also have to be set to true. This confers the program a sort of "blow up" or saturation behavior—either all triples of variables have exactly one variable set to true, or all variables are set to true.

What remains to be shown at this point is that f is a function from $\Sigma_{\mathrm{SAT}(R_1)}$ to $\Sigma_{\overline{\mathrm{SMC}}}$, that $\varphi \in \mathrm{SAT}(R_1)$ iff $f(\varphi) = (\Lambda, \Pi) \in \overline{\mathrm{SMC}}$ and that f is computable in polynomial time.

The first property is satisfied, for, as seen above, f turns any formula φ into a pair (Λ, Π) composed of a set of atoms Λ and a ground DLP Π.

[10] By abuse of terminology, "true" is said from here on, although for the case of ASP the term "assumed" would be more appropriate.

For the second property, $\varphi \in \mathrm{SAT}(R_1)$ is first assumed and $(\Lambda, \Pi) \in \overline{\textsc{Smc}}$, or equivalently that Λ is *not* an stable model of Π, is proven. Since $\varphi \in \mathrm{SAT}(R_1)$, φ is made true by some variable assignment $L \subseteq \mathrm{Var}(\varphi)$. Consider then the candidate stable model $\Lambda' = L$. Λ' satisfies Π_1, because L satisfies each clause of φ. Λ' also satisfies Π_2, because no rule within Π_2 is closed with respect to Λ'. More precisely, given a clause $R_1(x_i, y_i, z_i)$, three different cases can hypothetically occur:

1. x_i, y_i and z_i are all distinct, in which case the body of the rules
$$x_i \leftarrow y_i, z_i.$$
$$y_i \leftarrow x_i, z_i.$$
$$z_i \leftarrow x_i, y_i.$$
is not contained within Λ', because L sets exactly one of x_i, y_i, z_i to true and the other variables to false.
2. Two variables within x_i, y_i, z_i are equal. If, without loss of generality, $x_i = y_i$, $R_1(x_i, y_i, z_i)$ becomes $R_1(x_i, x_i, z_i)$, and the Π_2 rule associated to it is
$$x_i \leftarrow x_i, z_i.$$
$$x_i \leftarrow x_i, z_i.$$
$$z_i \leftarrow x_i, x_i.$$
In such a case, L needs to have x_i assigned to false and z_i to true, in such a way that no rule above has its body contained in Λ'.
3. All variables within $R_1(x_i, y_i, z_i)$ are equal, that is, $R_1(x_i, y_i, z_i)$ becomes $R_1(x_i, x_i, x_i)$. Given that L satisfies φ, it also has to satisfy $R_1(x_i, x_i, x_i)$, but this is absurd, because x_i would need to be simultaneously set to true and false. Hence not all variables within $R_1(x_i, y_i, z_i)$ can be the same.

Finally, Λ' satisfies Π_3 because, similarly as for Π_2, no rule of Π_3 is closed with respect to Λ'. Therefore, $\Lambda' \vDash \Pi$. Observe at this point that $\Lambda \vDash \Pi$ is also true, and that $\Pi = \Pi^{\Lambda} = \Pi^{\Lambda'}$. Hence Λ is a model of Π^{Λ}, but it is not a minimal one, because $\Lambda' \subsetneq \Lambda$ and $\Lambda' \vDash \Pi$. This proves that $(\Lambda, \Pi) \notin \textsc{Smc}$, or equivalently that $(\Lambda, \Pi) \in \overline{\textsc{Smc}}$.

For the opposite direction, that $\varphi \in \Sigma_{\mathrm{SAT}(R_1)}$ and $f(\varphi) = (\Lambda, \Pi) \in \overline{\textsc{Smc}}$ is supposed. Then $(\Lambda, \Pi) \notin \textsc{Smc}$, meaning that Λ is not a model of $\Pi^{\Lambda} = \Pi$ or that it is a model, but not a minimal one. Clearly, $\Lambda \vDash \Pi$, so the only case left is that Λ is not a minimal model. Let then $\Lambda' \subsetneq \Lambda$ be one such minimal model. Then no rule r of $\Pi_2 \cup \Pi_3$ is closed with respect to Λ', for, if it was, then $\Lambda' = \Lambda$, which would be absurd. Further, $\Lambda' \vDash \Pi_1$. It can at this point be seen that Λ' induces a variable assignment $L = \Lambda' \subseteq \mathrm{Var}(\varphi)$ such that $L \vDash \varphi$. Therefore, $\varphi \in \mathrm{SAT}(R_1)$.

Finally, if the number m of clauses within φ is taken to be the size $|\varphi|$ of φ, f computes $f(\varphi)$ in polynomial, in fact linear, time with respect to m. Π_1 is in fact composed of m rules, while Π_2 and Π_3 by $3m$ each. ■

What remains to be done for the Smc problem is show its membership to coNP.

Proposition 3.10 Given a DLP Π and a set of atoms Λ, computing the Gelfond-Lifschitz reduct Π^{Λ} of Π with respect to Λ can be done in time $O(|\Pi|^2 \cdot |\Lambda|)$, where $|\Pi|$ is the number of symbols in Π. □

```
1 CLP GELFOND-LIFSCHITZ-REDUCT (DLP Π, SET Λ)
2     let Π′ ← ∅
3     foreach H_1 ∨ ··· ∨ H_u ← P_1, ..., P_n, not P_{n+1}, ..., not P_m. ∈ Π
4         if ({P_{n+1}, ..., P_m} ∩ Λ = ∅)
5             let r′ ← H_1 ∨ ··· ∨ H_u ← P_1, ..., P_n.
6             Π′ ← Π′ ∪ {r′}
7     return Π′
```

Algorithm 1: Gelfond-Lifschitz reduct algorithm.

PROOF A possible algorithm to compute the Gelfond-Lifschitz reduct Π^Λ of Π with respect to Λ is Algorithm 1. This algorithm executes the outermost for loop r times, where r is the number of rules in Π. For each iteration of the loop, the check $\{P_{n+1}, ..., P_m\} \cap \Lambda = \emptyset$ can be performed in $O(|\Pi| \cdot |\Lambda|)$ time, by asking for each atom in $\{P_{n+1}, ..., P_m\}$ whether it is equal to some atom in Λ. Hence the total time of the algorithm is $O(r\ |\Pi|\ |\Lambda|)$, and since $r \leq |\Pi|$, it is $O(|\Pi|^2\ |\Lambda|)$. ■

PROPOSITION 3.11 SMC ∈ coNP. □

PROOF Given an arbitrary instance $(\Lambda, \Pi) \in \Sigma_{\text{SMC}}$, to decide whether $(\Lambda, \Pi) \in$ SMC or not the Gelfond-Lifschitz reduct can be computed first. By Proposition 3.10, computing Π^Λ can be done in deterministic polynomial time with respect to the size of Λ and Π. Once Π^Λ has been computed, a subset $\Lambda' \subset \Lambda$ can be guessed in nondeterministic polynomial time. If $n = |\Lambda|$, then Λ' can be constructed by deciding for each atom $l \in \Lambda$, whether $l \in \Lambda'$ or not. Since the number of possible subsets Λ' is 2^n, it is also finite. Once a subset $\Lambda' \subset \Lambda$ has been computed, whether $\Lambda' \vDash \Pi^\Lambda$ or not can be decided. If $\Lambda' \vDash \Pi^\Lambda$, then the current branch of the computation outputs a positive answer, meaning that $(\Lambda, \Pi) \notin$ SMC. If, instead, $\Lambda' \vDash \Pi^\Lambda$, then a negative answer is returned. If all branches of the computation tree return a negative answer, $(\Lambda, \Pi) \in$ SMC is declared. ■

What said so far showed the coNP–completeness of deciding the SMC problem. Even though EL is an extension of the syntax of ASP programs, deciding this problem is just as hard as deciding the SMC one. The way this is shown is by a reduction from the SMC problem to the ELMC one for the coNP–hardness, and by a standard argument for the coNP membership.

A key property in reducing the SMC problem to the ELMC one is the following one, stating that the semantics of EL generalizes the one of stable models (in fact, of answer sets).

PROPOSITION 3.12 (D. Pearce [49, Proposition 10].) *Let Π be a logic program. A consistent set H of literals is an answer set of Π if and only if H is the set of literals true in some equilibrium model of Π.* □

What meant by "logic progam" above corresponds to the definition of EDLP given in Section 2.4, but in D. Pearce [49] the same term is also used to refer to an EL theory.

THEOREM 3.13 Deciding ELMC is coNP–complete. □

PROOF Consequence of Proposition 3.14 and Proposition 3.16. ■

Proposition 3.14 Deciding ELMC is coNP–hard. □

Proof To show that deciding ELMC is coNP–hard, a Karp reduction $f : \Sigma_{\text{SMC}} \to \Sigma_{\text{ELMC}}$ is defined. Since by Proposition 3.9 deciding SMC is coNP–hard, deciding ELMC also turns out to be coNP–hard.

Given an instance $(\Lambda, \Pi) \in \Sigma_{\text{SMC}}$,

$$f((\Lambda, \Pi)) \stackrel{\text{def}}{=} (\Lambda, \varphi),$$

where φ is the HT formula obtained by turning each ground rule in Π into a subformula of φ, joined to the others by the $\wedge$ operator. Specifically, if $\Pi \stackrel{\text{def}}{=} \{r_1, \cdots, r_q\}$, where each rule r_i is of the form

$$r_i \stackrel{\text{def}}{=} H_1 \vee \cdots \vee H_u \leftarrow P_1, ..., P_n, \text{not } P_{n+1}, ..., \text{not } P_m.,$$

then $\varphi \stackrel{\text{def}}{=} r'_1 \wedge \cdots \wedge r'_q$, where

$$r'_i \stackrel{\text{def}}{=} P_1 \wedge \cdots \wedge P_n \wedge \neg P_{n+1} \wedge \cdots \wedge \neg P_m \to H_1 \vee \cdots \vee H_u.$$

What needs to be shown at this point is that f is a function form Σ_{SMC} to Σ_{ELMC}, that $(\Lambda, \Pi) \in \text{SMC}$ iff $(\Lambda, \varphi) = f((\Lambda, \Pi)) \in \text{ELMC}$ and that f is computable in polynomial time with respect to the size of (Λ, Π).

The first property is satisfied, because for a given $(\Lambda, \Pi) \in \Sigma_{\text{SMC}}$, $(\Lambda, \varphi) = f((\Lambda, \varphi))$, where Λ is a set of atoms and φ an EL formula.

For the second property, $(\Lambda, \Pi) \in \text{SMC}$ is first assumed. If $(\Lambda, \Pi) \in \text{SMC}$, then Λ is a stable model of Π, and by Proposition 3.12 Λ is the set of atoms true in some equilibrium model of φ, where $(\Lambda, \varphi) = f((\Lambda, \Pi))$. Hence $(\Lambda, \varphi) \in \text{ELMC}$.

If, on the other hand, for an arbitrary $(\Lambda, \Pi) \in \Sigma_{\text{SMC}}$, $(\Lambda, \varphi) = f((\Lambda, \Pi)) \in \text{ELMC}$, then Λ is an equilibrium model of φ, and by Proposition 3.12 again, Λ is a stable model of Π, meaning that $(\Lambda, \Pi) \in \text{SMC}$.

Finally, f can be computed in time polynomial with respect to the size of an arbitrary instance $(\Lambda, \Pi) \in \Sigma_{\text{SMC}}$, because $f((\Lambda, \Pi)) = (\Lambda', \varphi)$. $\Lambda' = \Lambda$, and φ is an EL formula whose size is at most two times that of the DLP program Π. ■

What remains to be shown for the coNP–completeness of ELMC is its membership to coNP.

Proposition 3.15 Given an EL formula φ and an HT interpretation (H, T), deciding whether $(H, T) \vDash_{\text{HT}} \varphi$ can be done in time $O(|H| + |T| + |\varphi|)$, where $|H|$ denotes the number of atoms in H, $|T|$ the number of atoms in T and $|\varphi|$ the number of symbols in φ. □

Proof By structural induction over the syntax of φ.

If $\varphi = \bot$, then a negative answer can be returned in constant time.

If $\varphi = a$, where a is an atomic symbol, then whether $a \in H$ can be decided in time $O(|H|)$ by looking up a within H. Hence, since $H \subseteq T$, deciding whether $a \in H$ can be done in time $O(|H| + |T| + |\varphi|)$.

If $\varphi = \varphi_1 \vee \varphi_2$, then to decide whether $(H, T) \vDash_{\text{HT}} \varphi_1 \vee \varphi_2$, deciding whether $(H, T) \vDash_{\text{HT}} \varphi_1$ or $(H, T) \vDash_{\text{HT}} \varphi_2$ can be done instead. By the inductive hypothesis (IH), deciding whether $(H, T) \vDash_{\text{HT}} \varphi_1$ and whether $(H, T) \vDash_{\text{HT}} \varphi_2$ can be done in time $O(|H| + |T| + |\varphi|)$.

If $\varphi = \varphi_1 \wedge \varphi_2$, to decide whether $(H, T) \vDash_{\text{HT}} \varphi_1 \wedge \varphi_2$, deciding whether $(H, T) \vDash_{\text{HT}} \varphi_1$ and $(H, T) \vDash_{\text{HT}} \varphi_2$ can be done instead. By the IH, deciding whether $(H, T) \vDash_{\text{HT}} \varphi_1$ and whether $(H, T) \vDash_{\text{HT}} \varphi_2$ can be done in time $O(|H| + |T| + |\varphi|)$.

If $\varphi = \varphi_1 \to \varphi_2$, then deciding whether $(H,T) \vDash_{\text{HT}} \varphi_1 \to \varphi_2$ requires deciding whether

- $(H,T) \vDash_{\text{HT}} \varphi_1$ classically implies $(H,T) \vDash_{\text{HT}} \varphi_2$, and whether
- $(T,T) \vDash_{\text{HT}} \varphi_1$ classically implies $(T,T) \vDash_{\text{HT}} \varphi_2$.

By the IH, deciding whether $(H,T) \vDash_{\text{HT}} \varphi_1$ and whether $(H,T) \vDash_{\text{HT}} \varphi_2$ can be done in time $O(|H| + |T| + |\varphi|)$. Similarly, by the IH deciding whether $(T,T) \vDash_{\text{HT}} \varphi_1$ and $(T,T) \vDash_{\text{HT}} \varphi_2$ can be done in time $O(2 \cdot |T| + |\varphi|)$. ■

Proposition 3.16 ELMC $\in$ coNP. □

Proof To decide whether an instance $(\Lambda, \varphi) \in \Sigma_{\text{ELMC}}$ belongs to ELMC, a set of atoms $\Lambda' \subset \Lambda$ can be guessed in nondeterministic polynomial time. Specifically, if $n = |\Lambda|$, then constructing Λ' consists of deciding, for each of the n atoms l in Λ, whether l should be added to Λ' or not. Since the number of possible sets of atoms $\Lambda' \subset \Lambda$ is at most 2^n, it is finite. Once a set of atoms $\Lambda' \subset \Lambda$ is obtained, whether $(\Lambda', \Lambda) \vDash_{\text{HT}} \varphi$ can be checked. By Proposition 3.15, this check can be performed in deterministic linear time with respect to $|\Lambda'| + |\Lambda| + |\varphi|$. If $(\Lambda', \Lambda) \vDash_{\text{HT}} \varphi$, then this branch of the nondeterministic computation returns a positive answer, meaning that $(\Lambda, \varphi) \notin$ ELMC; if, instead, $(\Lambda', \Lambda) \nvDash_{\text{HT}} \varphi$, a negative answer is returned for this branch of the nondeterministic computation. If all branches return a negative answer, $(\Lambda, \varphi) \in$ ELMC is declared. ■

3.2. Temporal model checking

Temporal model checking is the problem of deciding whether an LTL trace is the TEM of a given THT formula. As already anticipated in the previous sections, this problem is divided into two variants, a finite-trace one, TELMC$_f$, and an infinite-trace one, TELMC.

3.2.1. Model checking over finite traces

TELMC$_f$ is the problem of checking whether a trace T of finite length is the TEM of a temporal formula φ.

Definition 3.39 *(TELMC$_f$.)* The problem of model checking for TEL in the finite case is the language

$$\text{TELMC}_f \stackrel{\text{def}}{=} \{(T, \varphi) \mid T \text{ is a TEM of } \varphi\},$$

where T is a finite-sized LTL trace and φ a THT formula. □

Similarly to ELMC, the host language Σ_{TELMC_f} is the language of all pairs (T, φ) formed by a THT formula φ and a finite trace T, irrespective of whether T is a TEM of φ.

Theorem 3.17 Deciding TELMC$_f$ is coNP–complete. □

Proof Consequence of Proposition 3.18 and Proposition 3.22. ■

Proposition 3.18 Deciding TELMC$_f$ is coNP–hard. □

Proof To show the coNP–hardness of TELMC$_f$, a Karp reduction $f : \Sigma_{\text{ELMC}} \to \Sigma_{\text{TELMC}_f}$ is defined.

Given an instance $(\Lambda, \varphi) \in \Sigma_{\text{ELMC}}$, where Λ is a set of atoms and φ an EL formula, $f((\Lambda, \varphi)) = (T, \varphi')$, where T is an LTL trace of size $|T| = 1$ and φ' a THT formula. The only state T_0 of T is defined as the set of literals Λ, that is $T_0 = \Lambda$, while φ' is defined as φ. Then, given that φ' contains no temporal operators, Λ is an EM of φ iff T is a TEM of φ'.

More formally, if $(\Lambda, \varphi) \in \textsc{Elmc}$ then Λ is an EM of φ. Since $T_0 = \Lambda$, T is a TEM of φ, meaning that $(T, \varphi) \in \textsc{Telmc}_f$.

Vice versa, if for an instance $(\Lambda, \varphi) \in \Sigma_{\textsc{Elmc}}$, $(T, \varphi') \in \textsc{Telmc}_f$, then T is a TEM of φ'. Since $\Lambda = T_0$ and $\varphi' = \varphi$, Λ is an EM of φ. Hence $(\Lambda, \varphi) \in \textsc{Elmc}$.

Finally, given that $\varphi' = \varphi$ and $|T| = 1$, f can be computed in polynomial time with respect to the size of (Λ, φ). ■

What remains to be done for $\textsc{Telmc}_f$ is show its membership to coNP.

Definition 3.40 Given a THT formula φ, the set of subformulas $\mathrm{sub}(\varphi)$ of φ is the least set such that:

1. $\varphi \in \mathrm{sub}(\varphi)$;
2. If $\varphi = \bigcirc\varphi_1$, then $\varphi_1 \in \mathrm{sub}(\varphi)$;
3. If $\varphi = \varphi_1 \odot \varphi_2$, for $\odot \in \{\vee, \wedge, \rightarrow, \mathbb{U}, \mathbb{R}\}$, then $\varphi_1, \varphi_2 \in \mathrm{sub}(\varphi)$.

□

Example 3.41 If $\varphi = (\bigcirc a)\ \mathbb{R}\ (b \wedge c)$, then

$$\mathrm{sub}(\varphi) \stackrel{\text{def}}{=} \{(\bigcirc a)\ \mathbb{R}\ (b \wedge c), \bigcirc a, b \wedge c, a, b, c\}.$$

□

Proposition 3.19 Given a THT formula φ, $|\mathrm{sub}(\varphi)| \leq 2|\varphi| + 1$, where $|\varphi|$ is the number of symbols in φ. □

Proof By structural induction over the syntax of φ.

If $\varphi = \bot$, then $\mathrm{sub}(\varphi) \stackrel{\text{def}}{=} \{\bot\}$, and $|\mathrm{sub}(\varphi)| = 1 \leq 3 = 2|\varphi| + 1$ holds.

If $\varphi = a$, a reasoning similar to that of the $\varphi = \bot$ case follows.

If $\varphi = \bigcirc\varphi_1$, then $|\varphi| = |\varphi_1| + 1$ and $\mathrm{sub}(\varphi) \stackrel{\text{def}}{=} \{\varphi\} \cup \mathrm{sub}(\varphi_1)$. By the inductive hypothesis (IH) $|\mathrm{sub}(\varphi_1)| \leq 2|\varphi_1| + 1$, meaning that

$$\begin{aligned} |\mathrm{sub}(\varphi)| = |\{\varphi\}| + |\mathrm{sub}(\varphi_1)| &\leq 2|\varphi_1| + 2 \\ &= 2|\varphi| \\ &\leq 2|\varphi| + 1. \end{aligned}$$

If $\varphi = \varphi_1 \odot \varphi_2$ for $\odot \in \{\vee, \wedge, \rightarrow, \mathbb{U}, \mathbb{R}\}$, then $|\varphi| = |\varphi_1| + |\varphi_2| + 1$ and $\mathrm{sub}(\varphi) \stackrel{\text{def}}{=} \{\varphi\} \cup \mathrm{sub}(\varphi_1) \cup \mathrm{sub}(\varphi_2)$. By the IH applied to φ_1 and φ_2, respectively, $|\mathrm{sub}(\varphi_1)| \leq 2|\varphi_1| + 1$ and $|\mathrm{sub}(\varphi_2)| \leq 2|\varphi_2| + 1$, meaning that

$$\begin{aligned} |\mathrm{sub}(\varphi)| &= 1 + |\mathrm{sub}(\varphi_1)| + |\mathrm{sub}(\varphi_2)| \\ &\leq 1 + 2|\varphi_1| + 1 + 2|\varphi_2| + 1 \\ &= 2|\varphi_1| + 2|\varphi_2| + 2 + 1 \\ &= 2(|\varphi_1| + |\varphi_2| + 1) + 1 \\ &= 2|\varphi| + 1. \end{aligned}$$

■

The memberhip of $\textsc{Telmc}_f$ to coNP relies on a so-called marking algorithm for finite THT traces, which given a THT trace (H, T), a natural k and a THT formula φ, decides whether $(H, T), k \vDash \varphi$ or not. This algorithm is illustrated in Algorithm 2.

```
1  BOOLEAN ⊨_{THT_f} (FORMULA φ, TRACE (H,T), NATURAL k)
2      let L ← |(H,T)|
3      % S is a STACK where inner formulas are extracted prior to outer ones
4      let S ← SUBFORMULAS(φ)
5      % ψ ∈ M_k if the subformula ψ is true in (H,T) at position k
6      let M ← VECTOR(L)
7      % ψ ∈ N_k if the subformula ψ is true in (T,T) at position k
8      let N ← VECTOR(L)

9      while S ≠ ∅
10         ψ ← S.POP()

11         if ψ = ⊥
12             skip
13         else if ψ = a
14             M_k ← M_k ∪ {ψ} if a ∈ H_k
15             N_k ← N_k ∪ {ψ} if a ∈ T_k
16         else if ψ = ψ_1 ∨ ψ_2
17             M_k ← M_k ∪ {ψ} if ψ_1 ∈ M_k or ψ_2 ∈ M_k
18             N_k ← N_k ∪ {ψ} if ψ_1 ∈ N_k or ψ_2 ∈ N_k
19         else if ψ = ψ_1 ∧ ψ_2
20             M_k ← M_k ∪ {ψ} if ψ_1 ∈ M_k and ψ_2 ∈ M_k
21             N_k ← N_k ∪ {ψ} if ψ_1 ∈ N_k and ψ_2 ∈ N_k
22         else if ψ = ψ_1 → ψ_2
23             M_k ← M_k ∪ {ψ} if ψ_1 ∈ M_k implies ψ_2 ∈ M_k and ψ_1 ∈ N_k implies ψ_2 ∈ N_k
24             N_k ← N_k ∪ {ψ} if ψ_1 ∈ N_k implies ψ_2 ∈ N_k
25         else if ψ = ○ψ_1
26             M_k ← M_k ∪ {ψ} if k < L and ψ_1 ∈ M_{k+1}
27             N_k ← N_k ∪ {ψ} if k < L and ψ_1 ∈ N_{k+1}
28         else if ψ = ψ_1 𝕌 ψ_2
29             if ∃j ∈ [k..L] such that ψ_2 ∈ M_j and ∀j' ∈ [k..j), ψ_1 ∈ M_{j'}
30                 M_k ← M_k ∪ {ψ}
31             if ∃j ∈ [k..L] such that ψ_2 ∈ N_j and ∀j' ∈ [k..j), ψ_1 ∈ N_{j'}
32                 N_k ← N_k ∪ {ψ}
33         else if ψ = ψ_1 ℝ ψ_2
34             if ∀j ∈ [k..L], ψ_2 ∈ M_j
35                 M_k ← M_k ∪ {ψ}
36             else if ∃j ∈ [k..L] such that ψ_1 ∈ M_j and ∀j' ∈ [k..j], ψ_2 ∈ M_{j'}
37                 M_k ← M_k ∪ {ψ}
38             % Analogous for N

39     return true if φ ∈ M_k else false
```

Algorithm 2: Marking algorithm for $(H,T), k \vDash_{\mathrm{THT}_f} \varphi$.

```
1  Stack Subformulas (Formula φ)
2      let S ← Stack()
3      let Unvisited ← Queue()
4      S.Append(φ)
5      Unvisited.Enqueue(φ)
6      while Unvisited ≠ ∅
7          let ψ ← Unvisited.Dequeue()
8          if ψ = ⊥
9              S.Append(⊥)
10         else if ψ = a
11             S.Append(a)
12         else if ψ = ○ψ1
13             S.Append(ψ1)
14             Unvisited.Enqueue(ψ1)
15         else if ψ = ψ1 ⊙ ψ2, ⊙ ∈ {∨, ∧, →, 𝕌, ℝ}
16             S.Append(ψ1)
17             S.Append(ψ2)
18             Unvisited.Enqueue(ψ1)
19             Unvisited.Enqueue(ψ2)
20     return S
```

Algorithm 3: Algorithm to compute sub(φ) for a THT formula φ.

Proposition 3.20 *(Correctness of the marking algorithm.)* For all THT formulas φ, finite-sized THT traces (H,T) and naturals k, $0 \leq k \leq |(H,T)|$, Algorithm 2 returns a positive answer iff $(H,T),k \vDash_{\text{THT}} \varphi$. □

Proposition 3.21 Given a THT formula φ, a finite-sized THT trace (H,T) and a natural k, $0 \leq k \leq |(H,T)|$, deciding whether $(H,T),k \vDash_{\text{THT}} \varphi$ can be performed in $O(|(H,T)| \cdot |\varphi|)$ time. □

Proof By Proposition 3.20, Algorithm 2 can be used to decide whether $(H,T),k \vDash_{\text{THT}} \varphi$. The S variable of Algorithm 2 can be computed in $O(|\varphi|)$ time using Algorithm 3, while the outer **for each** loop in Algorithm 2 is executed at most $|\text{sub}(\varphi)|$ times, which by Proposition 3.19 is at most $2|\varphi|+1$ times. Within this loop, at most $2|(H,T)|$ operations are performed each iteration, meaning that the total running time of the algorithm is $O(|\varphi| \cdot |(H,T)|)$. ■

The membership of Telmc_f to coNP can at this point be proven.

Proposition 3.22 $\text{Telmc}_f \in \text{coNP}$. □

Proof To decide whether an instance $(T,\varphi) \in \Sigma_{\text{Telmc}_f}$ belongs to Telmc_f, a subtrace $H \subset T$ of T can be guessed in nondeterministic polynomial time. Specifically, if T is

constructed over the alphabet A, then constructing H means determining a subset H_i of atoms for each state T_i of T. Since each state T_i of T has at most $|A|$ atoms, at most $2^{|A|}$ possible subsets of T_i need to be considered; since T is of finite size, at most $2^{|A|} \cdot |T| = 2^{|A| + \log_2 |T|}$ possible subtraces H of T are possible.

Once a subtrace H of T is obtained, $(H, T) \vDash_{\text{THT}} \varphi$ can be checked; by Proposition 3.21, this check can be performed in deterministic polynomial time with respect to $|\varphi|$ and $|T|$. If $(H, T) \vDash_{\text{THT}} \varphi$, then the current computation branch outputs a positive answer, meaning that $(T, \varphi) \notin \textsc{Telmc}_f$. If, instead, $(H, T) \nvDash_{\text{THT}} \varphi$, then the current computation branch outputs a negative answer. If all computation branches output a negative answer, $(T, \varphi) \in \textsc{Telmc}_f$ is declared. ■

3.2.2. Model checking over infinite traces

Telmc is the problem of checking whether a trace T of infinite length is the TEM of a THT formula φ. In contrast to the previous section, where traces of finite length are considered, this section needs a way to represent in a finite encoding a trace of infinite length. This is done by restricting the focus to *periodically infinite* LTL traces.

Definition 3.42 A *periodically infinite* LTL trace T, also referred to as *lasso-shaped trace*, is an infinite LTL trace expressible as $T_1 T_2^\omega$, where T_1 and T_2 are finite LTL traces and T_2^ω represents the infinite repetition of T_2. □

Although periodically infinite LTL traces allow to specify LTL traces of a rather specific shape, this is not particularly restrictive when it comes to reasoning in TEL. A TEM of a formula φ can in fact be represented as a single LTL trace T, for which no $H < T$ exists such that $(H, T) \vDash_{\text{THT}} \varphi$. The TEL semantics makes no specific assumptions about the nature of H, and this subtrace is also allowed to be aperiodic, that is, unable to be specified using a finite encoding.

Definition 3.43 *(Telmc.)* The problem of model checking for TEL in the infinite case is the language

$$\textsc{Telmc} \stackrel{\text{def}}{=} \{(T, \varphi) \mid T \text{ is a TEM of } \varphi\},$$

where T is a periodically infinite LTL trace and φ a THT formula. □

Before jumping into the Pspace-hardness of deciding Telmc, a small observation might help clarify one of the intermediate steps in the proof of it.

