

# **Model Checking for Flat Systems and Temporal Logic with Counting**

Normann Decker



UNIVERSITÄT ZU LÜBECK  
INSTITUTE FOR SOFTWARE ENGINEERING  
AND PROGRAMMING LANGUAGES

From the  
Institute for Software Engineering and Programming Languages  
of the University of Lübeck

Director: Prof. Dr. Martin Leucker

## **Model Checking for Flat Systems and Temporal Logic with Counting**

Dissertation  
for Fulfillment of  
Requirements  
for the Doctoral Degree  
of the University of Lübeck

from the Department of Computer Sciences

Submitted by  
Normann Decker  
from Erfurt

Lübeck, 2019

First referee: Prof. Dr. Martin Leucker  
Second referee: Univ.-Prof. Dr. Dr. h. c. Javier Esparza

Date of oral examination: 26 July 2019

Approved for printing: 6 February 2020



**Model Checking for Flat Systems and Temporal Logic with Counting**

© Normann Decker. *This work is licensed under a [CC BY 4.0 license](https://creativecommons.org/licenses/by/4.0/).*

# Acknowledgement

Many people contributed directly or indirectly to this dissertation.

First of all, I am grateful to my supervisor Martin Leucker for his support and guidance in all the facets of academic work and the variety of research topics and collaborations that I was offered. The present report is only the most manifest outcome of a path that was guided by curiosity, choice, and advice rather than necessity or constraint. I also want to thank Javier Esparza for reviewing this dissertation as well as Till Tantau for chairing the examination board.

I had the great pleasure to collaborate with many wonderful and brilliant people. I'm very grateful for the opportunity to work on the present and a variety of other topics with Daniel Thoma, my fellow student, collaborator, office neighbour, and friend—an academic brother in the best sense. In our countless discussions, his sharp analyses and ideas gave me invaluable insights and motivation. I'm also indebted to Anton Pirogov, Arnaud Sangnier, Benedikt Bollig, and Peter Habermehl with whom I had the privilege and pleasure of closely working together on the series of publications this dissertation is based upon.

I'm thankful to Grigory Markin and all other colleagues and friends I met at the University of Lübeck for their inspiring company over the years. My colleagues at ISP created the supportive, competent, and casual environment that made our office more than a workplace. Thank you all for taking part, sharing thoughts and always holding together. Beyond ISP and the work presented here, I feel very lucky for having had the opportunity to participate in exciting projects and interesting discussions. Representative of all the people and research groups I met and who enriched these past years in numerous ways, I thank the CONIRAS team, the LEMON/FREQS team, César Rodríguez, Hernán Ponce de León, and Wanwei Liu.

Finally, I want to express my deepest love and gratitude to my parents and to Sophie for their endless love and unconditional support.

# Abstract

Temporal logic is a well-established specification framework for correctness properties of computational systems. While the traditional variants LTL, CTL, and CTL\* entail important property types such as safety and liveness, they lack the ability to *count*. Since counting is a fundamental and ubiquitous concept in computing, this dissertation studies counting extensions of temporal logic and their verification problems on counter systems. To this end, it joins different lines of research on counting temporal logics and model-checking flat counter systems. Counting temporal logics allow for the specification of complex, quantitative patterns by counting specific positions on an execution and formulating arithmetic constraints over the corresponding quantities. Further, the variants considered here are interpreted over counter systems and can impose arithmetic constraints on the valuation of their counters.

Previous work has shown that even limited means of counting in temporal logic lead to undecidability of essential problems such as satisfiability and model checking. This applies also to most of the logics considered here but two exceptions are identified where decidability is recovered by distinct kinds of restrictions. For the remaining family of logics, it is shown that decidability of the model-checking problem can be recovered by a sensible restriction on the model side, called *flatness*. Flatness demands, essentially, that each control state of the system belongs to at most one simple loop. While even basic reachability properties are undecidable for systems with just two counters, it is shown that flatness makes verifying powerful classes of counting properties decidable. To this end, the problem is shown to be interreducible with the satisfiability problem of a decidable extension of Presburger arithmetic.

The best-known decision procedure for the latter problem, however, is of non-elementary computational complexity. Therefore, an alternative method is developed for a temporal logic that features a restricted variant of the counting mechanism. The notion of *augmented path schemas* is introduced and serves as symbolic representation for a set of system executions exhibiting a similar shape. Path schemas are employed as witness structures for the satisfaction of a given formula and by proving a small-model property, an exponential bound on the complexity of the model-checking problem is established.

Further, augmented path schemas serve as underlying structures of an SMT-based approximation approach to the (undecidable) model-checking problem over arbitrary counter systems. Similar to bounded model checking, the presented method allows for verifying temporal and counting properties by finding satisfying witnesses while a depth parameter flexibly determines the trade-off between exhaustiveness and computational effort. Essential advantages of augmented path schemas are that they represent potentially infinite subsets of runs and can be more concise than a representation in terms of finite prefixes.

# Zusammenfassung

Temporallogik als formale Grundlage für Spezifikationsprachen ist ein etabliertes Konzept, um Korrektheitseigenschaften von operativen Systemen, insbesondere Computerprogrammen, zu formulieren. Die traditionellen Varianten LTL, CTL und CTL\* sind geeignet, wichtige Klassen wie z. B. Sicherheits- und Lebendigkeitseigenschaften auszudrücken. Jedoch fehlt ihnen im Allgemeinen die Möglichkeit des Zählens.

Da Zählen an sich ein fundamentaler und allgegenwärtiger Aspekt des Berechnens und der Datenverarbeitung ist, werden in der vorliegenden Dissertation Zähl-Erweiterungen von Temporallogiken und ihre Verifikationsprobleme untersucht. Dazu werden verschiedene existierende Forschungsergebnisse kombiniert und weiterentwickelt, die zum einen um Zählmechanismen erweiterte Temporallogiken betreffen und zum anderen Model-Checking-Verfahren für sogenannte flache Zählerysteme beschreiben.

Zählende Temporallogiken erlauben es, komplexe, quantitative Muster zu beschreiben, indem bestimmte Positionen entlang des Laufes eines Systems gezählt werden. Über die Anzahl solcher Positionen können dann Nebenbedingungen in Form arithmetischer Gleichungen formuliert werden. Darüber hinaus werden die hier betrachteten Logikvarianten über Zählerystemen interpretiert und erlauben es, arithmetische Bedingungen auch über die Belegung der Zähler auszudrücken.

Vorausgegangene Arbeiten zeigen, dass selbst eingeschränkte Zählmechanismen in Temporallogik die Unentscheidbarkeit wesentlicher Fragestellungen wie Erfüllbarkeit und Model-Checking zur Folge haben. Dies trifft ebenso auf die meisten der hier betrachteten Formalismen zu. Jedoch werden zwei Ausnahmen identifiziert, für welche Entscheidbarkeit durch verschiedenartige syntaktische Einschränkungen wiederhergestellt werden kann.

Für die verbleibende Familie von Logiken wird gezeigt, dass das Model-Checking-Problem unter der Einschränkung auf sogenannte *flache* Modelle entscheidbar wird. Flache Systeme zeichnen sich im Wesentlichen dadurch aus, dass jeder Kontrollzustand zu höchstens einem einfachen Kreis im Kontrollgraphen gehört. Während grundlegende Probleme, wie z. B. Erreichbarkeit, im Allgemeinen schon für Systeme mit nur zwei Zählern unentscheidbar ist, ermöglicht Flachheit das Verifizieren selbst mächtiger Klassen von durch Zählmechanismen ausdrückbarer Eigenschaften. Es wird gezeigt, dass die

entsprechenden Probleme auf das entscheidbare Erfüllbarkeitsproblem einer Erweiterung der Presburger'schen Arithmetik reduziert werden kann.

Die Komplexität der besten bekannten Entscheidungsprozedur für dieses Erfüllbarkeitsproblem ist jedoch nicht elementar. Daher wird eine alternative Methode entwickelt, die auf eine Temporallogik mit eingeschränktem Zählmechanismus anwendbar ist. Das Konzept der *ergänzten Pfadschemata* wird eingeführt und dient als symbolische Repräsentation für eine Menge von Systemläufen ähnlicher Form. Pfadschemata werden als Beweisobjekte verwendet, welche die Erfüllbarkeit einer gegebenen Formel bezeugen. Es wird nachgewiesen, dass, wenn überhaupt, immer auch kleine Modelle existieren. Daraus lässt sich folgern, dass die Komplexität des Model-Checking-Problems exponentiell beschränkt ist.

Darüber hinaus liefern ergänzte Pfadschemata die Grundlage für ein SMT-basiertes Approximationsverfahren für das (unentscheidbare) Model-Checking-Problem über beliebigen Zählersystemen. Ähnlich zu Bounded-Model-Checking erlaubt es die vorgestellte Methode, Läufe zu identifizieren, die eine gegebene temporale Eigenschaft erfüllen. Dabei bestimmt ein Parameter die Größe der in Betracht zu ziehenden Schemata und damit flexibel die Balance zwischen Umfänglichkeit der Approximation und Berechnungsaufwand. Im Gegensatz zur Verwendung von endlichen Präfixen beschränkt eine festgelegte Maximalgröße für Pfadschemata die Analyse nicht grundsätzlich auf eine endliche Menge von Läufen. Zudem können Pfadschemata Läufe deutlich kompakter repräsentieren.

# Contents

<b>Acknowledgement</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>Zusammenfassung</b>	<b>viii</b>
<b>List of Figures</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Contributions . . . . .	9
1.3 Overview . . . . .	13
<b>2 Counting in Temporal Logic</b>	<b>16</b>
2.1 Preliminaries . . . . .	16
2.2 Temporal Logic with Counting . . . . .	21
2.3 Fragments . . . . .	26
<b>3 Satisfiability of LTL with Counting</b>	<b>30</b>
3.1 Satisfiability of LTL over Counter Systems . . . . .	30
3.2 Undecidability Results . . . . .	32
3.3 A Decidable Fragment of LTL <sub>#</sub> . . . . .	34
<b>4 Model-checking cLTL over Flat Counter Systems</b>	<b>42</b>
4.1 Towards A Guess-and-Check Procedure . . . . .	43
4.2 Augmented Path Schemas . . . . .	44
4.3 Consistent APS are Correct . . . . .	52
4.4 Consistency-preserving Operations . . . . .	61
4.5 Existence of Consistent APS . . . . .	76
4.6 Discussion . . . . .	97

<b>5</b>	<b>SMT-based Flat Model Checking</b>	<b>99</b>
5.1	From Flat Model Checking to Presburger Arithmetic . . . . .	103
5.2	Properties of the Encoding . . . . .	119
5.3	Evaluation . . . . .	121
5.4	Conclusion . . . . .	122
<b>6</b>	<b>Model-checking cCTL over Kripke Structures</b>	<b>124</b>
6.1	Reachability and Emptiness in a Subclass of One-counter Systems . . . . .	125
6.2	Deciding $MC(cCTL,KS)$ in Polynomial Time . . . . .	129
<b>7</b>	<b><math>CTL_{\#}^*</math> and Presburger Arithmetic with Counting</b>	<b>134</b>
7.1	Presburger Arithmetic with Counting Quantifier . . . . .	134
7.2	A Reduction from $SAT(PH)$ to $MC(LTL_{\#}, FKS)$ . . . . .	136
7.3	A Reduction from $MC(CTL_{\#}^*, FCS)$ to $SAT(PH)$ . . . . .	139
<b>8</b>	<b>Conclusion</b>	<b>146</b>
8.1	Summary . . . . .	146
8.2	Outlook . . . . .	147
	<b>Bibliography</b>	<b>149</b>

# List of Figures

1.1	Examples of control graphs illustrating flatness. . . . .	8
1.2	Example of explicit position counting in $LTL_{\#}$ . . . . .	10
1.3	Overview of the counting temporal logics studied in this dissertation. . . . .	10
2.1	Flat Kripke structure (Example 2.4). . . . .	25
3.1	The counter system $\mathcal{S}_{\hat{w}}$ built from a symbolic word model $\hat{w} = uv^{\omega}$ over the extended set of propositions $AP \cup \Gamma_{\varphi}$ . . . . .	32
4.1	A Kripke structure $\mathcal{K}$ labelled by $\Sigma = \{a, b\}$ with $\llbracket \mathcal{K} \rrbracket^{\Sigma} = \Sigma^{\omega}$ . . . . .	42
4.2	A flat counter system $\mathcal{S}$ and (a sketch of) an augmented path schema $\mathcal{P}$ in $\mathcal{S}$ . . . . .	45
4.3	Sketch of the APS $\mathcal{P}$ from Figure 4.2 showing updates and guards of a balance counter (cf. Example 4.10). . . . .	51
4.4	Sketch of the run $\sigma$ and the location of the relevant parts and indices. . . . .	59
4.5	Consistency-preserving operations on augmented path schemas. . . . .	62
4.6	Sketch of an augmented path schema with witness state situated on a loop and an alternative witness. . . . .	71
4.7	Sketch of an augmented path schema with a defect state on a loop and a copy of the state occurring on the preceding row. . . . .	71
4.8	An augmented path schema constructed from the run of a counter system. . . . .	77
4.9	Example for stabilising a loop in $\mathcal{P}$ (from Figure 4.2) by duplicating and unfolding it. . . . .	82
4.10	Sketch of a loop that can result in a large number of unstable iterations. . . . .	98
5.1	A diagram showing a counter system $\mathcal{S}$ and a sketch of an augmented path schema that alternates two loops of $\mathcal{S}$ . . . . .	101
5.2	Example of encoding the run and path schema shown in Figure 5.1 in terms of PA variables. . . . .	105

- 7.1 Sketch of the bookmark placement on a run as expressed by the encoding of the PH formula  $\exists_{x_1}.\exists_{x_2}.x_1 = 2x_2 \wedge \exists_{x_3}^{x_1}.x_3 \leq 3$  discussed in Example 7.2. 138

# Introduction

## 1.1 Background

### 1.1.1 Program Verification and Temporal Logic

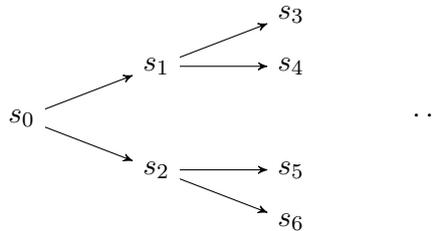
In 1977, Amir Pnueli published an article [Pnu77] on the use of the rather philosophical concept of *tense logic*, later called *temporal logic*, in the engineering task of *program verification*. He had observed that many approaches to the latter aspect of computer programming shared conceptual similarities and suggested to use a unified framework to formulate and prove the various correctness properties that developers were interested in to verify. At the time, reoccurring patterns were discovered in specifications that could be solved by specific proof techniques. An important and widely accepted classification is due to Lamport [Lam77] who distinguishes *safety* and *liveness* properties (stating that *nothing bad will ever happen* and *something good will happen continuously*, respectively) and devises appropriate proof techniques. Drawing from the developments on formalised temporal logic initiated by Prior [Pri57], Pnueli was able to capture this intuition by a simple extension of classical propositional logic that provides a mechanism to formulate *temporal succession*. Choosing the temporal operators **F** (eventually in the future) and **G** (globally, always in the future), he provided a simple, yet sufficiently expressive system to reason conveniently about many of the properties that one may wish to verify, especially for concurrent and interactive programs. For example, a simple formula **G**  $p$  would express the safety guarantee that an execution will never violate the proposition  $p$ , identifying, for example, all but the failure states of a program. In this *linear-time temporal logic (LTL)*, the behaviour of a program, or more generally a system model, is represented as a sequence

$$s_0 s_1 s_2 \dots$$

of computation steps or system states. In terms of the structural operational semantics of a program, as defined later by Plotkin [Plo04], such a sequence is a single path through its configuration graph, starting at the initial configuration. This view matches well

the semantics of temporal logic given in terms of a *Kripke structure*: such a structure was proposed by Kripke [Kri59] and represents a graph of which each node assigns a truth value to a number of atomic propositions. In this formal model, the “future” is determined by what is reachable from a given node.<sup>1</sup> Gabbay et al. [Gab+80] rounded up LTL to its form used today including the *next-time* operator **X** for referring to a successive computation step and the binary temporal *until* operator **U**. They showed LTL to be equally expressive as first-order logic over linear structures, sharpening the result by Kamp [Kam68] who came to that conclusion in the presence of additional *past-time* operators.

Soon after the introduction of LTL to the field of program verification, *computation-tree logic (CTL)* was considered as an alternative by Clarke and Emerson [CE81]. In contrast to the individual, linear execution traces described by LTL, the *branching-time* logic CTL reasons over the structure of the whole configuration graph, i.e., all potential executions, in terms of the unfolded computation tree



of a program. The logic features, in addition to the (linear) temporal operators, the *path quantifiers* **A** (universal) and **E** (existential) expressing, intuitively, the *possibilities* a program has to proceed at some specific state. For example, the formula **EF(A G p)** states that there is some execution (**E**) on which eventually (**F**) all possible continuations (**A**) satisfy  $p$  globally at every position (**G**). Clearly, the use of quantifiers ranging over possible continuations makes only a difference in the presence of non-determinism as exhibited, e.g., by concurrent programs, because otherwise the computation tree degenerates to a single path. Finally, Emerson and Halpern [EH83] suggested the temporal logic CTL\*. Entailing both LTL and CTL by unifying the branching- and linear-time view the authors aimed to overcome the “controversy” that “has arisen in the computer science community regarding the differences between and the appropriateness of branching versus linear time temporal logic” [EH83]. Thereby, they improved upon an earlier suggestion by Lamport [Lam80].

---

<sup>1</sup>Kripke [Kri59] provided structural semantics for modal logic (cf. [BBW06]) in terms of a countable set of interpretations of propositions and predicates (“worlds”). He assumed in [Kri63] a binary relation over this set to determine what is *possible* and hence to interpret modal operators.