Observation 3.44 If L is Pspace–hard, then $\overline{\text{L}}$ is Pspace–hard. □

Proof Let L be a language in Pspace. Then, by definition of complementation of computational classes, $\overline{\text{L}} \in \text{co}\textsc{Pspace}$, and since $\text{co}\textsc{Pspace} = \textsc{Pspace}$, there exists a Karp reduction f for which $\overline{\text{L}} \leq_m^P L$. This means that

$$\begin{aligned} x \in \overline{\text{L}} &\text{ iff } f(x) \in L, \\ x \notin \overline{\text{L}} &\text{ iff } f(x) \notin L, \\ x \in L &\text{ iff } f(x) \in \overline{\text{L}}, \\ L &\leq_m^P \overline{\text{L}}. \end{aligned}$$

■

Theorem 3.23 Deciding Telmc is Pspace–complete. □

Proof Consequence of Theorem 3.24 and Corollary 3.33. ■

Theorem 3.24 Deciding Telmc is Pspace–hard. □

Proof To show the Pspace–hardness of deciding Telmc, it is shown how any language L in Pspace can be reduced to $\overline{\text{Telmc}}$, the complement of the Telmc problem, by proving the existence of a Karp reduction $f : \Sigma_L \to \Sigma_{\overline{\text{Telmc}}}$ assigning to each word $x \in \Sigma_L$ a pair $(T, \varphi) \in \Sigma_{\overline{\text{Telmc}}}$, where T is an LTL trace and φ a THT formula. If deciding $\overline{\text{Telmc}}$ is Pspace–hard, by Observation 3.44 deciding Telmc also results to be Pspace–hard. It needs to be noted that the proof put forth here is not entirely constructive, and is limited to showing the existence of the reduction f, without providing all means to construct it[11].

If $L \in$ Pspace, then a single-tape Turing machine $M = (Q, \Gamma, \delta, q_0, F)$ deciding L in deterministic polynomial space exists[12], where Q is its non-empty, finite set of states, Γ its tape alphabet, $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$ its transition function, $q_0 \in Q$ its initial state and $F = \{q_{\text{accept}}, q_{\text{reject}}\} \subseteq Q$ its set of halting states. $\{-1, 0, 1\}$ denotes the possible directions the tape head of M can move to, corresponding to left, no movement, and right, respectively; they are represented as integers to facilitate the construction of f.

Since M decides L in polynomial space, a polynomial $p(n)$ exists such that for any $x \in \Sigma_L$, M decides in $p(|x|)$ space whether x belongs to L or not, where $|x|$ is the number of symbols in x. Because of this, each configuration of M at step i can be encoded by storing the state q at step i, the position of the head of M at step i and at most $p(|x|)$ tape symbols. To encode the state q of M, a set of symbols Q is used, in such a way that for a $q \in Q$, $q \in T_i$ iff the state of M at time step i is q. To encode the head position of M at time step i, a set of variables h_k, $k \in [0..p(|x|))$ is used, so that $h_k \in T_i$ iff the head of M at time step i points to cell k. The tape content of M at time step i can be encoded into state T_i of T by the use of $|\Gamma| \cdot p(|x|)$ many propositional atoms. For each tape symbol $\gamma \in \Gamma$ and $k \in [0..p(|x|))$, the propositional variable $\sigma_{k,\gamma}$ is added to the alphabet A of φ to signify that at the tape position k, the symbol is γ.

To sum it up, the alphabet A over which φ is defined is $A \stackrel{\text{def}}{=} A_Q \cup A_h \cup A_\sigma$, where

$$
\begin{aligned}
A_Q &\stackrel{\text{def}}{=} Q, \\
A_h &\stackrel{\text{def}}{=} \{h_k \mid 0 \leq k < p(|x|)\}, \\
A_\sigma &\stackrel{\text{def}}{=} \{\sigma_{k,\gamma} \mid 0 \leq k < p(|x|), \gamma \in \Gamma\},
\end{aligned}
$$

showing in particular that the cardinality of A is polynomial with respect to the size of M and $p(|x|)$.

Given an arbitrary word $x \in \Sigma_L$, $f(x) \stackrel{\text{def}}{=} (T, \varphi)$, where $T \stackrel{\text{def}}{=} \left(2^A\right)^\omega$ and $\varphi \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq 7} \varphi_i$ is the formula encoding the behavior of M, split into a number of subformulas for easier understanding. These subformulas are:

$$
\varphi_1 \stackrel{\text{def}}{=} q_0 \wedge h_0 \wedge \sigma_{0,x_0} \wedge \sigma_{1,x_1} \wedge \cdots \wedge \sigma_{|x|,x_{|x|}} \wedge \sigma_{|x|+1,b} \wedge \cdots \wedge \sigma_{p(|x|)-1,b},
$$

$$
\varphi_2 \stackrel{\text{def}}{=} \bigwedge_{q \in Q} \bigwedge_{0 \leq k < p(|x|)} \bigwedge_{\gamma \in \Gamma} \square\big(q \wedge h_k \wedge \sigma_{k,\gamma} \wedge \delta(q, \gamma) = (q', \gamma', \Delta) \to \bigcirc\big(q' \wedge h_{k+\Delta} \wedge \sigma_{k,\gamma'}\big)\big),
$$

$$
\varphi_3 \stackrel{\text{def}}{=} \square\left(\bigwedge_{0 \leq k < p(|x|)} \bigwedge_{\substack{0 \leq k' < p(|x|) \\ k' \neq k}} \bigwedge_{\gamma \in \Gamma} h_k \wedge \sigma_{k',\gamma} \to \bigcirc \sigma_{k',\gamma}\right),
$$

[11] This is, in fact, also the case with the proof of the Cook-Levin theorem that shows the NP–hardness of SAT.

[12] The reason why the proof has been said to be non-constructive before is due to the Turing machine M. M is known to exist, but it is not given as input. Hence, even though much of its behavior is characterized, the only thing that can be said about the reduction f is that it exists. In a sense, this kind of proof could be seen as semi-constructive proof, or in other words a constructive proof where some elements for the effective computation of the reduction it proves to exist is still missing.

$$\varphi_4 \stackrel{\text{def}}{\equiv} \Diamond \left(\bigvee_{q \in Q} \bigvee_{q' \in Q \setminus \{q\}} q \wedge q' \right) \rightarrow \varphi^{\uparrow},$$

$$\varphi_5 \stackrel{\text{def}}{\equiv} \Diamond \left(\bigvee_{0 \leq k < p(|x|)} \bigvee_{\substack{0 \leq k' < p(|x|) \\ k' \neq k}} h_k \wedge h_{k'} \right) \rightarrow \varphi^{\uparrow},$$

$$\varphi_6 \stackrel{\text{def}}{\equiv} \Diamond \left(\bigvee_{0 \leq k < p(|x|)} \bigvee_{\gamma \in \Gamma} \bigvee_{\gamma' \in \Gamma \setminus \{\gamma\}} \sigma_{k,\gamma} \wedge \sigma_{k,\gamma'} \right) \rightarrow \varphi^{\uparrow},$$

$$\varphi_7 \stackrel{\text{def}}{\equiv} \Diamond q_{\text{reject}} \rightarrow \varphi^{\uparrow}.$$

The subformulas make use of the auxiliary formula $\varphi^{\uparrow}$, defined as

$$\varphi^{\uparrow} \stackrel{\text{def}}{\equiv} \Box \bigwedge_{a \in A} a$$

and used to induce a state of saturation over the TEMs of φ.

The subformula φ_1 encodes the initial configuration of M on input x, consisting of the presence of the initial state q_0, the tape head set to position 0 and the content of x written on the tape of this first configuration, followed by a number of white space characters b.

φ_2 encodes the behavior of the transition relation δ of M into T, saying that if at any time the transition relation instructs M to write a certain symbol on the tape and move the tape head to a certain direction, then this change should be reflected in the next state of T.

φ_3 encodes inertia into T, ensuring that if at a certain time step a symbol is present on T and the tape head is pointing elsewhere, then that same symbol is also present on the tape during the next time step.

φ_4 enforces a state of saturation on a TEM of φ whenever more than one state $q \in Q'$ is present at a single time step. Rather than encode the constraint that only a single state q be present at each time step using negation, saturation is used: Whenever this constraint is violated, only the (T, T) trace is a TEM of φ.

Similarly to φ_4, φ_5 uses saturation to enforce the presence of at most one h_k atom at each time step, and φ_6 to enforce the presence of at most one $\sigma_{k,\gamma}$ atom at each time step for each k, $0 \leq k < p(|x|)$.

Finally, the formula φ_7 enforces a state of saturation in case a reject state is reached by M on input x. If the q_{reject} state is not reached, the q_{accept} one eventually will be, as M is guaranteed to halt on input x.

By performing an individual check for each of the φ_i formulas, it can be noted how (T, T) is a total THT interpretation satisfying φ.

What is left to show at this point is that f correctly maps x to a pair (T, φ), with T a periodically infinite LTL trace and φ a THT formula; that $x \in L$ iff T is not a TEM of φ; and that f is computable in polynomial time with respect to $|x|$.

The first statement holds, because $T = \left(2^A\right)^{\omega}$ is a periodically infinite LTL trace and φ, by the fact that it uses ordinary logical operators and $\bigcirc, \mathbb{U}$ and $\mathbb{R}$, a THT formula.

To show the second statement, $x \in L$ is first supposed. Since $x \in L$, M halts on x within $p(|x|)$ space by following a computation C, understood as a finite sequence of configurations, whose last configuration contains the q_{accept} state. C can be used to construct an LTL trace $H < T$ such that $(H, T) \vDash \varphi$ by populating each state H_i of H with the q, h_k and $\sigma_{k,\gamma}$ propositional variables that correspond to configuration C_i of C. It can be checked how a THT trace (H, T) thus constructed satisfies φ under the THT

semantics. Hence an LTL trace $H < T$ exists such that $(H, T) \vDash \varphi$, meaning that T is not a TEM of φ, or equivalently that $(T, \varphi) \in \overline{\text{TELMC}}$.

For the other direction, $(T, \varphi) \in \overline{\text{TELMC}}$ is supposed. If $(T, \varphi) \in \overline{\text{TELMC}}$, T is not a TEM of φ, meaning that there exists an LTL trace $H < T$ for which $(H, T) \vDash \varphi$. Thanks to the φ_4, φ_5 and φ_6 formulas, the atoms in H have been minimized in a way that for each state H_i of H only one state atom q, one head atom h_k and one tape symbol atom $\sigma_{k,\gamma}$ for all k, $0 \leq k < p(|x|)$, is present. Additionally, thanks to the φ_1 formula, the initial configuration of M on input x is imposed on state H_0, whereas thanks to the φ_2 formula, the transition relation δ of M is respected. Since φ_3 is also satisfied, the $p(|x|) - 1$ tape atoms $\sigma_{k,\gamma}$ that were not explicitly specified by the transition relation δ are copied, for each state, from the previous one. Finally, since H is not a saturated trace, the q_{reject} state of M is never hit, meaning that the q_{accept} one is. Hence M accepts x, meaning that $x \in L$.

Finally, f can be computed in polynomial time with respect to the size of M and x, because constructing $T = \left(2^A\right)^\omega$ entails constructing a single $\left\{a_1, a_2, ..., a_{|A|}\right\}$ expression with the polynomially many literals of A, while the time needed to construct φ is bounded by a polynomial function of $p(|x|)$. ■

The PSPACE–hardness of TELMC is one of the central results of this work, so much so that it alone justifies the presence of Chapter 4, where an attempt to study fragments of the TELMC problem with a lower upper-bound complexity is made. For the very reason that the PSPACE–hardness of TELMC is an important result, an alternative proof of it is given in Chapter 4, specifically in Theorem 4.53, by constructing a reduction from a variant of the domino tiling problem (P. v. E. Boas [59]).

3.2.2.1. A few intermediate techniques

Before continuing to discuss the complexity of TELMC and show its PSPACE membership, a number of techniques used in the sequel is introduced. The first of these techniques is the *star translation* (P. Cabalar and S. Demri [52]), a transformation that allows a THT interpretation to be encoded into an LTL one. THT interpretations are represented by a pair (H, T) of LTL traces, and this translation allows to merge (H, T) into a single LTL trace M.

DEFINITION 3.45 Given a THT trace (H, T) with atoms drawn from a finite alphabet $A = \{a_1, ..., a_n\}$, the *star translation* of (H, T), denoted by $(H, T)^*$, is the LTL trace M defined by

$$M_i \stackrel{\text{def}}{=} \{a' \mid a \in H_i\} \cup \{a \mid a \in T_i\}.$$

□

Setting $X' \stackrel{\text{def}}{=} \{x' \mid x \in X\}$, M can therefore be said to be defined over the alphabet $A^* \stackrel{\text{def}}{=} A \cup A'$, in such a way that if $a' \in H_i$, then $a \in T_i$. Equivalently, M can be also said to satisfy the formula

$$\varphi_{\leq}(A) \stackrel{\text{def}}{=} \bigwedge_{a \in A} \Box(a' \rightarrow a),$$

corresponding to the requirement that $H_i \subseteq T_i$ for a THT interpretation (H, T).

Conversely, for any LTL trace M defined over a set $A \cup A'$ of symbols such that $M \vDash_{\text{LTL}} \varphi_{\leq}(A)$, a THT trace (H, T) can be found such that $(H, T)^* = M$. More precisely, the following holds.

LEMMA 3.25 *(P. Cabalar and S. Demri [52, Lemma 1].)* Given any LTL trace M constructed over a set of atoms $A \cup A'$ such that $M \vDash_{\text{LTL}} \varphi_{\leq}(A)$, there exists a *unique* THT trace (H, T) over A such that $H \leq T$ and $(H, T)^* = M$. □

From THT and LTL traces, the star translation can be also extended to THT formulas.

Definition 3.46 The *star translation* for THT formulas is a function $\cdot^*$ from THT formulas to THT formulas, defined by structural recursion as:

$$\begin{aligned}
\bot^* &\stackrel{\text{def}}{=} \bot,\\
\top^* &\stackrel{\text{def}}{=} \top,\\
a^* &\stackrel{\text{def}}{=} a',\\
(\varphi_1 \vee \varphi_2)^* &\stackrel{\text{def}}{=} \varphi_1^* \vee \varphi_2^*,\\
(\varphi_1 \wedge \varphi_2)^* &\stackrel{\text{def}}{=} \varphi_1^* \wedge \varphi_2^*,\\
(\varphi_1 \rightarrow \varphi_2)^* &\stackrel{\text{def}}{=} (\varphi_1^* \rightarrow \varphi_2^*) \wedge (\varphi_1 \rightarrow \varphi_2),\\
(\bigcirc\varphi)^* &\stackrel{\text{def}}{=} \bigcirc\varphi^*,\\
(\varphi_1 \,\mathbb{U}\, \varphi_2)^* &\stackrel{\text{def}}{=} \varphi_1^* \,\mathbb{U}\, \varphi_2^*,\\
(\varphi_1 \,\mathbb{R}\, \varphi_2)^* &\stackrel{\text{def}}{=} \varphi_1^* \,\mathbb{R}\, \varphi_2^*.
\end{aligned}$$

□

Example 3.47 Let $\varphi \stackrel{\text{def}}{=} \Box(a \rightarrow b) \wedge \Diamond(\neg b)$. Then

$$\varphi^* = \Box(a' \rightarrow b' \wedge a \rightarrow b) \wedge \Diamond(b' \rightarrow \bot \wedge b \rightarrow \bot).$$

□

The reason the star translation is said to translate THT satisfiability into LTL satisfiability is motivated by the following result.

Lemma 3.26 *(F. Aguado et al. [69, Theorem 4].)* Let φ be a THT formula and (H,T) a THT interpretation. Then

$$(H,T), k \vDash_{\text{THT}} \varphi \text{ iff } (H,T)^*, k \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A).$$

□

This equivalence between THT and LTL traces can be brought one step further by introducing a constraining formula that requires THT interpretations to be non-total.

Definition 3.48 Given alphabet A, the *non-totality* formula for A is the formula

$$\varphi_{<}(A) \stackrel{\text{def}}{=} \bigvee_{a \in A} \Diamond(\neg a' \wedge a).$$

□

For this formula, the following holds.

Lemma 3.27 For any LTL trace M constructed over a set of atoms $A \cup A'$ such that $M \vDash_{\text{LTL}} \varphi_{\leq}(A) \wedge \varphi_{<}(A)$, there exists a *unique* THT trace (H,T) over A such that $H < T$ and $(H,T)^* = M$. □

Proof A simple consequence of P. Cabalar and S. Demri [52, Lemma 4]. ■

As a consequence, a restriction of Lemma 3.26 to non-total traces also holds.

Lemma 3.28 Let (H,T) be a THT interpretation for which $H < T$. Then

$$(H,T), k \vDash_{\text{THT}} \varphi \text{ iff } (H,T)^*, k \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A).$$

□

3.2.2.2. Embedding traces into formulas

To continue the journey through the various reshapings of temporal formulas, another technique is introduced which shows how the behavior of an LTL or THT trace can be embedded into a formula. This method will be first illustrated with an example. If the periodically infinite LTL trace $T = \{a, b\} \cdot \{c, d\} \cdot \emptyset \cdot (\{a\} \cdot \{a, b\} \cdot \{c\})^\omega$, depicted in Figure 7, is considered, an LTL formula φ_T can be associated to it in such a way that $T' \vDash_{\text{LTL}} \varphi_T$ iff $T = T'$. For this specific case,

$$
\begin{aligned}
\varphi_T \stackrel{\text{def}}{\equiv}\ & s_1 \\
& \wedge \Box(s_1 \to a \wedge b \wedge \neg c \wedge \neg d) && \wedge \Box(s_1 \to \bigcirc s_2) \\
& \wedge \Box(s_2 \to \neg a \wedge \neg b \wedge c \wedge d) && \wedge \Box(s_2 \to \bigcirc s_3) \\
& \wedge \Box(s_3 \to \neg a \wedge \neg b \wedge \neg c \wedge \neg d) && \wedge \Box(s_3 \to \bigcirc s_4) \\
& \wedge \Box(s_4 \to a \wedge \neg b \wedge \neg c \wedge \neg d) && \wedge \Box(s_4 \to \bigcirc s_5) \\
& \wedge \Box(s_5 \to a \wedge b \wedge \neg c \wedge \neg d) && \wedge \Box(s_5 \to \bigcirc s_6) \\
& \wedge \Box(s_6 \to \neg a \wedge \neg b \wedge c \wedge \neg d) && \wedge \Box(s_6 \to \bigcirc s_3) \\
& \wedge \Box(s_1 \to \neg s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5 \wedge \neg s_6) \\
& \wedge \Box(s_2 \to \neg s_1 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5 \wedge \neg s_6) \\
& \wedge \Box(s_3 \to \neg s_1 \wedge \neg s_2 \wedge \neg s_4 \wedge \neg s_5 \wedge \neg s_6) \\
& \wedge \Box(s_4 \to \neg s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge \neg s_5 \wedge \neg s_6) \\
& \wedge \Box(s_5 \to \neg s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_6) \\
& \wedge \Box(s_6 \to \neg s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5).
\end{aligned}
$$

Informally, this translation identifies each state of the input trace T, constrains the way each state can be reached from another and enforces the corresponding set of literals (and only that corresponding set of literals) to hold at a state. This type of construction is termed T-*embedding*, and can be generally formulated as follows.

Definition 3.49 Given a periodically infinite LTL trace $T = T_1 \cdot T_2 \cdot \cdots \cdot T_n \cdot (T_{n+1} \cdot T_{n+2} \cdot \cdots \cdot T_m)^\omega$ over the alphabet A, the T-*embedding* of T is the formula

$$
\begin{aligned}
\varphi_T \stackrel{\text{def}}{\equiv}\ & s_1 \\
& \wedge \bigwedge_{1 \le i \le m-1} \Box(s_i \to \bigcirc s_{i+1}) \wedge \Box(s_m \to \bigcirc s_{n+1}) \\
& \wedge \bigwedge_{1 \le i \le m} \Box\left(s_i \to \bigwedge_{a \in A \cap T_i} a \wedge \bigwedge_{a \in A \setminus T_i} \neg a\right) \\
& \wedge \bigwedge_{1 \le i \le m} \Box\left(s_i \to \bigwedge_{\substack{1 \le j \le m \\ j \ne i}} \neg s_j\right),
\end{aligned}
$$

where s_i, $1 \le i \le m$, are newly-introduced atoms not occurring in A. □

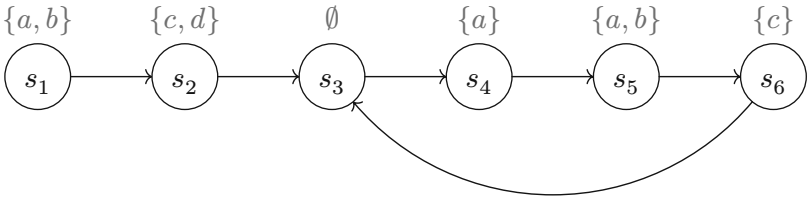


Figure 7: A lasso-shaped LTL trace.

PROPOSITION 3.29 *(Characterization property of T-embeddings.)* Let φ be an LTL formula and $T = T_1 \cdot \cdots \cdot T_n \cdot (T_{n+1} \cdot \cdots \cdot T_m)^\omega$ an LTL trace, both defined over a common alphabet A. Then $T \vDash \varphi$ iff $\varphi \wedge \varphi_T$ is satisfiable. □

PROOF IDEA If $T \vDash \varphi$, then the behavior of T can be embedded into a formula φ_T, so that $\varphi \wedge \varphi_T$ is satisfiable. Vice versa, if $\varphi \wedge \varphi_T$ is satisfiable, then, by definition, a trace over $A \cup \{s_1, ..., s_m\}$ exists which satisfies it. If this trace is restricted to A, a new trace that satisfies φ and is equal to T is obtained. In other words, T satisfies φ. ■

PROOF If $T \vDash \varphi$, then an LTL trace T', defined as

$$T' \stackrel{\text{def}}{=} (T_1 \cup \{s_1\}) \cdot \cdots \cdot (T_n \cup \{s_n\}) \cdot (T_{n+1} \cup \{s_{n+1}\} \cdot \cdots \cdot T_m \cup \{s_m\})^\omega$$

exists which satisfies $\varphi \wedge \varphi_T$. $T' \vDash \varphi$, because $T \vDash \varphi$ and T' extends T by atoms outside the alphabet of φ, while $T' \vDash \varphi_T$ because T' satisfies all of the subformulas of φ_T:

1. $T' \vDash s_1$;
2. If $T'_k \vDash s_i$ for $i \in [1..m-1]$, then $T'_{k+1} \vDash s_{i+1}$, and if $T'_k \vDash s_m$, then $T'_{k+1} \vDash s_{n+1}$;
3. If $T'_k \vDash s_i$, then $T'_k \vDash a$ for any $a \in A \cap T_i$ and $T'_k \vDash \neg a$ for any $a \in A \setminus T_i$;
4. If $s_i \in T'_k$, then by construction no other state s_j belongs to it.

Hence $T' \vDash \varphi \wedge \varphi_T$, meaning that $\varphi \wedge \varphi_T$ is satisfiable.

Conversely, if $\varphi \wedge \varphi_T$ is satisfiable, then a trace T' constructed over the alphabet $A \cup \{s_1, ..., s_m\}$ exists which satisfies $\varphi \wedge \varphi_T$. If T' is restricted to A, a new trace U, defined as $U_k \stackrel{\text{def}}{=} T_k \cap A$, is obtained which satisfies φ and is equal to T; in other words, when restricted to A, T' still satisfies φ and it is equal to T.

U satisfies φ because φ is defined over A, and the assignment of atoms of U in A is not different than that of T'. Further, $U = T$ because, for any $k \geq 1$, $U_k \subseteq T_k$ and $T_k \subseteq U_k$. $T_k \subseteq U_k$ because $T_k = T_i$ for some $i \in [1..m]$, and for the corresponding state T'_k of T', $T'_k \vDash \bigwedge_{a \in A \cap T_i} a$, meaning that $T'_k \supseteq T_i = T_k$. Since U is just a restriction of T' to A, this condition still holds for U, that is $U_k \supseteq T_i = T_k$. Additionally, $U_k \subseteq T_k$; this statement can be shown by contraposition, by observing that

$$\begin{aligned} U_k \subseteq T_k \quad & \text{iff} \\ \forall a \in A, a \in U_k \Rightarrow a \in T_k \quad & \text{iff} \\ \forall a \in A, a \notin T_k \Rightarrow a \notin U_k. & \end{aligned}$$

Let then a $k \geq 1$ be fixed, and an $x \in A$ such that $x \notin T_k$, if any, be taken. By a similar reasoning to the one used in the previous inclusion, $T_k = T_i$ for some $i \in [1..m]$, meaning $x \notin T_i$. Since $T'_k \vDash \bigwedge_{a \in A \setminus T_i} \neg a$, or equivalently $\overline{T_i} \subseteq \overline{T'_k}$, $x \notin T'_k$. Given that U is, again, but a restriction of T' to A and no assignment of atoms from this alphabet is changed, $a \notin U_k$ still holds. Hence $U_k \subseteq T_k$. ■

The usefulness of this translation is that it allows turning an LTL model checking problem into a satisfiability one. But the benefits of the T-embedding do not end here. In fact, when combined with the star translation, this technique can also be applied to THT traces. The caveat to consider in this case is that since a THT trace (H, T) is a pair of LTL traces, there is a difference on which of the two traces, either H or T, is embedded into a formula; there is, indeed, also a difference on whether the embedding is performed into φ or φ^*. Loosely speaking, if H is embedded into φ^* (φ_H is considered) then all literals in H are being asked to be *justified* (in an ASP sense), while if T is embedded into φ^* (φ_T is considered) then all literals in T are being asked to be *assumed*.

Lemma 3.30 Let $T = T_1 \cdot \dots \cdot T_n \cdot (T_{n+1} \cdot \dots \cdot T_m)^\omega$ be a periodically infinite LTL trace and φ a THT formula. Then some $H \leq T$ exists such that $(H,T) \vDash_{\text{THT}} \varphi$ iff $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$ is satisfiable. □

Observation 3.50 The formula $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$ is constructed over the alphabet $A \cup A' \cup \{s_1, ..., s_m\}$, but the φ_T formula imposes a constraint only over the A atoms. □

Proof If some $H \leq T$ exists such that $(H,T) \vDash_{\text{THT}} \varphi$ and $M \stackrel{\text{def}}{=} (H,T)^*$, then by Lemma 3.26 $M \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A)$, meaning that $\varphi^* \wedge \varphi_{\leq}(A)$ is LTL satisfiable. $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$ can be shown to be LTL satisfiable by extending M by M', an LTL trace defined as

$$M' \stackrel{\text{def}}{=} (M_1 \cup \{s_1\}) \cdot \dots \cdot (M_n \cup \{s_n\}) \cdot ((M_{n+1} \cup \{s_{n+1}\}) \cdot \dots \cdot (M_m \cup \{s_m\}))^\omega.$$

Since $M \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A)$ and M' extends M with atoms outside the alphabet of $\varphi^* \wedge \varphi_{\leq}(A)$, $M' \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A)$. Further, $M' \vDash_{\text{LTL}} \varphi_T$ because, as shown in the proof of Proposition 3.29, M' satisfies all of the subformulas of φ_T. Hence $M' \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$, meaning that $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$ is LTL satisfiable.

Conversely, if $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$ is LTL satisfiable, then there exists an LTL trace T' over the alphabet $A \cup A' \cup \{s_1, ..., s_m\}$ that satisfies it. From T', an LTL trace U restricted over the alphabet $A \cup A'$, defined as $U_k \stackrel{\text{def}}{=} T'_k \cap (A \cup A')$, can be obtained. This trace still satisfies $\varphi^* \wedge \varphi_{\leq}(A)$, because $\varphi^* \wedge \varphi_{\leq}(A)$ is constructed over the alphabet $A \cup A'$ and T' already satisfies it. Further, for any state k restricted to A, U_k contains all and only the atoms from T_k, or more precisely, $U_k \cap A = T_k$. This holds because it already holds for T', by virtue of the fact that $T' \vDash \varphi_T$, and because the assignment of atoms in A of U is not different than that of T'.

Given that U is constructed over the alphabet $A \cup A'$, by Lemma 3.25 there exists a THT trace (H,X) such that $H \leq X$ and $(H,X)^* = U$. Since the atoms of X are determined exclusively by the atoms of U restricted to A and $U_k \cap A = T_k$, $X = T$, so that $(H,X) = (H,T)$. Further, given that $U = (H,T)^* \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A)$, by Lemma 3.26 $(H,T) \vDash_{\text{THT}} \varphi$. Hence $\exists H \leq T$ for which $(H,T) \vDash_{\text{THT}} \varphi$. ■

Of particular importance for the Pspace membership proof of Telmc is the following property.

Lemma 3.31 Let $T \stackrel{\text{def}}{=} T_1 \cdot \dots \cdot T_n \cdot (T_{n+1} \cdot \dots \cdot T_m)^\omega$ be a periodically infinite LTL trace and φ a THT formula. Then $\exists H < T$ such that $(H,T) \vDash_{\text{THT}} \varphi$ iff $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$ is LTL satisfiable. □

The proof of this lemma is analogous to that of Lemma 3.30, differing only by the fact that the formula $\varphi_{<}(A)$ is taken into account, but for the sake of completeness it is reported here without abbreviations.

Proof If some $H < T$ exists such that $(H,T) \vDash_{\text{THT}} \varphi$ and $M \stackrel{\text{def}}{=} (H,T)^*$, then by Lemma 3.28 $M \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$, meaning that $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$ is LTL satisfiable. $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$ can be shown to be LTL satisfiable by extending M by M', an LTL trace defined as

$$M' \stackrel{\text{def}}{=} (M_1 \cup \{s_1\}) \cdot \dots \cdot (M_n \cup \{s_n\}) \cdot ((M_{n+1} \cup \{s_{n+1}\}) \cdot \dots \cdot (M_m \cup \{s_m\}))^\omega.$$

Since $M \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$ and M' extends M with atoms outside the alphabet of $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$, $M' \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$. Further, $M' \vDash_{\text{LTL}} \varphi_T$ because, as

shown in the proof of Proposition 3.29, M' satisfies all of the subformulas of φ_T. Hence $M' \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$, meaning that $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$ is satisfiable.