Since then, the trio of CTL\*, LTL, and CTL has enjoyed ongoing active research as basis for specification languages. The formalisms have been extended in numerous ways in order to address specific aspects of computing systems. Two dimensions concerned by such extensions are of particular interest:

- **Extended model features** may be exposed to the logic in terms of specific access and constraint mechanisms. In the propositional setting, every state of the system is characterised by finitely many fixed propositions that may hold or not. However, program models may provide further information such as physical execution time or program data stored in variables that is to be taken into account in a specification. For example, timed temporal logics (cf. [Bou+17]) deal with additional (real-valued) time stamps, allowing a formula to state, e.g., that some observation is made within arbitrarily many sequential steps but within a given time frame. Other variants recognise the recursive structure of a program flow (cf. [Alu+08]) or a spatial dimension within the system model (cf. [Kon+07]).
- **Temporal patterns** may be specified by additional constructs that increase the expressiveness beyond that of the standard temporal operators. For example, Wolper [Wol81] suggested an extension of LTL by regular grammars to increase expressiveness to the full class of regular languages. The same was achieved by integrating regular expressions [HT99; DKL10; LS10; DV13]. Explicit fixed point operators may be integrated to generalise the temporal modalities [BKP86; Var88]. Another option are mechanisms to reason on the number of occurrences of specific events (or patterns), globally or within a given scope [BER94; BEH95; LMP12; LMP10; BDL12].

The latter aspect of *counting* as extension to temporal logics in both of these dimensions is the central subject of study in this dissertation.

### 1.1.2 Counting in Temporal Logic

Counting is a fundamental principle in the theory of computation and well-established in the study and verification of finite and infinite-state systems. The concept is ubiquitous in programming and counting mechanisms are a natural notion of quantitative measurement in specification formalisms. For example, they are useful for expressing properties such as “the number of acknowledgements never exceeds the number of requests” or “the relative error frequency stays below some threshold”.

It is well-known that LTL, CTL, and also CTL\* provide only very limited means for counting. Even specific, finite distances can be characterised only in terms of nesting

the corresponding number of next-time (**X**) operators, effectively encoding them in a unary fashion. Beyond that, properties expressed in the temporal logics are invariant to what is commonly called *stuttering*, i.e., they cannot discriminate different numbers of repetitions. For example, the words  $a^n b^\omega$  and  $a^{n+1} b^\omega$  cannot be distinguished by any LTL formula with less than  $n \geq 1$  next-time operators. Further, quantitative aspects cannot be described, for example that some property holds twice as often during a computation as another one.

**Specification of counting patterns.** An approach to solve this issue is to extend temporal logic with some ability to count positions of an execution that satisfy some property and to impose constraints over these numbers. Bouajjani, Echahed and Robbana [BER94] and Bouajjani, Echahed and Habermehl [BEH95] investigated extensions of CTL and LTL, respectively, that allow for binding the number of positions where some particular event occurs to a variable and impose (Presburger) arithmetic constraints on them. For example, with variables  $x, y$  and propositions  $p, q, r$  a formula  $\mathbf{F}[x : p \wedge q][y : r] \neg \mathbf{F}(x \geq 2y)$  reads: from some point in the future on, the number of positions satisfying both propositions  $p$  and  $q$  must never be twice as large as the number of positions satisfying the proposition  $r$ . Both branching- and linear-time variants are shown to be undecidable, but the authors identify decidable fragments (with restricted negation) and verification problems for different system models.

Similar extensions were considered by Laroussinie, Meyer and Petonnet [LMP12; LMP10] where decidability is obtained by restricting the type of arithmetic constraints (e.g., with only positive coefficients) and a notation tying the *scope* of counting constraints to that of an individual temporal operator. In their notation, the property above can be formulated as  $\mathbf{F} \neg \mathbf{F}_{[1 \cdot \#(p \wedge q) + (-2) \cdot \#(r) \geq 0]} \text{true}$ . Thus, the property gives an example relying on negative coefficients in constraints if formulae (or corresponding variables) occur only on one side of equations. Introducing counting mechanisms leads quickly to undecidable satisfiability and verification problems and significant restrictions are necessary to recover decidability.

**Reasoning on counter systems.** Although counting logics were also considered for the verification of non-regular processes, they pertain clearly to the second dimension of extensions identified above as they only provide access to the underlying model in terms of a finite set of atomic propositions. However, as fundamental computing concept, counting is also an important aspect in system models. Therefore, temporal logics were also studied in combination with various notions of *counter systems*. They extend finite

state systems by so called counters, being integer variables that can be updated and constraint by transitions. The configuration sequence of a counter system provides a state and a counter valuation for every execution step. A natural way of expressing properties of such valuations of integer variables is to use *arithmetic constraints*, i.e., (in)equations over variables. To this end, temporal logic can be combined with (some fragment of) *Presburger arithmetic*—first-order logic over integers in which addition appears as only operation, named after Presburger [Pre29] who showed decidability of the corresponding theory. The constraints over counter values can be used instead of (or in addition to) atomic propositions in temporal formulae. For example,  $\mathbf{G}(c_1 \leq c_2)$  would then specify that the value of a counter  $c_1$  never exceeds the value of a counter  $c_2$  during an execution of some counter system.

Such extensions were used by Čerāns [Čer94] and Comon and Cortier [CC00]. A general framework of *constraint temporal logics* was studied by Demri and D’Souza [DD07] where the type of constraints and the domain of variables is a parameter of the logic and not restricted to, e.g., integers. The survey by Demri [Dem06] provides a comprehensive overview of the results and developments for temporal logic over arithmetic constraints.

### 1.1.3 Model Checking

Besides the better understanding of the proof techniques, the construction of a unified, powerful and convenient specification and reasoning framework brought a crucial milestone into reach: proof automation. In their seminal contributions Clarke and Emerson [CE81], Queille and Sifakis [QS82] and Lichtenstein and Pnueli [LP85] proposed algorithmic approaches to efficiently verify programs (modelled as Kripke structures) with respect to properties expressed in CTL and LTL, respectively.

Manually repeating similar proof arguments on individual programs is not only inconvenient and inefficient but also prone to reasoning mistakes, incomplete case analysis and, as programs and system descriptions grow larger, becomes infeasible with reasonable effort. With the number of programmers increasing, it becomes worthwhile to put substantial effort into sophisticated tool support, in terms of compilers, editors, management tools, and, not to the least, error detection.

Clarke and Emerson [CE81] used the term *model checking* for their algorithmic method. It refers, technically, to the decision problem of a formal logic answering the question whether a formula is valid in (i.e., satisfied by or part of the theory of) a given formal structure. For application of this concept in program verification, the implementation or a sufficiently simplified version thereof is represented as logical structure and, therefore, the verification task is phrased as model-checking problem in that framework. The intention

to generalise and unify the steps required to prove properties and the mechanical nature of formal proof systems (where truth arises from syntactic form rather than semantic interpretation) matched perfectly the concept of automation in terms of an algorithm executed by a computing machine. Algorithms solving the model-checking problem of a logical system (in particular temporal logic), applied to solve verification tasks represented such an advancement over the individual and manual approach that the term model checking is now effectively used synonymously to automatic system verification.

Nevertheless, it should be noted that automation of other techniques related to logic made also large advances: automated proof assistants, SAT-, constraint-, and SMT solvers proliferated in the past decades and in fact heavily influence also the techniques used to solve model-checking problems today.

For an in-depth treatment of the topic, the reader is referred to the very influential textbooks by Clarke, Grumberg and Peled [CGP01] and Baier and Katoen [BK08]. Further, the recently published handbook edited by Clarke et al. [Cla+18] provides an impressive compilation of the most important aspects of verification techniques based on model checking.

### **Approximation**

An inherent boundary and the reason for the plethora of different model-checking techniques targeting specific types of specifications and system models is that there cannot be a single method to analyse all relevant aspects of computation. Such a method would itself have to exceed what is considered computable, in the sense of Church and Turing, witnessed by undecidable problems, foremost the halting problem of Turing machines. Even for decidable combinations of a computation model and a specification formalism, the complexity of the model-checking problem can be quite high. It is therefore necessary to find a balance between leaving out complicating details of the program and the requirements on one side and faithfulness of the result on the other. An approximation should be chosen that appropriately reflects the relevant facets of a system and the correctness properties to be verified. Approximations can thus be made with respect to three aspects: the system model, the property specification, and the verification result.

**Reduced system model.** In many cases the concrete implementation is given in terms of a too powerful or ambiguous representation, for example, as source code in some general-purpose programming language. The challenge is hence to faithfully model the relevant aspect of a system using the capabilities of the chosen formalism while leaving

out irrelevant details. For example, using Kripke structures to model concurrent or interactive programs can capture their non-deterministic nature, avoiding any assumption on synchronising effects or input, respectively. Within the finite state space of a Kripke structure, on the other hand, essentially the program flow can be modelled while the state space of actual data structures must usually be ignored for scalability reasons.

**Simplified specification.** The correctness properties must be formulated using the means provided by the specification languages. As for the system model, it requires a balance between precisely representing the (possibly informally given) system requirements and the computational effort to treat more general specification mechanisms algorithmically. Formulating a correctness property, e.g., in terms of the simple (non-)reachability of a specific program state often allows for a very efficient analysis. This may, however, require ignoring some rare but possible cases such as program interrupts with uncontrollable handlers that would, technically, make any program state reachable from everywhere. Including this possibility in the formal specification by stating that a state is not reachable via only non-interrupt states, is not a plain reachability question any more but requires more powerful specification constructs (such as in LTL) and thus a possibly more expensive verification technique.

**Approximated verification result.** If the combination of specification language and system model appears infeasible to be checked accurately in general, the results of the formal problem itself can be approximated while still being useful. Accepting one-sided errors can significantly reduce the needed computational effort and thereby provide pragmatic solutions for verification tasks. Under-approximations of the actual model may reveal witnesses for satisfaction or violation of a property and their absence may be proven in an over-approximation. Such techniques have proven to be extremely efficient and hence useful, even for problems where computing an always accurate and conclusive result is theoretically and practically no option.

A popular over-approximating method is *counter example guided abstraction refinement* [Cla+03] where the program is first radically simplified, e.g., to the program-flow graph without any conditions. If the given property can be proven to hold in this over-approximation of the behaviour, the verification process can be terminated immediately and conclusively. Otherwise, a potentially spurious counter example is obtained that either witnesses the actual violation of the property or can be used to refine the abstraction and start the verification on a slightly more complex but more accurate abstraction.

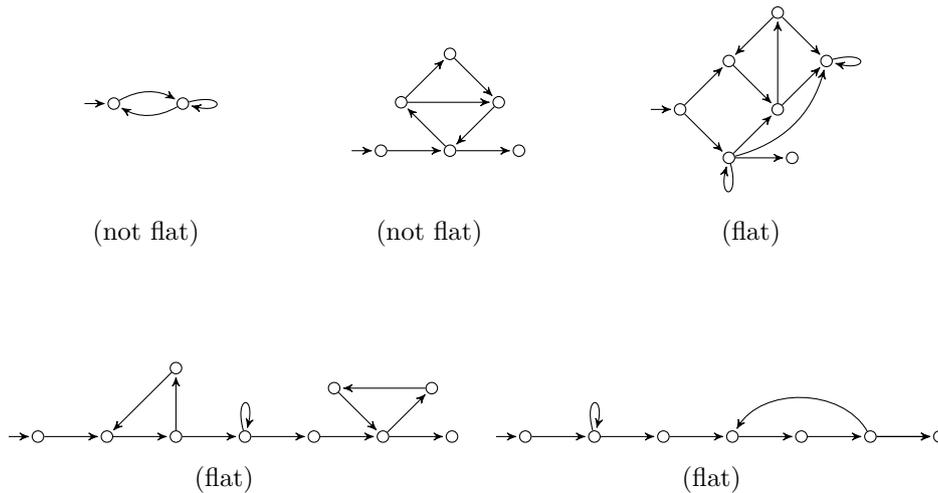


Figure 1.1: Examples of control graphs illustrating flatness.

A very effective method based on under-approximations is *bounded model checking* [Bie09] where the considered computations of a program are represented by a truncated prefix of specific length. This provides a bounded, finite view even into possibly infinite state spaces and the method can find witnesses (for errors or property fulfilment) as early as they actually occur, without having to analyse the entire state space.

#### 1.1.4 Flatness

The notion of *flat systems* (sometimes also called *weak*) was investigated as a sensible restriction to reduce the computational complexity of various verification problems. It is a structural property of transition systems such those underlying Kripke structures or counter systems and demands that no control state is part of more than one simple cycle. This means, essentially, that cycles in the control graph of the system cannot be alternated during any execution. Figure 1.1 shows some examples.

While model-checking LTL properties on arbitrary Kripke structures is well-known to be PSPACE-complete [SC85], it was shown by Kutz and Finkbeiner [KF11] to become NP-complete<sup>2</sup> under the flatness condition. A similar impact is observable for (infinite state) counter systems. While reachability is already undecidable for two-counter systems [Min67], results by Comon and Jurski [CJ98] provide that flatness recovers

<sup>2</sup>The authors consider the universal formulation of the model-checking problem for LTL and show completeness for CONP, implying NP-completeness for the existential variant.

decidability with an arbitrary number of counters (see also the follow-up work by Comon and Cortier [CC00]). Later, it was shown by Demri, Dhar and Sangnier [DDS12; DDS15] that LTL properties (including past-time operators) can generally be evaluated in NP. A comprehensive exposition on these results can be found in the dissertation by Dhar [Dha14]. Bardin et al. [Bar+05] argue that flatness is also a desired property in symbolic verification that allows for effectively computing a symbolic representation of the effect of program cycles.

Even though flatness seems to impose a quite strong restriction, Leroux and Sutre [LS05; LS06] have shown that many classes of counter systems are at least *virtually*<sup>3</sup> flat, i.e., their actual behaviour can be represented using a flat system even if the given representation is not flat syntactically. This particularly supports using flat systems as approximations since in these cases the exact behaviour can actually be represented. We will come back to this in Chapter 5.

## 1.2 Contributions

The primary subject of this dissertation are powerful counting extensions of the temporal logics LTL, CTL, and CTL\* and their model-checking problems. The counting temporal logic CTL\*<sub>#</sub> is introduced along with a number of natural syntactical fragments that feature both aspects of counting discussed above: On one hand, they allow for the specification of complex, quantitative patterns by counting specific positions of a run and imposing arithmetic constraints on the corresponding quantities. On the other hand, the logics are interpreted over counter systems and arithmetic constraints can also be imposed on the valuation of their counters.

The logic CTL\*<sub>#</sub> features a bookmarking mechanism to define the scope of counting: as shown in Figure 1.2, positional variables called *bookmarks* can be introduced at any time to store the current position. Within their scope, the number of positions satisfying some property can be used in linear arithmetic constraints. The logics LTL<sub>#</sub> and CTL<sub>#</sub> are defined by restricting the use of path quantifiers analogous to LTL and CTL. While bookmarks define counting scopes that can potentially span arbitrarily many nested temporal operators, the logics cLTL, cCTL, and cCTL\* are defined to restrict the mechanism in a natural way. They tie counting constraints to the scope of a single temporal operator.

Figure 1.3 shows an overview of the studied fragments. This family is based on the logics defined by Bouajjani, Echahed and Robbana [BER94], Bouajjani, Echahed and

---

<sup>3</sup>Bardin et al. [Bar+05] use the term *flatable* for systems for which an equivalent flat system exists while Leroux and Sutre [LS05; LS06] simply use the term *flat* in this more general sense.

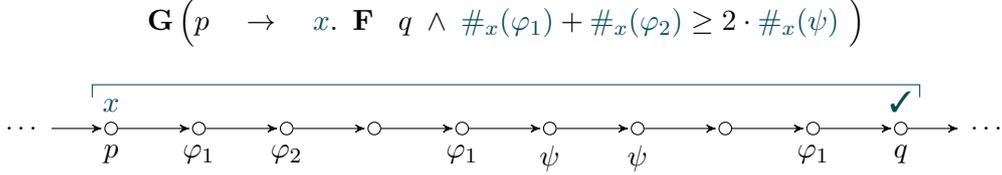


Figure 1.2: Example of explicit position counting within a scope defined by a bookmark  $x$ . Within the scope of  $x$  the term  $\#_x(\varphi_1)$  denotes the number of positions that satisfy  $\varphi_1$  since the placement of the bookmark. Hence, the formula states that whenever proposition  $p$  is satisfied, the position is to be bookmarked and eventually at some position satisfying  $q$ , the number of positions passed that satisfy  $\varphi_1$  or  $\varphi_2$  must be twice as large as the number of positions satisfying the formula  $\psi$ .

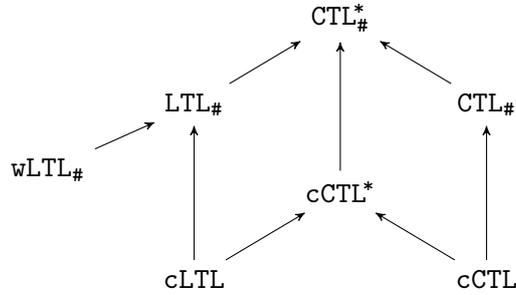


Figure 1.3: Overview of the counting temporal logics studied in this dissertation. Arrows point towards entailing fragments.

Habermehl [BEH95] and Laroussinie, Meyer and Pettonnet [LMP10; LMP12]. Their investigations have shown that even limited means of counting in temporal logic lead to undecidability of essential problems such as satisfiability and (Kripke-structure) model checking. Only essential restrictions to the counting mechanisms achieved to recover decidability. On the other hand, Kuhtz and Finkbeiner [KF11] and Demri et al. [DDS15; Dem+10; DDS18] investigated the effect of flatness on model-checking LTL, CTL, and CTL\* over Kripke structures and counter systems. They observed a significant impact on the computational complexity of the respective problems.

The present work joins these lines of research on counting temporal logics and model-checking flat systems. It extends the techniques and results in order to better understand counting mechanisms in temporal specification and provide solutions to the corresponding verification problems. The major technical contributions of this dissertation are the following.