Conversely, if $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$ is satisfiable, then there exists an LTL trace T' over the alphabet $A \cup A' \cup \{s_1, ..., s_m\}$ that satisfies it. From T', an LTL trace U restricted over the alphabet $A \cup A'$, defined as $U_k \stackrel{\text{def}}{=} T'_k \cap (A \cup A')$, can be obtained. This trace still satisfies $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$, because $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$ is constructed over the alphabet $A \cup A'$ and T' already satisfies it. Further, for any state k restricted to A, U_k contains all and only the atoms from T_k, or more precisely, $U_k \cap A = T_k$. This holds because it already holds for T', by virtue of the fact that $T' \vDash \varphi_T$, and because the assignment of atoms in A of U is not different than that of T'.

Given that U is constructed over the alphabet $A \cup A'$, by Lemma 3.27 there exists a THT trace (H, X) such that $H < X$ and $(H, X)^* = U$; since the atoms of X are determined exclusively by the atoms of U restricted to A and $U_k \cap A = T_k$, $X = T$, so that $(H, X) = (H, T)$. Further, given that $U = (H, T)^* \vDash_{\text{LTL}} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A)$, by Lemma 3.28 $(H, T) \vDash_{\text{THT}} \varphi$. Hence $\exists H < T$ for which $(H, T) \vDash_{\text{THT}} \varphi$. ■

Similarly to Proposition 3.29, this lemma allows to reduce the problem of searching for a non-total THT interpretation (H, T) that satisfies a given THT formula into a satisfiability problem.

THEOREM 3.32 $\overline{\text{TELMC}} \in$ PSPACE. □

PROOF To decide whether a given $(T, \varphi) \in \overline{\text{TELMC}}$, with $T = T_1 T_2^{\omega}$ a periodically infinite LTL trace and φ a THT formula, one of the following three conditions has to be verified:

1. (T, T) is not a total interpretation;
2. $(T, T) \nvDash_{\text{THT}} \varphi$, or
3. $\exists H < T \mid (H, T) \vDash_{\text{THT}} \varphi$.

The first condition is already falsified, because (T, T) is a total trace.

To check whether $(T, T) \nvDash_{\text{THT}} \varphi$, given that $(T, T) \vDash_{\text{THT}} \varphi$ iff $T \vDash_{\text{LTL}} \varphi$, a reduction to LTL model checking can be performed. Checking whether $T \vDash_{\text{LTL}} \varphi$ can be done in time $O((|T_1| + |T_2|) \cdot |\varphi|)$ (N. Markey and P. Schnoebelen [70]), which means it can also be done in polynomial space with respect to $(|T_1| + |T_2|) \cdot |\varphi|$.

To verify the third condition, the formula $\psi \stackrel{\text{def}}{=} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$ is constructed and checked for satisfiability. By Lemma 3.31, if ψ is satisfiable, then $\exists H < T$ such that $(H, T) \vDash_{\text{THT}} \varphi$, in which case (T, φ) is declared to belong to $\overline{\text{TELMC}}$. If, instead, ψ is unsatisfiable, by contraposition of Lemma 3.31 $\nexists H < T$ for which $(H, T) \vDash_{\text{THT}} \varphi$, and (T, φ) is declared not to belong to $\overline{\text{TELMC}}$.

ψ can be computed in polynomial time with respect to $(|T_1| + |T_2|) \cdot |\varphi|$, and its size is also polynomial with respect to this product. Since the problem of determining whether an LTL formula is satisfiable by an infinite trace is PSPACE–complete (A. P. Sistla and E. M. Clarke [55]), checking whether ψ is satisfiable can be carried out in polynomial space with respect to $(|T_1| + |T_2|) \cdot |\varphi|$. ■

For ease of understanding, the proof of Theorem 3.32 is illustrated with an example. The procedure implicitly contained in it is made explicit by Algorithm 4.

EXAMPLE 3.51 Let T be the LTL trace depicted in Figure 8 and φ the formula $\Box(\neg a \rightarrow \bigcirc a)$, both constructed over the alphabet $A = \{a\}$. It is already known that the unique TEM of φ is the trace $T' \stackrel{\text{def}}{=} (\emptyset \cdot \{a\})^{\omega}$ (P. Cabalar and S. Demri [52, pp. 5–6]), so, by exclusion, T is not a TEM of φ.

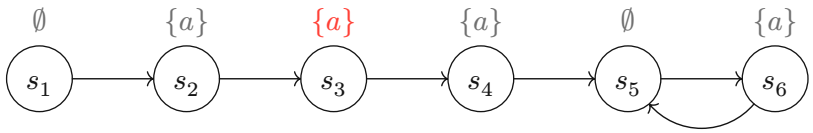


Figure 8: The trace T of Example 3.51.

The procedure illustrated in the proof of Theorem 3.32 starts by verifying whether $(T,T) \not\models_{\text{THT}} \varphi$ by checking whether $T \not\models_{\text{LTL}} \varphi$. However, $T \models_{\text{LTL}} \varphi$, so the third condition is investigated next.

To check this condition, the formula $\psi \stackrel{\text{def}}{=} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_T$ is constructed. This formula can be rewritten as

$$
\begin{aligned}
\psi = {} & \overbrace{\Box(((\neg a \wedge \neg a') \rightarrow \bigcirc a') \wedge (\neg a \rightarrow \bigcirc a))}^{\varphi^*} \\
& \wedge \overbrace{\Box(a' \rightarrow a)}^{\varphi_{\leq}(A)} \\
& \wedge \overbrace{\Diamond(\neg a' \wedge a)}^{\varphi_{<}(A)} \\
& \left.\begin{array}{l}
\wedge\, s_1 \\
\wedge\, \Box(s_1 \rightarrow \bigcirc s_2) \wedge \cdots \wedge \Box(s_6 \rightarrow \bigcirc s_5) \\
\wedge\, \Box(s_1 \rightarrow \neg a) \wedge \cdots \wedge \Box(s_3 \rightarrow a) \wedge \cdots \wedge \Box(s_6 \rightarrow a) \\
\wedge\, \Box(s_1 \rightarrow \neg s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5 \wedge \neg s_6) \\
\wedge\, \vdots \\
\wedge\, \Box(s_6 \rightarrow \neg s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5).
\end{array}\right\} \varphi_T
\end{aligned}
$$

The task is now to decide whether ψ is satisfiable by an LTL interpretation defined over the alphabet $A \cup A' \cup \{s_1, ..., s_6\}$. $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T$ is satisfiable, but $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_T \wedge \varphi_{<}(A)$ is also satisfiable. In fact, what the T-embedding φ_T does is enforce the satisfaction of $\Box(s_3 \rightarrow a)$, not of $\Box(s_3 \rightarrow a')$, so it is possible to satisfy $\varphi_{<}(A)$ by finding an H for which $H, s_3 \models_{\text{LTL}} \neg a'$ and $H, s_3 \models_{\text{LTL}} a$. One such trace H can, for instance, be

$$\{s_1\} \cdot \{s_2, a', a\} \cdot \{s_3, a\} \cdot \{s_4, a', a\} \cdot (\{s_5\} \cdot \{s_6, a', a\})^{\omega}.$$

Hence ψ is satisfiable, and by Lemma 3.31 $\exists H < T$ for which $(H,T) \models_{\text{THT}} \varphi$, showing T not to be a TEM of φ. □

```
1 Boolean ξ_{\overline{TELMC}} (Trace T, Formula φ)
2     if ((T, T) ⊭_THT φ)
3         return true
4     let ψ ← φ* ∧ φ_≤(A) ∧ φ_<(A) ∧ φ_T
5     if (ψ is satisfiable)
6         return true
7     else
8         return false
```

Algorithm 4: Decision algorithm for $\overline{\text{TELMC}}$.

COROLLARY 3.33 TELMC $\in$ PSPACE. □

PROOF Since $\overline{\text{TELMC}} \in$ PSPACE, TELMC $\in$ coPSPACE. But coPSPACE = PSPACE, so TELMC $\in$ PSPACE. ■

Chapter 4

Fragments of the TELMC problem

"When one teaches, two learn."
— Robert A. Heinlein.

The preceding chapter examined the complexity of model checking for TEL in two general settings, the one of finite-size interpretations and the one of infinite-size ones. Whereas for the finite-size case, TELMC_f, the complexity turns out to be coNP–complete—sitting just at the bottom of the polynomial hierarchy—the one of TELMC displays a far wider gap: Model checking for TELMC is PSPACE–complete. That model checking for TELMC_f is coNP–complete suggests that, in this case, the temporal operators confer no complexity to the problem, as the atemporal version is already coNP–hard.

But what about TELMC? By the results of Theorem 3.24, it is manifest that the presence of temporal operators contributes to the increased complexity of the problem, as it is instrumental in the encoding of deterministic space-bounded Turing machines. It will therefore be the aim of this chapter to investigate the complexity of this problem when the set of operators, in particular the set of temporal operators, is restricted in some appropriate way. A summary of the results of this chapter is shown in Table 6. A visual representation of some of these results is displayed in Figure 9.

Problem	Complexity
TELMC_2^1	PSPACE–complete (Corollary 4.34)
TELMC_1	coNP–complete (Theorem 4.42)
$\text{TELMC}(\bigcirc, \mathbb{U})$	coNP–complete (Theorem 4.44)
$\text{TELMC}(\bigcirc)$	coNP–complete (Corollary 4.47)
$\text{TELMC}(\mathbb{U})$	coNP–complete (Corollary 4.48)
$\text{TELMC}(\mathbb{U}, \mathbb{R})$	PSPACE–complete (Theorem 4.52)
$\text{TELMC}_2^1(\mathbb{U}, \mathbb{R})$	PSPACE–complete (Corollary 4.54)
$\text{TELMC}(\bigcirc, \mathbb{R})$	PSPACE–complete (Theorem 4.55)
$\text{TELMC}_2^1(\bigcirc, \mathbb{R})$	PSPACE–complete (Corollary 4.58)

Table 6: Summary of the results of the chapter.

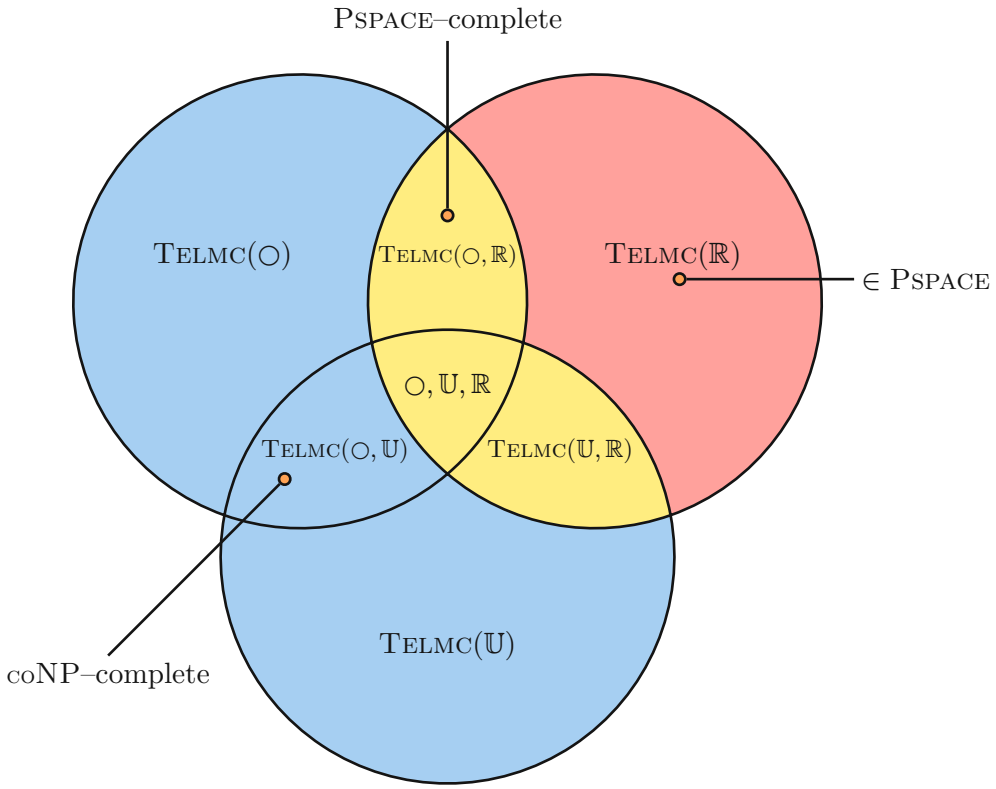


Figure 9: Visualisation of the complexity of the Telmc(S) problem, $S \subseteq \{\bigcirc, \mathbb{U}, \mathbb{R}\}$, $S \neq \emptyset$.

4.1. Basic setting

The restriction of the syntax of THT that will be taken under consideration is inspired from L. Bozzelli and D. Pearce [53]. For a THT formula φ, the *implication height* of φ is the maximum number of nested implication operators in φ, keeping in mind that $\neg\psi$ is a syntactic sugar for $\psi \rightarrow \bot$. For instance, the formula $\varphi = a \rightarrow (b \rightarrow c)$ has implication height equal to 2, whereas the formula $\varphi = a \rightarrow (b \rightarrow \neg c)$ has implication height equal to 3. Likewise, the *temporal height* of φ is the maximum number of nested temporal operators in it. For example, $\varphi = a \,\mathbb{U}\, \bigcirc b$ has temporal height equal to 2, and $\Box\Diamond a \rightarrow \Diamond a$ also has temporal height 2.

By THT_k^m, then, the syntactic fragment of THT where all formulas have temporal height at most k and implication height at most m is meant. Further, $\mathrm{THT}(f_1, ..., f_n)$ denotes the syntactic fragment where only the temporal operators $f_1, ..., f_n$ are allowed to occur. For instance, $\mathrm{THT}(\bigcirc, \mathbb{R})$ refers to the syntactic fragment of THT where only the temporal operators $\bigcirc$ and $\mathbb{R}$ are allowed. The two types of restrictions can be simultaneously employed together. $\mathrm{THT}_2^1(\bigcirc, \mathbb{U})$ denotes, for instance, the THT fragment of temporal height 2 and implication height 1 making use of the temporal operators $\bigcirc$ and $\mathbb{U}$ only.

The chapter will investigate the complexity of Telmc under this line of restriction, in the attempt of both identifying tractable fragments and determining when intractability begins to emerge. Under this setting, one immediate negative result can already be stated.

Corollary 4.34 Deciding Telmc_2^1 is Pspace–complete. □

Proof The proof of Theorem 3.24, showing the Pspace–hardness of Telmc, employs formulas whose temporal height does not exceed 2, and whose implication height does

not exceed 1. Hence $\textsc{Telmc}_2^1$ is Pspace–hard. Further, since $\textsc{Telmc}_2^1 \subset \textsc{Telmc}$ and by Corollary 3.33 $\textsc{Telmc} \in \textsc{Pspace}$, $\textsc{Telmc}_2^1 \in \textsc{Pspace}$. ■

A relatively shallow temporal height seems therefore to already impact the complexity of model checking for TEL in a considerable way. What, then, might be said of the immediately preceding syntactic fragment with respect to the temporal height, THT_1?

4.2. The $\textsc{Telmc}_1$ fragment

THT_1 is the syntactic fragment of THT where all formulas have temporal height at most 1. One important characteristic of this fragment is that for any temporal formula $\bigcirc\varphi$, $\varphi_1 \mathbb{U} \varphi_2$ and $\varphi_1 \mathbb{R} \varphi_2$, φ, φ_1 and φ_2 are all formulas without temporal operators in them. It will be shown next how, for these kind of formulas, any LTL or THT interpretation can be reduced down to a restricted number of states, thus making model checking for TEL computationally easier.

Definition 4.52 An *extraction pattern* is an infinite sequence $P = (n_0, n_1, ...)$ of strictly increasing natural numbers. Given a THT interpretation $M = (H, T)$, the *extraction* of M over P is the THT interpretation $M_P = M_{n_0} M_{n_1} \cdots$. □

An extraction pattern is, by default, an infinite sequence. When otherwise specified, however, finite extraction patterns are also considered.

Definition 4.53 Given a THT_1 formula φ and a periodically infinite THT interpretation $M = (H, T) = (H_1 H_2^\omega, T_1 T_2^\omega)$, a *witness pattern* of M for φ is *one of* the shortest finite extraction patterns W satisfying the following conditions:

1. $0 \in W$;
2. If M is not total, then $i \in W$, where i is such that $H_i \subset T_i$;
3. If $\bigcirc\psi$ is a subformula of φ, then $1 \in W$;
4. If $\varphi_1 \mathbb{U} \varphi_2$ is a subformula of φ:
   - If $M \vDash \varphi_1 \mathbb{U} \varphi_2$, then the smallest position i such that $M, i \vDash \varphi_2$ is in W;
   - If $M \nvDash \varphi_1 \mathbb{U} \varphi_2$ and $M \vDash \Diamond\varphi_2$, then the smallest position i for which $M, i \nvDash \varphi_1$ is in W;
5. If $\varphi_1 \mathbb{R} \varphi_2$ is a subformula of φ:
   - If $M \vDash \varphi_1 \mathbb{R} \varphi_2$ and $M \nvDash \Box\varphi_2$, then the smallest position i for which $M, i \vDash \varphi_1 \wedge \varphi_2$ is in W;
   - If $M \nvDash \varphi_1 \mathbb{R} \varphi_2$, then the smallest position i for which $M, i \nvDash \varphi_2$ is in W.

□

To make the discourse more concrete, an example is provided.

Example 4.54 Let $\varphi = ((a \vee \neg c) \mathbb{U} b) \wedge (b \mathbb{R} d) \wedge \bigcirc d$ be a THT_1 formula and $M = (H, T)$ be a THT interpretation, where

$$H = \{a, d\} \cdot \{e, d\} \cdot \quad \{a, c\} \quad \cdot \{b\} \cdot (\emptyset \cdot \{b\} \cdot \{c\})^\omega,$$
$$T = \{a, d\} \cdot \{e, d, a\} \cdot \{a, c, d\} \cdot \{b\} \cdot (\{a\} \cdot \{b\} \cdot \{c\})^\omega.$$

Then a witness pattern W of M for φ is $W = (0, 1, 2, 3)$. □

Proposition 4.35 Given a THT_1 formula φ and an interpretation $M = (H, T)$, a witness pattern W of M for φ exists. □

Proposition 4.36 Given a THT_1 formula φ and an interpretation $M = (H, T)$, each witness pattern W of M for φ has length at most $|\text{sub}(\varphi)| + 2$, where $\text{sub}(\varphi)$ denotes the set of subformulas of φ. □

Proof By Definition 4.53, W contains 0. Further, if $H < T$, then an index i is also contained in W for which $H_i \subsetneq T_i$. This gives a total of 2 indices contained in W. All the other indices occurring in the set W of Definition 4.53 (in other words, occurring before the periodicity of W) are determined by the number of temporal subformulas $\bigcirc\varphi_1$, $\varphi_1 \mathbb{U} \varphi_2$ and $\varphi_1 \mathbb{R} \varphi_2$ of φ, which is bounded by $|\mathrm{sub}(\varphi)|$. ■

Lemma 4.37 Let φ be a THT_1 formula, $M = (H, T)$ a periodically infinite THT interpretation and W a witness pattern of M for φ. Then for each extraction pattern P containing all positions of W, $M \vDash \varphi$ iff $M_P \vDash \varphi$. □

Proof By structural induction over the syntax of φ. In the following, it is assumed that $P \stackrel{\text{def}}{=} (n_0, n_1, ...)$.

If $\varphi = \bot$, then clearly $M \vDash \varphi$ iff $M_P \vDash \varphi$.

If $\varphi = a$, then since $0 \in P$, if $M \vDash \varphi$ then $M_P \vDash \varphi$. Vice versa, if $M_P \vDash a$, then clearly $M \vDash a$, because the set of states of M_P is a subset of the ones of M.

If $\varphi = \varphi_1 \vee \varphi_2$, then if $M \vDash \varphi$, $M \vDash \varphi_1$ or $M \vDash \varphi_2$. Without loss of generality, $M \vDash \varphi_1$ is supposed. Then, by inductive hypothesis (IH), $M_P \vDash \varphi_1$, meaning that $M_P \vDash \varphi_1 \vee \varphi_2 \equiv \varphi$. Vice versa and in an analogous way, if $M_P \vDash \varphi$, then either $M_P \vDash \varphi_1$ or $M_P \vDash \varphi_2$. If, without loss of generality, $M_P \vDash \varphi_1$, then by inductive hypothesis $M \vDash \varphi_1$, meaning $M \vDash \varphi_1 \vee \varphi_2$.

If $\varphi = \varphi_1 \wedge \varphi_2$, a reasoning similar to the $\varphi = \varphi_1 \vee \varphi_2$ case is followed.

If $\varphi = \varphi_1 \rightarrow \varphi_2$ and $M \vDash \varphi$, then

1. $(H, T) \vDash \varphi_1$ implies $(H, T) \vDash \varphi_2$, and
2. $(T, T) \vDash \varphi_1$ implies $(T, T) \vDash \varphi_2$.

What needs to be proven is that this same relation also holds for M_P. Let then $(H_P, T_P) \vDash \varphi_1$, then

$$\begin{aligned} (H, T) &\vDash \varphi_1 && \text{(IH)}, \\ (H, T) &\vDash \varphi_2 && \text{(property 1.)}, \\ (H_P, T_P) &\vDash \varphi_2 && \text{(IH)}. \end{aligned}$$

If $(T_P, T_P) \vDash \varphi_1$, then by the IH applied to the (T, T) trace $(T, T) \vDash \varphi_1$, and by point 2. above, $(T, T) \vDash \varphi_2$. By the IH applied to the (T, T) trace again, $(T_P, T_P) \vDash \varphi_2$.

If $(H_P, T_P) \vDash \varphi_1 \rightarrow \varphi_2$, then

1. $(H_P, T_P) \vDash \varphi_1$ implies $(H_P, T_P) \vDash \varphi_2$, and
2. $(T_P, T_P) \vDash \varphi_1$ implies $(T_P, T_P) \vDash \varphi_2$.

What needs to be proven is that this same relation also holds for M_P. If $(H, T) \vDash \varphi_1$, then

$$\begin{aligned} (H_P, T_P) &\vDash \varphi_1 && \text{(IH)}, \\ (H_P, T_P) &\vDash \varphi_2 && \text{(property 1.)}, \\ (H, T) &\vDash \varphi_2 && \text{(IH)}. \end{aligned}$$

Finally, if $(T, T) \vDash \varphi_1$, then by the IH applied to the (T, T) trace, $(T_P, T_P) \vDash \varphi_1$, and by point 2. above, $(T_P, T_P) \vDash \varphi_2$. By the IH applied to the (T, T) trace again, $(T, T) \vDash \varphi_2$.

If $\varphi = \bigcirc\varphi_1$, then $1 \in P$. If $M \vDash \bigcirc\varphi_1$, $M, 1 \vDash \varphi_1$, and since $1 \in P$, $M_P \vDash \bigcirc\varphi_1$. Vice versa, if $M_P \vDash \bigcirc\varphi_1$, then since $0 \in P$ and $1 \in P$, $M \vDash \bigcirc\varphi_1$.

If $\varphi = \varphi_1 \mathbb{U} \varphi_2$ then if $M \vDash \varphi_1 \mathbb{U} \varphi_2$, a smallest $k \geq 0$ exists for which $M, k \vDash \varphi_2$ and $M, k' \vDash \varphi_1$ for all $k' \in [0..k)$. Since $M \vDash \varphi_1 \mathbb{U} \varphi_2$, by Definition 4.53 $k \in P$, and since M_P is a subset of the states of M where φ_1 already holds, $M_P \vDash \varphi_1 \mathbb{U} \varphi_2$. The opposite direction is proven by contraposition, by assuming $M \nvDash \varphi_1 \mathbb{U} \varphi_2$. If $M \nvDash \varphi_1 \mathbb{U} \varphi_2$ because φ_2 never

holds, that is because $M \not\models \Diamond\varphi_2$, then the same also happens for M_P and $M_P \not\models \varphi_1 \mathbb{U} \varphi_2$. If, instead, $M \models \Diamond\varphi_2$, specifically $M, k \models \varphi_2$ for some position $k \geq 1$, then $M \not\models \varphi_1 \mathbb{U} \varphi_2$ holds because there exists a position $i < k$ for which $M, i \not\models \varphi_1$. Let i be the smallest position for which $M, i \not\models \varphi_1$; then, by Definition 4.53 $i \in P$ and $i = n_j$ for some $j \geq 0$. Clearly, $M_P, n_j \not\models \varphi_1$, and $M_P, n_{j'} \not\models \varphi_2$ for all $j' < j$, meaning that φ_2 holds in M_P at position n_{j+1} or later, if it ever does. Hence, $M_P \not\models \varphi_1 \mathbb{U} \varphi_2$.

Finally, if $\varphi = \varphi_1 \mathbb{R} \varphi_2$, then either $M \models \Box\varphi_2$ or $M \not\models \Box\varphi_2$. In the first case, since M_P is a subset of the states of M, $M_P \models \Box\varphi_2$ also holds, meaning that $M_P \models \varphi_1 \mathbb{R} \varphi_2$. In the second, by Definition 4.53 the smallest position i for which $M, i \models \varphi_1 \wedge \varphi_2$ is in P, meaning that $i = n_j$ for some $j \geq 0$. By definition of the release operator, $M, i' \models \varphi_2$ for all $i' \leq i$, and the same also holds for M_P, that is $M_P, n_{j'} \models \varphi_2$ for all $j' \leq j$. Hence $M_P \models \varphi_1 \mathbb{R} \varphi_2$. The other direction is shown by contraposition. If $M \not\models \varphi_1 \mathbb{R} \varphi_2$, then by Definition 4.53, the smallest position i such that $M, i \not\models \varphi_2$ is in P, and the position k for which $M, k \models \varphi_1$, if it exists, is such that $i < k$. Since $i \in P$, $i = n_j$ for some $j \geq 0$; therefore $M_P, n_j \not\models \varphi_2$ and, similarly to M, the position $n_{j'}$ at which $M_P, n_{j'} \models \varphi_1$ is greater than n_j, meaning that $M_P \not\models \varphi_1 \mathbb{R} \varphi_2$. ■

LEMMA 4.38 Let φ be a THT$_1$ formula and $T = T_1 T_2^\omega$ an LTL interpretation. Then for any arbitrary LTL trace $H \leq T$, an extraction $H' = H_1' H_2'^\omega$ of H exists such that $|H_1'| \leq |\text{sub}(\varphi)| + 2$, $|H_2'| = |T_2|$ and $(H, T) \models \varphi$ iff $(H', T) \models \varphi$. □

PROOF Let $M = (H, T)$. By Proposition 4.35, a witness pattern $W = (w_0, ..., w_k)$ of M for φ exists, and by Proposition 4.36, W has size at most $|\text{sub}(\varphi)| + 2$. Additionally, an extraction pattern $P = (w_0, ..., w_k, n_{k+1}, n_{k+2}, ...)$ exists which extends W to an infinite sequence, giving rise to a THT interpretation $M_P = (H_W H_2'^\omega, T_W T_2^\omega)$, whose H_W segment has length at most $|\text{sub}(\varphi)| + 2$ and whose $H_2' = H_{n_{k+1}} H_{n_{k+2}} \cdots$ segment has length $|T_2|$. Since P contains all the positions of the witness pattern W, by Lemma 4.37 $M \models \varphi$ iff $M_P \models \varphi$. ■

THEOREM 4.39 Let φ be a THT$_1$ formula and $T = T_1 T_2^\omega$ a periodically infinite LTL trace such that $T \models \varphi$. Then T is an equilibrium model of φ iff for all traces $H = H_1 H_2^\omega$, $H < T$, such that $|H_1| + |H_2| \leq |\text{sub}(\varphi)| + 2 + |T_2|$, $(H, T) \not\models \varphi$. □

PROOF If T is an equilibrium model of φ, then for any $H = H_1 H_2^\omega$, $H < T$, $(H, T) \not\models \varphi$ holds, including the ones for which $|H_1| + |H_2| \leq |\text{sub}(\varphi)| + 2 + |T_2|$.

The other direction is shown by contraposition. Suppose T not to be an equilibrium model of φ. Then, by definition, a trace $H < T$ exists such that $(H, T) \models \varphi$. By Lemma 4.38, a THT trace $S = (H_1 H_2^\omega, T_1' T_2^\omega)$ exists such that H_1 has length at most $|\text{sub}(\varphi)| + 2$, $|H_2| = |T_2|$ and $S \models \varphi$. ■

THEOREM 4.40 TELMC$_1$ $\in$ coNP. □

PROOF Given a THT$_1$ formula φ and an LTL interpretation $T = T_1 T_2^\omega$, a trace $H = H_1 H_2^\omega$ such that $|H_1| \leq |\text{sub}(\varphi)| + 2$, $|H_2| = |T_2|$ and $H < T$ can be guessed in nondeterministic polynomial time. Deciding whether $(H, T) \models \varphi$ can be subsequently decided in deterministic polynomial time. If $(H, T) \models \varphi$, then (T, φ) is declared not to belong to TELMC$_1$. ■

PROPOSITION 4.41 Deciding TELMC$_1$ is coNP–hard. □

PROOF Similarly as in the coNP–hardness proof of TELMC$_f$ (Proposition 3.18). ■

THEOREM 4.42 Deciding TELMC$_1$ is coNP–complete. □

PROOF Consequence of Proposition 4.41 and Theorem 4.40. ■

4.3. The Telmc($\bigcirc, \mathbb{U}$) fragment

Telmc($\bigcirc, \mathbb{U}$) is the fragment of the Telmc problem where only the $\bigcirc$ and $\mathbb{U}$ operators, within any arbitrary nesting depth, are allowed to occur.

A moment of reflection should suffice to realize that, under these specific constraints, no TEM of a THT($\bigcirc, \mathbb{U}$) formula has infinite length. For a THT interpretation to be so, an infinite number of propositions needs to be enforced[13], but this is not possible with the $\bigcirc$ or $\mathbb{U}$ operators. A THT interpretation M is, in fact, a TEM of a formula $\bigcirc\varphi$ if φ holds at second state of it; all atoms at states other than it are superfluous, and will be minimized by the TEL semantics. Similarly, M is a TEM of a formula $\varphi_1 \mathbb{U} \varphi_2$ if φ_2 holds at a certain state k, and φ_1 holds at all states j, $0 \leq j < k$. Atoms present after state k do not affect the satisfaction of φ_2, and will be minimized by semantics of TEL. Similarly, atoms occurring at one of the states j that do not contribute to the satisfaction of φ_1 will be minimized.

This intuition is made precise by the following notions. Given an interpretation $M = (H, T)$, a position i of M is said to be *empty* if $H_i = \emptyset$, and *non-empty* otherwise. A total interpretation $M = (T, T)$ is said to be *almost-empty* if its number of non-empty positions is finite[14].

L. Bozzelli and D. Pearce [53] offers a result for the THT($\bigcirc, \mathbb{U}$) fragment that is directly applicable to the case of Telmc($\bigcirc, \mathbb{U}$).

Lemma 4.43 (L. Bozzelli and D. Pearce [53, Lemma IV.3].) *Let φ be a* THT($\bigcirc, \mathbb{U}$) *formula and M be an equilibrium model of φ. Then, M is almost-empty.* □

By contraposition of this statement, any total interpretation $M = (T, T)$ which is not almost-empty is not a TEM of a THT($\bigcirc, \mathbb{U}$) formula. Model checking for infinite traces is therefore trivial, while for finite ones it can be solved in nondeterministic polynomial time.

Theorem 4.44 Deciding Telmc($\bigcirc, \mathbb{U}$) is coNP–complete. □

Proof Consequence of Proposition 4.45 and Proposition 4.46. ■

Proposition 4.45 Telmc($\bigcirc, \mathbb{U}$) $\in$ coNP. □

Proof Given a pair $(T, \varphi) \in \Sigma_{\text{Telmc}(\bigcirc,\mathbb{U})}$, where $T = T_1 T_2^{\omega}$ is a periodically infinite trace and φ a THT($\bigcirc, \mathbb{U}$) formula, both constructed over an alphabet A, it can be first checked whether $T_2 = \emptyset$. If $T_2 \neq \emptyset$, then by Lemma 4.43 $(T, \varphi) \notin$ Telmc($\bigcirc, \mathbb{U}$). Otherwise, if $T_2 = \emptyset$, a guess of an $H < T$ trace can be made in nondeterministic polynomial time. Since T has a finite number of non-empty positions, the number of possible H traces is at most $2^{|A| + \log_2 |T_1|}$, and hence finite[15]. Once a specific trace H has been obtained, it can be checked in deterministic polynomial time whether $(H, T) \vDash_{\text{THT}} \varphi$ or not. If $(H, T) \vDash_{\text{THT}} \varphi$, then (T, φ) is declared not to belong to Telmc($\bigcirc, \mathbb{U}$). If, otherwise, $\forall H, (H, T) \nvDash_{\text{THT}} \varphi$, (T, φ) is declared to belong to Telmc($\bigcirc, \mathbb{U}$). ■

Proposition 4.46 Deciding Telmc($\bigcirc, \mathbb{U}$) is coNP–hard. □

Proof Similarly as in the coNP–hardness proof of Telmc$_f$ (Proposition 3.18). ■

[13] Or *justified*, in an ASP sense.

[14] Since M is a total interpretation, if the *here* trace is empty at some position, then the *there* one also is.

[15] The H trace can be constructed by guessing, for each position i in T_1 and atom a present in this position, whether a is included in H_i or not. Since each state T_i of T has at most $|A|$ atoms in it, each state of H can be constructed in $2^{|A|}$ possible ways, and since $|T_1|$ many states in H need to be guessed, the total number of possible H traces is $2^{|A|} \cdot |T_1|$, or equivalently $2^{|A| + \log_2 |T_1|}$.

COROLLARY 4.47 Deciding TELMC(○) is coNP–complete. □

COROLLARY 4.48 Deciding TELMC(𝕌) is coNP–complete. □

4.4. The TELMC(𝕌, ℝ) fragment

TELMC(𝕌, ℝ) is another relevant fragment of the TELMC problem, in which the presence of the ○ operator is prohibited, and only the 𝕌 and ℝ operators, within any arbitrary implication height, can be used. Studying the complexity of deciding TELMC(𝕌, ℝ) can be done by relying on what is already known about the complexity of *tilings*, a class of problems whose task is to determine whether a so-called *tiling* exists for a certain set of elements called *tiles*. P. v. E. Boas [59] and F. Schwarzentruber [60] offer several details about this type of problems.

Given a countable set of colours C, a *tile* is a quadruple $z \in C^4$. z_l, z_t, z_r and z_b denote the left, top, right and bottom side of z, respectively.

DEFINITION 4.55 Given a finite set of colours C, a set of tiles $T \subseteq C^4$ and a width and height parameters $w, h \in \mathbb{N}$, a *tiling* of the $w \times h$ plane is a function $f : w \times h \to T$, assigning to each *cell* of the $w \times h$ plane a tile from T. f is said to be *admissible* if

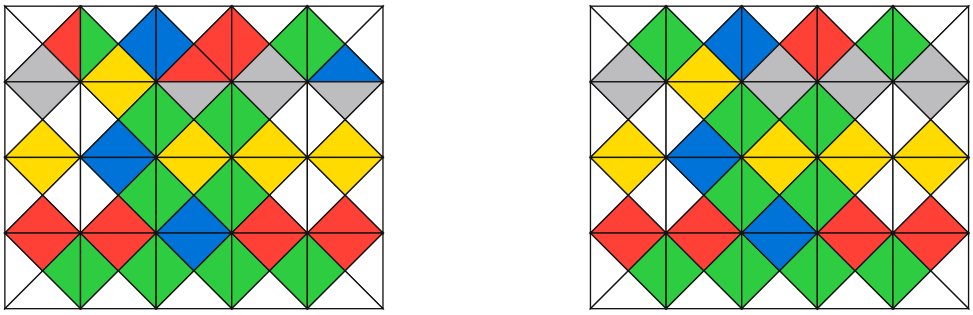
1. $\forall i \in [0..w), \forall j,\ f(i,j)_r = f(i+1,j)_l$;
2. $\forall i, \forall j \in [0..h),\ f(i,j)_b = f(i,j+1)_t$.

□

An example of two tilings over the 4×5 grid, only one of which is admissible, is given in Figure 10. Different decision problems over tilings can be defined, but for the purposes of the current section only a specific one is relevant.

DEFINITION 4.56 *(Corridor tiling problem.)* The *corridor tiling* problem is the language $\text{CT} \stackrel{\text{def}}{=} \{(X, U, B)\}$, where

1. X is a set of tiles;
2. U and B are sequences of tiles from X of length $n \in \mathbb{N}$;
3. An integer m exists which admits an admissible tiling of the $m \times n$ plane, where
    1. The top edge corresponds to U;
    2. The bottom edge corresponds to B;
    3. All tiles on the left and right sides have white sides.



(a) A 4×5 tiling. (b) An admissible 4×5 tiling.

Figure 10: Two 4×5 tilings.

□

In other words, given a set of tiles X and two segments U and B of n tiles from X, the corridor tiling problem asks whether an admissible tiling for the $m \times n$ plane exists for some $m \in \mathbb{N}$, where the top segment corresponds to U and the lower one to B. P. v. E. Boas [59] shows that deciding this problem is Pspace–complete.

One slight variant of the corridor tiling problem is the recurring one, where the upper and lower segments are required to be the same. Finding an admissible tiling for a given instance of this problem is, in a way, equivalent to tiling a cylinder[16], for if the B edge corresponds to the U one, then the same tiling that was used from U to B the first time can be repeated an infinite number of times[17].

Definition 4.57 *(Recurring corridor tiling problem.)* The *recurring corridor tiling* problem is the language $\textsc{Rct} \stackrel{\text{def}}{=} \{(X, S)\}$, where

1. X is a set of tiles;
2. S is a sequence of tiles from X of length $n \in \mathbb{N}$;
3. An integer m exists which admits an admissible tiling of the $m \times n$ plane, where
    1. The top and bottom edges corresponds to S;
    2. All tiles on the left and right sides have white sides.

□

As it turns out, considering a subset of the admissible tilings of the Ct problem in the way it is done by the Rct one does not affect its computational complexity. Like the Ct problem, the Rct problem is in fact also Pspace–complete, but it lends more easily in the reduction to the $\textsc{Telmc}(\mathbb{U}, \mathbb{R})$ problem, as it will be seen in the proof of Theorem 4.53.

Proposition 4.49 Deciding Rct is Pspace–complete. □

Proof Consequence of Proposition 4.50 and Proposition 4.51. ■

Proposition 4.50 Deciding Rct is Pspace–hard. □

Proof idea A visual illustration of the proof of this statement is given in Figure 11, where an instance of the Ct problem composed of two sequences U and B of 8 tiles each is given, and where X is assumed to be the set of tiles appearing in U and B.

To the $I = (X, U, B)$ instance, the $I' = (X', S)$ instance of the Rct problem is associated. The new set of tiles X' is obtained by adding to the old set of tiles X a new, special tile G, n new tiles obtained from U and n other new tiles obtained from B. The sequence S of tiles is defined as the tile G laid next to itself 8 times.

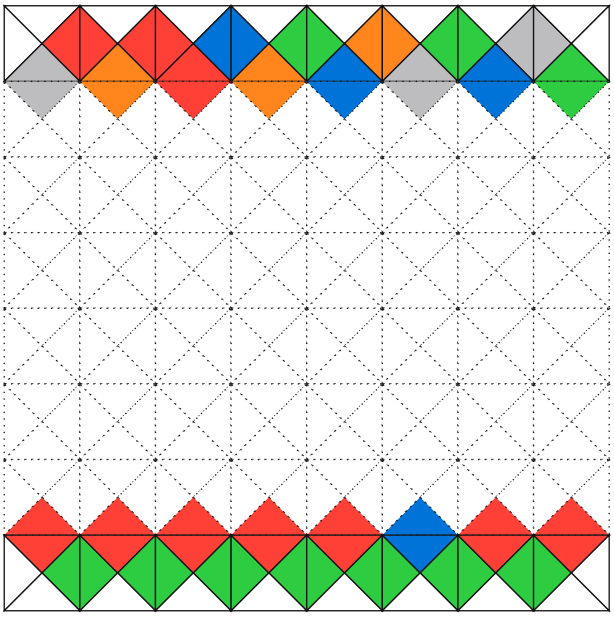
The n new tiles obtained from U have a dark shade of gold on top, the same colours as the tiles from U at the bottom and a sequence of distinct shades of light blue in between them. Similarly, the n tiles obtained from B have a light shade of gold at the bottom, the same colours as the tiles of B on top and a sequence of distinct shades of fuchsia in between them.

If the I instance admits an admissible tiling, then the I' one also admits one by filling the empty space between the sequence of tiles at the extremities with exactly the same tiles used for I. Vice versa, if the I' instance admits an admissible tiling, then the I one

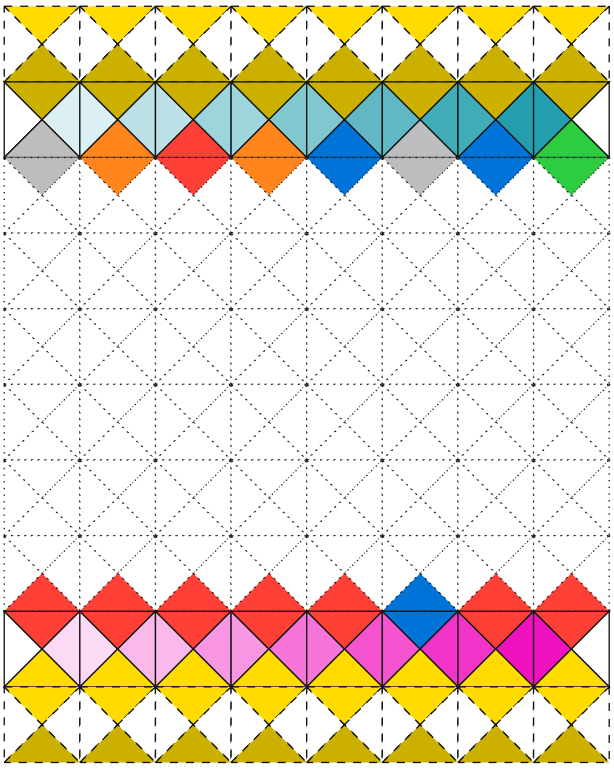
[16] In fact, since the constraint that the left and right sides of an admissible tiling be white is imposed, this variant is equivalent to the tiling of a torus.

[17] Although, clearly, it does not have to.

also admits one. In fact, none of the special, new tiles introduced in X' can be used to find an admissible tiling for I', meaning that only the old ones can be used. The same arrangement of tiles used for I' can therefore be also used for I. ■



(a) An instance $I = (X, U, B)$ of the CT problem.



(b) A corresponding instance $I' = (X', S)$ of the RCT problem. The shaded colours in the second and second to last rows are assumed to be all distinct from each other.

Figure 11: Proof idea of Proposition 4.50.

Proof To show the Pspace–hardness of Rct, a Karp reduction f from Σ_{Ct} to Σ_{Rct} is defined such that $I \in \text{Ct}$ iff $f(I) \in \text{Rct}$.

Given an instance $I \in \Sigma_{\text{Ct}}$, $I = (X, U, B)$, an instance $I' \stackrel{\text{def}}{=} f(I) = (X', S)$ is defined in which $2n$ new colours d_i not appearing in X are introduced, and in which $2n+1$ new tiles are also introduced. n of these tiles correspond to U, and are used to define a new sequence U' of tiles; the other n correspond to B, and are used to define a new sequence B' of tiles; the final tile is a special tile and is used to construct S.

Suppose $U = (u_1, ..., u_n)$ to be the sequence of tiles composing U, so that $u_i = (l_i, t_i, r_i, b_i)$. Then n new tiles u'_i are introduced in which the upper edges of the corresponding u_i tiles are replaced by the d_1 colour, and in which horizontally adjacent colours of successive tiles are replaced by a unique colour. In other words, if

$$U = ((w, t_1, r_1, b_1), (r_1, t_2, r_2, b_2), ..., (r_{n-1}, t_n, w, b_n)),$$

then

$$U' \stackrel{\text{def}}{=} ((w, d_1, d_2, b_1), (d_2, d_1, d_3, b_2), ..., (d_n, d_1, w, b_n)).$$

A similar transformation is applied to B. If $B = (b_1, ..., b_n)$, then n new tiles b'_i are introduced in which the lower edges of the corresponding b_i tiles are replaced by the d_{2n} colour, and in which horizontally adjacent colours of successive tiles are placed by another unique colour. Similarly as before, if

$$B = ((w, t_1, r_1, b_1), (r_1, t_2, r_2, b_2), ..., (r_{n-1}, t_n, w, b_n)),$$

then

$$B' \stackrel{\text{def}}{=} ((w, t_1, d_{n+1}, d_{2n}), (d_{n+1}, t_2, d_{n+2}, d_{2n}), ..., (d_{2n-1}, t_n, w, d_{2n})).$$

The sequence S of tiles is then constructed in a way that its lower edges can be matched to the upper edges of the U' sequence, while its upper edges can be matched to the lower edges of the B' sequence:

$$S \stackrel{\text{def}}{=} ((w, d_{2n}, w, d_1), (w, d_{2n}, w, d_1), ..., (w, d_{2n}, w, d_1)).$$

What needs to be proven at this point is that f is a function from Σ_{Ct} to Σ_{Rct}, that $I \in \text{Ct}$ iff $f(I) \in \text{Rct}$ and that f is computable in polynomial time.

f is a function from Σ_{Ct} to Σ_{Rct}, because the instance $I' = (X', S)$ that is produced consists of a set of tiles X' and a sequence of tiles of length n.

To prove the second point, $I \in \text{Ct}$ is supposed. If $I \in \text{Ct}$, then a natural number $m \in \mathbb{N}$ exists for which the $m \times n$ plane admits a tiling composed of tiles from X, and whose upper and lower edges correspond to U and B, respectively. From this tiling, a corresponding one of dimension $(m+2) \times n$ using the tiles from $X' \supset X$ and having S as an upper and lower edge exists. In fact, if S is used as an upper edge of the tiling, the second row of it can (in fact, it *has* to) be matched by the tiles from U'. Similarly, if S is used as a lower edge of the tiling, then the second-to-last row has to be matched by the sequence of tiles from B'. The lower edge of U' corresponds to the lower edge of U, and similarly, the upper edge of B' corresponds to the upper edge of B, so the tiles lying between U' and B' can be placed in exactly the same way as they were for the I instance.

To show the other direction, contraposition is used. If $I \notin \text{Ct}$, then no $m \in \mathbb{N}$ exists for which a tiling of the $m \times n$ plane using the tiles from X and having U and B as upper and lower edges, respectively, exists. Imposing the constraint that the upper and lower edges of a tile correspond to S is, as has already been seen, equivalent to imposing that the second row matches U', and that the second-to-last one matches B'. None of the newly

introduced tiles from X' can be used to fill the intermediate area lying between these two rows. The single tile composing S, in fact, is of the type (w, d_{2n}, w, d_1), whereas the tiles introduced to define U' and B' can only be laid to construct U' and B', respectively. Hence, since none of the tiles from X can also be used to tile the area between U' and B', no tiling for the I' instance exists, and $I' \notin$ RCT.

Finally, f can be computed in polynomial time, because it consists of adding $2n + 1$ new tiles and $2n$ new colours. ■

PROPOSITION 4.51 RCT ∈ PSPACE. □

PROOF RCT can be reduced to CT. Given an instance $I = (X, S)$ of the RCT problem, a new instance $I' = (X, S, S)$ of the CT one is constructed. Clearly, $I \in$ RCT iff $I' \in$ CT. Since CT ∈ PSPACE, RCT ∈ PSPACE too. ■

THEOREM 4.52 Deciding TELMC(𝕌, ℝ) is PSPACE–complete. □

PROOF Consequence of Theorem 4.53 and Corollary 3.33. ■

THEOREM 4.53 Deciding TELMC(𝕌, ℝ) is PSPACE–hard. □

PROOF To show the PSPACE–hardness of TELMC(𝕌, ℝ), a Karp reduction f from the $\overline{\text{RCT}}$ problem to TELMC(𝕌, ℝ) is performed. $\overline{\text{RCT}}$ is the complement of RCT, and asks whether for a sequence of tiles S of length n *no* height $m \in \mathbb{N}$ exists such that the $m \times n$ plane admits a tiling having S as upper and lower edges. By Observation 3.44, since the RCT problem is PSPACE–hard, the $\overline{\text{RCT}}$ problem is also PSPACE–hard.

Given an instance $I = (X, S)$ of the $\overline{\text{RCT}}$ problem, f associates to it an instance $I' = (T, \varphi)$ of the TELMC(𝕌, ℝ) problem, such that I does not admit a tiling iff T is a TEM of φ. The trace T is used to encode a grid of width $n \stackrel{\text{def}}{=} |S|$ and infinite height, by using each state to encode a cell of a tiling. Each state is marked by a column number from 0 to $n - 1$, a tile from X and a chromatic color of either white or black, so that the states forming a row are all either white and black, and so that white and black rows alternate each other. More precisely, the alphabet composing φ and T is made up of

1. n column atoms c_i, $0 \leq i < n$;
2. $|X|$ tile atoms $x \in X$;
3. Two chromatic atoms w and b.

For a chromatic atom $v \in \{w, b\}$, $\overline{v}$ denotes the opposite of v; that is $\overline{v} = b$ if $v = w$ and $\overline{v} = w$ if $v = b$.

If the instance $I = (X, S)$ is such that $X = \{x_1, ..., x_r\}$ and $S = (s_1, ..., s_n)$, then T is defined as

$$T \stackrel{\text{def}}{=} \left(\underbrace{\{w, x_1, ..., x_r, c_0\} \cdot ... \cdot \{w, x_1, ..., x_r, c_{n-1}\}}_{\text{white row}} \cdot \underbrace{\{b, x_1, ..., x_r, c_0\} \cdot ... \cdot \{b, x_1, ..., x_r, c_{n-1}\}}_{\text{black row}} \right)^{\omega}.$$

The formula φ obtained by the reduction is split into a number of subformulas, whose intent is summarised by the following table:

Formula	Description
φ_1	Horizontally adjacent cells have to share the same colour
φ_2	Vertically adjacent cells have to share the same colour
φ_3	At least one tile
φ_4	No more than one tile
φ_5	At least one column variable
φ_6	At least one chromatic variable
φ_7	The left side of the cells in the first column has to be white
φ_8	The right side of the cells in the last column has to be white
φ_9	The first row has to correspond with S
φ_{10}	Infinitely many rows have to correspond with S

Some of these formulas rely on the auxiliary formula

$$\varphi^{\uparrow} \stackrel{\text{def}}{\equiv} \Box \bigwedge_{x \in X} x,$$

which is used to induce a state of saturation, imposing (T, T) as the only THT model and thereby signaling that some of the constraints of φ have not been fulfilled.

The φ_1 formula specifies that horizontally adjacent cells share the same colour along the shared edge. For instance, if the fourth cell of a white row has a blue edge on its right side, then the fifth cell of the same row needs to have a blue edge on its left side. Given that no use of the $\bigcirc$ operator is allowed, there is no direct way to specify the notion of "next cell." Reliance has to be instead made on the $\mathbb{U}$ operator.

Referring back to the previous example, if the current state corresponds to the fourth column of a white row and the right side of its tile is blue, then $w \wedge c_4 \wedge x$ holds, where $x_r = \text{blue}$. The $\mathbb{U}$ operator alone allows to specify something of the sort

$$\Diamond \left(w \wedge c_5 \wedge \bigvee_{\substack{x' \in X \\ x'_l \neq \text{ blue}}} \right) \rightarrow \varphi^{\uparrow},$$

but the cell satisfying this constraint might belong to another white row altogether. How to enforce the constraint that this cell belong to the current row instead? The intuition here is that if the constraint of φ_1 were to be fulfilled by another white row, then at a certain point a black row would be crossed. φ_1 is therefore defined as

$$\varphi_1 \stackrel{\text{def}}{\equiv} \bigvee_{0 \leq k < n-1} \bigvee_{x \in X} \bigvee_{v \in \{b,w\}} \Diamond \left[c_k \wedge x \wedge v \wedge \left(v \,\mathbb{U}\, c_{k+1} \wedge v \wedge \bigvee_{\substack{x' \in X \\ x_r \neq x'_l}} x' \right) \right] \rightarrow \varphi^{\uparrow}.^{18}$$

A similar behavior is enforced by the φ_2 formula. If the sixth cell of a white row has a tile with a green lower edge, then the formula $w \wedge c_6 \wedge x$ holds, where $x_b = \text{green}$. In this

[18] φ_1, like other φ_i formulas, makes use of the saturation technique, and in a way corresponds to so-called *constraint* rules in classical ASP programming. Rather than encoding a desired condition explicitly, φ_1 encodes the negation of it, imposing a state of saturation whenever this negation is met. Formulas of this type have the property of being satisfiable by the THT interpretation (T, T), which is used to signal an impossibility to meet all constraints simultaneously.

case, the sixth cell of the immediately successive black row needs to have a tile with a green upper edge. Similarly as before, the formula

$$\Diamond \left(b \wedge c_6 \wedge \bigvee_{\substack{x' \in X \\ x'_t \neq \text{ green}}} x \right) \rightarrow \varphi^{\uparrow}$$

is only good to disqualify any cell from the 6th column of *any* subsequent black row from possessing a tile with a green upper edge. In order to target only the cells of the immediate next row, the condition that only cells of the current row with a greater (or equal) column number, or cells of the next row with a strictly less column number be visited is imposed. φ_2 is therefore defined as

$$\varphi_2 \stackrel{\text{def}}{=} \bigvee_{0 \leq k \leq n-1} \bigvee_{x \in X} \bigvee_{v \in \{w,b\}} \Diamond \left[c_k \wedge x \wedge v \wedge \left(\text{BeforeNext}(c_k, v) \,\mathbb{U}\, \overline{v} \wedge c_k \wedge \bigvee_{\substack{x' \in X \\ x_b \neq x'_t}} x' \right) \right] \rightarrow \varphi^{\uparrow},$$

where

$$\text{BeforeNext}(c_k, v) \stackrel{\text{def}}{=} \bigvee_{k \leq k' \leq n-1} (v \wedge c_{k'}) \vee \bigvee_{0 \leq k' < k} (\overline{v} \wedge c_{k'}).$$

Informally, $\text{BeforeNext}(c_k, v)$ evaluates to true at all cells of the current row (of colour v) having a column number $k' \geq k$, or to true at all cells of the next row (of colour $\overline{v}$) having a column number $k' < k$. It stops holding true at the cell where $\overline{v} \wedge c_k$ holds.

One important requirement is making sure that at each state of a non-trivial TEM, one and only one $x \in X$ atom is present. This is achieved by the two formulas

$$\varphi_3 \stackrel{\text{def}}{=} \Box \left(\bigvee_{x \in X} x \right),$$

$$\varphi_4 \stackrel{\text{def}}{=} \left(\Diamond \bigvee_{\substack{x_1, x_2 \in X \\ x_1 \neq x_2}} x_1 \wedge x_2 \right) \rightarrow \varphi^{\uparrow}.$$

A similar condition has to be imposed for the column and chromatic variables, although in this case no upper bound needs to be imposed, as the candidate interpretation T already does this implicitly:

$$\varphi_5 \stackrel{\text{def}}{=} \Box \left(\bigvee_{0 \leq k \leq n-1} c_k \right),$$

$$\varphi_6 \stackrel{\text{def}}{=} \Box (b \vee w).$$

Additionally, the constraint that tiles at the left and right margin of the tiling have left and right white sides, respectively, needs be imposed. This is enforced by the formulas φ_7 and φ_8:

$$\varphi_7 \stackrel{\text{def}}{=} \Diamond \left(c_0 \wedge \bigvee_{\substack{x \in X \\ x_l \neq \text{ white}}} x \right) \rightarrow \varphi^{\uparrow},$$

$$\varphi_8 \stackrel{\text{def}}{=} \Diamond \left(c_{n-1} \wedge \bigvee_{\substack{x \in X \\ x_r \neq \text{ white}}} x \right) \rightarrow \varphi^{\uparrow}.$$

Finally, the requirement that the first row of the tiling matches S and that some subsequent, distinct row matches S one further time needs to be imposed. A major discrepancy between the $\overline{\text{RCT}}$ and the $\text{TELMC}(\mathbb{U}, \mathbb{R})$ problem is that the former is defined over an $m \times n$ grid of finite size, whereas the latter is defined over a trace T of infinite length. What to do, then, with all the states in T after the sequence of states encoding S a second time? Since the sequence S was already used to function as a "starter row," it is possible to repeat the same tiling after the second occurrence of S in T, and in fact do so infinitely often. φ_9 and φ_{10} are consequently defined as

$$\varphi_9 \stackrel{\text{def}}{=} (c_0 \wedge w \wedge S_0) \wedge (w \,\mathbb{U}\, (c_1 \wedge w \wedge S_1)) \wedge \cdots \wedge (w \,\mathbb{U}\, (c_{n-1} \wedge w \wedge S_{n-1})),$$

$$\varphi_{10} \stackrel{\text{def}}{=} \Box\Diamond \left(\bigvee_{v \in \{b,w\}} (c_0 \wedge v \wedge S_0) \wedge (v \,\mathbb{U}\, (c_1 \wedge v \wedge S_1)) \wedge \cdots \wedge (v \,\mathbb{U}\, (c_{n-1} \wedge v \wedge S_{n-1})) \right).$$

Since it is not known beforehand whether the second occurrence of S is over a white or a black row, a disjunction among the two chromatic values has to be included.

As anticipated before, the whole formula φ is defined as

$$\varphi \stackrel{\text{def}}{=} \bigwedge_{1 \leq i < 10} \varphi_i.$$

What needs to be proven at this point is that f is a function from $\Sigma_{\overline{\text{RCT}}}$ to $\Sigma_{\text{TELMC}(\mathbb{U},\mathbb{R})}$, that $I \in \overline{\text{RCT}}$ iff $f(I) \in \text{TELMC}(\mathbb{U}, \mathbb{R})$ and that f is computable in polynomial time.

f is a function from $\Sigma_{\overline{\text{RCT}}}$ to $\Sigma_{\text{TELMC}(\mathbb{U},\mathbb{R})}$, because the pair (T, φ) produced by it corresponds to an LTL trace and a $\text{THT}(\mathbb{U}, \mathbb{R})$ formula, where no occurrence of the $\bigcirc$ operator appears.

To show the second point, $I \in \overline{\text{RCT}}$ is first supposed. If $I \in \overline{\text{RCT}}$, then for any $m \in \mathbb{N}$, no tiling of the $m \times n$ plane having S as upper and lower delimiters exists, meaning that some of the constraints of a tiling cannot be satisfied. By manually going over the individual φ_i subformulas of φ, it can be checked how the saturated interpretation (T, T) is a THT model of φ. Since for any m, no tiling of the $m \times n$ plane exists, one or more of the constraints of φ is not satisfied, and the saturated model (T, T) is imposed. Hence no $H < T$ exists such that $(H, T) \vDash \varphi$, and (T, T) is a TEM of φ.

Vice versa, if $I = (X, S) \notin \overline{\text{RCT}}$, then an $m \in \mathbb{N}$ exists for which a tiling of the $m \times n$ plane having S as upper and lower edges exists. (T, T) is still a THT model of φ, but the tiling of the $m \times n$ plane allows to obtain a subtrace $H < T$ encoding it, and for which $(H, T) \vDash \varphi$ holds. Hence (T, T) is not a TEM of φ, and $(T, \varphi) \notin \text{TELMC}(\mathbb{U}, \mathbb{R})$.

Finally, f can be computed in polynomial time, because the trace T and the formula φ that are constructed from (X, S) have polynomial size with respect to $|X| + |S|$. ■

By inspection at the various φ_i formulas that occur within the proof of Theorem 4.53, it can be seen how no more than two nested temporal operators $\mathbb{U}$ and $\mathbb{R}$, and no more than one single implication operator $\rightarrow$, ever appear in any of these formulas. In other words, the temporal and the implication height of the φ_i formulas, and as a consequence of the

φ formula, is at most 2 and 1, respectively. The result of Theorem 4.52 can therefore be further restricted to the following one.