1. The powerful counting temporal logic CTL\*<sub>#</sub> is defined to subsume many similar logics proposed earlier. Its model-checking problem, and hence that of its fragments, is shown to be decidable over flat counter systems. This is achieved by a reduction to the satisfiability problem of a decidable extension of Presburger arithmetic. In addition, also the reverse direction is provided by an encoding of the satisfiability into the model-checking problem. This correspondence is of conceptual interest: On one hand this establishes an 2NEXP lower bound for the model-checking problem. On the other hand, only non-elementary decision procedures are known. The result therefore relates a not yet entirely understood arithmetic theory to a concrete temporal-logic verification problem.
2. A weak variant of LTL<sub>#</sub> called wLTL<sub>#</sub> is introduced. It is shown that both the satisfiability problem over counter systems and the model-checking problem over Kripke structures is decidable and in NEXP. Although wLTL<sub>#</sub> is not closed under negation, the logic provides a type of restriction that recovers decidability without limiting the powerful bookmark-based counting mechanism. The presented construction reduces satisfiability to reachability in integer vector addition systems. It revises and extends a previous result [Dec11; BDL12] on LTL with only frequency counting constraints where no complexity upper bound was given.
3. The technically most involved result concerns the fragment cLTL of LTL<sub>#</sub>. The restriction to operator-bound counting scopes does not recover decidability of the model-checking problem in general but a procedure is developed to show that flat counter systems can be verified against a cLTL property non-deterministically in

exponential time. To this end, the notion of *augmented path schemas* is introduced along with the syntactic property of *consistency*. It is shown that consistent augmented path schemas are suitable symbolic witnesses for satisfaction of a formula and exhibit a small-model property.

4. Based on the theoretical developments for model-checking  $\text{cLTL}$  on flat counter systems, an approximation technique is presented that is applicable also to non-flat systems. The approach verifies *flat under-approximations* of a specific depth given as parameter. Similarly to bounded model checking, the parameter allows the user to flexibly adjust the trade-off between exhaustiveness and computational effort. However, using a symbolic representation based on augmented path schemas instead of bounded prefixes allows for representing infinite subsets of runs instead of only a finitely many. The technique is exhaustive for flat systems but (necessarily) incomplete in the general case. Nevertheless, it is directly applicable and may well provide conclusive results within the given depth. In any way, the incremental procedure identifies “simple” witnesses early whenever they exist. For the (approximated) model-checking problem, an explicit formulation in the quantifier-free fragment of Presburger arithmetic is developed. This allows the approach to be implemented using competitive SMT solvers since this theory is well supported. The construction is parametrised by the depth of the flat approximation that is to be verified, and the resulting arithmetic formula is linear in the problem size and the chosen depth. Using a prototype implementation by Pirogov [Pir17], the effectiveness of the approach was demonstrated on verification tasks of the RERS Challenge [How+14] and quantified variations.
5. The model-checking problem of the branching-time fragment  $\text{cCTL}$  is proven to be decidable in polynomial time for arbitrary Kripke structures. To this end, specific variants of one-counter systems are employed for which (repeated) control-state reachability is shown to be decidable in polynomial time. In combination with the results on  $\text{cLTL}$ , this provides an exponential-space decision procedure for model-checking  $\text{cCTL}^*$  over flat Kripke structures.

Table 1.1 provides an overview of the results on the model-checking problem for the fragments of  $\text{CTL}_\#^*$  that are considered. As mentioned above, the undecidability results follow mostly from previous work. Multiple references in the table indicate independent reasons for undecidability. For example, model-checking  $\text{LTL}_\#$  over counter systems ( $\text{CS}$ ) is found to be undecidable because the problem entails the undecidable reachability problem in counter systems [Min67] and, independently, because the undecidable satisfiability

	FKS	KS	FCS	CS
cCTL	P (Thm. 6.5)	P (Thm. 6.5)	dec. (Thm. 7.1)	undec. [Min67]
cLTL	NP-h. [KF11] NEXP (Thm. 4.2)	undec. (Thm. 4.1)	NP-h. [KF11] NEXP (Thm. 4.2)	undec. [Min67],(Thm. 4.1)
cCTL*	EXPSPACE (Thm. 6.7)	undec. (Thm. 4.1)	dec. (Thm. 7.1)	undec. [Min67],(Thm. 4.1)
wLTL#	NEXP (Cor. 3.7)	NEXP (Cor. 3.7)	dec. (Thm. 7.1)	undec. [Min67]
LTL#	2NEXP-h. (Thm. 7.5) dec. (Thm. 7.1)	undec. (Thm. 4.1)	2NEXP-h. (Thm. 7.5) dec. (Thm. 7.1)	undec. [Min67],(Thm. 4.1)
CTL#	2NEXP-h. (Thm. 7.5) dec. (Thm. 7.1)	undec. [LMP12]	2NEXP-h. (Thm. 7.5) dec. (Thm. 7.1)	undec. [Min67],[LMP12]
CTL*#	2NEXP-h. (Thm. 7.5) dec. (Thm. 7.1)	undec. [LMP12],(Thm. 4.1)	2NEXP-h. (Thm. 7.5) dec. (Thm. 7.1)	undec. [LMP12],(Thm. 4.1)

Table 1.1: Overview of the results on the model-checking problems—(F)KS and (F)CS denote (flat) Kripke structures and (flat) counter systems, respectively.

problem of cLTL reduces to it (Theorem 4.1).

## 1.3 Overview

### Chapter 2

We start the technical developments by providing the basic definitions in Chapter 2. Presburger arithmetic and counter systems are introduced formally. Syntax and semantics of  $\text{CTL}_\#^*$  are defined and the various restrictions are formalised in terms of syntactical fragments. The central decision problems of satisfiability and model checking are stated formally.

### Chapter 3

Chapter 3 is concerned with the satisfiability problem that is found to be undecidable for most of the considered fragments. To understand and characterise the influence of the features of the logics on this (negative) result,  $\text{wLTL}_\#$  is defined as a weak fragment

of  $LTL_{\#}$ . To show that the satisfiability problem of  $wLTL_{\#}$  is decidable in exponential time, a reduction to the reachability problem in a subclass of counter systems ( $\mathbb{Z}$ -VASS) is presented.

#### Chapter 4

Chapter 4 is dedicated to one of the main results: it is shown that verifying  $cLTL$  formulae on flat counter systems is decidable. The chapter presents the technically most involved part of this dissertation. The developed decision procedure follows a *guess-and-check* scheme based on a specific variant of flat counter systems, called *augmented path schemas* that can certify the satisfaction of a property. Following a general overview of the chapter, path schemas and the important syntactic property of *consistency* are defined in Section 4.2. Subsequently, the connection of syntax and semantics is established, proving soundness of the procedure: Section 4.3 shows that a syntactically consistent schema reliably certifies that a property holds. Crucially for the decision procedure to be complete, Section 4.5 establishes that if a given property is satisfied by some flat counter system, then a witnessing consistent schema always exists of (exponentially) bounded size. The proof relies on technical constructions on augmented path schemas that are discussed in Section 4.4.

#### Chapter 5

Chapter 5 presents an approximation approach to verifying arbitrary counter systems with respect to properties formulated in  $cLTL$ . Since the model-checking problem is undecidable in general, flat counter systems are used as under-approximations. This provides a flexibly adjustable trade-off between exhaustiveness and computational effort, similar to bounded model checking. The techniques and results, especially augmented path schemas as symbolic witnesses are employed to construct a parametrised encoding of the (approximated) problem in quantifier-free Presburger arithmetic.

#### Chapter 6

Chapter 6 is concerned with the branching-time fragment  $cCTL$ . In contrast to the linear-time fragments for which the model-checking problem is of high complexity or even undecidable in many cases, it is shown that  $cCTL$  properties can be verified over Kripke structures in polynomial time. The problem is reduced to reachability questions in specific subclasses of counter systems with only one active counter. They are shown to be decidable in polynomial time.

## **Chapter 7**

Finally, the model-checking problem of  $\text{CTL}_\#^*$  is shown to be decidable over flat counter systems. An extension of Presburger arithmetic is presented that features a quantifier for counting the solutions of a formula, called Härtig quantifier, and is known to be decidable. We show that its satisfiability problem can be reduced to the model-checking problems of  $\text{LTL}_\#$  and  $\text{CTL}_\#$  over flat Kripke structures in linear time. For the reverse direction, the model-checking problem of  $\text{CTL}_\#^*$  is encoded into the satisfiability problem, producing an arithmetic formula of exponential size.

## **Publications**

Parts of this dissertation have been published in the proceedings of conferences. The content of Chapter 3 revises and extends part of [BDL12]. The content of Chapters 4, 6 and 7 is based on and extends the results of [Dec+17]. Further, Chapter 5 develops the results presented in [DP19].

# Counting in Temporal Logic

## 2.1 Preliminaries

### 2.1.1 Basic Notation

The set of all integers is denoted by  $\mathbb{Z}$  and  $\mathbb{N} := \{0, 1, \dots\} \subset \mathbb{Z}$  denotes the set of all non-negative integers, that is, all natural numbers and zero. Further, let  $\mathbb{Z}_\infty := \mathbb{Z} \cup \{\infty, -\infty\}$  and  $\mathbb{N}_\infty := \mathbb{N} \cup \{\infty\}$  be their respective extensions by the symbols  $\infty$  and  $-\infty$ . Arithmetically, the latter are treated as absorbing, extremal elements, i.e.,  $-\infty = -\infty + x < x < x + \infty = \infty$  is assumed to hold for every  $x \in \mathbb{Z}$  while  $-\infty + \infty$  remains undefined. Intervals of integers are denoted by  $[n, m] := \{n, n + 1, \dots, m\} \subset \mathbb{Z}$  for  $n, m \in \mathbb{Z}$ . Notice that  $[n, m] = \emptyset$  if  $n > m$ . We consider numbers  $n \in \mathbb{Z}$  to be represented binary and define their representation size to be  $size(n) := \lceil \log_2(|n| + 1) \rceil$ .

The set of all functions  $f : A \rightarrow B$  from a domain  $A$  to a co-domain  $B$  is denoted  $B^A$  and  $2^A$  is the powerset of  $A$  being comprised of all subsets of  $A$ . The domain of a function  $f : A \rightarrow B$  is denoted by  $dom(f) = A$ . For elements  $a \in A$  and  $b \in B$ , let  $f[a \mapsto b]$  denote the function that maps  $a$  to  $f[a \mapsto b](a) = b$  and every other other element  $a' \in A \setminus \{a\}$  to the original value  $f[a \mapsto b](a') = f(a')$ . For functions  $f, g$ , we write  $f \sqsubseteq g$  if  $dom(f) \subseteq dom(g)$  and  $f(x) = g(x)$  for all  $x \in dom(f)$ . Arithmetic operations are lifted point-wise to integer-valued functions  $f, g : A \rightarrow \mathbb{Z}$ , i.e.,  $(f + g)(a) := f(a) + g(a)$  for all  $a \in A$ . The symbols  $\mathbf{0}$  and  $\mathbf{1}$  may be used to refer to constant functions assigning 0 and 1, respectively, to every element, if the domain is understood. The entirely undefined function is called  $\varepsilon : \emptyset \rightarrow \emptyset$ .

The sets of finite and infinite sequences of elements from a set  $A$  are denoted by  $A^*$  and  $A^\omega$ , respectively, and we may write  $A^\infty := A^* \cup A^\omega$  for their union. The length  $|w|$  of a sequence  $w$  is the cardinality of its position set. That is, if  $w = \varepsilon$  is the empty sequence,  $|w| := 0$ ; if  $w = a_0 \dots a_n$  is finite,  $|w| := |[0, n]| = n + 1$ ; and if  $w$  is infinite, then  $|w| := |\mathbb{N}| = \omega$ . Whenever convenient, a sequence  $w = a_0 \dots a_n \in A^*$  may be considered as function  $w : [0, n] \rightarrow A$  where  $w(i) := a_i$  is the  $(i + 1)$ th element if

$i \in [0, n]$  is a position (index) on the sequence. This applies equally to infinite sequences  $w \in A^\omega$  interpreted as function  $w : \mathbb{N} \rightarrow A$ . Further, let the inverse-like function  $pos_w : A^+ \rightarrow 2^{\mathbb{N}}$  provide for an element or a sequence of elements  $a_1, \dots, a_n \in A$  the set  $pos_w(a_1 \dots a_n) := \{i \in \mathbb{N} \mid i < |w|, w(i) \in \{a_1, \dots, a_n\}\}$  of positions carrying any of them. We extend the element-of relation ( $\in$ ) from sets to sequences and write  $a \in w$  if  $pos_w(a) \neq \emptyset$ . A sequence  $w$  is called *simple* if no element occurs at more than one position. Finite powers  $w^n$  of finite sequences  $w$  denote their ( $n$  times) iterated concatenation and  $w^\omega$  denotes the sequence of  $w$  repeated infinitely.

Landau notation is used to denote for a function  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  the set  $\mathcal{O}(f) := \{g : \mathbb{N}^n \rightarrow \mathbb{N} \mid \exists a, b \in \mathbb{N} \forall x_1, \dots, x_n \in \mathbb{N} : x_1, \dots, x_n \geq a \rightarrow b \cdot g(x_1, \dots, x_n) \geq f(x_1, \dots, x_n)\}$  of dominating functions. It is extended to application of functions  $h : \mathbb{N} \rightarrow \mathbb{N}$  such that  $h(\mathcal{O}(f)) := \{h(g) \mid g \in \mathcal{O}(f)\}$  denotes the class of functions dominating  $h(g)$  for some function  $g$  dominating  $f$ ; specifically, we use  $2^{\mathcal{O}(f)} = \{2^g \mid g \in \mathcal{O}(f)\}$ .

The standard computational complexity classes (see, e.g., the handbook article by Johnson [Joh90]) are denoted by P and NP (deterministic and non-deterministic polynomial time), PSPACE and EXPSPACE (polynomial and exponential space), EXP and NEXP (deterministic and non-deterministic exponential time), and 2NEXP (non-deterministic doubly-exponential time). A problem is considered of *elementary* complexity if it can be solved (by a Turing machine) in time bounded by a  $k$ -exponential function  $f_1 \circ \dots \circ f_k$ , for some  $k \in \mathbb{N}$  and  $f_1, \dots, f_k \in 2^{\mathcal{O}(n)}$ .

### 2.1.2 Presburger Arithmetic

A *linear arithmetic term*  $\tau$  over a set  $X$  of indeterminate *placeholders* (typically variable symbols) is a finite expression produced by the grammar

$$\tau ::= a \cdot x \mid \tau + \tau$$

for arbitrary integer coefficients  $a \in \mathbb{Z}$  and placeholders  $x \in X$ . Where convenient, the multiplication symbol ( $\cdot$ ) and factors  $a = 1$  may be omitted, as well as the addition symbol ( $+$ ) before a negative number sign ( $-$ ). For example,  $1 \cdot x + (-2) \cdot y$  may be written as  $x - 2y$  for  $x, y \in X$ .

The set  $\text{PA}(X)$  of *Presburger arithmetic* formulae  $\varphi$  over  $X$  is defined by the grammar

$$\varphi ::= \tau \geq b \mid \neg \varphi \mid \varphi \wedge \varphi \mid \exists x. \varphi$$

for linear arithmetic terms  $\tau$  over  $X$  and  $x \in X$ . The *quantifier-free* fragment  $\text{qfPA}(X) \subseteq \text{PA}(X)$  consists of those formulae not using the quantifier  $\exists$ . Further, a simple inequation

of the form  $\tau \geq b$  is called *guard* and we denote the set of all guards over  $X$  by  $\mathbf{Grd}(X) \subseteq \mathbf{qfPA}(X)$ . The parameter  $X$  may be omitted for convenience if determined by the context.

The semantics of terms and PA formulae is defined in terms of *valuations* assigning integer values to the respective placeholders or variables. The arithmetic evaluation  $\llbracket \tau \rrbracket(\theta) \in \mathbb{Z}$  of a constraint term  $\tau$  over  $X$  with respect to a valuation  $\theta : X \rightarrow \mathbb{Z}$  is defined inductively by

$$\begin{aligned} \llbracket a \cdot x \rrbracket(\theta) &:= a \cdot \theta(x) \\ \llbracket \tau_1 + \tau_2 \rrbracket(\theta) &:= \llbracket \tau_1 \rrbracket(\theta) + \llbracket \tau_2 \rrbracket(\theta) \end{aligned}$$

for  $a \in \mathbb{Z}$ ,  $x \in X$ , and terms  $\tau_1, \tau_2$ . The satisfaction relation  $\models_{\mathbf{PA}}$  is defined for PA( $X$ ) formulae  $\varphi, \psi, \tau \geq b$ , placeholders  $x \in X$ , and valuations  $\theta : X \rightarrow \mathbb{Z}$  by

$$\begin{aligned} \theta \models_{\mathbf{PA}} \tau \geq b & \quad :\Leftrightarrow \quad \llbracket \tau \rrbracket(\theta) \geq b \\ \theta \models_{\mathbf{PA}} \neg \varphi & \quad :\Leftrightarrow \quad \theta \not\models \varphi \\ \theta \models_{\mathbf{PA}} \varphi \wedge \psi & \quad :\Leftrightarrow \quad \theta \models_{\mathbf{PA}} \varphi \text{ and } \theta \models_{\mathbf{PA}} \psi \\ \theta \models_{\mathbf{PA}} \exists x. \varphi & \quad :\Leftrightarrow \quad \exists z \in \mathbb{Z} : \theta[x \mapsto z] \models_{\mathbf{PA}} \varphi. \end{aligned}$$

The relation be extended to finite sets  $M \subseteq \mathbf{PA}(X)$  of formulae by interpreting  $M$  as conjunctive clause, that is,  $\theta \models_{\mathbf{PA}} M$  if and only if  $\theta \models_{\mathbf{PA}} \bigwedge_{\varphi \in M} \varphi$ . For easier reading, we may omit the subscript and simply write  $\models$  instead of  $\models_{\mathbf{PA}}$  if no ambiguity arises. A formula  $\varphi \in \mathbf{PA}(X)$  is *satisfiable* if there is some valuation  $\theta : X \rightarrow \mathbb{Z}$  such that  $\theta \models_{\mathbf{PA}} \varphi$  and the set of all satisfiable PA( $X$ ) formulae, for arbitrary  $X$ , is denoted by  $\mathbf{SAT}(\mathbf{PA})$ . The *satisfiability problem* of PA is the question whether a given formula is satisfiable or, equivalently, a member of the set  $\mathbf{SAT}(\mathbf{PA})$  and therefore identified with the latter. Notice that  $\mathbf{SAT}(\mathbf{PA})$  is well defined because satisfiability of any formula  $\varphi \in \mathbf{PA}(X)$  is independent of the choice of  $X$  (being necessarily a superset of the placeholders used in  $\varphi$ ).