COROLLARY 4.54 Deciding TELMC$^1_2(\mathbb{U}, \mathbb{R})$ is PSPACE–complete. □

4.5. The TELMC($\bigcirc, \mathbb{R}$) fragment

The long journey through the complexity study of the fragments of the TELMC problem is continued by the TELMC($\bigcirc, \mathbb{R}$) problem, the TELMC fragment where only the $\bigcirc$ and $\mathbb{R}$ operators, within any arbitrary nesting depth, are allowed. As it turns out, restricting the set of operators to only the $\bigcirc$ and $\mathbb{R}$ ones does not affect the complexity of the problem. For TELMC($\bigcirc, \mathbb{R}$), in fact, the following holds.

THEOREM 4.55 Deciding TELMC($\bigcirc, \mathbb{R}$) is PSPACE–complete. □

PROOF Consequence of Theorem 4.56 and Corollary 4.57. ■

THEOREM 4.56 Deciding TELMC($\bigcirc, \mathbb{R}$) is PSPACE–hard. □

PROOF To show the PSPACE–hardness of deciding TELMC($\bigcirc, \mathbb{R}$), the same reasoning in the proof of Theorem 3.24 is repeated. The only difference with respect to that reasoning is that, since no $\mathbb{U}$ and $\Diamond$ operators are allowed, a rewriting φ' of the φ formula used in it with only the $\bigcirc$ and $\mathbb{R}$ operators needs to be found. All other details of the proof remain unchanged.

A rewriting of the φ formula into a THT($\bigcirc, \mathbb{R}$) formula φ' can be achieved by adding a special atom $a^{\uparrow}$, referred to as the *saturation atom*, to the $A \stackrel{\text{def}}{=} A_Q \cup A_h \cup A_\sigma$ alphabet over which φ is defined, obtaining the $A' \stackrel{\text{def}}{=} A \cup \{a^{\uparrow}\}$ alphabet. Each φ_i subformula of φ can be subsequently rewritten into a subformula φ'_i of φ' in the following way:

$$
\begin{aligned}
\varphi'_1 &\stackrel{\text{def}}{=} \varphi_1, \\
\varphi'_2 &\stackrel{\text{def}}{=} \varphi_2, \\
\varphi'_3 &\stackrel{\text{def}}{=} \varphi_3, \\
\varphi'_4 &\stackrel{\text{def}}{=} \Box\left(\bigvee_{q \in Q} \bigvee_{q' \in Q \setminus \{q\}} q \wedge q' \rightarrow a^{\uparrow}\right), \\
\varphi'_5 &\stackrel{\text{def}}{=} \Box\left(\bigvee_{0 \leq k < p(|x|)} \bigvee_{\substack{0 \leq k' < p(|x|) \\ k' \neq k}} h_k \wedge h_{k'} \rightarrow a^{\uparrow}\right), \\
\varphi'_6 &\stackrel{\text{def}}{=} \Box\left(\bigvee_{0 \leq k < p(|x|)} \bigvee_{\gamma \in \Gamma} \bigvee_{\gamma' \in \Gamma \setminus \{\gamma\}} \sigma_{k,\gamma} \wedge \sigma_{k,\gamma'} \rightarrow a^{\uparrow}\right), \\
\varphi'_7 &\stackrel{\text{def}}{=} \Box\left(q_{\text{reject}} \rightarrow a^{\uparrow}\right).
\end{aligned}
$$

An additional set of formulas, denoted by ψ_i, also needs to be used to govern the behavior of the saturation atom $a^{\uparrow}$:

$$
\begin{aligned}
\psi_1 &\stackrel{\text{def}}{=} \Box(a^{\uparrow} \rightarrow \bigcirc a^{\uparrow}), \\
\psi_2 &\stackrel{\text{def}}{=} \Box(\bigcirc a^{\uparrow} \rightarrow a^{\uparrow}), \\
\psi_3 &\stackrel{\text{def}}{=} \Box a^{\uparrow} \rightarrow \varphi^{\uparrow}.
\end{aligned}
$$

The $\varphi^{\uparrow}$ formula is also slightly readapted to take the $a^{\uparrow}$ atom into account:

$$\varphi^{\uparrow} \stackrel{\text{def}}{\equiv} \Box \bigwedge_{a' \in A'} a'.$$

Finally, the φ' formula is defined as

$$\varphi' \stackrel{\text{def}}{\equiv} \bigwedge_{1 \leq i \leq 7} \varphi'_i \wedge \bigwedge_{1 \leq i \leq 3} \psi_i.$$

An intuitive explanation of the subformulas φ'_i and ψ_i is as follows.

φ'_1, φ'_2 and φ'_3 need no modifications, as the φ_1, φ_2 and φ_3 formulas of φ are already written using only the $\bigcirc$ and $\mathbb{R}$ operators[19].

The rewriting of the φ_4 formula into the φ'_4 one, instead, is such that if a pair of distinct state atoms q and q' are ever found in the state H_i of a THT interpretation (H, T) for φ', then the saturation atom $a^{\uparrow}$ should also be placed on H_i. Similarly, φ'_5 imposes the condition that if a pair of two distinct h_k and $h_{k'}$ atoms are ever found in H_i, $a^{\uparrow}$ should be also present in it. The same goes for φ'_6 and φ'_7: If two distinct $\sigma_{k,\gamma}$ and $\sigma_{k,\gamma'}$ atoms, or if the q_{reject} atom, are present in H_i, then $a^{\uparrow}$ should be also present there.

The ψ_i formulas control the behavior of the $a^{\uparrow}$ atom. ψ_1 says that if the $a^{\uparrow}$ atom is found in a state, then it should be also found in the next one, and ψ_2 says that if it is found in the next one, then it should also be found in the present one[20]. Taken together, ψ_1 and ψ_2 mean that if $a^{\uparrow}$ ever appears in a state, then it should appear in *all* states of a THT interpretation (H, T) for φ': Hence $a^{\uparrow}$ either never appears, or it appears in every state of (H, T). Finally, ψ_3 triggers the saturation of (H, T) whenever $a^{\uparrow}$ is found in it. ■

Corollary 4.57 $\text{Telmc}(\bigcirc, \mathbb{R}) \in \text{Pspace}$. □

Proof Since $\text{Telmc}(\bigcirc, \mathbb{R}) \subset \text{Telmc}$ and $\text{Telmc} \in \text{Pspace}$, $\text{Telmc}(\bigcirc, \mathbb{R}) \in \text{Pspace}$ too. ■

Since all subformulas in the reduction of Theorem 4.56 have a temporal height of value at most 2 and an implication height of value at most 1, the following also holds.

Corollary 4.58 Deciding $\text{Telmc}_2^1(\bigcirc, \mathbb{R})$ is Pspace–complete. □

[19] It might be useful remembering that $\Box\varphi \stackrel{\text{def}}{\equiv} \bot \mathbb{R} \varphi$.

[20] The syntax of TEL defined in this work makes no use of temporal past operators, and a writing of the form $\Box(\bigcirc a^{\uparrow} \rightarrow a^{\uparrow})$ is a little trick to refer to the past without the use of such operators.

Chapter 5

Model checking over transition systems

"The universe is a pretty big place. If it's just us, seems like an awful waste of space."

— Carl Sagan; Contact (1985).

The previous two chapters studied the problem of model checking for TEL over linear structures, first in a general setting and, subsequently, in restricted ones. The current chapter investigates a similar problem, but over structures that generalize linear traces and allow a sort of ramification, known as *Kripke structures* or, following the terminology adopted here, *transition systems*. Similarly to the previous two chapters, it will be shown how performing model checking over transition systems is just as hard, if not harder, than performing it over linear structures. Table 7 shows a summary of the results of the chapter.

Problem	Complexity
$\textsc{Telts}_\forall$	Pspace–complete (Theorem 5.67)
$\textsc{Telts}_\exists$	Expspace–complete (Theorem 5.68)

Table 7: Summary of the results of the chapter.

5.1. Transition systems

This section introduces the notion of transition system. A more thorough exposition, suitable for a first study, can be found in C. Baier and J.-P. Katoen [62].

Transition systems are graph-like structures composed of vertices, also called *states*, and edges, also called *transitions*, finding application in the description of the behavior of concurrent systems.

Definition 5.58 A *transition system* is a tuple $\tau = (S, \alpha, \rightarrow, I, A, L)$, where:

- S is a set of *states*;
- α is a set of *actions*;
- $\rightarrow \subseteq S \times \alpha \times S$ is a transition relation;
- $I \subseteq S$ is the set of initial states;
- A is a set of atomic propositions, and
- $L : S \rightarrow 2^A$ is a labeling function, assigning to each state $s \in S$ the set of atomic propositions holding on it.

□

A transition system is said to be *finite* if S, α and A are finite.

The intended behavior of a transition system is to model a system that starts operating in any state $s_0 \in I$, and then moves from state to state according to the transition relation $\rightarrow$. $(s, a, t) \in \rightarrow$ means that a transition from state s to state t is available, and that in going from state s to state t, action a is executed. A transition system may come to a halt if a state s is reached for which no outgoing connections are available, or execute forever if any of the states the system goes through has at least one outgoing transition leading to a next state.

An example transition system, modeling a simplified automated teller machine (ATM) is given in Figure 12. In this transition system, $S = \{\text{Card}, \text{Operation}, \text{Withdraw}, \text{Deposit}, \text{Draw card}\}$, $\alpha = \{\text{Pin}, \text{Withdraw}, \text{Terminate}, \text{Insert}, \text{Exit}\}$, $I = \{\text{Card}\}$, while $\rightarrow$ is the relation induced by the graph structure in Figure 12, omitted here for the sake of brevity. A and L are not shown in the figure, but they are specified as $A = \{p, q\}$ and

$$L(\text{Withdraw}) = \{p\},$$
$$L(\text{Deposit}) = \{q\},$$
$$L(x) = \emptyset \; \forall x \in S \setminus \{\text{Withdraw}, \text{Deposit}\}.$$

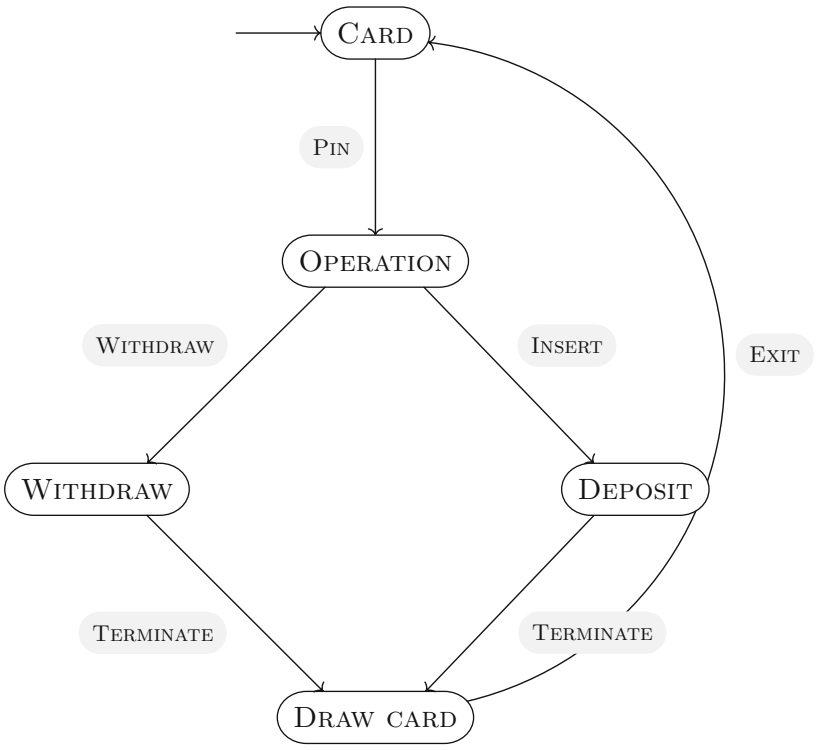


Figure 12: The transition system of a simple ATM.

For a given transition system, the following functions can be defined that encode the states a single state is related to:

$$\begin{aligned}
\textsc{Post}(s,a) &\stackrel{\text{def}}{=} \{t \mid (s,a,t) \in \rightarrow\},\\
\textsc{Post}(s) &\stackrel{\text{def}}{=} \bigcup_{a \in \rightarrow} \textsc{Post}(s,a),\\
\textsc{Pre}(s,a) &\stackrel{\text{def}}{=} \{t \mid (t,a,s) \in \rightarrow\},\\
\textsc{Pre}(s) &\stackrel{\text{def}}{=} \bigcup_{a \in \rightarrow} \textsc{Pre}(s,a).
\end{aligned}$$

Making use of this notation, a state s is said *terminal state* if $\textsc{Post}(s) = \emptyset$.

For a transition system τ, a finite *execution fragment* is an alternating sequence $\rho = s_0 a_1 s_1 a_2 s_2 \cdots a_n s_n$ of states and actions of finite length, such that $\forall i \in \{0, ..., n-1\}, (s_i, a_{i+1}, s_{i+1}) \in \rightarrow$. Similarly, an infinite execution fragment is a sequence $\rho = s_0 a_1 s_1 a_2 s_2 \cdots$ of infinite length, for which $\forall i \geq 0, (s_i, a_{i+1}, s_{i+1}) \in \rightarrow$. It is possible to abstract away from the actions present in execution fragments, and end up with sequences of states, referred to as *path fragments*. Formally, a finite path fragment is a sequence $\pi = s_0 s_1 s_2 \cdots s_n$ of finite length such that $s_i \in \textsc{Post}(s_{i-1})\ \forall i \in \{1, ..., n\}$. Similarly, an infinite path fragment is a path $\pi = s_0 s_1 s_2 \cdots$ of infinite length such that $\forall i \geq 1, s_i \in \textsc{Post}(s_{i-1})$. The focus of this chapter is on *trace fragments*, which are obtained from paths by considering the set of propositions holding on a path. For a finite path fragment $\pi = s_0 s_1 \cdots s_n$, a trace fragment is defined as

$$\textsc{Trace}(\pi) \stackrel{\text{def}}{=} L(s_0)L(s_1) \cdot \cdots \cdot L(s_n),$$

where L is the labeling function of the transition system under consideration. Trace fragments of infinite path fragments are defined analogously. Therefore, trace fragments are sequences of sets of atomic propositions from A, that is, sequences of elements from 2^A.

An execution fragment ρ is said *initial* if the first state s_0 of it belongs to I, and it is said *maximal* if it is either finite, and its last state s_n is a terminal state, or if it is infinite. Said otherwise, a maximal execution fragment is a fragment that cannot be made any longer. An *execution*, then, is an initial and maximal execution fragment. Analogous notions to initial and maximal execution fragments exist for path fragments and trace fragments. A *path* is an initial and maximal path fragment, while a *trace* is an initial and maximal trace fragment.

For the transition system of Figure 12, $\rho_1 = \textsc{Card} \cdot \textsc{Pin} \cdot \textsc{Operation} \cdot \textsc{Withdraw} \cdot \textsc{Withdraw} \cdot \textsc{Terminate} \cdot \textsc{Draw card} \cdot \textsc{Exit} \cdot \textsc{Card} \cdot \cdots$ is an execution fragment which, being initial and maximal, is also an execution. On the other hand, neither $\rho_2 = \textsc{Operation} \cdot \textsc{Withdraw} \cdot \textsc{Withdraw} \cdot \textsc{Terminate} \cdot \textsc{Draw card} \cdot \textsc{Exit} \cdot \textsc{Card} \cdot \cdots$ nor $\rho_3 = \textsc{Card} \cdot \textsc{Pin} \cdot \textsc{Operation} \cdot \textsc{Withdraw} \cdot \textsc{Withdraw} \cdot \textsc{Terminate} \cdot \textsc{Draw card} \cdot \textsc{Exit} \cdot \textsc{Card}$ are executions, for ρ_2 is not initial and ρ_3 is not maximal. From ρ_1, ρ_2 and ρ_3, corresponding path fragments can be obtained by omitting the actions, leading to

$$\begin{aligned}
\pi_1 &= \textsc{Card} \cdot \textsc{Operation} \cdot \textsc{Withdraw} \cdot \textsc{Draw card} \cdot \textsc{Card} \cdot \cdots,\\
\pi_2 &= \textsc{Operation} \cdot \textsc{Withdraw} \cdot \textsc{Draw card} \cdot \textsc{Card} \cdot \cdots,\\
\pi_3 &= \textsc{Card} \cdot \textsc{Operation} \cdot \textsc{Withdraw} \cdot \textsc{Draw card} \cdot \textsc{Card}.
\end{aligned}$$

Finally, the trace fragments corresponding to π_1, π_2 and π_3 are given by

$$\begin{aligned}
\textsc{Trace}(\pi_1) &= \emptyset \cdot \emptyset \cdot \{p\} \cdot \emptyset \cdot \emptyset \cdot \cdots,\\
\textsc{Trace}(\pi_2) &= \emptyset \cdot \{p\} \cdot \emptyset \cdot \emptyset \cdot \cdots,\\
\textsc{Trace}(\pi_3) &= \emptyset \cdot \emptyset \cdot \{p\} \cdot \emptyset \cdot \emptyset.
\end{aligned}$$

5.2. The Telts problem

The focus of this chapter is the Telts problem, the problem of model checking for TEL over transition systems. This problem is divided into two variants, the universal one, denoted $\text{Telts}_\forall$, and the existential one, denoted $\text{Telts}_\exists$. Informally, for a given transition system τ and temporal formula φ, the $\text{Telts}_\forall$ problem asks whether all traces obtained from τ are a TEM of φ, whereas for the same transition system, $\text{Telts}_\exists$ asks whether at least a single trace from τ is a TEM of φ.

5.2.1. The universal Telts problem

The first of the two variants of the Telts problem to be studied is the universal one.

Definition 5.59 The universal variant of the problem of model checking for TEL over transition systems is the language

$$\text{Telts}_\forall \stackrel{\text{def}}{=} \{(\tau, \varphi) \mid \forall T \text{ in } \tau, T \text{ is a TEM of } \varphi\},$$

where τ is a finite transition system without terminal states and T a periodically infinite trace from τ. □

The assumption that τ contains no terminal states in Definition 5.59 serves to ensure that every trace obtained from it is infinite, for an outgoing edge can be found for each state that is visited.

Before studying the complexity of the $\text{Telts}_\forall$ problem, it is helpful introducing Ltlts, the problem of model checking for LTL over transition systems. Similarly to Telts, this problem comes in two variants, $\text{Ltlts}_\forall$ and $\text{Ltlts}_\exists$.

Definition 5.60 The universal variant of the problem of model checking for LTL over transition systems is the language

$$\text{Ltlts}_\forall \stackrel{\text{def}}{=} \{(\tau, \varphi) \mid \forall T \text{ in } \tau, T \vDash_{\text{LTL}} \varphi\},$$

where τ is a finite transition system without terminal states and T a periodically infinite trace from τ. □

Definition 5.61 The existential variant of the problem of model checking for LTL over transition systems is the language

$$\text{Ltlts}_\exists \stackrel{\text{def}}{=} \{(\tau, \varphi) \mid \exists T \text{ in } \tau, T \vDash_{\text{LTL}} \varphi\},$$

where τ is a finite transition system without terminal states and T a periodically infinite trace from τ. □

Theorem 5.59 Deciding $\text{Ltlts}_\exists$ is Pspace–hard. □

Proof See A. P. Sistla and E. M. Clarke [55]. ■

Theorem 5.60 Deciding $\text{Ltlts}_\forall$ is Pspace–hard. □

Showing this fact can be done with a reduction from the $\overline{\text{Telts}_\exists}$ problem (C. Baier and J.-P. Katoen [62]) which, given an instance (τ, φ), asks whether no trace T in τ is the LTL model of φ. Since by Theorem 5.59 deciding $\text{Telts}_\exists$ is Pspace–hard, by Observation 3.44 deciding $\overline{\text{Telts}_\exists}$ is also Pspace–hard.

Proof To show the Pspace–hardness of $\text{Ltlts}_\forall$, a Karp reduction $f : \Sigma_{\overline{\text{Ltlts}_\exists}} \to \Sigma_{\text{Ltlts}_\forall}$ is constructed. Given an instance $(\tau, \varphi) \in \Sigma_{\overline{\text{Ltlts}_\exists}}$,

$$f((\tau, \varphi)) \stackrel{\text{def}}{=} (\tau, \neg\varphi).$$

If $(\tau, \varphi) \in \overline{\text{LTLTS}_\exists}$, then no periodically infinite trace T within τ is an LTL model of φ, meaning that for all such traces T within τ, $T \vDash_{\text{LTL}} \neg\varphi$. Vice versa, if $(\tau, \neg\varphi) \in \text{LTLTS}_\forall$, then all periodically infinite traces T within τ are an LTL model of $\neg\varphi$, meaning that no such trace within τ is an LTL model of φ. ■

Another result needed in the study of the complexity of $\text{TELTS}_\forall$ is the following.

PROPOSITION 5.61 *(P. Cabalar and S. Demri [52, Proposition 1].)* Given a THT formula φ defined over the alphabet A and a THT interpretation (H, T),

$$(H, T) \vDash_{\text{THT}} \varphi \wedge \varphi_{\text{EM}}(A)$$

if and only if

$$H = T \wedge T \vDash_{\text{LTL}} \varphi,$$

where

$$\varphi_{\text{EM}}(A) \stackrel{\text{def}}{=} \bigwedge_{p \in A} \Box(p \vee \neg p).$$

□

In other words, Proposition 5.61 asserts that a THT interpretation can be brought to collapse into an LTL one by the addition of the $\varphi_{\text{EM}}(A)$ formula, referred to as the *excluded middle axiom*. A THT interpretation satisfying this formula is forced to set any literal $p \in A$ to either "true" or "false," but not "assumed." That is, $p \in H_i = T_i$ or $p \notin H_i = T_i$, with no possibility to have $p \in T_i \setminus H_i$.

THEOREM 5.62 Deciding $\text{TELTS}_\forall$ is PSPACE–hard. □

PROOF IDEA If it is possible to solve $\text{TELTS}_\forall$, then it is also possible to solve $\text{LTLTS}_\forall$ by "switching off" the minimization process in the semantics of TEL. The "switching off" of the minimization process is achieved by the $\varphi_{\text{EM}}(A)$ formula. ■

PROOF To show the PSPACE–hardness of $\text{TELTS}_\forall$, a Karp reduction $f : \Sigma_{\text{LTLTS}_\forall} \to \Sigma_{\text{TELTS}_\forall}$ is constructed. This reduction is defined as

$$f((\tau, \varphi)) \stackrel{\text{def}}{=} (\tau', \psi),$$

where

$$\tau' \stackrel{\text{def}}{=} \tau,$$
$$\psi \stackrel{\text{def}}{=} \varphi \wedge \varphi_{\text{EM}}(A),$$

and where A is the set of literals appearing in φ. Informally, ψ enforces φ and the requirement that all of its THT models be total. What needs to be shown at this point is that f is a function from $\Sigma_{\text{LTLTS}_\forall}$ to $\Sigma_{\text{TELTS}_\forall}$, that $(\tau, \varphi) \in \text{LTLTS}_\forall$ iff $f((\tau, \varphi)) \in \text{TELTS}_\forall$ and that f is computable in polynomial time.

The first statement follows immediately by the observation that $\tau' = \tau$, and that ψ is a THT formula, given that φ and $\varphi_{\text{EM}}(A)$ are LTL formulas[21].

For the second statement, $(\tau, \varphi) \in \text{LTLTS}_\forall$ is supposed. For $f((\tau, \varphi)) = (\tau, \psi)$ to be in $\text{TELTS}_\forall$, any trace T from τ needs to be a TEM of ψ. Let then T be a trace from τ. T is a TEM of ψ if

[21] Although LTL has not been explicitly defined in this work, the formulation of THT given in the existing literature makes use of the same syntax as LTL.

1. (T, T) is a total THT interpretation;
2. $(T, T) \vDash_{\text{THT}} \psi$, and
3. $\nexists T' < T \mid (T', T) \vDash_{\text{THT}} \psi$.

The first property is satisfied. The second is also satisfied, because $(\tau, \varphi) \in \text{LTLTS}_\forall$ and therefore any trace of τ, including T, is an LTL model of φ. Therefore, given that $T \vDash_{\text{LTL}} \varphi$, by Proposition 5.61 $(T, T) \vDash_{\text{THT}} \varphi \wedge \varphi_{\text{EM}}(A)$.

To show the third point, $\exists T' < T \mid (T', T) \vDash_{\text{THT}} \psi$ is supposed. Then, for a certain $i \geq 0$ and $p \in A$, $p \in T_i \setminus T'_i$, meaning that

$$\begin{aligned}(T'_i, T_i) &\nvDash_{\text{HT}} (p \vee \neg p),\\ (T', T) &\nvDash_{\text{THT}} \varphi_{\text{EM}}(A),\\ (T', T) &\nvDash_{\text{THT}} \varphi \wedge \varphi_{\text{EM}}(A),\\ (T', T) &\nvDash_{\text{THT}} \psi.\end{aligned}$$

This is in contradiction with $(T', T) \vDash_{\text{THT}} \psi$, so one such T' cannot exist and point 3. is satisfied.

For the opposite side of the implication, given a $(\tau, \varphi) \in \Sigma_{\text{LTLTS}_\forall}$, $f((\tau, \varphi)) = (\tau, \psi) \in \text{TELTS}_\forall$ is supposed. Then for all traces T in τ

$$\begin{aligned}&T \text{ is a TEM of } \psi,\\ (T, T) &\vDash_{\text{THT}} \psi \wedge \nexists T' < T \mid (T', T) \vDash_{\text{THT}} \psi,\\ (T, T) &\vDash_{\text{THT}} \psi,\\ (T, T) &\vDash_{\text{THT}} \varphi \wedge \varphi_{\text{EM}}(A),\\ (T, T) &\vDash_{\text{THT}} \varphi,\\ &T \vDash_{\text{LTL}} \varphi,\end{aligned}$$

meaning that $(\tau, \varphi) \in \text{LTLTS}_\forall$.

Finally, f can be computed in polynomial time with respect to the size of (τ, φ), because $\tau' \overset{\text{def}}{=} \tau$ and $\psi \overset{\text{def}}{=} \varphi \wedge \varphi_{\text{EM}}(A)$, where $\varphi_{\text{EM}}(A) \overset{\text{def}}{=} \bigwedge_{p \in A} \Box(p \vee \neg p)$ takes at most linear space with respect to the size of φ. ■

What remains to be done at this point is show the PSPACE membership of $\text{TELTS}_\forall$. To that end, it might be useful to remember that when discussing the complexity of TELMC, specifically its PSPACE membership, Subsection 3.2.2 from Chapter 3 introduced the notion of T-embedding for an LTL trace T, an LTL formula encoding the behavior of T. That notion was instrumental in the proof of PSPACE membership of TELMC, because it allowed reducing a model checking problem into a satisfiability one. But the scope of T-embeddings is not restricted to LTL and THT traces. It is in fact possible to extend this notion to transition systems, and use it in such a way that a model checking problem over transition systems is reduced to, again, a satisfiability one. As in the previous case, this extension of T-embeddings is first introduced by an example.

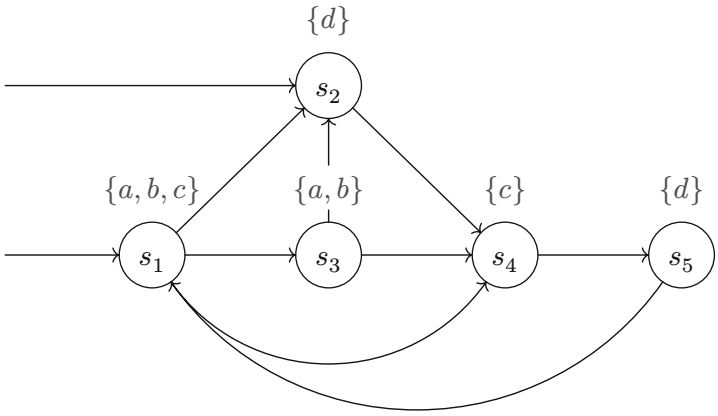


Figure 13: An example transition system.

If the transition system τ of Figure 13 is taken into consideration, in which each state is labeled by its own name and the set of atoms holding on it is displayed on top of it, an LTL formula φ_τ can be associated to it in such a way that for any LTL trace T, $T \vDash_{\text{LTL}} \varphi_\tau$ iff T is a trace within τ. For the specific instance of Figure 13

$$
\begin{aligned}
\varphi_\tau \stackrel{\text{def}}{=}\ & s_1 \vee s_2 \\
& \wedge \Box(s_1 \rightarrow a \wedge b \wedge c \wedge \neg d) && \wedge\ \Box(s_1 \rightarrow \bigcirc(s_2 \vee s_3 \vee s_4)) \\
& \wedge \Box(s_2 \rightarrow \neg a \wedge \neg b \wedge \neg c \wedge d) && \wedge\ \Box(s_2 \rightarrow \bigcirc s_4) \\
& \wedge \Box(s_3 \rightarrow a \wedge b \wedge \neg c \wedge \neg d) && \wedge\ \Box(s_3 \rightarrow \bigcirc(s_2 \vee s_4)) \\
& \wedge \Box(s_4 \rightarrow \neg a \wedge \neg b \wedge c \wedge \neg d) && \wedge\ \Box(s_4 \rightarrow \bigcirc s_5) \\
& \wedge \Box(s_5 \rightarrow \neg a \wedge \neg b \wedge \neg c \wedge d) && \wedge\ \Box(s_5 \rightarrow \bigcirc s_1) \\
& \wedge \Box(s_1 \rightarrow \neg s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5) \\
& \wedge \Box(s_2 \rightarrow \neg s_1 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5) \\
& \wedge \Box(s_3 \rightarrow \neg s_1 \wedge \neg s_2 \wedge \neg s_4 \wedge \neg s_5) \\
& \wedge \Box(s_4 \rightarrow \neg s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge \neg s_5) \\
& \wedge \Box(s_5 \rightarrow \neg s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge \neg s_4).
\end{aligned}
$$

As in the case of T-embeddings, this type of construction identifies each state of the input transition system τ, constrains the way each state can be reached by the others and enforces the corresponding set of atoms holding at a state (and only those atoms) to hold. This type of construction is termed *TS-embedding* and can be more generally defined as follows.

DEFINITION 5.62 Given a transition system $\tau = (S, \alpha, \rightarrow, I, A, L)$, the *$TS$-embedding* of τ is the formula

$$\varphi_\tau \stackrel{\text{def}}{=} \bigvee_{s \in I} s$$
$$\wedge \bigwedge_{s \in S} \Box \left(s \rightarrow \bigcirc \bigvee_{s \rightarrow t} t \right)$$
$$\wedge \bigwedge_{s \in S} \Box \left(s \rightarrow \bigwedge_{a \in A \cap L(s)} a \wedge \bigwedge_{a \in A \setminus L(s)} \neg a \right)$$
$$\wedge \bigwedge_{s \in S} \Box \left(s \rightarrow \bigwedge_{\substack{t \in S \\ t \neq s}} \neg t \right),$$

constructed over the alphabet $A \cup S$. □

As anticipated, TS-embeddings allow to turn a model checking problem into a satisfiability one, by encoding the behavior of a transition system into a formula. The intuitive meaning of this property is formalized by the following proposition.

Proposition 5.63 Given an LTL formula φ constructed over an alphabet A[22] and a transition system $\tau = (S, \alpha, \rightarrow, I, A, L)$, there exists a periodically infinite LTL trace $T = T_1 \cdot \dots \cdot T_n \cdot (T_{n+1} \cdot \dots \cdot T_m)^\omega$ within τ such that $T \vDash_{\text{LTL}} \varphi$ iff $\varphi \wedge \varphi_\tau$ is LTL satisfiable. □

Proof If a periodically infinite LTL trace T within τ exists for which $T \vDash_{\text{LTL}} \varphi$, then an extension T' of T over $A \cup S$ exists which marks the states of τ T passes through. Specifically, since T is a trace in τ, then a sequence of states $t_1, ..., t_m \in S$ exists such that $\forall i, 1 \leq i \leq m, T_i = L(s_i)$. This extension of T is then equal to

$$T' \stackrel{\text{def}}{=} (T_1 \cup \{t_1\}) \cdot \dots \cdot (T_n \cup \{t_n\}) \cdot (T_{n+1} \cup \{t_{n+1}\}) \cdot \dots \cdot (T_m \cup \{t_m\})^\omega.$$

Since T satisfies φ and $T \leq T'$, T' also satisfies φ. Additionally, T' also satisfies φ_τ, because

1. T', by virtue of being an initial path fragment, contains a state variable $s \in I$ in T'_0;
2. If $s \in T'_k$, then $t \in T'_{k+1}$ for some t such that $s \rightarrow t$;
3. If $s \in T'_k$, then $T'_k \cap A = L(s)$ and $T'_k \cap (A \setminus L(s)) = \emptyset$;
4. If $s \in T'_k$, then no other $s' \neq s$ belongs to T'_k;

Hence T' satisfies φ and φ_τ, meaning that $\varphi \wedge \varphi_\tau$ is satisfiable.

Vice versa, if $\varphi \wedge \varphi_\tau$ is LTL satisfiable, then a periodically infinite LTL trace T over the alphabet $A \cup S$ exists which satisfies it. From this trace, a restriction T' to A, defined as $T'_k \stackrel{\text{def}}{=} T_k \cap A$, can be obtained; since φ is constructed only over A and it is already satisfied by T, it is also satisfied by T'. Further, given that $T \vDash \varphi_\tau$, for a given $s \in S$ each state T_k of T contains all and only the atoms in $L(s)$, except for the single atom s; in other words, $T_k \setminus \{s\} = L(s)$. Since T' is the restriction of T to A, this property continues to hold for T', that is $T'_k = L(s)$ for some $s \in S$; in particular, coupled with the fact that $T \vDash \varphi_\tau$, this shows that T' is a trace from τ. Then a periodically infinite trace from τ satisfying φ exists, and that trace is T' itself. ■

[22] A can be seen as the smallest set of atoms φ builds on. In particular, φ makes no use of the atoms found in S.

OBSERVATION 5.63 The statement of Proposition 5.63 supposes φ to be constructed over an alphabet A, and the states of the transition system τ to draw atoms from the alphabet $A \cup S$. Different settings where this hypothesis is violated, such as in Lemma 5.64 below, can occur. In such cases, it might no longer be the case that the trace T obtained is completely within τ, for T might also contain atoms beyond the set $A \cup S$. However, when restricted to $A \cup S$, each state T_k of T contains exactly the set of atoms of some state of τ. □

Proposition 5.63 stated one of the basic properties of TS-embeddings. The aim of this section, however, is to show the PSPACE membership of $\text{TELTS}_\forall$, and to that end it is the following lemma that is needed.

LEMMA 5.64 Let φ be an LTL formula constructed over an alphabet A and $\tau = (S, \alpha, \rightarrow, I, A, L)$ a transition system. Then a periodically infinite LTL trace $T = T_1 \cdot \dots \cdot T_n \cdot (T_{n+1} \cdot \dots \cdot T_m)^\omega$ within τ exists such that $(H, T) \vDash \varphi$ for some $H < T$ iff $\varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A) \wedge \varphi_\tau$ is LTL satisfiable. □

PROOF If a periodically infinite LTL trace T within τ exists such that $(H, T) \vDash \varphi$ for some $H < T$, then by Lemma 3.28 $M \stackrel{\text{def}}{=} (H, T)^* \vDash_{\text{LTL}} \varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A)$. Given that T is periodically infinite and from within τ, each state of T draws its atoms from a single state of τ. This makes it possible to label each state of T with an atom $s \in S$, indicating which state from τ a single state of T_k obtains its atoms from; however, rather than perform this labeling to T, it is performed to M. Let $t_1, \dots, t_m \in S$ be the atoms that distinguish which state from τ M obtains its atoms from. Then a new trace M' can be constructed such that

$$M' \stackrel{\text{def}}{=} (M_1 \cup \{t_1\}) \cdot \dots \cdot (M_n \cup \{t_n\}) \cdot (M_{n+1} \cup \{t_{n+1}\}) \cdot \dots \cdot (M_m \cup \{t_m\})^\omega.$$

M' continues to satisfy $\varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A)$ and, additionally, it also satisfies φ_τ because, as shown in the proof of Proposition 5.63, all of the subformulas of φ_τ are satisfied. Hence $\varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A) \wedge \varphi_\tau$ is satisfiable.

Conversely, if $\psi \stackrel{\text{def}}{=} \varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A) \wedge \varphi_\tau$ is LTL satisfiable, then a periodically infinite LTL trace M over the alphabet $A \cup A' \cup S$ exists which satisfies it[23]. Since M satisfies φ_τ, the "unprimed" atoms from A are forced to occur according to τ; in other words, when restricted to A, M is a trace from τ. From M, a restriction M' to $A \cup A'$, defined as $M'_k \stackrel{\text{def}}{=} T_k \cap (A \cup A')$, can be obtained; given that $\varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A)$ is constructed only over $A \cup A'$ and it is already satisfied by M, it continues to be satisfied by M'. Then, due to Lemma 3.27 a THT trace (H, T) exists such that $H < T$ and $(H, T)^* = M'$. Furthermore, since $M' \vDash_{\text{LTL}} \varphi^* \wedge \varphi_\leq(A) \wedge \varphi_<(A)$ and $M' = (H, T)^*$, by Lemma 3.28 $(H, T) \vDash_{\text{THT}} \varphi$. Finally, given that T is the restriction of M to the atoms in A and, as previously said, M restricted to A is a trace from τ, T is also a trace from τ; additionally, T is also periodically infinite, because by hypothesis M already is. Hence a periodically infinite LTL trace T such that $(H, T) \vDash \varphi$ for some $H < T$ exists. ■

PROPOSITION 5.65 $\overline{\text{TELTS}_\forall} \in$ PSPACE. □

PROOF IDEA To prove $(\tau, \varphi) \in \overline{\text{TELTS}_\forall}$ intuitively means to prove that not all of the traces of τ are a TEM for φ. In order to verify this, what needs to be done is find at least

[23] The formula φ_τ is constructed over the alphabet $A \cup S$, and in order to be satisfied, it imposes the presence of a single atom $s_i \in S$ for each state M_k of M. It is important to realize that these atoms do *not* occur in their primed version, as do the atoms from φ^*, and that $\varphi_\leq(A)$ and $\varphi_<(A)$ impose their constraints only to the atoms in A.

one trace from τ that is not in equilibrium for φ. The insights dispensed in the previous sections, specifically the one of Lemma 5.64, show that in order to find one such trace in τ, an LTL trace T can be looked for that satisfies $\varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_{\tau}$. Thanks to φ^* and $\varphi_{\leq}(A)$, if such a trace exists, then a corresponding THT interpretation (X, Y) exists such that $(X, Y)^* = T$ and $X \leq Y$; thanks to $\varphi_{<}(A)$, $X < Y$; and thanks to φ_{τ}, the "there" trace of (X, Y), namely Y, corresponds to a trace from τ. ■

Proof In order to prove whether a given $(\tau, \varphi) \in \Sigma_{\overline{\text{TELTS}_\forall}}$ belongs to $\overline{\text{TELTS}_\forall}$, with $\tau = (S, \alpha, \rightarrow, I, A, L)$ a transition system and φ a THT formula over A, a periodically infinite trace T within τ needs to be found which is not a TEM of φ. A periodically infinite trace T within τ is not a TEM of φ if either

1. (T, T) is not a total THT trace; or
2. $(T, T) \not\models_{\text{THT}} \varphi$; or
3. $\exists H < T$ for which $(H, T) \models_{\text{THT}} \varphi$.

Condition 1. never holds, so in order to verify whether a trace T which is not a TEM of φ exists or not, condition 2. and condition 3. need to be verified.

To verify whether a trace T exists such that $(T, T) \not\models_{\text{THT}} \varphi$, a trace T such that $T \not\models_{\text{LTL}} \varphi$ is looked for. The problem of looking for such a trace can, in turn, be reduced to the validity problem of a formula over a transition system, that is to the problem of verifying whether $T \models_{\text{LTL}} \varphi$ for all traces T in τ. If $T \not\models_{\text{LTL}} \varphi$ for some trace T in τ, then (τ, φ) is declared to belong to $\overline{\text{TELTS}_\forall}$; conversely, if $T \models_{\text{LTL}} \varphi$ for all traces T in τ, then the next condition is verified. Since the problem of validity of an LTL formula over a transition system is in Pspace, this type of check also lies in Pspace.

To verify whether a periodically infinite trace T within τ such that $(H, T) \models_{\text{THT}} \varphi$ for some $H < T$ exists, the formula $\psi \stackrel{\text{def}}{=} \varphi^* \wedge \varphi_{\leq}(A) \wedge \varphi_{<}(A) \wedge \varphi_{\tau}$ is constructed and checked for satisfiability. By Lemma 5.64, if ψ is satisfiable, then a periodically infinite trace T in τ such that $(H, T) \models_{\text{THT}} \varphi$ for some $H < T$ exists, in which case (τ, φ) is declared to belong to $\overline{\text{TELTS}_\forall}$. If, instead, ψ is unsatisfiable, then by contraposition of Lemma 5.64 no periodically infinite trace T from τ such that $(H, T) \models_{\text{THT}} \varphi$ for some $H < T$ exists, in which case (τ, φ) is declared not to belong to $\overline{\text{TELTS}_\forall}$. Since verifying whether an LTL formula such as ψ is satisfiable is in Pspace, this third check is also in Pspace. ■

The algorithm implicit in the proof of Proposition 5.65 is shown in Algorithm 5; $\xi_{\text{LTLTS}_\forall}$ is supposed to be an algorithm for the LTL model checking problem on transition systems, which, given a transition system τ and a formula φ, returns true if φ holds on all traces T of τ, and false otherwise.

```
1 Boolean ξ_{TELTS_∀} (TransitionSystem τ, Formula φ)
2     if (¬ ξ_{LTLTS_∀}(τ, φ))
3         return true
4     let ψ ← φ* ∧ φ_≤(A) ∧ φ_<(A) ∧ φ_τ
5     if (ψ is satisfiable)
6         return true
7     else
8         return false
```

Algorithm 5: Decision algorithm for the $\overline{\text{TELTS}_\forall}$ problem.

THEOREM 5.66 $\text{TELTS}_\forall \in \text{PSPACE}$. □

PROOF By Proposition 5.65, $\overline{\text{TELTS}_\forall} \in \text{PSPACE}$, meaning that $\text{TELTS}_\forall \in \text{coPSPACE}$. But $\text{coPSPACE} = \text{PSPACE}$, therefore $\text{TELTS}_\forall \in \text{PSPACE}$. ■

THEOREM 5.67 Deciding $\text{TELTS}_\forall$ is PSPACE–complete. □

PROOF Consequence of Theorem 5.62 and Theorem 5.66. ■

5.2.2. The existential TELTS problem

The second of the two variants of the TELTS problem to be studied is the existential one which, given a transition system τ and a THT formula φ, asks whether a trace T in τ exists which is a TEM of φ.

DEFINITION 5.64 The existential variant of the problem of model checking for TEL over transition systems is the language

$$\text{TELTS}_\exists \stackrel{\text{def}}{=} \{(\tau, \varphi) \mid \exists T \text{ in } \tau, T \text{ is a TEM of } \varphi\},$$

where τ is a finite transition system without terminal states and T a periodically infinite trace from τ. □

As in the case of the $\text{TELTS}_\forall$ problem, all states in τ are assumed to have at least one outgoing connection, ensuring all traces from τ to be infinite.

Even though the $\text{TELTS}_\forall$ and $\text{TELTS}_\exists$ problems look very similar to each other, their computational complexity is considerably different. For $\text{TELTS}_\exists$, in fact, the following holds:

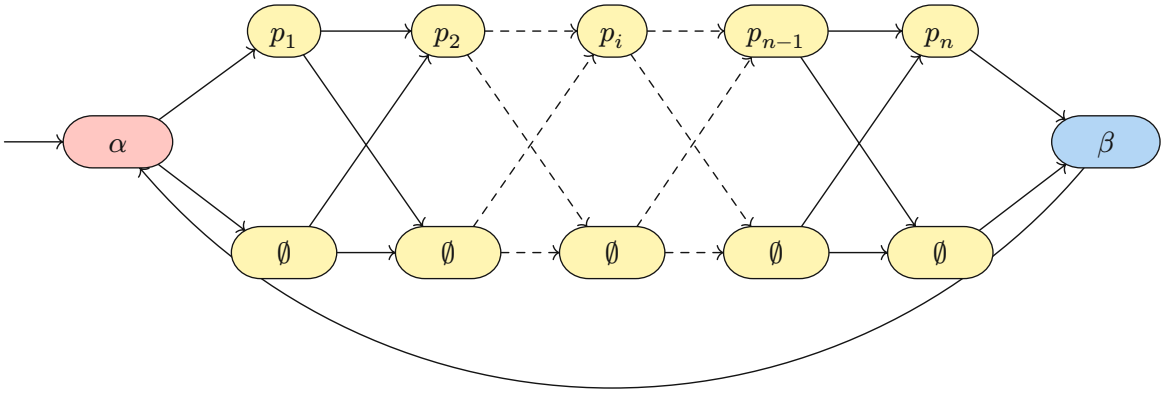
THEOREM 5.68 Deciding $\text{TELTS}_\exists$ is EXPSPACE–complete. □

PROOF Consequence of Theorem 5.72 and Theorem 5.76. ■

The problem of satisfiability for TEL, denoted as TEL, asks whether a THT formula φ admits a TEM T and L. Bozzelli and D. Pearce [53] show deciding it to be EXPSPACE–hard. The way deciding $\text{TELTS}_\exists$ can be also shown to be EXPSPACE–hard is by a reduction from this problem, by assigning to each THT formula φ a transition system τ and a THT formula φ' such that φ has a TEM iff φ' has a TEM which is a trace from τ.

If φ is a THT formula defined over the alphabet $A = \{p_1, p_2, ..., p_n\}$, where an arbitrary ordering of the atoms has been fixed, a transition system $\tau = (S, \emptyset, \rightarrow, \{\alpha\}, A \cup \{\alpha, \beta\}, L)$, referred to as the *atom selection (AS) transition system*, can be associated to it[24]. The set of states S of τ contains a pair of states for each atom p_i in A, plus two additional states, α and β. The way these states are connected to each other by the transition relation $\rightarrow$ is illustrated in Figure 14. For each state p_i in the top row, $L(p_i) \stackrel{\text{def}}{=} \{p_i\}$, while $L(\alpha) \stackrel{\text{def}}{=} \{\alpha\}$ and $L(\beta) \stackrel{\text{def}}{=} \{\beta\}$. For all other states s, $L(s) \stackrel{\text{def}}{=} \emptyset$. A terminology is also introduced to refer to the states of τ. The α state is referred to as a "start" state, the β one as "end" state and all the intermediate ones as "inner" states.

[24] In particular, τ is a function of the alphabet A of φ only.



(a) Transition system τ obtained from the THT formula φ defined over the alphabet A.

State	Also referred to as
α	Start
p_i $\emptyset$	Inner
β	End

(b) Terminology for the states in τ.

Figure 14: Transition system obtained by the reduction from the Tel problem to the Telts$_\exists$ one.

The idea of the AS transition system is that if an infinite LTL trace T over the alphabet 2^A is obtained, an equivalent infinite LTL trace T' over the alphabet $A' \stackrel{\text{def}}{=} A \cup \{\alpha, \beta\}$ can be also obtained, in which the literals of a single state T_i of T are *unfolded* (or *flattened*) over a finite segment of length $n+2$. A trace T' of this sort is said to be an *unfolding* of T over τ. To give an example, if $A = \{a, b, c, d\}$ and

$$T = \{a, b, c\} \cdot \{a, c, d\} \cdot \{a\}^\omega$$

then

$$T' = \underbrace{\{\alpha\} \cdot \{a\} \cdot \{b\} \cdot \{c\} \cdot \emptyset \cdot \{\beta\} \cdot}_{\{a,b,c\}} \underbrace{\{\alpha\} \cdot \{a\} \cdot \emptyset \cdot \{c\} \cdot \{d\} \cdot \{\beta\} \cdot}_{\{a,c,d\}} \underbrace{(\{\alpha\} \cdot \{a\} \cdot \emptyset \cdot \emptyset \cdot \emptyset \cdot \{\beta\})^\omega}_{\{a\}^\omega}.$$

It is clear that if the formula φ is satisfied by an LTL interpretation T, then it will likely not be also satisfied by T'. However, from φ another formula φ' can be obtained which satisfies this property. More precisely, φ' can be defined in such a way that for any $k \geq 0$

$$T, k \vDash \varphi \text{ iff } T', (n+2)k \vDash \varphi'.$$

To accommodate this property, φ' needs to be defined in a specific way. One possibility is to follow the L-transformation, defined below.

Definition 5.65 *(L-transformation.)* The *L-transformation* is a function from THT formulas to THT formulas, defined by structural recursion as:

$$\begin{aligned}
\bot^L &\stackrel{\text{def}}{=} \bot, \\
p_i^L &\stackrel{\text{def}}{=} \bigcirc^i p_i, \\
(\varphi_1 \vee \varphi_2)^L &\stackrel{\text{def}}{=} \varphi_1^L \vee \varphi_2^L, \\
(\varphi_1 \wedge \varphi_2)^L &\stackrel{\text{def}}{=} \varphi_1^L \wedge \varphi_2^L, \\
(\varphi_1 \to \varphi_2)^L &\stackrel{\text{def}}{=} \varphi_1^L \to \varphi_2^L, \\
(\bigcirc\varphi)^L &\stackrel{\text{def}}{=} \bigcirc^{n+2}\varphi^L, \\
(\varphi_1 \,\mathbb{U}\, \varphi_2)^L &\stackrel{\text{def}}{=} \neg\alpha \vee \varphi_1^L \,\mathbb{U}\, \alpha \wedge \varphi_2^L, \\
(\varphi_1 \,\mathbb{R}\, \varphi_2)^L &\stackrel{\text{def}}{=} \alpha \wedge \varphi_1^L \,\mathbb{R}\, \neg\alpha \vee \varphi_2^L.
\end{aligned}$$

□

The following then holds.

Lemma 5.69 Let φ be a THT formula defined over the finite alphabet A and τ the AS transition system of φ. Then for any THT trace (H, T) over 2^A and any $k \geq 0$

$$(H, T), k \vDash_{\text{THT}} \varphi \text{ iff } (H', T'), (n+2)k \vDash_{\text{THT}} \varphi^L,$$

where H' and T' are the unfolding of H and T over τ, respectively. □

Proof By structural induction over the syntax of THT. For convenience, in the sequel $k' \stackrel{\text{def}}{=} (n+2)k$ and $(H, T), k \vDash \varphi$ is written to signify $(H, T), k \vDash_{\text{THT}} \varphi$. The statement can then be rewritten into

$$(H, T), k \vDash \varphi \text{ iff } (H', T'), k' \vDash \varphi^L.$$

The same notational convention for k will be used throughout the proof even for other variables, such as u, z etcetera, so that, for instance, $u' \stackrel{\text{def}}{=} (n+2)u$. When a variable v' is made use of without introducing the corresponding variable v first, then it is supposed that $\exists v \geq 0 \mid v' = (n+2)v$.

If $\varphi = \bot$, then $(H, T), k \nvDash \bot$ and the $\Rightarrow$ side of the statement is vacuously satisfied. Similarly, since $(H', T'), k' \nvDash \bot$, the $\Leftarrow$ side of the statement is also vacuously satisfied.

If $\varphi = p_i$, then $\varphi^L = \bigcirc^i p_i$. If $(H, T), k \vDash p_i$, then $p_i \in H_k$, and since H' is an unfolding of H over τ, $p_i \in H'_{k'+i}$, meaning that $(H', T'), k' \vDash \bigcirc^i p_i$. Similarly, if $(H', T'), k' \vDash \bigcirc^i p_i$, $p_i \in H'_{k'+i}$, and since H' is an unfolding of H, $p_i \in H_k$, meaning that $(H, T), k \vDash p_i$.

If $\varphi = \varphi_1 \vee \varphi_2$, then $\varphi^L = \varphi_1^L \vee \varphi_2^L$. If $(H, T), k \vDash \varphi$, then either $(H, T), k \vDash \varphi_1$ or $(H, T), k \vDash \varphi_2$. In the first case, by the inductive hypothesis (IH) $(H', T'), k' \vDash \varphi_1^L$, meaning that $(H', T'), k' \vDash \varphi_1^L \vee \varphi_2^L$. In the second case, the reasoning is analogous. Vice versa, if $(H', T'), k' \vDash \varphi_1^L \vee \varphi_2^L$, then either $(H', T'), k' \vDash \varphi_1^L$ or $(H', T'), k' \vDash \varphi_2^L$. A reasoning by cases and the use of the IH can then be applied again.

If $\varphi = \varphi_1 \wedge \varphi_2$, then $\varphi^L = \varphi_1^L \wedge \varphi_2^L$. If $(H, T), k \vDash \varphi_1 \wedge \varphi_2$, then $(H, T), k \vDash \varphi_1$ and $(H, T), k \vDash \varphi_2$, and by applying the IH separately two times, $(H', T'), k' \vDash \varphi_1^L$ and $(H', T'), k' \vDash \varphi_2^L$, meaning that $(H', T'), k' \vDash \varphi_1^L \wedge \varphi_2^L$. Vice versa, if $(H', T'), k' \vDash \varphi_1^L \wedge$

φ_2^L, then $(H',T'),k' \vDash \varphi_1^L$ and $(H',T'),k' \vDash \varphi_2^L$, and by applying the IH separately two times, $(H,T),k \vDash \varphi_1$ and $(H,T),k \vDash \varphi_2$, meaning that $(H,T),k \vDash \varphi_1 \wedge \varphi_2$.

If $\varphi = \varphi_1 \rightarrow \varphi_2$, then $\varphi^L = \varphi_1^L \rightarrow \varphi_2^L$. If $(H,T),k \vDash \varphi$, then by the semantics of the $\rightarrow$ operator for THT $\varphi = \varphi_1 \rightarrow \varphi_2$

1. either $(H,T),k \nvDash \varphi_1$ or $(H,T),k \vDash \varphi_2$, and
2. either $(T,T),k \nvDash \varphi_1$ or $(T,T),k \vDash \varphi_2$.

In the first case, if $(H,T),k \nvDash \varphi_1$, by contraposition of the IH $(H',T'),k' \nvDash \varphi_1^L$, meaning that $(H',T'),k' \vDash \varphi_1^L \rightarrow \varphi_2^L$. If, instead, $(H,T),k \vDash \varphi_2$, then by the IH $(H',T'),k' \vDash \varphi_2^L$ and $(H',T'),k' \vDash \varphi_1^L \rightarrow \varphi_2^L$ follows again.

In the second case, if $(T,T),k \nvDash \varphi_1$, then by contraposition of the IH $(T',T'),k' \nvDash \varphi_1^L$, and $(T',T'),k' \vDash \varphi_1^L \rightarrow \varphi_2^L$ follows. If, instead, $(T,T),k \vDash \varphi_2$, then by the IH $(T',T'),k' \vDash \varphi_2^L$, and $(T',T'),k' \vDash \varphi_1^L \rightarrow \varphi_2^L$ follows again.

In the opposite direction, if $(H',T'),k' \vDash \varphi_1^L \rightarrow \varphi_2^L$ then $\Longleftarrow$

1. either $(H',T'),k' \nvDash \varphi_1^L$ or $(H',T'),k' \vDash \varphi_2^L$, and
2. either $(T',T'),k' \nvDash \varphi_1^L$ or $(T',T'),k' \vDash \varphi_2^L$.

In the first case, if $(H',T'),k' \nvDash \varphi_1^L$, then by contraposition of the IH $(H,T),k \nvDash \varphi_1$, meaning that $(H,T),k \vDash \varphi_1 \rightarrow \varphi_2$. If, instead, $(H',T'),k' \vDash \varphi_2^L$, then by the IH $(H,T),k \vDash \varphi_2$, and $(H,T),k \vDash \varphi_1 \rightarrow \varphi_2$ follows again.

In the second case, if $(T',T'),k' \nvDash \varphi_1^L$, then by contraposition of the IH $(T,T),k \nvDash \varphi_1$, meaning that $(T,T),k \vDash \varphi_1 \rightarrow \varphi_2$. If, instead, $(T',T'),k' \vDash \varphi_2^L$, then by the IH $(T,T),k \vDash \varphi_2$, and $(T,T),k \vDash \varphi_1 \rightarrow \varphi_2$ follows again.

If $\varphi = \bigcirc\psi$, then $\varphi^L = \bigcirc^{n+2}\psi^L$. If $(H,T),k \vDash \bigcirc\psi$, then $(H,T),k+1 \vDash \psi$, and by IH $(H',T'),(k+1)' \vDash \psi^L$. Since $(k+1)' = k' + (n+2)$, this means that $(H',T'),k' \vDash \bigcirc^{n+2}\psi^L \equiv \varphi^L$. Vice versa, if $(H',T'),k' \vDash \bigcirc^{n+2}\psi^L$, then $(H',T'),(k+1)' \vDash \psi^L$, and by IH $(H,T),k+1 \vDash \psi$, meaning that $(H,T),k \vDash \bigcirc\psi$. $\varphi = \bigcirc\psi$

If $\varphi = \varphi_1 \mathbb{U} \varphi_2$, then $\varphi^L = \neg\alpha \vee \varphi_1^L \mathbb{U} \alpha \wedge \varphi_2^L$. If $(H,T),k \vDash \varphi_1 \mathbb{U} \varphi_2$, then by definition of the $\mathbb{U}$ operator $\exists j, k \leq j$, such that $(H,T),j \vDash \varphi_2$ and $(H,T),z \vDash \varphi_1$ for all $z \in [k..j)$. Since $(H,T),j \vDash \varphi_2$ and φ_2 is a subformula of φ, by the IH $(H',T'),j' \vDash \varphi_2^L$. Since $j' = (n+2)j$, j' is a start state, $(H',T'),j' \vDash \alpha$ and, consequently, $(H',T'),j' \vDash \alpha \wedge \varphi_2^L$. What is at this point left to show is that $(H',T'),u \vDash \neg\alpha \vee \varphi_1^L$ for all $u \in [k'..j')$. The proof of this goes by cases. If u is an inner or end state, then $(H',T'),u \vDash \neg\alpha$, meaning that $(H',T'),u \vDash \neg\alpha \vee \varphi_1^L$. If, instead, u is a start state, $u = (n+2)z$ for some $z \in [k..j)$, meaning that by the IH applied to $(H,T),z \vDash \varphi_1$, $(H',T'),u \vDash \varphi_1^L$, meaning, again, that $(H',T'),u \vDash \neg\alpha \vee \varphi_1^L$. Hence $(H',T'),j' \vDash \alpha \wedge \varphi_2^L$, while $(H',T'),u \vDash \neg\alpha \vee \varphi_1^L$ for all $u \in [k'..j')$, showing that $(H',T'),k' \vDash \neg\alpha \vee \varphi_1^L \mathbb{U} \alpha \wedge \varphi_2^L \equiv (\varphi_1 \mathbb{U} \varphi_2)^L$. $\varphi = \varphi_1 \mathbb{U} \varphi_2$

Vice versa, if $(H',T'),k' \vDash \neg\alpha \vee \varphi_1^L \mathbb{U} \alpha \wedge \varphi_2^L$, then by definition of $\mathbb{U}$ there exists a j', $k' \leq j'$, such that $(H',T'),j' \vDash \alpha \wedge \varphi_2^L$ and $(H',T'),z' \vDash \neg\alpha \vee \varphi_1^L$ for all $z' \in [k'..j')$. Since $(H',T'),j' \vDash \varphi_2^L$, by the IH $(H,T),j \vDash \varphi_2$, $k \leq j$. What needs to be proven at this point is that for all $z \in [k..j)$, $(H,T),z \vDash \varphi_1$. Let then z be one such index. Then $z' \in [k'..j')$, and since z' is a start state, $(H',T'),z' \nvDash \neg\alpha$, so that $(H',T'),z' \vDash \varphi_1^L$. By the IH, then, $(H,T),z \vDash \varphi_1$. $\Longleftarrow$

Finally, if $\varphi = \varphi_1 \mathbb{R} \varphi_2$ then $\varphi^L = \alpha \wedge \varphi_1^L \mathbb{R} \neg\alpha \vee \varphi_2^L$. If $(H,T),k \vDash \varphi_1 \mathbb{R} \varphi_2$, then either $\varphi = \varphi_1 \mathbb{R} \varphi_2$

1. $\forall j \geq k, (H,T),j \vDash \varphi_2$, or
2. $\exists j \geq k, (H,T),j \vDash \varphi_1$ and $\forall z \in [k..j], (H,T),z \vDash \varphi_2$.