For convenience, the relation symbols  $\leq$ ,  $<$ , and  $>$  may be used to denote equivalent constraints by their standard arithmetic interpretation. For example,  $2x_1 + x_2 < 3$  denotes  $-2x_1 - x_2 \geq -2$ . The *dual* of a constraint  $\tau \geq b$  is denoted by  $\overline{\tau \geq b}$  and defined as the equivalent of  $\tau < b$ . Notice that neither conjunction nor negation is necessary to express these relations since  $\mathbb{Z}$  contains inverses with respect to addition. However, in the actual presence of conjunction, also equality ( $=$ ) may be used to abbreviate formulae of the form  $\tau \geq b \wedge \tau \leq b$  by  $\tau = b$ .

The representation size of placeholders  $x \in X$  is assumed to be  $\mathit{size}(x) = 1$ , unless noted otherwise. The size of arithmetic terms and PA( $X$ ) formulae is defined inductively

by

$$\begin{aligned}
 |a \cdot x| &:= \text{size}(a) + \text{size}(x) \\
 |\tau_1 + \tau_2| &:= |\tau_1| + |\tau_2| \\
 |\tau \geq b| &:= |\tau| + \text{size}(b) + 1 \\
 |\neg\varphi| &:= |\varphi| + 1 \\
 |\varphi \wedge \psi| &:= |\varphi| + |\psi| + 1 \\
 |\exists x.\varphi| &:= \text{size}(x) + |\varphi| + 1.
 \end{aligned}$$

### 2.1.3 Counter Systems

Counter systems are comprised of a finite control that manipulates a fixed set of integer variables, called counters. Every transition updates each counter by adding a fixed integer value that may be zero. In addition, each transition carries a set of linear arithmetic constraints (guards) over these variables that needs to be satisfied after the updates are applied. Various notions of counter systems exist with the most important distinction being whether they subsume *Minsky machines* [Min67] or not because this determines decidability of essential problems such as reachability.

Let  $\Lambda$  be a set of *labels* and  $C_S$  a finite set of *system counters*. A *counter system (CS)* over  $\Lambda$  and  $C_S$  is a tuple  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  where

- $S$  is a finite set of *control states*,  $s_I \in S$  is the *initial state*,
- $\lambda : S \rightarrow \Lambda$  is a *labelling function* and
- $\Delta \subseteq S \times \mathbb{Z}^{C_S} \times 2^{\text{Grd}(C_S)} \times S$  is a finite set of *transitions* carrying an *update*  $\mu : C_S \rightarrow \mathbb{Z}$  to the system counters and a finite set of *guards*  $\Gamma \subseteq \text{Grd}(C_S)$  over them.

A *configuration* of  $\mathcal{S}$  is a pair  $(s, \theta)$  comprised of a state  $s \in S$  and a *valuation*  $\theta : C_S \rightarrow \mathbb{Z}$ . A *run* of  $\mathcal{S}$  is an infinite sequence  $\rho = (s_0, \theta_0)(s_1, \theta_1) \dots \in (S \times \mathbb{Z}^{C_S})^\omega$  such that  $(s_0, \theta_0) = (s_I, \mathbf{0})$  and for all positions  $i \in \mathbb{N}$  there is a transition  $(s_i, \mu_i, \Gamma_i, s_{i+1}) \in \Delta$  such that  $\theta_{i+1} = \theta_i + \mu_i$  and  $\theta_{i+1} \models \Gamma_i$ . The set of all runs of  $\mathcal{S}$  is denoted  $\text{runs}(\mathcal{S})$ .

Let  $\text{counters}(\mathcal{S}) = C_S$  denote the set of counters used by  $\mathcal{S}$ . For a configuration  $(s, \theta)$ , let  $\text{st}((s, \theta)) := s$  and  $\text{val}((s, \theta)) := \theta$  denote the projections to its state- and valuation component, respectively. This notation is extended point-wise to sequences of configurations, especially runs. Correspondingly, let us denote by  $\text{pos}_\rho(s) := \{i \in \mathbb{N} \mid \text{st}(\rho(i)) = s\}$  the set of positions on  $\rho$  carrying a state  $s \in S$ . By means of the labelling function, a run  $\rho$  of  $\mathcal{S}$  defines an abstraction subject to a set of relevant labels  $\Sigma$  in terms

of a word  $\llbracket \rho \rrbracket_{\mathcal{S}}^{\Sigma} = (a_0, \theta_0)(a_1, \theta_1) \dots \in (\Sigma \times \mathbb{Z}^{C_S})^{\omega}$  over the alphabet  $\Sigma \times \mathbb{Z}^{C_S}$  defined by  $\llbracket \rho \rrbracket_{\mathcal{S}}^{\Sigma}(i) := (\lambda(st(\rho(i)) \cap \Sigma), val(\rho(i)))$ . That is, states are substituted point-wise by their labelling restricted to  $\Sigma$ . The sub- and superscripts may be omitted if no ambiguity arises. Then, a counter system  $\mathcal{S}$  defines a language  $\llbracket \mathcal{S} \rrbracket^{\Sigma} := \{\llbracket \rho \rrbracket_{\mathcal{S}}^{\Sigma} \mid \rho \in \mathcal{S}\}$  over the potentially infinite alphabet  $\Sigma \times \mathbb{Z}^{C_S}$ . In case of Kripke structures, no counters are being used and, for convenience, in this case we identify the corresponding alphabet  $\Sigma \times \mathbb{Z}^{\emptyset}$  with  $\Sigma$  and consider languages over  $\Sigma^{\omega}$ .

Let  $\text{succ}_{\mathcal{S}}(s) := \{s' \in S \mid \exists_{\mu, \Gamma} : (s, \mu, \Gamma, s') \in \Delta\}$  denote the set of direct successors of some state  $s \in S$ . Its transitive closure is denoted  $\text{succ}_{\mathcal{S}}^+(s)$  while  $\text{succ}_{\mathcal{S}}^*(s)$  refers to the transitive and reflexive closure. A finite or infinite *path* in  $\mathcal{S}$  is a finite or infinite state sequence  $w \in S^*$  or  $w \in S^{\omega}$ , respectively, with  $w(i+1) \in \text{succ}_{\mathcal{S}}(w(i))$  for all positions  $0 \leq i < |w|$  if  $w$  is finite and all position  $i \in \mathbb{N}$  if  $w$  is infinite. A finite path  $w = s_0 \dots s_n$  is a *loop* in  $\mathcal{S}$  if  $s_0 \in \text{succ}_{\mathcal{S}}(s_n)$ , i.e., if there is a transition from  $s_n$  to  $s_0$ . The path  $w$  is a *row* if no state  $s \in w$  is part of any loop in  $\mathcal{S}$ . The counter system  $\mathcal{S}$  is *flat* if for every state  $s \in S$  there is at most one simple loop  $s_0 \dots s_n$  (a loop with no repeating states) with  $s_0 = s$ . Recall Figure 1.1 showing examples of flat and non-flat control graphs. Let  $\text{CS}$  denote the set of all counter systems and  $\text{FCS}$  the set of all flat counters systems. A counter system  $\mathcal{S}$  is called a *Kripke structure* if  $\text{counters}(\mathcal{S}) = \emptyset$ . The sets  $\text{KS}$  and  $\text{FKS} \subseteq \text{KS}$  are comprised of all Kripke structures and all flat Kripke structures, respectively.

The *representation size* of a set  $\Delta$  of transitions is defined as

$$\text{size}(\Delta) := \sum_{(s, \mu, \Gamma, s') \in \Delta} 1 + \left( \sum_{\gamma \in \Gamma} |\gamma| \right) + \sum_{c \in \text{dom}(\mu)} \text{size}(\mu(c)).$$

Notice that  $\text{size}(\Delta) = |\Delta|$  if the transitions carry neither updates nor guards and that, in general,  $\text{size}(\Delta) \geq |\Delta|$ . The size of a counter system  $\mathcal{S} = (S, \Delta_{\mathcal{S}}, s_I, \lambda_{\mathcal{S}})$  is now defined in terms of the representation of its transition relation denoted by  $|\mathcal{S}| := \text{size}(\Delta_{\mathcal{S}})$ .

## Reachability

An important problem for counter systems is the question whether a particular control state is visited (once or infinitely often) by any execution. Formally, the *control-state reachability problem* of a class  $K \subseteq \text{CS}$  of counter systems is to decide for an instance  $\mathcal{S} = (S, \Delta, s_I, \lambda) \in K$  and a state  $s \in S$  whether there is a run  $\rho \in \text{runs}(\mathcal{S})$  and a position  $i \in \mathbb{N}$  such that  $st(\rho(i)) = s$ . The *repeated control-state reachability problem* is to decide whether there is a run  $\rho \in \text{runs}(\mathcal{S})$  and an infinite subset  $I \subseteq \mathbb{N}$  of positions such that  $st(\rho(i)) = s$  for all  $i \in I$ .

At their core, many verification algorithms build on solving these questions and this is also the case for most procedures developed here. Therefore, it is important to realise that counter systems in their general form represent an extremely powerful model of computation. This is foremost witnessed by the well-known result by Minsky [Min67] providing the equivalence of counter systems (even with only two counters and zero-testing guards) and Turing machines. It is hence undecidable whether a particular configuration or a particular control state is (repeatedly) reachable.

► **Theorem 2.1** ([Min67]). *The (repeated) control-state reachability problem for counter systems is undecidable.*

Flatness, on the other hand, recovers decidability of these fundamental problems. In fact, control-state reachability becomes NP-complete: Encoding the NP-complete subset-sum problem, e.g., as described by Haase et al. [Haa+09], requires only a flat system with one counter and hence witnesses NP-hardness. Further, (repeated) control-state reachability can be expressed in LTL and Demri et al. [DDS12; DDS15] have shown that evaluating LTL properties on flat counter systems is in NP.

► **Theorem 2.2** ([DDS15]). *The (repeated) control-state reachability problem for flat counter systems is NP-complete.*

## 2.2 Temporal Logic with Counting

In the following, we define the temporal logics that are the central object of study in this dissertation. Recall the diagram shown in Figure 1.3. Syntax and semantics are first defined for  $\text{CTL}_{\#}^*$ , the richest variant of the logic family. Subsequently, the other fragments are obtained by specific restrictions.

The logics  $\text{CTL}_{\#}^*$ ,  $\text{CTL}_{\#}$  and  $\text{CTL}$  feature a bookmarking mechanism based on position variables that store positions in a register-like fashion for later reference, similarly to the *freeze quantifier* used by Alur and Henzinger [AH94] for storing time stamps. Henzinger [Hen90] refers to it as “half-order” quantification (as opposed to first- or second order) because it binds a variable to a specific value exhibited by the part (state) of the model it is to be evaluated at. In  $\text{cCTL}^*$ ,  $\text{cCTL}$  and  $\text{cLTL}$ , access to the quantified variable is, essentially, restricted to the first temporal level. An alternative notation scheme based on annotating temporal operators is used to naturally impose this restriction.

### Syntax

Let  $AP$  be a fixed, finite set of *atomic propositions* and  $C$  be a fixed, infinite set of *counter names*. Although any formula can only use a finite number of counters, assuming  $C$  to be infinite provides the choice of an unused (fresh) counter name when required. Similarly, let  $B$  be a set of position variables, called *bookmarks*. The syntax of  $\text{CTL}_{\#}^*$  formulae  $\varphi$  and counting constraint terms  $\tau$  is given by the grammar rules

$$\begin{aligned} \varphi &::= \text{true} \mid p \mid \gamma \mid \varphi \wedge \varphi \mid \neg \varphi \mid \mathbf{X} \varphi \mid \varphi \mathbf{U} \varphi \mid \mathbf{E} \varphi \mid x.\varphi \mid \tau \geq a \\ \tau &::= a \cdot \#_x(\varphi) \mid \tau + \tau \end{aligned}$$

for atomic propositions  $p \in AP$ , counter guards  $\gamma \in \mathbf{Grd}(C)$ , bookmarks  $x \in B$ , and integer constants  $a \in \mathbb{Z}$ . The set of all  $\text{CTL}_{\#}^*$  formulae over  $AP$  and  $C$  is also denoted by  $\text{CTL}_{\#}^*$ . Analogously to counter systems, let  $\text{counters}(\varphi) \subseteq C$  denote the set of counter names actually occurring in a formula  $\varphi$ . We write  $\text{CTL}_{\#}^*(C')$  for the set of those  $\text{CTL}_{\#}^*$  formulae  $\varphi$  with  $\text{counters}(\varphi) \subseteq C'$ .

Notice that  $\text{CTL}_{\#}^*$  counting constraints, i.e. formulae of the form  $\tau \geq b$  for some constraint term  $\tau$  and  $b \in \mathbb{Z}$ , are in fact PA formulae (more precisely, guards) over the set  $\{\#_x(\varphi) \mid x \in B, \varphi \in \text{CTL}_{\#}^*\}$  used as placeholders. Thus, the corresponding abbreviations defined above may also be used where convenient. As for PA, the multiplication symbol may be omitted in terms to improve readability if no ambiguity arises. Observe, moreover, that constraints over system counters are syntactically distinct from constraints over counted quantities, e.g., a formula  $2\#_x(\varphi) + c \geq 0$ , for  $c \in C$ , is not valid because it contains both a system counter ( $c$ ) and a counting expression ( $\#_x(\varphi)$ ). Although both types of inequations are technically guards (over different placeholder sets), to avoid confusion we mostly refer to the first type (from  $\mathbf{Grd}(C)$ ) as *counter guards* and to the other as *counting constraints*.

**Derived operators.** Let us extend the core syntax by additional operators derived by using the equivalences

$$\begin{aligned} \text{false} &::= \neg \text{true} & \mathbf{F} \varphi &::= \text{true} \mathbf{U} \varphi & \varphi \mathbf{EU} \psi &::= \mathbf{E}(\varphi \mathbf{U} \psi) \\ \varphi \vee \psi &::= \neg((\neg \varphi) \wedge (\neg \psi)) & \mathbf{G} \varphi &::= \neg \mathbf{F} \neg \varphi & \varphi \mathbf{AU} \psi &::= \neg \mathbf{E} \neg(\varphi \mathbf{U} \psi) \\ \varphi \rightarrow \psi &::= (\neg \varphi) \vee \psi & \varphi \mathbf{R} \psi &::= \neg(\neg \varphi \mathbf{U} \neg \psi) \end{aligned}$$

for all  $\varphi, \psi \in \text{CTL}_{\#}^*$ . We will use them as convenient abbreviations and also to define syntactic fragments of the logic. By substituting derived operators as determined by

their defined equation, any formula can be reduced to the core syntax defined above. Therefore, it is no restriction to limit structural arguments to the core syntax. Derived operators will only be considered explicitly in the context of fragments that exclude core constructs necessary for expressing them.

**Subformulae.** The set of all *subformulae* of a formula  $\varphi \in \mathbf{CTL}_{\#}^*$  is defined to be the smallest set  $sub(\varphi) \subseteq \mathbf{CTL}_{\#}^*$  such that  $\varphi \in sub(\varphi)$  and further

- if  $\neg\psi \in sub(\varphi)$ ,  $\mathbf{X}\psi \in sub(\varphi)$ ,  $\mathbf{E}\psi \in sub(\varphi)$ , or  $x.\psi \in sub(\varphi)$ , then  $\psi \in sub(\varphi)$ ,
- if  $\chi \wedge \psi \in sub(\varphi)$  or  $\chi \mathbf{U}\psi \in sub(\varphi)$ , then  $\chi, \psi \in sub(\varphi)$ , and
- if  $a_0 \cdot \#_{x_0}(\chi_0) + \dots + a_n \cdot \#_{x_n}(\chi_n) \geq b \in sub(\varphi)$ , then  $\chi_0, \dots, \chi_n \subseteq sub(\varphi)$

for all  $n \in \mathbb{N}$ ,  $b, a_0, \dots, a_n \in \mathbb{Z}$ , and  $\psi, \chi, \chi_0, \dots, \chi_n \in \mathbf{CTL}_{\#}^*$ . Although counting terms are not considered as (sub)formulae, let us extend the notation and also define

$$sub(a_0 \cdot \#_{x_0}(\chi_0) + \dots + a_n \cdot \#_{x_n}(\chi_n)) := sub(\chi_0) \cup \dots \cup sub(\chi_n).$$

Let the size  $|\varphi|$  of a formula be the sum of the sizes of the atomic elements (propositions, atomic constraints) it contains and the number of operators connecting them. Formally, let it be defined recursively over the structure of  $\varphi$  by  $|p| := 1$  for  $p \in AP$  and

- $|\chi \wedge \psi| := |\chi \mathbf{U}\psi| := |\chi| + 1 + |\psi|$ ,
- $|\neg\psi| := |\mathbf{X}\psi| := |\mathbf{E}\psi| := |x.\psi| := 1 + |\psi|$ ,
- $|a_0 \cdot \#_{x_0}(\chi_0) + \dots + a_n \cdot \#_{x_n}(\chi_n) \geq b| := size(a_0) + |\chi_0| + \dots + size(a_n) + |\chi_n| + 1 + size(b)$

for all  $n \in \mathbb{N}$ ,  $a_0, \dots, a_n, b \in \mathbb{Z}$ , and  $\psi, \chi, \chi_0, \dots, \chi_n \in \mathbf{CTL}_{\#}^*$ . Let the size of atomic guards  $\gamma \in \mathbf{Grd}(C)$  be defined as above (Section 2.1.2), where  $size(c) := 1$  for each counter  $c \in C$ .

**Brackets and Operator Precedence.** In formulae, brackets are used to disambiguate the syntactic scope of a logical connective. However, since exhaustive use clutters notation let us assume precedence according to the levels assigned in Table 2.1. Quantification of a bookmark hides each bookmark with the same name outside of its scope. For example,

$$\begin{aligned} & \mathbf{G} x. \quad p_1 \wedge \neg p_2 \vee p_3 \quad \mathbf{U} \quad x. \quad p_2 \rightarrow \mathbf{F} \#_x(p_4) \geq 5 \wedge p_5 \\ \equiv & \mathbf{G} y. \left( ((p_1 \wedge \neg p_2) \vee p_3) \quad \mathbf{U} \quad (x. (p_2 \rightarrow \mathbf{F}(\#_x(p_4) \geq 5 \wedge p_5))) \right). \end{aligned}$$

Level	6	5	4	3	2	1
Connectives	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\mathbf{U}, \mathbf{U}_{[.]}$	$\mathbf{X}, \mathbf{G}, \mathbf{F}, \mathbf{F}_{[.]}, \mathbf{E}, \mathbf{A}, \cdot$

Table 2.1: Precedence levels of logical connectives. A connective with higher level is to be evaluated before those of lower levels. Within each level, all connectives are right-associative.