In the first case, by the IH $(H',T'),j' \vDash \varphi_2^L$ for all start states $j' \geq k'$, and, in particular, $(H',T'),j' \vDash \neg\alpha \vee \varphi_2^L$. For all the inner or end states $j \geq k'$, instead, $(H',T'),j \vDash \neg\alpha$,

meaning again $(H',T'),j \vDash \neg\alpha \vee \varphi_2^L$. Hence for all states $j \geq k'$, irrespective of whether start, inner or end, $(H',T'),j \vDash \neg\alpha \vee \varphi_2^L$, meaning that $(H',T'),k' \vDash \alpha \wedge \varphi_1^L \mathbb{R} \neg\alpha \vee \varphi_2^L$.

In the second case, by the IH $(H',T'),j' \vDash \varphi_1^L$, $k' \leq j'$, and since j' is a start state, $(H',T'),j' \vDash \alpha \wedge \varphi_1^L$. Further, by the IH again, for all start states $z \in [k'..j'],(H',T'),z \vDash \varphi_2^L$ and, in particular, $(H',T'),z \vDash \neg\alpha \vee \varphi_2^L$. That $(H',T'),z \vDash \neg\alpha \vee \varphi_2^L$ also holds for the inner and end states follows by the fact that in these states, $(H',T'),z \vDash \neg\alpha$. Hence, for (H',T'), there exists one $j' \geq k'$ such that $(H',T'),j' \vDash \alpha \wedge \varphi_1^L$, and for all $z \in [k'..j']$, irrespective of whether start, inner or end states, $(H',T'),z' \vDash \neg\alpha \vee \varphi_2^L$, that is, $(H',T'),k' \vDash \alpha \wedge \varphi_1^L \mathbb{R} \neg\alpha \vee \varphi_2^L$.

$\Leftarrow$ Finally, if $(H',T'),k' \vDash \alpha \wedge \varphi_2^L \mathbb{R} \neg\alpha \vee \varphi_2^L$, then either

1. $(H',T'),u \vDash \neg\alpha \vee \varphi_2^L$ for all $u \geq k'$, or
2. $\exists j' \geq k',(H,T),j' \vDash \alpha \wedge \varphi_1^L$ and $\forall z \in [k'..j'],(H',T'),z \vDash \neg\alpha \vee \varphi_2^L$.

In the first case, since $\forall u \geq k,(H',T'),u \vDash \varphi_2^L$, by the IH applied to only the start states $u' \geq k'$, $\forall u \geq k,(H,T),u \vDash \varphi_2$, meaning that $(H,T),k \vDash \varphi_1 \mathbb{R} \varphi_2$.

In the second case, given that $(H',T'),j' \vDash \varphi_1^L$, by the IH $(H,T),j \vDash \varphi_1$. Further, since $(H',T'),z' \vDash \varphi_2^L$ for all start states $z' \in [k'..j']$, by the IH $\forall z \in [k..j],(H,T),z \vDash \varphi_2$. Hence $\exists j \geq k,(H,T),j \vDash \varphi_1$ and $\forall z \in [k..j],(H,T),z \vDash \varphi_2$, meaning that $(H,T),k \vDash \varphi_1 \mathbb{R} \varphi_2$. ■

A special case of Lemma 5.69 occurs when $H = T$.

LEMMA 5.70 Let φ be a THT formula defined over the finite alphabet A and τ the AS transition system of φ. Then for any LTL trace T over 2^A and any $k \geq 0$

$$T,k \vDash_{\text{LTL}} \varphi \text{ iff } T',(n+2)k \vDash_{\text{LTL}} \varphi^L,$$

where T' is the unfolding of T over τ. □

One final lemma useful to show the EXPSPACE–hardness of $\text{TELTS}_\exists$ is the following, showing a correspondence between the TEMs of a given formula φ and the TEMs found within its AS transition system.

LEMMA 5.71 Let φ be a THT formula, τ the AS transition system of φ, T an LTL trace and T' the unfolding of T over τ. Then T is a TEM of φ iff T' is a TEM of φ', where

$$\varphi' \stackrel{\text{def}}{=} \varphi^L \wedge \varphi_\alpha \wedge \varphi_\beta,$$
$$\varphi_\alpha \stackrel{\text{def}}{=} \Box(\alpha \vee \neg\alpha),$$
$$\varphi_\beta \stackrel{\text{def}}{=} \Box(\beta \vee \neg\beta).$$

□

OBSERVATION 5.66 It is not possible to state that T is a TEM of φ iff T' is a TEM of φ^L, because the α and β atoms found in T' need to be justified by some rule. If they are not, then the semantics of TEL will allow them to be minimized, and supposing T' not to be a TEM of φ' will not allow to conclude that T is not a TEM of φ. □

OBSERVATION 5.67 One initial attempt at justifying α along T' is to define φ_α as

$$\alpha \wedge \Box(\alpha \rightarrow \bigcirc^{n+2}\alpha).$$

On a closer inspection, however, the definition

$$\Box(\alpha \vee \neg\alpha)$$

obtains the same behavior as the first one, with less symbols and without having to use the $\bigcirc$ operator. (A similar reasoning holds for β.) □

Proof If T is a TEM of φ, then, in particular, T is an LTL model of φ, and by Lemma 5.70, T' is an LTL model of φ^L. Furthermore, T' is a TEM of φ'. In fact, if T' was not a TEM of φ', then a subtrace $H' < T'$ would exist such that $(H', T') \vDash_{\text{THT}} \varphi'$. Since $(H', T') \vDash_{\text{THT}} \varphi_\alpha \wedge \varphi_\beta$, the α and β atoms along the (H', T') trace cannot be minimized, meaning that $H' < T'$ is due to the minimization of some atom in A. This, in turn, means that a subtrace $H < T$ can be obtained such that H' is the unfolding of H over τ. By Lemma 5.69, since $(H', T'), 0 \vDash_{\text{THT}} \varphi^L$, $(H, T), 0 \vDash_{\text{THT}} \varphi$ would also hold. But T was supposed to be a TEM of φ, so one such $H' < T'$ cannot exist and T' is a TEM of φ^L.

Vice versa, if T' is a TEM of φ', then, in particular, T' is an LTL model of φ', and so of φ^L, and by Lemma 5.70 T is an LTL model of φ. T is also a TEM of φ, because if it was not a trace $H < T$ would exist for which $(H, T) \vDash_{\text{THT}} \varphi$, and by Lemma 5.69 $(H', T') \vDash_{\text{THT}} \varphi^L$, where H', $H' < T'$, is the unfolding of H over τ. But T' was supposed to be a TEM of φ^L, so one such $H < T$ cannot exist and T is a TEM of φ. ■

Theorem 5.72 Deciding $\text{TELTS}_\exists$ is Expspace–hard. □

Proof To show the Expspace–hardness of deciding $\text{TELTS}_\exists$, a Karp reduction $f : \Sigma_{\text{TEL}} \to \Sigma_{\text{TELTS}_\exists}$ is defined such that $\varphi \in \text{TEL}$ iff $f(\varphi) \in \text{TELTS}_\exists$. Since deciding TEL is Expspace–hard (L. Bozzelli and D. Pearce [53]), deciding $\text{TELTS}_\exists$ also turns out to be Expspace–hard.

The reduction f is defined as $f(\varphi) = (\tau, \varphi')$, where τ is the AS transition system of φ and $\varphi' \stackrel{\text{def}}{=} \varphi^L \wedge \varphi_\alpha \wedge \varphi_\beta$.

What needs to be shown at this point is that f is a function from Σ_{TEL} to $\Sigma_{\text{TELTS}_\exists}$, that $\varphi \in \text{TEL}$ iff $f(\varphi) \in \text{TELTS}_\exists$ and that f is computable in polynomial time.

f is a function from Σ_{TEL} to $\Sigma_{\text{TELTS}_\exists}$, because τ is a transition system without terminal states and φ' a THT formula.

To show the second point, $\varphi \in \text{TEL}$ is supposed. Since $\varphi \in \text{TEL}$, an LTL trace T exists which is a TEM of φ, and by Lemma 5.71 the unfolding T' of T over τ is a TEM of φ'. Therefore a trace from within τ which is a TEM of φ' exists, and $(\tau, \varphi') \in \text{TELTS}_\exists$.

Conversely, if $\varphi \in \Sigma_{\text{TEL}}$ is such that $(\tau, \varphi') = f(\varphi)$ belongs to $\text{TELTS}_\exists$, then an LTL trace T' in τ exists which is a TEM of φ'. Further, an LTL trace T such that T' is the unfolding of T over τ also exists, and by Lemma 5.71 T is a TEM of φ. Therefore, $\varphi \in \text{TEL}$.

Finally, f can be computed in polynomial time with respect to the number of symbols in φ. In fact, the AS transition system τ obtained from f is only a function of the finite alphabet A used to construct φ. In particular, τ possesses $2 \cdot |A| + 2$ states and $4(|A| - 1) + 5$ edges, and the labeling function L of τ does not assign more than one propositional atom to each state. The number of symbols constituting the formula φ^L, instead, is at most $(|A| + 2) \cdot \text{sub}(\varphi)$, which is polynomial with respect to that of φ, whereas the number of symbols of φ_α and φ_β is constant. ■

What remains to be shown for the $\text{TELTS}_\exists$ problem is its membership to Expspace. To that end, it might be useful to remember that in the $\text{TELTS}_\forall$ section, the notion of T -embedding was extended from linear traces to transition systems, obtaining a similar notion named TS-embedding. TS-embeddings can, in turn, be modified to obtain what could be thought of as a weakened version, named *quasiembeddings* or, since transition systems are dealt with, *TS-quasiembeddings*.

Given a transition system τ, the TS-embedding φ_τ of τ *enforces* the satisfaction of the atoms present in the states of τ. In the context of a three-valued logic like HT or THT,

however, it might also be desirable to enforce the intermediate truth value of the atoms present in the states of τ, sometimes referred to as *assumed* or *unknown*. A formula like $\neg\neg a$, expressible in natural language as "not false," can be satisfied in the HT logic (and, by extension, in the THT one) in two different ways, either by the interpretation $(\emptyset, \{a\})$ or by the interpretation $(\{a\}, \{a\})$. The first interpretation assigns the intermediate truth value "assumed" or "unknown" to a, whereas the second the "true" truth value.

DEFINITION 5.68 Given a transition system $\tau = (S, \alpha, \rightarrow, I, A, L)$, the *TS-quasiembedding* of τ is the formula[25]

$$\begin{aligned}\psi_\tau \stackrel{\text{def}}{\equiv} & \bigvee_{s \in I} s \\ & \wedge \bigwedge_{s \in S} \Box \left(s \rightarrow \bigcirc \bigvee_{s \rightarrow t} t \right) \\ & \wedge \bigwedge_{s \in S} \Box \left(s \rightarrow \bigwedge_{a \in A \cap L(s)} \neg\neg a \wedge \bigwedge_{a \in A \setminus L(s)} \neg a \right) \\ & \wedge \bigwedge_{s \in S} \Box \left(s \rightarrow \bigwedge_{\substack{t \in S \\ t \neq s}} \neg t \right),\end{aligned}$$

constructed over the alphabet $A \cup S$. □

TS-quasiembeddings are defined almost exactly as *TS*-embeddings, with the only difference that each atom a in the state of a given transition system τ is required to be not false, expressed as $\neg\neg a$, rather than true, expressed as a. The purpose of *TS*-quasiembeddings remains however equivalent to those of *TS*-embeddings: *TS*-quasiembeddings allow turning a model checking problem into a satisfiability one.

LEMMA 5.73 Let φ be a THT formula constructed over the alphabet A and $\tau = (S, \alpha, \rightarrow, I, A, L)$ a transition system. Then an LTL trace T in τ exists which is a TEM of φ iff $\varphi \wedge \psi_\tau$ admits a TEM. □

As the proof of this lemma is somehow cumbersome to present in a single piece, it is split into two smaller ones.

PROOF Consequence of Lemma 5.74 and Lemma 5.75. ■

LEMMA 5.74 Let φ be a THT formula constructed over the alphabet A and $\tau = (S, \alpha, \rightarrow, I, A, L)$ a transition system, for which an LTL trace T that is a TEM of φ exists. Then a TEM of $\varphi \wedge \psi_\tau$ also exists. □

PROOF If an LTL trace T in τ which is a TEM of φ exists, then an LTL trace T' extending T also exists which is a TEM of $\varphi \wedge \psi_\tau$. T' is defined as that extension of T over the alphabet $A \cup S$ which marks each state T_i with a state atom $\sigma_i \in S$, indicating which state in τ T_i obtains its atoms from. Formally

$$T' \stackrel{\text{def}}{\equiv} (T_1 \cup \{\sigma_1\}) \cdot (T_2 \cup \{\sigma_2\}) \cdot \cdots.$$

Since T' is extended only over atoms in S and φ is defined over the alphabet A, $(T', T') \vDash_{\text{THT}} \varphi$ still holds. Further, $(T', T') \vDash_{\text{THT}} \psi_\tau$, for in fact

[25] Highlighted in blue, the only difference between a *TS*-quasiembedding and a *TS*-embedding.

1. (T', T') satisfies the first formula of ψ_τ, because an initial state from I is by construction present in T_1;
2. (T', T') satisfies the second formula of ψ_τ, because the transition relation $\rightarrow$ of τ is respected;
3. (T', T') satisfies the third formula of ψ_τ, because at each state T_i:
    1. For each atom $a \in A \cap L(\sigma_i)$, $\neg\neg a$ holds;
    2. For each atom $a \in A \setminus L(\sigma_i)$, $\neg a$ holds.
4. (T', T') satisfies the fourth formula of ψ_τ, because at most one state variable σ_i is always present in a state of T'.

Hence $(T', T') \vDash_{\text{THT}} \varphi$ and $(T', T') \vDash_{\text{THT}} \psi_\tau$, meaning that $(T', T') \vDash_{\text{THT}} \varphi \wedge \psi_\tau$. Since (T', T') is a total THT interpretation, what needs to be shown for it to be a TEM of $\varphi \wedge \psi_\tau$ is that $\nexists H' < T'$ for which $(H', T') \vDash_{\text{THT}} \varphi \wedge \psi_\tau$. If, by absurdity, one such H' existed, then some atom $a \in A$ or some atom $\sigma_i \in S$ initially present in T' would be removed from H'. However

1. If, for some i, $a \in A$ were to be removed from T'_i, the same atom could be removed from T_i, obtaining an $H < T$ that would show T not to be a TEM of φ, contrary to the initial hypothesis;
2. If, for some i, $\sigma_i \in S$ were to be removed from T'_i instead, then the second formula of ψ_τ would no longer hold, for the $\rightarrow$ relation of τ would not be respected.

Hence (T', T') is a total THT interpretation satisfying $\varphi \wedge \psi_\tau$, for which no $H' < T'$ exists such that $(H', T') \vDash_{\text{THT}} \varphi \wedge \psi_\tau$, meaning that (T', T') is a TEM of $\varphi \wedge \psi_\tau$. ■

Lemma 5.75 Let φ be a THT formula constructed over the alphabet A and $\tau = (S, \alpha, \rightarrow, I, A, L)$ a transition system, such that $\varphi \wedge \psi_\tau$ admits a TEM. Then an LTL trace T from τ exists which is a TEM of φ. □

Proof If $\varphi \wedge \psi_\tau$ admits a TEM, then an LTL trace T constructed over the alphabet $A \cup S$ exists which is a TEM of $\varphi \wedge \psi_\tau$. From T, its projection T' over the alphabet A, defined as $T'_i \stackrel{\text{def}}{=} T_i \cap A$, can be constructed. By virtue of the ψ_τ embedding, T' is a trace from within τ. Further, since φ is constructed over the alphabet A, $(T', T') \vDash_{\text{THT}} \varphi$ continues to hold. Since (T', T') is a total THT interpretation and $(T', T') \vDash_{\text{THT}} \varphi$, what remains to prove for (T', T') to be a TEM of φ is that $\nexists H' < T'$ such that $(H', T') \vDash_{\text{THT}} \varphi$.

If an $H' < T'$ existed for which $(H', T') \vDash_{\text{THT}} \varphi$, then an index i would exist such that $a \in T'_i \setminus H'_i$ for some $a \in A$. If this atom a was removed from position i of the corresponding (T, T) interpretation, to obtain a non-total (H, T) interpretation with $H < T$, then $(H, T) \vDash_{\text{THT}} \varphi \wedge \psi_\tau$ would still hold, for

1. $(H', T') \vDash_{\text{THT}} \varphi$, and consequently $(H, T) \vDash_{\text{THT}} \varphi$, still holds, and
2. $(H, T) \vDash_{\text{THT}} \psi_\tau$ continues to hold, because
    1. The first, second and fourth formulas of ψ_τ, defined only over S, are not affected by the removal of an atom from A, while
    2. For formula 3 of ψ_τ, if $(T, T), j \vDash_{\text{THT}} a$ used to hold at state j of (T, T), then $(H, T), j \vDash_{\text{THT}} \neg\neg a$ would continue to hold.

However, (T, T) was already supposed to be a TEM of $\varphi \wedge \psi_\tau$, so one such $H' < T'$ cannot exist. Hence (T', T') is a TEM of φ. ■

All the notions needed to prove the Expspace membership of $\text{Telts}_\exists$ are at this point available.

Theorem 5.76 $\text{Telts}_\exists \in$ Expspace. □

PROOF IDEA Given a transition system τ and a THT formula φ, when asked whether an LTL trace T in τ exists which is a TEM of φ, a TEM of $\varphi \wedge \psi_\tau$ can be searched for instead. ■

PROOF To show $\text{TELTS}_\exists$ to belong to EXPSPACE, a Karp reduction $f : \Sigma_{\text{TELTS}_\exists} \to \Sigma_{\text{TEL}}$ is constructed. Given that TEL belongs to EXPSPACE (P. Cabalar and S. Demri [52]), $\text{TELTS}_\exists$ also results to belong to EXPSPACE.

Given an instance $(\tau, \varphi) \in \Sigma_{\text{TELTS}_\exists}$,

$$f((\tau, \varphi)) \stackrel{\text{def}}{=} \varphi \wedge \psi_\tau,$$

where ψ_τ is the TS-quasiembedding of τ. What needs to be shown at this point is that f is a function from $\Sigma_{\text{TELTS}_\exists}$ to Σ_{TEL}, that $(\tau, \varphi) \in \text{TELTS}_\exists$ iff $\varphi \wedge \psi_\tau \in \text{TEL}$ and that f is computable in polynomial time with respect to the size of the instance (τ, φ).

f is certainly a function from $\Sigma_{\text{TELTS}_\exists}$ to Σ_{TEL}, for $\varphi \wedge \psi_\tau$ is a syntactically valid THT formula.

For the second property, $(\tau, \varphi) \in \text{TELTS}_\exists$ is first assumed. If $(\tau, \varphi) \in \text{TELTS}_\exists$, then an LTL trace T from within τ such that (T, T) is a TEM of φ exists, and by Lemma 5.74 a TEM for $\varphi \wedge \psi_\tau$ exists, showing $f((\tau, \varphi))$ to belong to TEL.

Vice versa, if $(\tau, \varphi) \in \Sigma_{\text{TELTS}_\exists}$ is an instance of $\text{TELTS}_\exists$ such that $f((\varphi, \tau)) = \varphi \wedge \psi_\tau \in \text{TEL}$, then a TEM of $\varphi \wedge \psi_\tau$ exists and by Lemma 5.75 an LTL trace T from τ exists which is a TEM of φ, meaning that $(\tau, \varphi) \in \text{TELTS}_\exists$.

Finally, f can be computed in polynomial time with respect to the size of τ and φ, because $f((\tau, \varphi))$ entails copying φ first and constructing ψ_τ afterwards, which can be done in linear time with respect to the size of ψ_τ. ■

Chapter 6

Fragments of the Telts problem

"Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better."

— *Edsger W. Dijkstra.*

In the same way Chapter 4 investigated fragments of the problem of model checking over infinite traces, it is possible to study fragments of the problem of model checking over transition systems. This chapter is devoted to that purpose, restricting the Telts problem in the same way the Telmc one was in Chapter 4.

In this quick-paced chapter, it will be shown how restricting the $\text{Telts}_\forall$ and $\text{Telts}_\exists$ problems to only the release operator $\mathbb{R}$ does not affect the complexity of deciding them. In fact, deciding $\text{Telts}_\forall(\mathbb{R})$ remains Pspace–complete (Theorem 6.81) and, similarly, deciding $\text{Telts}_\exists(\mathbb{R})$ Expspace–complete (Theorem 6.87).

A summary of the results of the chapter is shown in Table 8.

Problem	Complexity
$\text{Telts}_\forall(\mathbb{R})$	Pspace–complete (Theorem 6.81)
$\text{Telts}_\forall(\bigcirc, \mathbb{R})$	Pspace–complete (Corollary 6.82)
$\text{Telts}_\forall(\mathbb{U}, \mathbb{R})$	Pspace–complete (Corollary 6.83)
$\text{Telts}_\exists(\mathbb{R})$	Expspace–complete (Theorem 6.87)
$\text{Telts}_\exists(\bigcirc, \mathbb{R})$	Expspace–complete (Corollary 6.90)
$\text{Telts}_\exists(\mathbb{U}, \mathbb{R})$	Expspace–complete (Corollary 6.91)

Table 8: Summary of the results of the chapter.

6.1. The $\text{Telts}_\forall(\mathbb{R})$ fragment

$\text{Telts}_\forall(\mathbb{R})$ is the fragment of the $\text{Telts}_\forall$ problem where only the temporal operator $\mathbb{R}$ is allowed. Like $\text{Telts}_\forall$, deciding this problem is still Pspace–complete. Its Pspace–hardness results from the fact that $\text{Ltlts}_\forall(\mathbb{R})$ is still Pspace–hard, and that since the $\mathbb{R}$ operator, and in particular the $\Box$ operator, is allowed a reduction from $\text{Ltlts}_\forall(\mathbb{R})$ to $\text{Telts}_\forall(\mathbb{R})$ in the style of Theorem 5.62 is still possible. The membership of $\text{Telts}_\forall(\mathbb{R})$ to Pspace continues to hold because $\text{Telts}_\forall(\mathbb{R})$ is a subset of $\text{Telts}_\forall$.

Theorem 6.77 Deciding $\text{Ltlts}_\exists(\mathbb{R})$ is Pspace–hard. □

Proof A. P. Sistla and E. M. Clarke [55] show that deciding $\text{Ltlts}_{\exists}(\mathbb{U})$ is Pspace–hard. Since the until operator can be rewritten in terms of the release one as $\varphi_1 \mathbb{U} \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \mathbb{R} \neg\varphi_2)$, deciding $\text{Ltlts}_{\exists}(\mathbb{R})$ also results to be Pspace–hard. ■

Theorem 6.78 Deciding $\text{Ltlts}_{\forall}(\mathbb{R})$ is Pspace–hard. □

Proof Since by Theorem 6.77 deciding $\text{Ltlts}_{\exists}(\mathbb{R})$ is Pspace–hard, the same reasoning adopted in the proof of Theorem 5.60, showing the Pspace–hardness of $\text{Ltlts}_{\forall}$, can be replicated to obtain a reduction from $\overline{\text{Ltlts}_{\exists}(\mathbb{R})}$ to $\text{Ltlts}_{\forall}(\mathbb{R})$. ■

Theorem 6.79 Deciding $\text{Telts}_{\forall}(\mathbb{R})$ is Pspace–hard. □

Proof Since in $\text{Telts}_{\forall}(\mathbb{R})$ the use of the $\mathbb{R}$ operator, and in particular of the □ operator, is allowed the $\varphi_{\text{EM}}(A)$ formula from Proposition 5.61 can be used to construct a Karp reduction from $\text{Ltlts}_{\forall}(\mathbb{R})$ to $\text{Telts}_{\forall}(\mathbb{R})$ in the same way it is done in the proof of Theorem 5.62. ■

Corollary 6.80 $\text{Telts}_{\forall}(\mathbb{R}) \in$ Pspace. □

Proof Since $\text{Telts}_{\forall}(\mathbb{R}) \subset \text{Telts}_{\forall}$ and $\text{Telts}_{\forall} \in$ Pspace, $\text{Telts}_{\forall}(\mathbb{R}) \in$ Pspace. ■

Theorem 6.81 Deciding $\text{Telts}_{\forall}(\mathbb{R})$ is Pspace–complete. □

Proof Consequence of Theorem 6.79 and Corollary 6.80. ■

Corollary 6.82 Deciding $\text{Telts}_{\forall}(\bigcirc, \mathbb{R})$ is Pspace–complete. □

Corollary 6.83 Deciding $\text{Telts}_{\forall}(\mathbb{U}, \mathbb{R})$ is Pspace–complete. □

6.2. The $\text{Telts}_{\exists}(\mathbb{R})$ fragment

$\text{Telts}_{\exists}(\mathbb{R})$ is the fragment of the $\text{Telts}_{\exists}$ problem where only the release operator $\mathbb{R}$ is allowed, and like $\text{Telts}_{\exists}$, deciding it is still Expspace–complete. Showing the Expspace–hardness of the $\text{Telts}_{\exists}(\mathbb{R})$ problem can be done by readapting the proof of the Expspace–hardness of the unconstrained $\text{Telts}_{\exists}$ problem, by revising the definition of the L-transformation in a way that no $\bigcirc$ operators are used without need. For lack of a better name, this modified transformation is termed *M-transformation*.

Definition 6.69 *(M-transformation.)* The *M-transformation* is a function from THT formulas to THT formulas, defined by structural recursion as[26]:

$$
\begin{aligned}
\bot^M &\stackrel{\text{def}}{=} \bot, \\
p_i^M &\stackrel{\text{def}}{=} p_i \mathbb{R} \neg\beta, \\
(\varphi_1 \vee \varphi_2)^M &\stackrel{\text{def}}{=} \varphi_1^M \vee \varphi_2^M, \\
(\varphi_1 \wedge \varphi_2)^M &\stackrel{\text{def}}{=} \varphi_1^M \wedge \varphi_2^M, \\
(\varphi_1 \to \varphi_2)^M &\stackrel{\text{def}}{=} \varphi_1^M \to \varphi_2^M, \\
(\varphi_1 \mathbb{R} \varphi_2)^M &\stackrel{\text{def}}{=} \alpha \wedge \varphi_1^M \mathbb{R} \neg\alpha \vee \varphi_2^M.
\end{aligned}
$$

□

This modification requires proving again parts of the lemmas that were used in the case of the L-transformation. The first one is Lemma 5.69.

[26] Highlighted in blue, the only actual difference between the M-transformation and the L-transformation.

LEMMA 6.84 Let φ be a THT formula defined over the finite alphabet A and τ the AS transition system of φ. Then for any THT trace (H,T) over 2^A and any $k \geq 0$

$$(H,T),k \vDash_{\text{THT}} \varphi \text{ iff } (H',T'),(n+2)k \vDash_{\text{THT}} \varphi^M,$$

where H' and T' are the unfolding of H and T over τ, respectively. □

PROOF By structural induction over the syntax of THT. Since the M-transformation is defined almost exactly as the L-transformation, differing only by the p_i case, the proof of Lemma 5.69 is proposed again, by modifying the proof of the $\varphi = p_i$ case. In particular, the same notational conventions of the other proof will be also used here.

If $\varphi = p_i$, then $\varphi^M = p_i \,\mathbb{R}\, \neg\beta$. If $(H,T),k \vDash p_i$, then $p_i \in H_k$, and since H' is an unfolding of H over τ, $p_i \in H'_{k'+i}$. This means, in particular, that p_i eventually holds at or after position k' of (H',T'). Further, $\neg\beta$ holds from position k' of (H',T') up to position $k'+i$, meaning that $(H',T'),k' \vDash p_i \,\mathbb{R}\, \neg\beta$.

Vice versa, if $(H',T'),k' \vDash p_i \,\mathbb{R}\, \neg\beta$, then p_i holds at or after position k' of (H',T'), because $\neg\beta$ does not hold forever. In fact, p_i holds between position k' of (H',T') and position $k'+n+2$, $n \stackrel{\text{def}}{=} |A|$, that is within the finite segment of length $n+2$ starting at position k'. Since position $k'+i$ is the only state within this fragment where p_i can hold, $(H',T'),k'+i \vDash p_i$. Since H is an unfolding of H', $p_i \in H_k$, meaning that $(H,T),k \vDash p_i$. ■

The special case of Lemma 5.69 to LTL traces, Lemma 5.70, continues to hold for the M -transformation, and so does Lemma 5.71.