### Semantics

Intuitively, a bookmark  $x \in B$  is used to mark some position on the concerned run. It can be considered as a variable holding a position. In the formula within the scope of its quantification, a term  $\#_x(\varphi)$  refers to the number of times the formula  $\varphi$  holds between the current position and that marked by  $x$ . The semantics of a  $\text{CTL}_{\#}^*$  formula is hence defined with respect to a counter system  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  over counters  $C_{\mathcal{S}} = \text{counters}(\mathcal{S})$ , a run  $\rho \in \text{runs}(\mathcal{S})$ , a position  $i \in \mathbb{N}$  on  $\rho$  and a bookmark valuation function  $\beta : B \rightarrow \mathbb{N}$  assigning a position (index) on  $\rho$  to each bookmark. For  $p \in AP$ ,  $\gamma \in \text{Grd}(C_{\mathcal{S}})$ ,  $x \in B$ , and  $\tau \geq b$ ,  $\varphi, \psi \in \text{CTL}_{\#}^*(C_{\mathcal{S}})$  the satisfaction relation  $\models$  is defined inductively by

$$\begin{aligned}
 (\mathcal{S}, \rho, i, \beta) &\models \text{true} \\
 (\mathcal{S}, \rho, i, \beta) &\models p && :\Leftrightarrow p \in \lambda(\rho(i)) \\
 (\mathcal{S}, \rho, i, \beta) &\models \gamma && :\Leftrightarrow \text{val}(\rho(i)) \models_{\text{PA}} \gamma \\
 (\mathcal{S}, \rho, i, \beta) &\models \neg\varphi && :\Leftrightarrow (\mathcal{S}, \rho, i, \beta) \not\models \varphi \\
 (\mathcal{S}, \rho, i, \beta) &\models \varphi \wedge \psi && :\Leftrightarrow (\mathcal{S}, \rho, i, \beta) \models \varphi \text{ and } (\mathcal{S}, \rho, i, \beta) \models \psi \\
 (\mathcal{S}, \rho, i, \beta) &\models \mathbf{X}\varphi && :\Leftrightarrow (\mathcal{S}, \rho, i+1, \beta) \models \varphi \\
 (\mathcal{S}, \rho, i, \beta) &\models \varphi \mathbf{U} \psi && :\Leftrightarrow \exists_{j \geq i} : (\mathcal{S}, \rho, j, \beta) \models \psi \text{ and } \forall_{k \in [i, j-1]} : (\mathcal{S}, \rho, k, \beta) \models \varphi \\
 (\mathcal{S}, \rho, i, \beta) &\models \mathbf{E}\varphi && :\Leftrightarrow \exists_{\rho' \in \text{runs}(\mathcal{S})} \forall_{j \in [0, i]} : \rho'(j) = \rho(j) \text{ and } (\mathcal{S}, \rho', i, \beta) \models \varphi \\
 (\mathcal{S}, \rho, i, \beta) &\models x.\varphi && :\Leftrightarrow (\mathcal{S}, \rho, i, \beta[x \mapsto i]) \models \varphi \\
 (\mathcal{S}, \rho, i, \beta) &\models \tau \geq b && :\Leftrightarrow \llbracket \tau \rrbracket(\mathcal{S}, \rho, i, \beta) \geq b
 \end{aligned}$$

where the semantics of counting constraint terms  $\tau$  is given for  $a \in \mathbb{Z}$ , by

$$\begin{aligned}
 \llbracket \tau_1 + \tau_2 \rrbracket(\mathcal{S}, \rho, i, \beta) &:= \llbracket \tau_1 \rrbracket(\mathcal{S}, \rho, i, \beta) + \llbracket \tau_2 \rrbracket(\mathcal{S}, \rho, i, \beta) \\
 \llbracket a \cdot \#_x(\varphi) \rrbracket(\mathcal{S}, \rho, i, \beta) &:= a \cdot |\{j \in [\beta(x), i] \mid (\mathcal{S}, \rho, j, \beta) \models \varphi\}|.
 \end{aligned}$$

For improved readability, the bookmark valuation  $\beta$  and the position  $i$  may be omitted if they are equal to  $\mathbf{0}$  and  $0$ , respectively, i.e.,  $(\mathcal{S}, \rho) \models \varphi$  abbreviates  $(\mathcal{S}, \rho, 0, \mathbf{0}) \models \varphi$ . The run  $\rho$  of  $\mathcal{S}$  is said to satisfy the formula  $\varphi$  if  $(\mathcal{S}, \rho) \models \varphi$ . The counter system  $\mathcal{S}$  satisfies the formula  $\varphi$ , if there exists some run  $\rho \in \text{runs}(\mathcal{S})$  such that  $(\mathcal{S}, \rho) \models \varphi$  and in this case let us write  $\mathcal{S} \models \varphi$ .

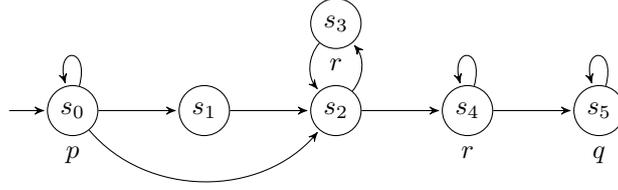


Figure 2.1: A flat Kripke structure labelled by propositions  $p$ ,  $q$ , and  $r$ . [Dec+17]

► **Remark 2.3.** Notice that the model relation is defined existentially, although it is mostly found to be defined universally in the model-checking literature. The methods and results developed here are mainly concerned with the existence of a run satisfying a formula, therefore the existential definition seems more natural in this context. Nevertheless, the considered logical formalisms are closed under negation and therefore the choice does not have consequences on the results apart from switching to complement-complexity classes, e.g., from NP and coNP.

► **Example 2.4** ([Dec+17]). Consider the Kripke structure shown in Figure 2.1 and the  $CTL_{\#}^*$  formula  $\varphi_1 = z. \mathbf{AG} q \rightarrow (\#_z(p) \leq \#_z(\mathbf{EX} r))$ . The latter basically states that on every path reaching the state  $s_5$  there must be a position where the states  $s_2$  and  $s_4$  together have been visited at least as often as the state  $s_0$ . The formula is violated as witnessed by the path  $s_0^3 s_1 s_2 s_4 s_5^{\omega}$ . The Kripke structure, however, satisfies  $\varphi_2 = z. \mathbf{AG} \neg q \rightarrow \mathbf{EF} \#_z(p) < \#_z(r)$  because from every state except  $s_5$  the number of positions that satisfy  $r$  can be increased arbitrarily without increasing the number of those satisfying  $p$ . Notice that this would not be the case if, e.g.,  $s_4$  were labelled by  $p$ .

### Decision Problems: Satisfiability and Model Checking

Henceforth, we define the decision problems to be studied. They are formalised in terms of sets and thus deciding them refers to algorithmically determine membership of a given object. Satisfiability and validity are fundamental aspects of a formal logic. They represent classical problems to be studied and semantic questions can often be reduced to them. Therefore, finding and understanding procedures to determine whether a given formula admits some model, or is a tautology satisfied by all models, provides valuable conceptual insights for understanding the nature of a formalism. We use a formulation that is parametrised by a specific fragment of  $CTL_{\#}^*$  and a class of models.

► **Definition 2.5** (Satisfiability). Let  $\mathcal{L} \subseteq CTL_{\#}^*$  be a fragment of  $CTL_{\#}^*$  and  $M \subseteq CS$  a class

of counter systems. The satisfiability problem of  $\mathcal{L}$  over  $M$  is the set

$$SAT(\mathcal{L}, M) := \{\varphi \in \mathcal{L} \mid \exists \mathcal{S} \in M : \mathcal{S} \models \varphi\}$$

of formulae satisfied by some system of the class  $M$ .

The initial configurations of counter systems are deliberately defined to assign zero to all counters. Consequently, the definition above considers a formula only satisfiable if there is a system and satisfying run that starts at such a configuration. Notice, however, that this is not essential: the questions whether there is such a zero-initialised run satisfying a formula and whether there is a run starting with *any* configuration are easily reduced to each other in the relevant classes<sup>1</sup> of counter systems. For example, a formula  $\varphi \in \mathbf{CTL}_{\#}^*$  is satisfiable by a flat counter system with arbitrary initial valuation if and only if  $\mathbf{F}\varphi$  is satisfiable by a flat counter system starting with all counters being zero.

The model-checking problem is essentially the model relation defined above to give semantics to logic formulae. As for satisfiability the problem is parametrised by a concrete logic and a class of models.

► **Definition 2.6** (Model checking). *Let  $\mathcal{L} \subseteq \mathbf{CTL}_{\#}^*$  be a fragment of  $\mathbf{CTL}_{\#}^*$  and  $M \subseteq CS$  a class of counter systems. The model-checking problem of  $\mathcal{L}$  over  $M$  is the set*

$$MC(\mathcal{L}, M) := \{(\varphi, \mathcal{S}) \in \mathcal{L} \times M \mid \mathcal{S} \models \varphi\}$$

relating each formula  $\varphi \in \mathcal{L}$  to those systems  $\mathcal{S} \in M$  that contain a run satisfying it.

## 2.3 Fragments

In the following, let us define formally the fragments of  $\mathbf{CTL}_{\#}^*$  to be studied. The restrictions consider essentially two dimensions: temporal navigation and the counting mechanism.

### Linear- and Branching-time Fragments

The first dimension to be restricted concerns the distinction between linear-time and branching-time properties characterised by the admitted combination of path- and temporal quantification. Historically, the temporal logic  $\mathbf{CTL}^*$  was designed to subsume the linear-time temporal logic LTL and the branching-time temporal logic CTL that are both well-known to be incomparable in terms of expressiveness, that is, which classes

<sup>1</sup>Clearly, classes can be constructed where the initial condition matters, e.g., a class containing only a single counter system.

of structures they can distinguish or not. Both LTL and CTL build on the temporal modalities **X** and **U** but while the path quantifier **E** is absent in LTL, the branching-time logic CTL imposes that every temporal operator be preceded by either **E** or its dual **A**.

Consequently, let the linear-time fragment  $\text{LTL}_\# \subseteq \text{CTL}_\#^*$  consist of those formulae that do not use the explicit path quantifier **E** (nor any operator derived from it). The branching-time fragment  $\text{CTL}_\# \subseteq \text{CTL}_\#^*$  consist of the formulae using only **AX**, **EX**, **AU**, and **EU** as temporal operators. Also, the stand-alone use of **E** (and **A**) is discarded in this fragment since it has no semantic effect. Notice that the formulae used in Example 2.4 above are contained in this fragment.

### Operator-bound Restriction of the Counting Mechanism

Since the object of study here is the extension of temporal logic by the ability to count, this latter mechanism constitutes the second dimension of restrictions. The ability of bookmarking and counting positions in  $\text{CTL}_\#^*$  is very general and restrictions can be imposed. One aspect that can be restricted is the means to specify the *scope* that is subject to counting constraints. The bookmarking mechanism allows for specifying the scope along an arbitrary number of temporal operators where different constraints can be imposed at any intermediate step. For example, consider the formula

$$x. (\#_x(p_1) - \#_x(p_2) \geq 0 \mathbf{U} \mathbf{X} \mathbf{X} \mathbf{G} \#_x(p_3) - \#_x(p_1) - \#_x(p_2) \geq 0),$$

stating that there are at least as many positions with  $p_1$  as with  $p_2$ , until  $p_3$  outnumbers both forever. The constraint  $\#_x(p_1) - \#_x(p_2) \geq 0$  is imposed on a number of scopes, namely every prefix of a run up to some position satisfying the right-hand side of the **U** operator. Moreover, the second constraint is imposed on scopes that are determined by the **U** operator, extended by two steps forward and then the further **G** operator. In contrast, the fragment  $\text{cCTL}^*$  restricts the evaluation of counting constraints to only scopes determined by a specific (extended) **U** operator. In a  $\text{cCTL}^*$  formula  $\varphi \mathbf{U}_{[\tau \geq b]} \psi$ , for example, the subformulae  $\varphi$  and  $\psi$  may use counting operators but cannot refer to the scope of the connecting  $\mathbf{U}_{[\cdot]}$  determining the scope in which the constraint  $\tau \geq b$  is evaluated.

The results presented in this work show that this restriction may have a significant effect on the complexity of the model-checking problem (cf. Table 1.1). A second aspect is the specific type of constraints that can be formulated. Laroussinie, Meyer and Petonnet [LMP12] have identified and examined various types of constraints, e.g., admitting only positive coefficients or so-called diagonal constraints. The forthcoming investigations

concentrate on conjunction-free constraints with arbitrary integer coefficients.

Let the set of  $\text{cCTL}^*$  formulae  $\varphi$  be denoted just by  $\text{cCTL}^*$  and defined by the grammar

$$\begin{aligned} \varphi &::= \text{true} \mid p \mid \gamma \mid \varphi \wedge \psi \mid \neg \varphi \mid \mathbf{X} \varphi \mid \varphi \mathbf{U}_{[\tau \geq b]} \psi \mid \mathbf{E} \varphi \\ \tau &::= a \cdot \varphi \mid \tau + \tau \end{aligned}$$

for atomic propositions  $p \in AP$ , guards on system counters  $\gamma \in \text{Grd}(C)$  and integer constants  $a \in \mathbb{Z}$ . That is, compared to  $\text{CTL}_{\#}^*$ , the bookmarking mechanism is dismissed in favour of the extended temporal until modality  $\mathbf{U}_{[\cdot]}$  that is parametrised by a counting constraint  $\tau \geq b \in \text{Grd}(\text{cCTL}^*)$ .

Let  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  be a counter system as before,  $\rho \in \text{runs}(\mathcal{S})$  a run, and  $i \in \mathbb{N}$  a position on  $\rho$ . For  $p \in AP$ ,  $\tau \geq b \in \text{Grd}(\text{cCTL}^*)$ ,  $\gamma \in \text{Grd}(C)$ , and  $\varphi, \psi \in \text{cCTL}^*$  the satisfaction relation  $\models$  is defined inductively by

$$\begin{aligned} (\mathcal{S}, \rho, i) &\models \text{true} \\ (\mathcal{S}, \rho, i) &\models p && :\Leftrightarrow p \in \lambda(\rho(i)) \\ (\mathcal{S}, \rho, i) &\models \gamma && :\Leftrightarrow \text{val}(\rho(i)) \models_{\text{PA}} \gamma \\ (\mathcal{S}, \rho, i) &\models \neg \varphi && :\Leftrightarrow (\mathcal{S}, \rho, i) \not\models \varphi \\ (\mathcal{S}, \rho, i) &\models \varphi \wedge \psi && :\Leftrightarrow (\mathcal{S}, \rho, i) \models \varphi \text{ and } (\mathcal{S}, \rho, i) \models \psi \\ (\mathcal{S}, \rho, i) &\models \mathbf{X} \varphi && :\Leftrightarrow (\mathcal{S}, \rho, i+1) \models \varphi \\ (\mathcal{S}, \rho, i) &\models \varphi \mathbf{U}_{[\tau \geq b]} \psi && :\Leftrightarrow \exists j \geq i : (\mathcal{S}, \rho, j) \models \psi \text{ and } \llbracket \tau \rrbracket(\mathcal{S}, \rho, i, j-1) \geq b \\ &&& \text{and } \forall k \in [i, j-1] : (\mathcal{S}, \rho, k) \models \varphi. \\ (\mathcal{S}, \rho, i) &\models \mathbf{E} \varphi && :\Leftrightarrow \exists \rho' \in \text{runs}(\mathcal{S}) \forall j \in [0, i] : \rho'(j) = \rho(j) \text{ and } (\mathcal{S}, \rho', i) \models \varphi \end{aligned}$$

where the semantics of counting constraint terms  $\tau$  is given for  $a \in \mathbb{Z}$ , by

$$\begin{aligned} \llbracket \tau_1 + \tau_2 \rrbracket(\mathcal{S}, \rho, i, j) &:= \llbracket \tau_1 \rrbracket(\mathcal{S}, \rho, i, j) + \llbracket \tau_2 \rrbracket(\mathcal{S}, \rho, i, j) \\ \llbracket a \cdot \varphi \rrbracket(\mathcal{S}, \rho, i, j) &:= a \cdot |\{k \in [i, j] \mid (\mathcal{S}, \rho, k) \models \varphi\}|. \end{aligned}$$

As for  $\text{CTL}_{\#}^*$ ,  $(\mathcal{S}, \rho) \models \varphi$  abbreviates  $(\mathcal{S}, \rho, 0) \models \varphi$ .

Now, it follows from the definition that the counting until operator can be expressed in  $\text{CTL}_{\#}^*$ . A formula  $\varphi \mathbf{U}_{[a_0 \chi_0 + \dots + a_n \chi_n \geq b]} \psi$  holds on some run  $\rho$  at position  $i$  if  $\psi$  holds at  $i$  and  $0 \geq b$ —i.e., precisely if  $(\mathcal{S}, \rho, i) \models x.(\#_x(\text{false}) \geq b \wedge \psi)$ —or  $\varphi$  holds up to and including some position  $j \geq i$  where the counting constraint is satisfied and that is followed by a position  $j+1$  satisfying  $\psi$ . The latter can be stated as

$$x.(\varphi \mathbf{U}(\varphi \wedge a_0 \cdot \#_x(\chi_0) + \dots + a_n \cdot \#_x(\chi_n) \geq b \wedge \mathbf{X} \psi))$$

und thus the equivalence

$$\begin{aligned} \varphi \mathbf{U}_{[a_0\chi_0+\dots+a_n\chi_n \geq b]} \psi &\equiv \\ x.(\#_x(\text{false}) \geq b \wedge \psi) \vee (\varphi \mathbf{U} \varphi \wedge a_0 \cdot \#_x(\chi_0) + \dots + a_n \cdot \#_x(\chi_n) \geq b \wedge \mathbf{X} \psi) \end{aligned}$$

holds for some fixed bookmark symbol  $x$ . Therefore, the  $\mathbf{U}_{[\cdot]}$  operator can be considered as derived and  $\text{cCTL}^*$  as fragment of  $\text{CTL}_\#^*$ .