LEMMA 6.85 Let φ be a THT formula defined over the finite alphabet A and τ the AS transition system of φ. Then for any LTL trace T over 2^A and any $k \geq 0$

$$T,k \vDash_{\text{LTL}} \varphi \text{ iff } T',(n+2)k \vDash_{\text{LTL}} \varphi^M,$$

where T' is the unfolding of T over τ. □

LEMMA 6.86 Let φ be a THT formula, τ the AS transition system of φ, T an LTL trace and T' the unfolding of T over τ. Then T is a TEM of φ iff T' is a TEM of φ', where

$$\begin{aligned}\varphi' &\stackrel{\text{def}}{=} \varphi^M \wedge \varphi_\alpha \wedge \varphi_\beta,\\ \varphi_\alpha &\stackrel{\text{def}}{=} \Box(\alpha \vee \neg\alpha),\\ \varphi_\beta &\stackrel{\text{def}}{=} \Box(\beta \vee \neg\beta).\end{aligned}$$

□

THEOREM 6.87 Deciding TELTS$_\exists(\mathbb{R})$ is EXPSPACE–complete. □

PROOF Consequence of Theorem 6.88 and Corollary 6.89. ■

THEOREM 6.88 Deciding TELTS$_\exists(\mathbb{R})$ is EXPSPACE–hard. □

PROOF Exactly as in the proof of Theorem 5.72, except that this time the Karp reduction f is defined as $f(\varphi) \stackrel{\text{def}}{=} (\tau,\varphi'')$, where $\varphi'' \stackrel{\text{def}}{=} \varphi^M \wedge \varphi_\alpha \wedge \varphi_\beta$. ■

COROLLARY 6.89 TELTS$_\exists(\mathbb{R}) \in$ EXPSPACE. □

PROOF Since TELTS$_\exists(\mathbb{R}) \subset$ TELTS$_\exists$ and TELTS$_\exists \in$ EXPSPACE, TELTS$_\exists(\mathbb{R}) \in$ EXPSPACE too. ■

Since TELTS$_\exists(\bigcirc,\mathbb{R})$ and TELTS$_\exists(\mathbb{U},\mathbb{R})$ are supersets of TELTS$_\exists(\mathbb{R})$, these two fragments remain EXPSPACE–hard.

Corollary 6.90 Deciding $\text{Telts}_{\exists}(\bigcirc, \mathbb{R})$ is Expspace–complete. □

Corollary 6.91 Deciding $\text{Telts}_{\exists}(\mathbb{U}, \mathbb{R})$ is Expspace–complete. □

Chapter 7

Conclusion

"Look at you," he said finally. "I say this in no spirit of contempt, but look at you! The material you are made of is soft and flabby, lacking endurance and strength, depending for energy upon the inefficient oxidation of organic material—like that." He pointed a disapproving finger at what remained of Donovan's sandwich. "Periodically you pass into a coma and the least variation in temperature, air pressure, humidity, or radiation intensity impairs your efficiency."

— Isaac Asimov; "I, Robot" (1950).

This work studied the complexity of the model checking problem for TEL, the problem of deciding whether a given interpretation I is the intended model of a THT formula φ. The work first analyzed the problem under linear traces (Chapter 3 and Chapter 4), where the interpretation I is taken to be a sequence of atomic propositions of finite or infinite length, and subsequently under transition systems (Chapter 5 and Chapter 6), where the interpretation I is a transition system.

Chapter 3 studied the complexity of ELMC, the model checking problem for EL, and subsequently the one of TELMC_f and TELMC, the model checking problem for the finite-trace and the infinite-trace version of TEL. Deciding ELMC was shown to be coNP–complete (Theorem 3.13), and deciding TELMC_f was also shown to be coNP–complete (Theorem 3.17). In contrast, deciding TELMC was proven PSPACE–complete (Theorem 3.23).

Chapter 4 analyzed the same problem as Chapter 3, but over formulas where some kind of restriction is imposed. As a first observation, it was pointed out how deciding TELMC_2^1 (the TELMC fragment where the input formulas have a temporal height of value no greater than 2 and an implication height of value no greater than 1) is PSPACE–complete (Corollary 4.34). It was then shown how deciding TELMC_1 (the fragment of the TELMC problem where the formulas have a temporal height of value no greater than 1) is coNP–complete (Theorem 4.42). Deciding the $\text{TELMC}(\bigcirc, \mathbb{U})$ problem (the TELMC fragment where only the $\bigcirc$ and $\mathbb{U}$ operators are allowed) was also proven coNP–complete (Theorem 4.44). However, deciding the $\text{TELMC}(\mathbb{U}, \mathbb{R})$ problem was shown PSPACE–complete (Theorem 4.52) and in fact it was so even for the $\text{TELMC}_2^1(\mathbb{U}, \mathbb{R})$ fragment (Corollary 4.54), that is even when restricted to formulas with a temporal height no greater than 2 and an implication height no greater than 1. Similarly, deciding the $\text{TELMC}(\bigcirc, \mathbb{R})$ and $\text{TELMC}_2^1(\bigcirc, \mathbb{R})$ fragments was shown to be PSPACE–complete (Theorem 4.55 and Corollary 4.58).

Chapter 5 generalized the setting of Chapter 3 from interpretations as linear traces to interpretations as transition systems. The chapter analyzed the complexity of the TELTS

problem, which was divided into two variants, the universal one, $\textsc{Telts}_\forall$, and the existential one, $\textsc{Telts}_\exists$. Deciding $\textsc{Telts}_\forall$ was shown to be Pspace–complete (Theorem 5.67), while deciding $\textsc{Telts}_\exists$ Expspace–complete (Theorem 5.68).

Chapter 6 finally showed that deciding $\textsc{Telts}_\forall(\mathbb{R})$, the $\textsc{Telts}_\forall$ fragment where only the release $\mathbb{R}$ operator is allowed, is still Pspace–complete (Theorem 6.81) and similarly, that deciding $\textsc{Telts}_\exists(\mathbb{R})$, the $\textsc{Telts}_\exists$ fragment where only the $\mathbb{R}$ operator is allowed, is still Expspace–complete (Theorem 6.87).

A summary of the results is given in Table 9. A comparison of the results of the satisfiability problem for TEL (L. Bozzelli and D. Pearce [53]) with the model checking one is given in Table 10.

	Problem	Complexity
Chapter 3	$\textsc{Elmc}$	coNP–complete (Theorem 3.13)
	$\textsc{Telmc}_f$	coNP–complete (Theorem 3.17)
	$\textsc{Telmc}$	Pspace–complete (Theorem 3.23)
Chapter 4	$\textsc{Telmc}_2^1$	Pspace–complete (Corollary 4.34)
	$\textsc{Telmc}_1$	coNP–complete (Theorem 4.42)
	$\textsc{Telmc}(\bigcirc, \mathbb{U})$	coNP–complete (Theorem 4.44)
	$\textsc{Telmc}(\bigcirc)$	coNP–complete (Corollary 4.47)
	$\textsc{Telmc}(\mathbb{U})$	coNP–complete (Corollary 4.48)
	$\textsc{Telmc}(\mathbb{U}, \mathbb{R})$	Pspace–complete (Theorem 4.52)
	$\textsc{Telmc}_2^1(\mathbb{U}, \mathbb{R})$	Pspace–complete (Corollary 4.54)
	$\textsc{Telmc}(\bigcirc, \mathbb{R})$	Pspace–complete (Theorem 4.55)
	$\textsc{Telmc}_2^1(\bigcirc, \mathbb{R})$	Pspace–complete (Corollary 4.58)
Chapter 5	$\textsc{Telts}_\forall$	Pspace–complete (Theorem 5.67)
	$\textsc{Telts}_\exists$	Expspace–complete (Theorem 5.68)
Chapter 6	$\textsc{Telts}_\forall(\mathbb{R})$	Pspace–complete (Theorem 6.81)
	$\textsc{Telts}_\forall(\bigcirc, \mathbb{R})$	Pspace–complete (Corollary 6.82)
	$\textsc{Telts}_\forall(\mathbb{U}, \mathbb{R})$	Pspace–complete (Corollary 6.83)
	$\textsc{Telts}_\exists(\mathbb{R})$	Expspace–complete (Theorem 6.87)
	$\textsc{Telts}_\exists(\bigcirc, \mathbb{R})$	Expspace–complete (Corollary 6.90)

	$\textsc{Telts}_{\exists}(\mathbb{U}, \mathbb{R})$	$\textsc{Expspace}$–complete (Corollary 6.91)

Table 9: Contributions of the work.

Language	Satisfiability	Model checking
THT	$\textsc{Expspace}$–complete	$\textsc{Pspace}$–complete
$\mathrm{THT}^{k+1}_{m+1}(\mathbb{U})$	$\textsc{Expspace}$–complete	coNP–complete
$\mathrm{THT}(\bigcirc)$, $\mathrm{THT}(\Diamond)$, $\mathrm{THT}(\bigcirc, \Diamond)$	Σ_2–complete	coNP–complete
THT_1, $\mathrm{THT}^{k+1}_1(\Diamond, \Box)$, $\mathrm{THT}^{k+1}_1(\mathbb{U})$, $\mathrm{THT}^{k+1}_1(\mathbb{R})$	$\textsc{Nexptime}$–complete	coNP–complete
$\mathrm{THT}_1(\bigcirc, \Box)$	Σ_2–complete	coNP–complete
$\mathrm{THT}^1(\bigcirc, \mathbb{R})$	$\textsc{Pspace}$–complete	$\textsc{Pspace}$–complete
$\mathrm{THT}^1(\mathbb{U})$, $\mathrm{THT}^1(\bigcirc, \mathbb{U})$	$\textsc{Pspace}$–complete	coNP–complete
THT^1_1	NP–complete	coNP–complete
THT^0	• $\textsc{Pspace}$–hard • $\in \textsc{Expspace}$	$\in \textsc{Pspace}$
$\mathrm{THT}_0 = \mathrm{HT}$	Σ_2–complete	coNP–complete

Table 10: Comparison of the complexity of the satisfiability and the model checking problem for TEL.

7.1. Future work

A number of problems within this work remain unaddressed which might be made the focus of future research efforts. First among these is the problem of determining the complexity of the $\textsc{Telmc}(\mathbb{R})$ problem, the $\textsc{Telmc}$ fragment where only the release operator $\mathbb{R}$ is allowed. For $\textsc{Telmc}(\mathbb{R})$, the following is supposed.

$\textsc{Conjecture}$ 7.92 Deciding $\textsc{Telmc}(\mathbb{R})$ is $\textsc{Pspace}$–complete. □

$\textsc{Proof}$ Consequence of Conjecture 7.93 and Corollary 7.94. ■

$\textsc{Conjecture}$ 7.93 Deciding $\textsc{Telmc}(\mathbb{R})$ is $\textsc{Pspace}$–hard. □

$\textsc{Conjectured proof}$ Even though no use of the next operator $\bigcirc$ is allowed, the $\textsc{Pspace}$–hardness proof of Theorem 4.56 can be readapted by populating the saturated trace T of the reduction with an alternating pair of propositions (like the black and white propositions

b and w of Theorem 4.53) allowing one state to be distinguished from the other by the use of the $\mathbb{R}$ operator only. ■

Corollary 7.94 $\textsc{Telmc}(\mathbb{R}) \in \textsc{Pspace}$. □

Proof Since $\textsc{Telmc}(\mathbb{R}) \subset \textsc{Telmc}$ and $\textsc{Telmc} \in \textsc{Pspace}$, $\textsc{Telmc}(\mathbb{R}) \in \textsc{Pspace}$. ■

If Conjecture 7.92 was proven true, then Figure 9 would look more like Figure 15.

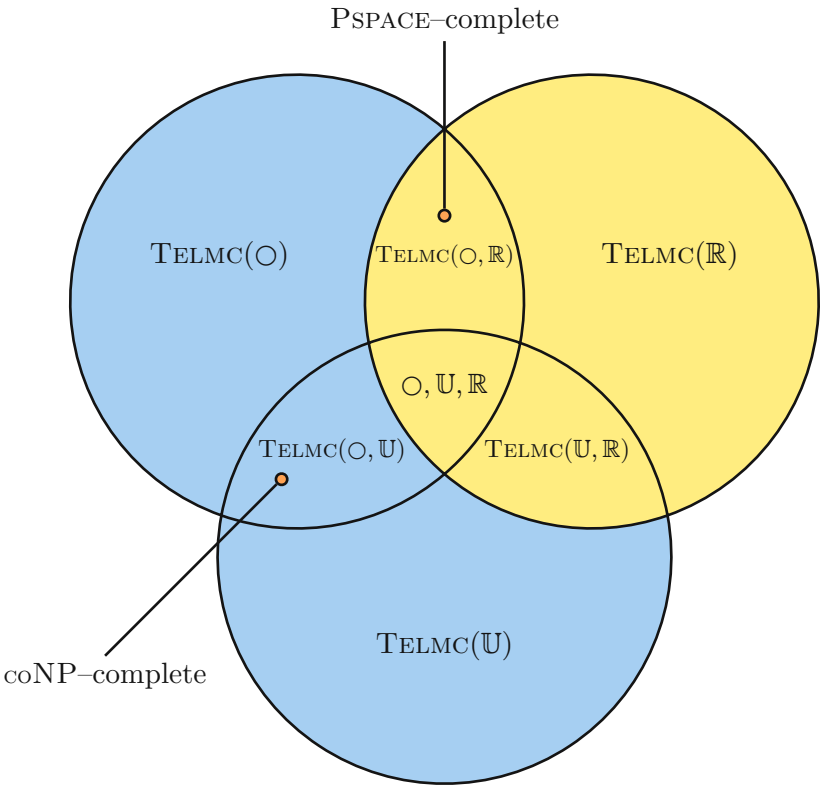


Figure 15: Visualisation of the complexity of the $\textsc{Telmc}(S)$ problem, $S \subseteq \{\bigcirc, \mathbb{U}, \mathbb{R}\}$, $S \neq \emptyset$, if deciding $\textsc{Telmc}(\mathbb{R})$ was proven Pspace–hard.

Another interesting problem to study is the $\textsc{Telmc}^0$ one, the Telmc fragment where no use of implication, and as a consequence of negation, is allowed. A peculiarity of this fragment is that, among the already studied ones, it could be the only one solvable in deterministic polynomial time. For $\textsc{Telmc}^0$, in fact, the following is supposed.

Conjecture 7.95 $\textsc{Telmc}^0 \in \mathrm{P}$. □

Conjectured proof By means of a dynamic programming-like notion of witness patterns. ■

Observation 7.70 Deciding the $\textsc{Telmc}^0$ problem cannot be shown coNP–hard by the standard reduction argument from the Elmc problem. In Elmc, in fact, the presence of the $\rightarrow$ operator is assumed. □

A few more fragments which could be made the subject of future study are the ones rising from the Telts problem. In particular, Lemma 4.43 of L. Bozzelli and D. Pearce [53] could be used to show that deciding $\textsc{Telts}_\forall(\bigcirc, \mathbb{U})$ and $\textsc{Telts}_\exists(\bigcirc, \mathbb{U})$ has a lower complexity than Pspace.

Conjecture 7.96 Deciding $\text{TELTS}_{\forall}(\bigcirc, \mathbb{U})$ is coNP–complete. □

Proof Consequence of Corollary 7.97 and Conjecture 7.98. ■

Corollary 7.97 Deciding $\text{TELTS}_{\forall}(\bigcirc, \mathbb{U})$ is coNP–hard. □

Proof By the standard reduction argument from ELMC. ■

Conjecture 7.98 $\text{TELTS}_{\forall}(\bigcirc, \mathbb{U}) \in \text{coNP}$. □

Conjectured proof Deciding $\text{TELTS}_{\forall}(\bigcirc, \mathbb{U})$ requires determining whether, for a given transition system τ and $\text{THT}(\bigcirc, \mathbb{U})$ formula φ, all traces T within τ are a TEM of φ. If an infinite trace T within τ exists, then by Lemma 4.43 T cannot be a TEM of φ, and a negative answer for the (τ, φ) instance is returned. If, instead, all traces T in τ are almost-empty ones, then a model checking algorithm for TEL over finite traces can be invoked. ■

Conjecture 7.99 Deciding $\text{TELTS}_{\exists}(\bigcirc, \mathbb{U})$ is coNP–complete. □

Proof Consequence of Corollary 7.100 and Conjecture 7.101. ■

Corollary 7.100 Deciding $\text{TELTS}_{\exists}(\bigcirc, \mathbb{U})$ is coNP–hard. □

Proof By the standard reduction argument from ELMC. ■

Conjecture 7.101 $\text{TELTS}_{\exists}(\bigcirc, \mathbb{U}) \in \text{coNP}$. □

Conjectured proof Similarly as in the proof of Conjecture 7.98, if an instance (τ, φ) is given, then thanks to Lemma 4.43 a finite trace T within τ needs to be found which is a TEM of φ. ■

The complexity of the $\text{TELTS}_{\forall,1}$ and $\text{TELTS}_{\exists,1}$ problems, the $\text{TELTS}_{\forall}$ and $\text{TELTS}_{\exists}$ fragments where the temporal height is limited to a value of 1, has not been considered yet. One more way of considering fragments of the TELTS problem would be to restrict the structure of the transition system τ that is given as input.

Table 11 displays a summary of the list of open problems.

Problem	Conjectured complexity
$\text{TELMC}(\mathbb{R})$	PSPACE–complete (Conjecture 7.93)
TELMC^0	$\in$ P (Conjecture 7.95)
$\text{TELTS}_{\forall}(\bigcirc, \mathbb{U})$	coNP–complete (Conjecture 7.96)
$\text{TELTS}_{\exists}(\bigcirc, \mathbb{U})$	coNP–complete (Conjecture 7.99)

Table 11: List of open problems.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] D. Pearce, "Equilibrium Logic," *Annals of Mathematics and Artificial Intelligence*, vol. 47, no. 1–2, pp. 3–41, Nov. 2006, doi: 10.1007/s10472-006-9028-z.

[2] V. Lifschitz, "What Is Answer Set Programming?," in *Proc. of AAAI*, MIT Press, 2008, pp. 1594–1597.

[3] T. Eiter, G. Ianni, and T. Krennwallner, "Answer Set Programming: A Primer," *Reasoning Web. Semantic Technologies for Information Systems*, vol. 5689. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 40–110, 2009. doi: 10.1007/978-3-642-03754-2_2.

[4] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge: Cambridge University Press, 2003. doi: 10.1017/CBO9780511543357.

[5] E. Erdem, M. Gelfond, and N. Leone, "Some Applications of ASP," 2015.

[6] V. Lifschitz, "Answer Set Programming and Plan Generation," *Artificial Intelligence*, vol. 138, pp. 39–54, 2002.

[7] J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang, "Monitoring Agents Using Declarative Planning," *Fundamenta Informaticae*, vol. 57, no. 2–4, pp. 345–370, 2003.

[8] T. C. Son, E. Pontelli, and C. Sakama, "Logic Programming for Multiagent Planning with Negotiation," in *Proc. of ICLP*, 2009, pp. 99–114.

[9] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*. in Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[10] M. D. Vos, T. Crick, J. Padget, M. Brain, and O. C. a. J. Needham, "A Multi-Agent Platform Using Ordered Choice Logic Programming," in *In Declarative Agent Languages and Technologies (DALT'05)*, 2005, pp. 72–88.

[11] E. Erdem, O. Erdem, and F. Ture, "HAPLO-ASP: Haplotype Inference Using Answer Set Programming," in *Proc. of LPNMR*, 2009, pp. 573–578.

[12] R. Schwitter, "The Jobs Puzzle: Taking on the Challenge via Controlled Natural Language Processing," *Theory and Practice of Logic Programming*, vol. 13, no. 4–5, pp. 487–501, 2013.

[13] R. Schwitter, "Controlled Natural Language Processing as Answer Set Programming: An Experiment," in *Proc. of CNL*, 2014, pp. 8–19.

[14] C. Sakama, "Learning by Answer Sets.," in *Proc. of AAAI Spring Symposium:Answer Set Programming*, 2001.

[15] K. Inoue and C. Sakama, "Abductive Framework for Nonmonotonic Theory Change.," in *Proc. of IJCAI*, 1995, pp. 204–210.

[16] T. Schaub and K. Wang, "A Comparative Study of Logic Programs with Preference.," in *Proc. of IJCAI*, 2001, pp. 597–602.

[17] G. Brewka, "Preferences, Contexts and Answer Sets," in *Proc. of ICLP*, 2007, p. 22.

[18] T. Eiter, W. Faber, N. Leone, and G. Pfeifer, "The Diagnosis Frontend of the Dlv System.," *AI Communications*, vol. 12, no. 1–2, pp. 99–111, 1999.

[19] M. Balduccini and M. Gelfond, "Diagnostic Reasoning with A-Prolog," *Theory and Practice of Logic Programming*, vol. 3, no. 4–5, pp. 425–461, 2003.

[20] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception Handling for Repair in Service-Based Processes," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 198–215, 2010.

[21] D. Calvanese, T. Eiter, and M. Ortiz, "Regular Path Queries in Expressive Description Logics with Nominals," in *Proc. of IJCAI*, 2009, pp. 714–720.

[22] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits, "Well-Founded Semantics for Description Logic Programs in the Semantic Web," in *Proc. of RuleML*, 2004, pp. 81–97.

[23] C. Baral, M. Gelfond, and J. N. Rushton, "Probabilistic Reasoning with Answer Sets," *Theory and Practice of Logic Programming*, vol. 9, no. 1, pp. 57–144, 2009.

[24] N. Leone *et al.*, "The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data," in *Proc. of SIGMOD*, 2005, pp. 915–917.

[25] M. Alviano, W. Faber, and N. Leone, "Disjunctive ASP with Functions: Decidable Queries and Effective Computation," *Theory and Practice of Logic Programming*, vol. 10, no. 4–6, pp. 497–512, 2010.

[26] G. Brewka, T. Eiter, and M. Truszczynski, "Answer Set Programming at a Glance," *Communications of The Acm*, vol. 54, no. 12, pp. 92–103, 2011.

[27] T. C. Son, E. Pontelli, N.-H. Nguyen, and C. Sakama, "Formalizing Negotiations Using Logic Programming," *ACM Trans. Comput. Logic*, vol. 15, no. 2, pp. 1–30, 2014.

[28] F. Toni and M. Sergot, "Argumentation and Answer Set Programming," in *Proc. of LPNMR*, 2011, pp. 164–180.

[29] M. Balduccini, "Industrial-Size Scheduling with ASP + CP," in *Proc. of LPNMR*, 2011, pp. 284–296.

[30] E. Erdem, V. Lifschitz, and M. F. Wong, "Wire Routing and Satisfiability Planning," in *Proc. of CL*, 2000, pp. 822–836.

[31] D. East and M. Truszczynski, "More on Wire Routing with ASP," in *Proc. of ASP*, 2001.

[32] E. Erdem and M. D. F. Wong, "Rectilinear Steiner Tree Construction Using Answer Set Programming," in *Proc. of ICLP*, 2004, pp. 386–399.

[33] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, "An A-Prolog Decision Support System for the Space Shuttle," *Practical Aspects of Declarative Languages*, vol. 1990. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 169–183, 2001. doi: 10.1007/3-540-45241-9_12.

[34] K. Heljanko and I. Niemelä, "Bounded LTL Model Checking with Stable Models," in *Logic Programming and Nonmotonic Reasoning*, T. Eiter, W. Faber, and M. law Truszczyński, Eds., Berlin, Heidelberg: Springer, 2001, pp. 200–212. doi: 10.1007/3-540-45402-0_15.

[35] D. V. Marina and D. Vermeir, "Logic Programming Agents and Game Theory." 2001.

[36] M. D. Vos and D. Vermeir, "Extending Answer Sets for Logic Programming Agents," *Annals of Mathematics and Artif. Intell.*, vol. 42, no. 1–3, pp. 103–139, 2004.

[37] R. Finkel, V. W. Marek, and M. Truszczynski, "Constraint Lingo: A Program for Solving Logic Puzzles and Other Tabular Constraint Problems." pp. 513–516, 2002.

[38] M. Celik, H. Erdogan, F. Tahaoglu, T. Uras, and E. Erdem, "Comparing ASP and CP on Four Grid Puzzles," in *Proc. of RCRA*, 2009.

[39] E. Erdem, V. Lifschitz, and D. Ringe, "Temporal Phylogenetic Networks and Logic Programming," *Theory and Practice of Logic Programming*, vol. 6, no. 5, pp. 539–558, 2006.

[40] D. R. Brooks, E. Erdem, S. T. Erdogan, and J. W. M. and D. Ringe, "Inferring Phylogenetic Trees Using Answer Set Programming," *Journal of Automated Reasoning*, vol. 39, no. 4, pp. 471–511, 2007.

[41] E. Erdem, "PHYLO-ASP: Phylogenetic Systematics with Answer Set Programming," in *Proc. of LPNMR*, 2009, pp. 567–572.

[42] E. Erdem, "Applications of Answer Set Programming in Phylogenetic Systematics," in *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning* -

Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, 2011, pp. 415–431.

[43] C. Baral and C. Uyan, "Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming," in *Proc. of LPNMR*, 2001, pp. 186–199.

[44] N. Tran and C. Baral, "Reasoning about Triggered Actions in AnsProlog and Its Application to Molecular Interactions in Cells," in *Proc. of KR*, 2004, pp. 554–564.

[45] T. Schaub and S. Thiele, "Metabolic Network Expansion with Answer Set Programming," in *Proc. of ICLP*, 2009, pp. 312–326.

[46] M. Gebser *et al.*, "Repair and Prediction (under Inconsistency) in Large Biological Networks with Answer Set Programming," in *Proc. of KR*, 2010.

[47] M. Gebser, T. Schaub, S. Thiele, and P. Veber, "Detecting Inconsistencies in Large Biological Networks with Answer Set Programming," *Theory and Practice of Logic Programming*, vol. 11, no. 2, pp. 1–38, 2011.

[48] G. Boenn, M. Brain, M. D. Vos, and J. Fitch, "ANTON: Composing Logic and Logic Composing," in *Proc. of LPNMR*, 2009, pp. 542–547.

[49] D. Pearce, "A New Logical Characterisation of Stable Models and Answer Sets," *Non-Monotonic Extensions of Logic Programming*, vol. 1216. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 57–70, 1997. doi: 10.1007/BFb0023801.

[50] M. Gelfond and V. Lifschitz, "The Stable Model Semantics for Logic Programming," 1988, Accessed: Aug. 19, 2025. [Online]. Available: https://www.cs.utexas.edu/~ai-lab/?gel88

[51] A. Heyting, "Die Formalen Regeln Der Intuitionistischen Logik," 1930. Accessed: Sep. 11, 2024. [Online]. Available: https://www.semanticscholar.org/paper/Die-formalen-Regeln-der-intuitionistischen-Logik-Heyting/8c20747578e4dde70731cc5faf582f8fdeced244

[52] P. Cabalar and S. Demri, "Automata-Based Computation of Temporal Equilibrium Models," *Logic-Based Program Synthesis and Transformation*, vol. 7225. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 57–72, 2012. doi: 10.1007/978-3-642-32211-2_5.

[53] L. Bozzelli and D. Pearce, "On the Complexity of Temporal Equilibrium Logic," in *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, Kyoto, Japan: IEEE, Jul. 2015, pp. 645–656. doi: 10.1109/LICS.2015.65.

[54] M. Cadoli, "The Complexity of Model Checking for Circumscriptive Formulae," *Information Processing Letters*, vol. 44, no. 3, pp. 113–118, Nov. 1992, doi: 10.1016/0020-0190(92)90049-2.

[55] A. P. Sistla and E. M. Clarke, "The Complexity of Propositional Linear Temporal Logics," *J. ACM*, vol. 32, no. 3, pp. 733–749, Jul. 1985, doi: 10.1145/3828.3837.

[56] S. Demri and P. Schnoebelen, "The Complexity of Propositional Linear Temporal Logics in Simple Cases," *Information and Computation*, vol. 174, no. 1, pp. 84–103, Apr. 2002, doi: 10.1006/inco.2001.3094.

[57] N. Leone *et al.*, "The DLV System for Knowledge Representation and Reasoning," *ACM Trans. Comput. Logic*, vol. 7, no. 3, pp. 499–562, Jul. 2006, doi: 10.1145/1149114.1149117.

[58] T. Eiter *et al.*, "Complexity of Model Checking and Bounded Predicate Arities for Non-Ground Answer Set Programming," *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning*. in Guide Proceedings. pp. 377–387, Jun. 02, 2004. doi: 10.5555/3029848.3029898.

[59] P. v. E. Boas, "The Convenience of Tilings," *Complexity, Logic, and Recursion Theory*. CRC Press, 1997.

[60] F. Schwarzentruber, "The Complexity of Tiling Problems." Accessed: Mar. 05, 2025. [Online]. Available: http://arxiv.org/abs/1907.00102

[61] J. van Leeuwen, Ed., *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. Cambridge, MA, USA: MIT Press, 1991.

[62] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[63] T. Soininen and I. Niemelä, "Developing a Declarative Rule Language for Applications in Product Configuration," in *Practical Aspects of Declarative Languages*, G. Gupta, Ed., Berlin, Heidelberg: Springer, 1998, pp. 305–319. doi: 10.1007/3-540-49201-1_21.

[64] S. A. Kripke, "Semantical Analysis of Intuitionistic Logic I," *Studies in Logic and the Foundations of Mathematics*, vol. 40. Elsevier, pp. 92–130, 1965. doi: 10.1016/S0049-237X(08)71685-9.

[65] D. Van Dalen, *Logic and Structure*. in Universitext. London: Springer, 2013. doi: 10.1007/978-1-4471-4558-5.

[66] P. Cabalar and G. Pérez Vega, "Temporal Equilibrium Logic: A First Approach," *Computer Aided Systems Theory – EUROCAST 2007*, vol. 4739. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 241–248, 2007. doi: 10.1007/978-3-540-75867-9_31.

[67] G. De Giacomo and M. Y. Vardi, "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, in IJCAI '13. Beijing, China: AAAI Press, Aug. 2013, pp. 854–860.

[68] T. J. Schaefer, "The Complexity of Satisfiability Problems," in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, in STOC '78. New York, NY, USA: Association for Computing Machinery, May 1978, pp. 216–226. doi: 10.1145/800133.804350.

[69] F. Aguado *et al.*, "Linear-Time Temporal Answer Set Programming," *Theory and Practice of Logic Programming*, vol. 23, no. 1, pp. 2–56, Jan. 2023, doi: 10.1017/S1471068421000557.

[70] N. Markey and P. Schnoebelen, "Model Checking a Path," *CONCUR 2003 - Concurrency Theory*, vol. 2761. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 251–265, 2003. doi: 10.1007/978-3-540-45187-7_17.

Generative AI tools

The German translation of the dedication and abstract sections of this work have been translated from English into German by use of the DeepL Translator[27], and then manually adapted by the author. All the other sections of this work are a result of the author's own effort.

[27] https://www.linguee.de/.