The linear-time and branching-time fragments  $\text{cLTL}$  and  $\text{cCTL}$  are defined by restricting the combinations of temporal operators and path quantifiers as above for  $\text{LTL}_\#$  and  $\text{CTL}_\#$ . For  $\text{cLTL}$ , path quantifiers are entirely discarded while in  $\text{cCTL}$  the path temporal operators  $\mathbf{X}$  and  $\mathbf{U}_{[\cdot]}$  are replaced by  $\mathbf{AX}$ ,  $\mathbf{EX}$ ,  $\mathbf{AU}_{[\cdot]}$ , and  $\mathbf{EU}_{[\cdot]}$ . In addition to the abbreviations introduced above we define  $\mathbf{F}_{[\tau \geq b]} \varphi := \text{true} \mathbf{U}_{[\tau \geq b]} \varphi$ .

## ■ CHAPTER 3

# Satisfiability of LTL with Counting

Satisfiability is a classical problem in formal logic and to study often provides valuable insights and helps to characterise and relate such formalisms. Therefore, we investigate the satisfiability problem of LTL with counting in this chapter. First, we observe that interpreting LTL over counter systems does not affect the complexity of the model checking problem, determined to be PSPACE-complete by Halpern and Reif [HR81] and Sistla and Clarke [SC82]. The reason is, essentially, that the only additional computation that may be necessary is solving linear equation systems of linear size and this does not involve significantly more resources than the rest of the procedure requires anyway.

The counting mechanism is therefore the more interesting<sup>1</sup> dimension: even very restricted forms of counting lead to an undecidable satisfiability problem and this carries over to all of the fragments defined so far. In contrast to the model-checking problems that are investigated in the later chapters, restricting to flat models does not recover decidability. However, we identify a decidable fragment of  $LTL_{\#}$  that does not directly restrict the counting mechanism.

## 3.1 Satisfiability of LTL over Counter Systems

Traditionally, LTL is defined primarily over words with a finite alphabet  $\Sigma = 2^{AP}$  but we can easily translate the results to the present setting.

► **Theorem 3.1.** *The problem  $SAT(LTL, CS)$  is PSPACE-complete.*

**Proof.** We can essentially adopt the result by Sistla and Clarke [SC85]. To this end, we will first show how to apply one of the well-known algorithms for LTL satisfiability with minor modification. Subsequently, it will be demonstrated that  $SAT(LTL, CS)$  entails the traditional problem and is thus equally hard.

---

<sup>1</sup>Especially undecidable logics are considered to be interesting, e.g., by Bollig [Bol11].

*SAT(LTL, CS) is in PSPACE.* Let  $\varphi \in \text{LTL}$  be an LTL formula,  $\Gamma_\varphi := \text{sub}(\varphi) \cap \text{Grd}(C)$  the guards used in  $\varphi$  and  $C_\varphi := \text{counters}(\varphi) \subseteq C$  the counters mentioned in  $\varphi$ . The semantics, as defined above, admits only runs starting with all counters being set to zero, i.e., an initial configuration of the form  $(q, \mathbf{0})$ . Therefore,  $\varphi$  is semantically equivalent to  $\varphi \wedge \bigwedge_{c \in C_\varphi} c = 0$ , and let us henceforth assume that  $\varphi$  explicitly specifies the initial configuration this way.

A well-known decision procedure for LTL satisfiability (described, e.g., in the textbook by Baier and Katoen [BK08]) translates a formula over propositions  $AP$  to an equivalent Büchi automaton over the alphabet  $\Sigma = 2^{AP}$ . A nested depth-first search is then used to examine the control graph and to determine whether an accepting run exist. Although the procedure assumes a finite set of propositions, it can equally be applied to  $\varphi$  by interpreting each guard in  $\varphi$  as atomic proposition from the set  $\hat{AP} := AP \cup \Gamma_\varphi$ . This may introduce spurious solutions because some letters over the corresponding extended alphabet  $\hat{\Sigma} := 2^{\hat{AP} \cup \Gamma_\varphi}$  are invalid: the procedure has to be modified such that it ignores those letters  $a \in \hat{\Sigma}$  that contain contradictory constraints, i.e., where the conjunction  $\bigwedge_{\gamma \in a \cap \Gamma_\varphi} \gamma$  is not satisfiable. This simply requires restricting the depth-first search to take only those transitions into account that carry valid letters. Checking the additional condition whenever a transition is about to be selected can be done (non-deterministically) in polynomial time [BT76], hence staying within a polynomial space bound.

The modified procedure is complete: If  $\varphi$  is in fact satisfiable by some run  $\rho$  in some counter system  $\mathcal{S}$ , then there is a corresponding word  $\llbracket \rho \rrbracket_{\hat{\Sigma}}$  over the alphabet  $\Sigma \times \mathbb{Z}^C$ . We can derive a symbolic word  $\hat{w} \in \hat{\Sigma}^\omega$  abstracting the actual valuation  $\text{val}(\rho(i))$  in terms of the maximal set  $\Gamma_i := \{\gamma \in \Gamma_\varphi \mid \text{val}(\rho(i)) \models_{\text{PA}} \gamma\}$  of satisfied guards, i.e., with  $\hat{w}(i) := (\lambda(\text{st}(\rho(i))) \cap AP) \cup \Gamma_i$ . This word  $\hat{w}$  then satisfies the formula  $\varphi$  interpreted over  $\hat{\Sigma}$  and therefore the algorithm will determine that  $\varphi$  is satisfiable (by a word model over  $\hat{\Sigma}$ ).

Concerning soundness of the procedure, assume that it terminates successfully. Then, there is a symbolic solution of the form  $\hat{w} = uv^\omega \in \hat{\Sigma}^\omega$  (with  $u, v \in \hat{\Sigma}^*$ ) satisfying the formula interpreted over the extended proposition set. This solution defines a counter system with a corresponding run that, by construction, satisfies the formula  $\varphi$ . Let  $k := |u|$  and  $n := |uv| - 1$  be the first and last position, respectively, of the segment  $v$  on  $uv$ . Further, let  $a_i := \hat{w}(i) \cap AP$  be the actual letter at position  $i$  and  $\theta_i : C_\varphi \rightarrow \mathbb{Z}$  be some solution of the conjunction of the guards at position  $i$ . That is,  $\theta_i \models_{\text{PA}} \bigwedge_{\gamma \in \hat{w}(i) \cap \Gamma_\varphi} \gamma$  and such solutions  $\theta_i$  exists due to the modification of the standard procedure. Let us, without loss of generality, choose the same solutions  $\theta_i = \theta_{i'}$  if the corresponding letters  $\hat{w}(i) = \hat{w}(i')$  are identical. Then, the sequence

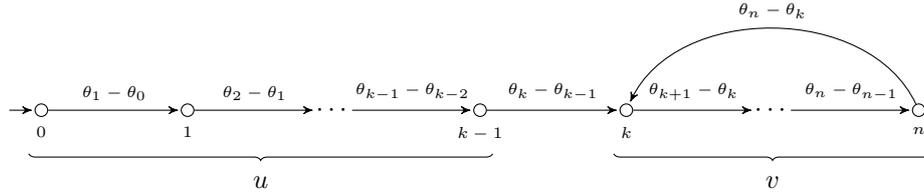


Figure 3.1: The counter system  $\mathcal{S}_{\hat{w}}$  built from a symbolic word model  $\hat{w} = uv^\omega$  over the extended set of propositions  $AP \cup \Gamma_\varphi$ .

of valuations is periodic in the same way as the letters in the word  $uv^\omega$ . Now, let  $\mathcal{S}_{\hat{w}} := ([0, n], \Delta_{\hat{w}}, 0, \lambda_{\hat{w}})$  be the counter system depicted in Figure 3.1 with the positions of the word  $uv$  used as states and labelled by  $\lambda_{\hat{w}}(i) := a_i$ . The updates on the transitions between states  $i$  and  $j$  are defined by  $\mu_{i,j} := \theta_j - \theta_i$ . The system precisely admits the run  $\rho = (0, \theta_0) \dots (k-1, \theta_{k-1})((k, \theta_k) \dots (n, \theta_n))^\omega$  and thus satisfies  $\varphi$ . Recall that  $\varphi$  was assumed to specify that the initial counter valuation is  $\theta_0 = \mathbf{0}$ .

*SAT(LTL, CS) is PSPACE-hard.* For LTL over finite alphabets, corresponding here to the case of  $\text{LTL}(\emptyset)$  where no counter name occurs syntactically, the satisfiability problem was shown to be PSPACE-complete by Sistla and Clarke [SC85]. From the corresponding decision procedure it follows that if such a formula  $\varphi \in \text{LTL}(\emptyset)$  without counter constraints is satisfiable by some word  $w' \in \Sigma^\omega$ , then it is satisfiable by a periodic word of the form  $w = uv^\omega \in \Sigma^\omega$  and thus by a corresponding Kripke structure with that shape (cf. Figure 3.1). On the other hand, if  $(\mathcal{S}, \rho) \models \varphi$  for any counter system  $\mathcal{S}$ , the counters cannot matter and the labelling along the corresponding path  $\rho \in \text{runs}(\mathcal{S})$  provides a word over  $\Sigma$ . This reduces the traditional PSPACE-hard satisfiability problem to  $\text{SAT}(\text{LTL}, \text{CS})$ . ■

Notice, that the system  $\mathcal{S}_{\hat{w}}$  constructed in the proof above is in fact flat. Thus, a formula is satisfiable if and only if it is satisfiable over flat structures. Further, checking whether a formula  $\varphi$  uses any counter is trivial.

► **Corollary 3.2.** *The satisfiability problems of LTL over KS, FKS, and FCS are PSPACE-complete.*

## 3.2 Undecidability Results

An inherent source of undecidability lies in the concept of counting itself, as witnessed most prominently by the well-known correspondence between Turing machines and

counter systems established by Minsky [Min67]. Very little is necessary to leave the realm of automatic analysis in this context and restrictions must be chosen carefully. In LTL, even frequency constraints allow for describing computations of a class of counter systems denying semantic analyses.

A *Minsky machine* is a configuration-deterministic counter system that is essentially limited to using only two counters, updates that either increment or decrement one of the counters by one, and zero tests. The problem of determining whether the unique computation reaches a given control state was shown to be undecidable by Minsky [Min67]. It was shown in [BDL12] that the computation of a given Minsky machine can be encoded as word over a finite alphabet and specified to reach a given control state by a formula in the logic  $\mathbf{fLTL}$ . The latter is an extension of LTL proposed in [BDL12] that allows for expressing relative frequencies and can be considered as a fragment of  $\mathbf{cLTL}$  with a very specific type of counting constraints.

If the Minsky machine traverses the target state, a corresponding word encoding the run exists and thus a counter system—more precisely, a Kripke structure—exists and satisfies the formula. In case the run does not exist, the formula is not satisfiable. In fact, the relevant part of an encoded run is finite. The specified words have the form  $wa^\omega$  for a finite word  $w$  and a distinct letter  $a$ . It follows that if such a word encoding a run exists, it is admitted by some *flat* Kripke structure with  $|w| + 1$  states having precisely the shape of the word. Since  $\mathbf{fLTL}$ , as fragment of  $\mathbf{cLTL}$ , and  $\mathbf{FKS}$  reside on the lower end of the hierarchy of logics and structure classes to be investigated, the negative result on satisfiability has a significant impact to the scene: The decision problem reduces trivially to the satisfiability problems of the logics  $\mathbf{cLTL}$ ,  $\mathbf{cCTL}^*$ ,  $\mathbf{LTL}_\#$ , and  $\mathbf{CTL}_\#^*$  over  $\mathbf{FKS}$  and hence  $\mathbf{KS}$ ,  $\mathbf{FCS}$  and  $\mathbf{CS}$ . The situation is similar for the branching-time logics.

As was observed also by Laroussinie, Meyer, and Petonnet [LMP12], the transition relation of a given Minsky machine can easily be described in  $\mathbf{CTL}$  with dedicated propositions indicating incrementation ( $\mathbf{inc}_1$ ,  $\mathbf{inc}_2$ ), decrementation ( $\mathbf{dec}_1$ ,  $\mathbf{dec}_2$ ), and zero-testing ( $\mathbf{eqz}_1$ ,  $\mathbf{eqz}_2$ ) of the first or second counter, respectively. The semantics of zero tests and thereby the validity of a run can then be expressed by a formula such as

$$(\neg \mathbf{EF}_{[\mathbf{inc}_1 - \mathbf{dec}_1 > 0]} \mathbf{eqz}_1) \wedge \neg \mathbf{EF}_{[\mathbf{inc}_2 - \mathbf{dec}_2 > 0]} \mathbf{eqz}_2$$

and further

$$(\neg \mathbf{EF}_{[\mathbf{inc}_1 - \mathbf{dec}_1 < 0]} \mathbf{true}) \wedge \neg \mathbf{EF}_{[\mathbf{inc}_2 - \mathbf{dec}_2 < 0]} \mathbf{true}$$

expresses that the counters never become negative. This way, a formula can be constructed that is satisfied precisely by those counter systems that exhibit a proper representation of

the computation of the Minsky machine on every run, up to reaching a designated target state. As above, if this computation exists, it can be represented as the unique run of a flat Kripke structure of corresponding shape. Otherwise, the formula is not satisfiable.

► **Theorem 3.3** ([BDL12; LMP12]). *The satisfiability problems of cCTL, cLTL and all their supersets are undecidable over FKS, KS, FCS, and CS.*

As will be discussed later in Chapters 4 and 6, also the model-checking problem for cLTL is undecidable, even over Kripke structures, as an immediate consequence of Theorem 3.3. On the contrary, the model-checking problem for cCTL is decidable in polynomial time. To understand what may be necessary to recover decidability in linear-time fragments, a weak fragment of LTL<sub>#</sub> is investigated next.

### 3.3 A Decidable Fragment of LTL<sub>#</sub>

Consider the fragment of LTL<sub>#</sub> formulae  $\varphi$  defined by the grammar

$$\varphi ::= \psi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid z.\varphi \mid \tau \geq a \quad \tau ::= a \cdot \#_x(\psi) \mid \tau + \tau$$

where  $\psi \in \text{LTL}$  can be any plain LTL formula,  $z \in B$  is any bookmark and  $a \in \mathbb{Z}$  is an arbitrary integer value. Let us call this fragment *weak LTL<sub>#</sub>* and denote it  $\text{wLTL}_{\#}$ .

Although the  $\text{wLTL}_{\#}$  fragment is substantially restricted, it can still express frequency constraints such as that  $\psi$  must hold at least two thirds of the time before  $\varphi$  by

$$z.\mathbf{F}\varphi \wedge 3\#_z(\psi) - 2\#_z(\text{true}) \geq 0$$

for  $\psi \in \text{LTL}$  and  $\varphi \in \text{wLTL}_{\#}$ . This is equal to  $\text{true} \mathbf{U}_{[3 \cdot \psi - 2 \cdot \text{true} \geq 0]} \varphi$ . More generally, the logic can express any formula  $\psi \mathbf{U}_{[a_1 \cdot \chi_1 + \dots + a_n \cdot \chi_n \geq b]} \varphi$  by

$$z.\mathbf{F}\varphi \wedge a_1 \cdot \#_z(\chi_1) + \dots + a_n \cdot \#_z(\chi_n) \geq b \wedge \#_z(\psi) - \#_z(\text{true}) \geq 0,$$

as long as  $\psi, \chi_1, \dots, \chi_n$  do not contain counting constraints. The concern of this section is to show why a restriction of this form is of interest: it constitutes a *linear-time* fragment for which satisfiability is decidable.

► **Theorem 3.4.** *The problems SAT(wLTL<sub>#</sub>,CS) and SAT(wLTL<sub>#</sub>,FCS) are in NEXP.*

The idea to prove Theorem 3.4 is to construct from a  $\text{wLTL}_{\#}$  formula an *integer vector addition system with states* ( $\mathbb{Z}$ -VASS) such that the formula is satisfiable if and only if a specific target state  $q$  is reachable with all counters being non-negative, i.e., the

configuration  $(q, \mathbf{0})$  is *coverable*. In terms of the definition in Section 2.1.3, a  $\mathbb{Z}$ -VASS is an entirely unlabelled and unguarded counter system. The coverability problem in  $\mathbb{Z}$ -VASS is well-known to be decidable in non-deterministic polynomial time by reduction to **qfPA** satisfiability (for example, see [Rei16]).

The construction is presented in Section 3.3.2 and assumes the formula to have a specific normal form that we discuss first. Finally, we observe that only little must be changed to obtain a model-checking procedure for Kripke structures.

### 3.3.1 A Normal Form for $\mathbf{wLTL}_\#$ Formulae

Let  $\Phi$  be a  $\mathbf{wLTL}_\#$  formula and assume without loss of generality explicit quantification of all used bookmarks. For technical reasons, assume further that  $\Phi$  has the form  $\Phi = \varphi \wedge \bigwedge_{c \in C_\varphi} c = 0$ . That is, it explicitly specifies that each used system counter is zero initially. Since this is assumed in the semantics anyway, such constraints can be added if necessary without affecting the statement of the formula.

The first step is to normalise the bookmarks occurring in the formula such that each corresponds uniquely to some scope and condition to be counted. For example, the formula

$$x. \mathbf{F} y. \mathbf{X} \mathbf{F} 2\#_y(p) - \#_x(p) + \#_x(q) \geq 0$$

is equivalently reformulated as

$$\hat{z}.z. \mathbf{F} y. \mathbf{X} \mathbf{F} (2 \cdot \#_y(p) - \#_{\hat{z}}(p) + \#_z(q) \geq 0)$$

where the variable  $z$  is only referenced by the term  $\#_z(q)$  and a fresh variable  $\hat{z}$  corresponds to  $\#_{\hat{z}}(p)$ . Technically, shadowed bookmarks are renamed to be unique or removed if unused and every subformula of the form  $z.\varphi$  that contains two terms  $\tau_1 = \#_z(\psi_1)$  and  $\tau_2 = \#_z(\psi_2)$  (not necessarily different) is replaced by  $\hat{z}.z.\hat{\varphi}$  for a fresh bookmark  $\hat{z}$  that is not yet in use. In  $\hat{\varphi}$ , (one occurrence of) the term  $\tau_1$  is replaced by  $\#_{\hat{z}}(\psi_1)$  and this substitution is repeated until there is a unique bookmark for every summand of every term.

Second, every plain LTL subformula  $\varphi \in \text{sub}(\Phi) \cap \text{LTL}$  is translated to *negation normal form* where negation only occurs in front of atomic propositions and counter guards. To

this end, let the *dual*  $\overline{\varphi}$  of a formula  $\varphi \in \text{LTL}$  be defined recursively by

$$\begin{array}{ll} \overline{\varphi \wedge \psi} := \overline{\varphi} \vee \overline{\psi} & \overline{p} := \neg p \\ \overline{\varphi \vee \psi} := \overline{\varphi} \wedge \overline{\psi} & \overline{\neg \varphi} := \varphi \\ \overline{\varphi \mathbf{U} \psi} := \overline{\varphi} \mathbf{R} \overline{\psi} & \overline{\mathbf{X} \varphi} := \mathbf{X} \overline{\varphi} \\ \overline{\varphi \mathbf{R} \psi} := \overline{\varphi} \mathbf{U} \overline{\psi} & \overline{\overline{\gamma}} := \neg \gamma \end{array}$$

for propositions  $p \in AP$ , guards  $\gamma \in \text{Grd}(C)$  over system counters, and formulae  $\varphi, \psi \in \text{LTL}$ . The LTL subformulae of the form  $\neg\varphi$ , where  $\varphi \notin AP \cup \text{Grd}(C)$  is not atomic, are replaced by the equivalent formula  $\overline{\varphi}$ . The procedure is applied recursively as long as necessary to establish negation normal form. Notice that  $\text{wLTL}_\#$  admits negation inside plain LTL formulae and therefore the resulting formulae are still in the fragment. However, to maintain negation normal form, the derived operators  $\mathbf{R}$  and  $\vee$  must be treated as core operators and not be replaced by their definition.

Let us assume henceforth, that the formula  $\Phi$  is given in such a normal form. Let  $B_\gamma$  denote for each counting constraint  $\gamma = (\tau \geq b) \in \text{sub}(\Phi) \setminus \text{Grd}(C)$  the set of bookmarks that occur in  $\gamma$ . Note that these sets are disjoint due to the normal form. Further, let  $b_\gamma := b$  denote the absolute part of the inequation,  $\psi_x$  the formula corresponding to  $x \in B_\gamma$  and  $a_x$  the corresponding coefficient. That is, every counting constraint has the form

$$\gamma = \left( \sum_{x \in B_\gamma} a_x \cdot \#_x(\psi_x) \right) \geq b_\gamma.$$

► **Remark 3.5.** Laroussinie, Meyer and Pettonnet [LMP12] define a logic they call  $\text{CCTL}^\vee$  where bookmarks (positional variables) are used similarly as in  $\text{CTL}_\#$ . An example for a  $\text{CCTL}^\vee$  formula is  $x[p \wedge \mathbf{X}q].y[r].\mathbf{F} \#_x + \#_y > 5$  for propositions  $p, q$  and  $r$  as well as variables  $x$  and  $y$ . A similar notation is used by Bouajjani, Echahed and Robbana [BER94] and Bouajjani, Echahed and Habermehl [BEH95]. The formulae to be evaluated are associated explicitly to one specific variable at the point of quantification. This ensures syntactically the uniqueness of scope and condition that the above translation achieves.

### 3.3.2 Constructing $\mathbb{Z}$ -VASS from $\text{wLTL}_\#$ Formulae

For a formula  $\Phi$  in this normal form we construct a  $\mathbb{Z}$ -VASS  $\mathcal{N} = (S, \Delta, \{\Phi\}, \lambda_\emptyset)$  where a state from  $S$  is a set  $s \subseteq \text{sub}_{\text{inf}}(\Phi) \cup \overline{\text{sub}}_{\text{LTL}}(\Phi) \cup B \cup B_{\mathbf{X}}$ . It is comprised of *obligations*

in terms of subformulae

$$\begin{aligned} \text{sub}_{\text{unf}}(\Phi) := & \text{sub}(\Phi) \cup \{\varphi \vee \mathbf{X} \mathbf{F} \varphi \mid \mathbf{F} \varphi \in \text{sub}(\Phi)\} \\ & \cup \{\varphi \vee (\psi \wedge \mathbf{X}(\varphi \mathbf{U} \psi)), \psi \wedge \mathbf{X}(\varphi \mathbf{U} \psi), \mathbf{X}(\varphi \mathbf{U} \psi) \mid \varphi \mathbf{U} \psi \in \text{sub}(\Phi)\} \\ & \cup \{\varphi \wedge (\psi \vee \mathbf{X}(\varphi \mathbf{R} \psi)), \psi \vee \mathbf{X}(\varphi \mathbf{R} \psi), \mathbf{X}(\varphi \mathbf{R} \psi) \mid \varphi \mathbf{R} \psi \in \text{sub}(\Phi)\} \end{aligned}$$

including their temporal unfoldings, duals of LTL subformulae  $\overline{\text{sub}}_{\text{LTL}}(\Phi) := \{\overline{\varphi} \mid \varphi \in \text{sub}_{\text{unf}}(\Phi) \cap \text{LTL}\}$ , as well as bookmarks that have been placed already. The set  $B_{\mathbf{X}} := \{\mathbf{X} x \mid x \in B\}$  consists essentially of a copy of the bookmarks from  $B$  marked as *deferred*.

The system counters  $C_{\mathcal{N}}$  of  $\mathcal{N}$  should not be confused with those being potentially used in  $\Phi$ . Rather, for the construction only unrelated counters  $c \notin C$  will be used that cannot occur in any formula. The system features exactly one counter  $c_{\gamma} \in C_{\mathcal{N}}$  for each counting constraint  $\gamma = \tau \geq b \in \text{sub}(\Phi) \setminus \text{Grd}(C)$  in  $\Phi$ . Note that the normal form guarantees that no such subformula occurs twice in  $\Phi$ .

Before formally defining the transition relation, let us illustrate how the constructed system is supposed to operate by means of an example.

► **Example 3.6.** Consider the  $w\text{LTL}_{\#}$  formula

$$\Phi = x.y. \mathbf{F} \varphi = x.y. \mathbf{F} (\#_x(q) \geq 5 \vee z. \mathbf{F} 3\#_z(p) - 2\#_y(p) \geq 0)$$

in normal form. The system we construct can choose to decompose a formula present in a state, splitting conjunctions into the two parts, selecting non-deterministically one part of a disjunction or stripping off bookmarks. Starting from the initial state  $\{\Phi\}$  the system can only perform the latter action resulting in the state  $\{x, y. \mathbf{F} \varphi\}$  and subsequently in  $\{x, y, \mathbf{F} \varphi\}$ . The temporal operators other than  $\mathbf{X}$  are replaced by their fixed-point unfolding. Here we use  $\mathbf{F} \varphi \equiv \varphi \vee \mathbf{X} \mathbf{F} \varphi$ . The system may choose the left part ( $\varphi$ ) of the conjunction but we assume the right-hand part ( $\mathbf{X} \mathbf{F} \varphi$ ) to be selected here, leading to the state  $\{x, y, \mathbf{X} \mathbf{F} \varphi\}$ .

The formula  $\Phi$  contains two counting constraints  $\gamma_1 = (\#_x(q) \geq 5)$  and  $\gamma_2 = (3\#_z(p) - 2\#_y(p) \geq 0)$  and therefore  $\mathcal{N}$  uses two counters  $c_1$  and  $c_2$  corresponding to  $\gamma_1$  and  $\gamma_2$ , respectively. They serve for tracking the value of the corresponding terms  $\#_x(q)$  and  $3\#_z(p) - 2\#_y(p)$ , respectively. In the current state  $\{x, y, \mathbf{X} \mathbf{F} \varphi\}$  the bookmark  $y \in B_{\gamma_2}$  has been placed. If  $\psi_y = p$  holds, we want to update the corresponding counter  $c_2$  by the

factor  $a_y = -2$ . Let thus  $\mu_y$  denote the update function with

$$\mu_y(c) = \begin{cases} -2 & \text{if } c = c_2 \\ 0 & \text{otherwise.} \end{cases}$$

As soon as a bookmark occurs in a state, the system has a non-deterministic choice of whether the associated formula is supposed to hold or not. The consequences of the choice, e.g. for  $y$ , is implemented by the two applicable transitions

$$\{x, y, \mathbf{X F} \varphi\} \xrightarrow{\mu_y} \{x, \mathbf{X} y, p, \mathbf{X F} \varphi\} \quad \text{and} \quad \{x, y, \mathbf{X F} \varphi\} \rightarrow \{x, \mathbf{X} y, \neg p, \mathbf{X F} \varphi\}$$

that defer  $y$  and record the choice in terms of the obligation  $p$  or its dual  $\neg p$ . In the former (positive) case, the update  $\mu_y$  is applied effectively adding  $\mu_y(c_2) = a_y = -2$  to the counter  $c_2$  tracking the value of  $\gamma_2$ .

If all elements of a state are either atomic (propositions or counter guards), a negated proposition, or starting with  $\mathbf{X}$  (formulae or deferred bookmarks), a transition can be applied that simulates an actual temporal step by stripping all  $\mathbf{X}$  prefixes and removing the propositions and counter guards. The only condition is that the removed obligations are not contradictory, that is, no proposition is contained in both positive and negative form and the conjunction of the counter guards is satisfiable. The latter consists only of guards that occur in  $\Phi$  and thus the conjunction is a **qfPA** formula not larger than  $\Phi$ . Checking it can be done non-deterministically in polynomial time.

The successor of the state  $\{\mathbf{X} x, \mathbf{X} y, p, \mathbf{X F} \varphi\}$  is  $\{x, y, \mathbf{F} \varphi\}$ , thus closing a cycle in the system.

Finally, bookmarks may be removed from a state along with their corresponding term. Once the system arrives at, e.g., the state  $\{x, \mathbf{X} y, \mathbf{X} z, 3\#_z(p) - 2\#_y(p) \geq 0\}$ , the counting constraint  $\gamma_2$  can be removed if all associated bookmarks from  $B_{\gamma_2} = \{y, z\}$  are deferred and removed at the same time. This results here in the state  $\{x\}$  containing no obligation anymore. Notice that counter  $c_2$  holds at this point precisely the value of term  $3\#_z(p) - 2\#_y(p)$  evaluated from the placement of the bookmarks  $y$  and  $z$  until the point the system chooses to select  $\varphi$  instead of  $\mathbf{X F} \varphi$  and thus to evaluate the constraint  $\gamma_2$  referring to  $x$  and  $y$ . Therefore, if the state is reachable with counter  $c_2$  being larger or equal to 0, then the counting constraint was actually satisfied at the time it was removed. Once the bookmarks  $y$  and  $z$  are removed from the state, the corresponding counters are not modified anymore by the system.

The complete set of transitions of the  $\mathbb{Z}$ -VASS  $\mathcal{N}$  is given by the rules

$$\begin{array}{ll}
 M \cup \{\varphi \wedge \psi\} \rightarrow M \cup \{\varphi, \psi\} & M \cup \{\mathbf{F} \varphi\} \rightarrow M \cup \{\varphi \vee \mathbf{X} \mathbf{F} \varphi\} \\
 M \cup \{\varphi \vee \psi\} \rightarrow M \cup \{\varphi\} & M \cup \{\varphi \mathbf{U} \psi\} \rightarrow M \cup \{\psi \vee (\varphi \wedge \mathbf{X} \varphi \mathbf{U} \psi)\} \\
 M \cup \{\varphi \vee \psi\} \rightarrow M \cup \{\psi\} & M \cup \{\psi \mathbf{R} \varphi\} \rightarrow M \cup \{\varphi \wedge (\psi \vee \mathbf{X} \psi \mathbf{R} \varphi)\} \\
 M \cup \{x.\varphi\} \rightarrow M \cup \{x, \varphi\} & M \cup \{x\} \xrightarrow{\mu_x} (M \setminus \{x\}) \cup \{\mathbf{X} x, \psi_x\} \\
 M \cup \{x\} \rightarrow M & M \cup \{x\} \rightarrow M \cup \{\mathbf{X} x, \overline{\psi_x}\} \\
 M \cup \hat{B}_\gamma \cup \{\gamma\} \xrightarrow{\alpha_\gamma} M \setminus \hat{B}_\gamma
 \end{array}$$

for (subsets of) states  $M$ , formulae  $\varphi$  and  $\psi$ , counting constraints of the form

$$\gamma = \sum_{x \in B_\gamma} a_x \#_x(\psi_x) \geq b_\gamma$$

and sets  $\hat{B}_\gamma = \{\mathbf{X} x \mid x \in B_\gamma\}$  consisting of precisely the variables (in deferred form) that are used in the constraint  $\gamma$ . The update functions  $\mu_x$  and  $\alpha_\gamma$  are defined for bookmarks  $x \in B$  as

$$\mu_x(c) = \begin{cases} a_x & \text{if } c = c_\gamma \text{ and } x \in B_\gamma \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \alpha_\gamma(c) = \begin{cases} b_\gamma & \text{if } c = c_\gamma \\ 0 & \text{otherwise.} \end{cases}$$

Further, the system  $\mathcal{N}$  admits transitions according to the rule

$$\left\{ \begin{array}{l} \mathbf{X} \varphi_1, \dots, \mathbf{X} \varphi_n, \mathbf{X} z_1, \dots, \mathbf{X} z_m, \\ \chi_1, \dots, \chi_k, \gamma_1, \dots, \gamma_\ell \end{array} \right\} \rightarrow \{\varphi_1, \dots, \varphi_n, z_1, \dots, z_m\} \quad (3.1)$$

for any  $n, m, k \geq 0$ ,  $z_1, \dots, z_m \in B$ ,  $\chi_1, \dots, \chi_k \in \{p, \neg p \mid p \in AP\}$ ,  $\gamma_1, \dots, \gamma_\ell \in \{\gamma, \neg\gamma \mid \gamma \in \text{Grd}(C)\}$  under the condition that the non-temporal part is satisfiable: the transition rule applies only if  $\chi_i \neq \overline{\chi_j}$  for any  $i, j \in [1, k]$ , i.e., no proposition occurs in both positive and negative form, and the conjunction  $\bigwedge_{i=1}^k \chi_i$  of all counter guards admits a satisfying solution.

Notice that the construction is similar to the tableaux approach to LTL satisfiability used by Lichtenstein and Pnueli [LP85], although the formula sets are not required to be maximal or even consistent here. Intuitively, the transition relation is more fine-grained and performs the decomposition of each formula individually, allowing also for treating their potential effect on the counters separately. Only rule (3.1) corresponds to advancing

one temporal step, and the necessary sanity checks are associated to it.

To complete the construction, it remains to identify the target configurations that are supposed to be coverable in the thus obtained system  $\mathcal{N}$ . A *target state* is a state  $s$  of  $\mathcal{N}$  where  $\hat{s} = s \setminus (B \cup B_{\mathbf{X}}) \subseteq \text{LTL}$  contains only plain LTL formulae and their conjunction  $\bigwedge_{\varphi \in \hat{s}} \varphi$  is satisfiable. Notice that this is again an LTL formula (and at most polynomially larger than  $\Phi$ ) for which satisfiability is decidable in PSPACE. Now, there is a target state  $s$  such that the configuration  $(s, \mathbf{0})$  is coverable from the initial configuration  $(\{\Phi\}, \mathbf{0})$ , if and only if  $\Phi$  is satisfiable. As mentioned before, this problem is decidable in NP.

In summary, the construction can be understood as a non-deterministic algorithm that first constructs the (reachable part of) the transition graph of  $\mathcal{N}$ , starting at state  $\{\Phi\}$ . Concerning rule (3.1), it guesses a (polynomial) solution for the conjunction of constraints and selects the transition if that succeeds. Finally, it guesses a target state and checks whether  $s$  represents a satisfiable LTL formula and whether  $(s, \mathbf{0})$  is coverable. Both is possible using space bounded polynomially in the size of the constructed system, which is potentially exponential in the size of  $\Phi$ . The whole non-deterministic procedure can thus be performed in exponential time with respect to the formula size. If successful, it provides a finite path from the initial to the target state that can be projected to a sequence  $u$  over  $2^{AP} \times \mathbb{Z}^C$  by selecting only the states on the path where each position corresponds to an application of rule (3.1). At each position  $i$  on  $u$ , let  $u(i) = (a, \theta)$  consist of the letter  $a$  containing precisely the positive propositions when applying rule (3.1) and  $\theta$  some solution to the (satisfiable) conjunction of the counter guards.

Upon reaching the target state, all obligations that involve position counting and bookmarks have been satisfied and all remaining obligations form a satisfiable LTL formula. Hence, for the letter, there is a flat counter system  $\mathcal{W}$  satisfying it. The sequence  $u$  defines a counter system  $\mathcal{U}$  containing one state  $s_i$  for each position of  $u$  labelled by the letter  $a_i$  and counter updates between  $s_i$  and  $s_{i+1}$  defined by  $\theta_{i+1} - \theta_i$  (cf. Figure 3.1). The last state of  $\mathcal{U}$  is then linked to the initial state of  $\mathcal{W}$  setting all counters to zero by means of the update  $-\theta_{|u|-1}$ . The combined structure has a run of the form  $uw$  that traverses  $\mathcal{U}$  and then  $\mathcal{W}$  and satisfies  $\Phi$ . This construction proves Theorem 3.4.

### 3.3.3 Application to Model Checking

Moreover, the construction can be used to solve the model-checking problem of  $\text{wLTL}_{\#}$  for a Kripke structure  $\mathcal{K} = (Q, \Delta_{\mathcal{K}}, q_I, \lambda_{\mathcal{K}})$  by means of a product construction that synchronises the transitions of the model  $\mathcal{K}$  with the transitions given by rule (3.1) corresponding to temporal steps. Technically, assuming that  $\mathcal{N} = (S, \Delta, \{\Phi\}, \lambda)$  is the

constructed  $\mathbb{Z}$ -VASS and  $\Delta_{\mathbf{X}} \subseteq \Delta$  consists of the transitions defined by rule (3.1), the transition relation of the product is comprised of transitions

- $((s, q), \mu, \emptyset, (s', q))$  (not advancing  $\mathcal{K}$ ) where  $(s, \mu, \emptyset, s') \in \Delta \setminus \Delta_{\mathbf{X}}$  and
- $((s, q), \mu, \emptyset, (s', q'))$  where  $(q, \mathbf{0}, \emptyset, q') \in \Delta_{\mathcal{K}}$ ,  $(s, \mu, \emptyset, s') \in \Delta_{\mathbf{X}}$  and where the states  $s$  are compatible with the labelling of  $q$ , i.e., for all propositions  $p \in AP$  we have  $p \in s \Rightarrow p \in \lambda_{\mathcal{K}}(q)$  and  $\neg p \in s \Rightarrow p \notin \lambda(q)$ .

Then, a target state is identified as a tuple state  $(s, q) \in S \times Q$  where  $s = \{\varphi_1, \dots, \varphi_n\}$  consists only of LTL formulae  $\varphi_1, \dots, \varphi_n$  and, instead of being just satisfiable, they have to hold upon reaching  $q$  in  $\mathcal{K}$ . Since  $\mathcal{K}$  is a Kripke structure, the satisfaction of the formula  $\varphi_s := \varphi_1 \wedge \dots \wedge \varphi_n$  is independent of the history, i.e., the sequence of states traversed before and thus the run can be extended to satisfy  $\varphi_s$  if and only if  $\mathcal{K}' \models \varphi_s$  where  $\mathcal{K}'$  is  $\mathcal{K}$  with initial state  $q$ . As discussed above (Corollary 3.2), this check can be performed in polynomial space and therefore we conclude that  $\text{MC}(\text{wLTL}_{\#}, \text{KS}) \in \text{NEXP}$ .

► **Corollary 3.7.** *The problem  $\text{MC}(\text{wLTL}_{\#}, \text{KS})$  is in NEXP.*

## Model-checking cLTL over Flat Counter Systems

The presence of counting operators and constraints, even those that are essentially restricted such as the relative frequencies expressible in fLTL and consequently cLTL, confront us immediately with undecidable decision problems. We have seen this for the satisfiability problems (Theorem 3.3) which leads us straight-forward to the same conclusion for the model-checking problems, even over the finite state spaces of Kripke structures.

► **Theorem 4.1.** *The model-checking problem of all supersets of cLTL is undecidable over Kripke structures and counter systems.*

**Proof.** Consider the universal Kripke structure  $\mathcal{K}$  over  $\Sigma = 2^{AP}$ , i.e., representing the language  $\Sigma^\omega$  (cf. Figure 4.1). For any Kripke structure  $\mathcal{K}'$  we have  $\llbracket \mathcal{K}' \rrbracket^\Sigma \subseteq \llbracket \mathcal{K} \rrbracket^\Sigma$  and thus  $\mathcal{K}' \models \varphi$  implies  $\mathcal{K} \models \varphi$  for any cLTL formula  $\varphi$  (recall that the model relation is defined existentially). Thus,  $\varphi \in \text{SAT}(\text{cLTL}, \text{KS})$  if and only if  $\mathcal{K} \models \varphi$ . The problem trivially reduces to model-checking any superset of cLTL both over KS and CS. ■

This reveals that there are two independent sources of undecidability in the family of logics we investigate. On one hand, the counting mechanisms of the logics are generally powerful enough to encode undecidable problems. On the other hand, even without any counting constraints, temporal logic intentionally subsumes (control-state) reachability [Pnu77; EC80] which is itself undecidable in counter systems [Min67].

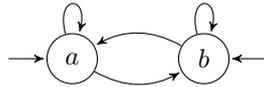


Figure 4.1: A Kripke structure  $\mathcal{K}$  labelled by  $\Sigma = \{a, b\}$  with  $\llbracket \mathcal{K} \rrbracket^\Sigma = \Sigma^\omega$ .

In that light the question arises whether counting extensions are worthwhile to be considered in the context of automatic verification or whether such mechanisms are simply too powerful and the goal therefore too ambitious. A common approach to recover decidability of verification problems is to restrict the specification language just as far as is necessary. This has the potential of providing insights on which aspects particularly harden the problem in question and whether it can be avoided. In the family of formalisms that we aim to investigate, the combination of  $cLTL$  and finite-state systems represents already one of the most restricted cases. Recall that the positive result for  $wLTL_{\#}$  (Theorem 3.4) was obtained by sacrificing symmetric duality of temporal logic and greatest fixed points.

To avoid such restrictions, let us investigate an alternative compromise pertaining to the model side: the special case of structures that are *flat*. Recall that flatness demands, essentially, that cycles of the system cannot be alternated during an execution. It is thus a strong restriction that can, however, be used to under-approximate arbitrary systems and thereby provide a realistic application domain. We will come back to this application later in Chapter 5. The most general members of the counting logic family,  $LTL_{\#}$ ,  $CTL_{\#}$  and  $CTL_{\#}^*$ , will be approached by means of a different technique in Chapter 7. The aim of this chapter is to develop a model-checking procedure specifically for the logic  $cLTL$  and thereby prove the following result. It is based on and extends the developments presented in [Dec+17].

► **Theorem 4.2.** *The problem  $MC(cLTL, FCS)$  is in  $NEXP$ .*

## 4.1 Towards A Guess-and-Check Procedure

The decision procedure follows a *guess-and-check* scheme: To solve the decision problem, a non-deterministic algorithm selects a witness object supposed to demonstrate a positive result and subsequently checks the credibility of that witness. In order to be feasible, this scheme relies on the following aspects to be established.

- **Representation.** The potentially witnessing objects need to have a finite, syntactically enumerable representation, in order to be selected (or searched for).
- **Analysis.** Given such an object, it must be decidable whether it actually certifies the positive result.
- **Soundness.** The existence of any object (representation) that passes the check must imply a positive result.

- **Completeness.** There can only be a finite number of candidates and if the result is positive, then at least one of them must pass the check.

The necessary and sufficient condition for a given flat counter system  $\mathcal{S}$  to satisfy a cLTL formula  $\Phi$  is that  $\mathcal{S}$  admits a satisfying run. However, the object that will be used to certify that such a run exists, is a (non-empty) *set* of runs that satisfy  $\Phi$ . Clearly, this is not a restriction since such a set may have a single element. As representation for such sets, Section 4.2 will introduce *augmented path schemas* that extend the concept of path schemas [LS04; DDS15]. Augmented path schemas are, essentially, a class of degenerated flat counter systems that provide for each of its runs a labelling to indicate which formulae are satisfied. We are going to define a notion of *consistency* as a syntactic property vouching for the correctness of the labelling and thus the credibility of the structure as witness. This gives rise to the following non-deterministic procedure to decide the problem  $\text{MC}(\text{cLTL}, \text{FCS})$ .

1. **Read as input** a counter system  $\mathcal{S}$  and a cLTL formula  $\Phi$ .
2. **Guess** an augmented path schema  $\mathcal{P}$  in  $\mathcal{S}$  of at most exponential size.
3. **Terminate** if  $\mathcal{P}$  is consistent, labels its runs initially by  $\Phi$ , and is non-empty.

It is shown in Section 4.3 that if an augmented path schema is consistent, then the associated runs satisfy precisely the formulae they are labelled with. This provides the essential soundness argument. Finally, a construction is developed in Sections 4.4 and 4.5 that starts on an arbitrary run satisfying  $\Phi$  and yields a consistent schema of at most exponential size containing it. Thereby, completeness of the procedure is established.

For the remainder of this chapter let us fix the counter system  $\mathcal{S} = (S_{\mathcal{S}}, \Delta_{\mathcal{S}}, s_I, \lambda_{\mathcal{S}})$  as well as the formula  $\Phi \in \text{cLTL}$  in order to avoid additional explicit parameters in the notation.

## 4.2 Augmented Path Schemas

In the following we introduce a representation of (subsets of) paths in  $\mathcal{S}$  called *augmented path schemas (APS)*. They serve as finitely representable symbolic witnesses for proving that the counter system  $\mathcal{S}$  satisfies the formula  $\Phi$ . Therefore, they are essential for the decision procedure and provide also the basis for the encoding of the model-checking problem into quantifier-free Presburger arithmetic and henceforth the verification schema based on flat under-approximations presented in Chapter 5.

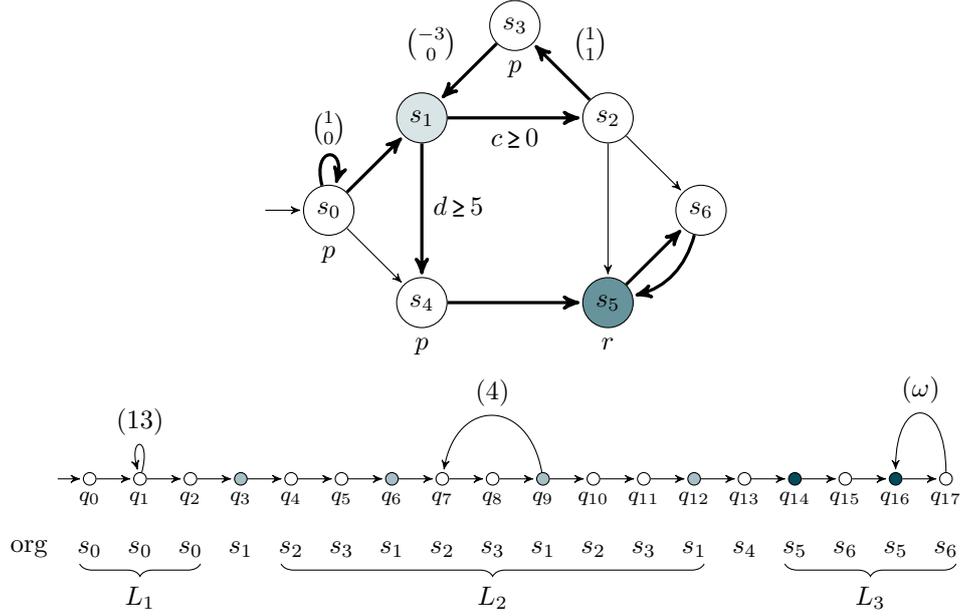


Figure 4.2: A flat counter system  $\mathcal{S}$  (top) using propositions  $p, r \in AP$  and counters  $c, d \in C$ , and (a sketch of) an augmented path schema  $\mathcal{P}$  (bottom) with states  $Q = \{q_0, \dots, q_{17}\}$  representing runs of  $\mathcal{S}$  that traverse the loops  $L_1 = s_0$ ,  $L_2 = s_2s_3s_1$ , and finally  $L_3 = s_5s_6$ . The numbers above the loops of  $\mathcal{P}$  uniquely determine one specific run of  $\mathcal{S}$ .

Path schemas [LS04] represent a subset of valid runs of a counter system with a common shape. More precisely, they represent a (connected) sequence  $u_0v_0u_1v_1 \dots u_nv_n$  of (not necessarily simple) paths  $u_i$  and loops  $v_i$  of a system and thereby all the runs  $\rho$  of  $\mathcal{S}$  that traverse a state sequence of the form  $st(\rho) = u_0v_0^{\ell_1} \dots u_{n-1}v_{n-1}^{\ell_{n-1}}u_nv_n^\omega$ . The augmented variant of path schemas essentially allows for attaching an additional labelling that provides information not directly accessible in the original system. Moreover, additional counters and constraints can be added to precisely refine the set of runs represented by an APS.

► **Example 4.3.** Figure 4.2 depicts an example of the flat counter system  $\mathcal{S}$  and an augmented path schema  $\mathcal{P}$  with states  $Q = \{q_0, \dots, q_{17}\}$ . The latter is itself a flat counter system (of a specific degenerated shape) of which each state  $q \in Q$  corresponds to some origin state  $\text{org}(q)$  in  $\mathcal{S}$ , e.g., the three first (leftmost) states of  $\mathcal{P}$  correspond to  $\text{org}(q_0) = \text{org}(q_1) = \text{org}(q_2) = \text{org}(q_3) = s_0$ . The same applies to the transitions in a consistent fashion, e.g., the transitions from  $q_0$  to  $q_1$  and from  $q_1$  to  $q_1$  correspond to the transition between their origins, that is from  $s_0$  to  $s_0$ . Thus, every run  $\rho \in \text{runs}(\mathcal{P})$  represents a

run  $\text{org}(\rho)$  in  $\mathcal{S}$  by projecting the states to their origins. In fact,  $\mathcal{P}$  represents a subset of runs of  $\mathcal{S}$  that only use the highlighted transitions. Notice, however, that the schema  $\mathcal{P}$  features unfoldings of each loop, i.e., states that explicitly represent the first or last iterations. The shape of  $\mathcal{P}$  thus imposes that only those runs of  $\mathcal{S}$  are represented that traverse the loops  $L_1$  and  $L_2$  at least three times. To represent a run with less iterations, e.g., of  $L_2$ , a different schema would have to be chosen, such as  $\mathcal{P}$  with the second loop cut out. Finally, observe that the shape of  $\mathcal{P}$  admits also a concise representation of one specific run by providing a concrete number of iterations for each (but the last) loop.

► **Definition 4.4** (Augmented Path Schema). *An augmented path schema (APS) in  $\mathcal{S}$  is a structure  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$  where*

- $(Q, \Delta_{\mathcal{P}}, q_0, \lambda_{\mathcal{P}})$  is a flat counter system over  $Q = \{q_0, \dots, q_n\}$ , for some  $n \in \mathbb{N}$ , with labelling  $\lambda_{\mathcal{P}} : Q \rightarrow 2^{\text{sub}(\Phi) \cup \text{AP}}$  and simple path  $q_0 \dots q_n$ ;
- $\text{org} : Q \rightarrow S_{\mathcal{S}}$  maps every state to an origin such that  $\lambda_{\mathcal{P}}(q) \cap \text{AP} = \lambda_{\mathcal{S}}(\text{org}(q)) \cap \text{AP}$  and  $\text{org}(q_0) = s_I$ ;
- for each transition  $(q, \mu, \Gamma, q') \in \Delta_{\mathcal{P}}$  there is  $(\text{org}(q), \hat{\mu}, \hat{\Gamma}, \text{org}(q')) \in \Delta_{\mathcal{S}}$  with  $\hat{\Gamma} \subseteq \Gamma$  and  $\hat{\mu} \sqsubseteq \mu$ ;
- $\Delta_{\mathcal{P}} = \Delta_{\text{fwd}} \dot{\cup} \Delta_{\text{bwd}}$  is comprised of forward- and backward transitions where
  - $\Delta_{\text{fwd}} = \{(q_0, \mu_0, \Gamma_0, q_1), \dots, (q_{n-1}, \mu_{n-1}, \Gamma_{n-1}, q_n)\}$ ,
  - there is  $(q_n, \mu_n, \Gamma_n, q_{n'}) \in \Delta_{\text{bwd}}$  for some  $n' \leq n$  closing the last loop, and
  - for all  $(q_j, \mu, \Gamma, q_i), (q_k, \mu', \Gamma', q_h) \in \Delta_{\text{bwd}}$  we have  $i \leq j$ ,  $h \leq k$ , and the corresponding loops  $q_h q_{h+1} \dots q_k$  and  $q_i q_{i+1} \dots q_j$  are disjoint; and
- for each loop  $L = q_i q_{i+1} \dots q_{i+\ell}$  there is a front row  $F = q_{i-\ell-1} \dots q_{i-1}$  and, if  $i + \ell < n$ , a rear row  $R = q_{i+\ell+1} \dots q_{i+2\ell+1}$  with  $\lambda_{\mathcal{P}}(F) = \lambda_{\mathcal{P}}(R) = \lambda_{\mathcal{P}}(L)$ .

For an APS  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, q_I, \text{org})$ , the paths of the underlying counter system  $\mathcal{Q} = (Q, \Delta_{\mathcal{P}}, q_I, \lambda_{\mathcal{P}})$  are considered also as the paths of  $\mathcal{P}$ . Let  $\text{lastl}(\mathcal{P})$  denote the last loop of  $\mathcal{P}$  (containing the last state  $q_n$ ). The simple path  $q_0 \dots q_n$  is the only one that spans the entire control graph of  $\mathcal{P}$  and thereby defines an ordering  $\preceq_{\mathcal{P}}$  on  $Q$  with  $q_i \preceq_{\mathcal{P}} q_j$  if and only if  $i \leq j$ .

Observe that the definition requires loops to be preceded and (except for  $\text{lastl}(\mathcal{P})$ ) succeeded by one unfolding. This is due to technical reasons that are discussed later in the course of this chapter. Since each loop can be unfolded this is no restriction and increases the necessary schema size at most by factor three. By  $\text{loops}(\mathcal{P})$  we denote

the set of all simple loops of the form  $L = q_i q_{i+1} \dots q_{i+\ell} \in Q^+$ , i.e. up to rotation. For each such loop, let  $front_{\mathcal{P}}(L) = q_{i-\ell-1} \dots q_{i-1}$  denote the front row of  $L$  and, unless  $L = lastl(\mathcal{P})$ , let  $rear_{\mathcal{P}}(L) = q_{i+1} \dots q_{i+1+\ell}$  be the rear row of  $L$ . We mostly omit the subscript for easier reading if no ambiguity arises.

The runs of  $\mathcal{P}$  are those runs of the underlying counter system  $\mathcal{Q}$  that visit the last state of  $\mathcal{P}$ , that is  $runs(\mathcal{P}) := \{\rho \in runs(\mathcal{Q}) \mid q_n \in st(\rho)\}$ . Nevertheless, we use  $\mathcal{P}$  instead of  $\mathcal{Q}$  when using the model relation and write  $(\mathcal{P}, \rho, i) \models \varphi$  instead of  $(\mathcal{Q}, \rho, i) \models \varphi$  for  $\rho \in runs(\mathcal{P})$ . The origin mapping is lifted to configurations by  $org((q, \theta)) := (org(q), \hat{\theta})$  where  $\hat{\theta}$  is the restriction of  $\theta$  to the domain  $counters(\mathcal{S})$ . Further, it applies point-wise to paths and runs of  $\mathcal{P}$ .

Reachability of the last state of an APS is decidable in NP (Theorem 2.2) and this implies the same for non-emptiness.

► **Lemma 4.5** ([DDS15]). *The non-emptiness problem of augmented path schemas is in NP.*

### 4.2.1 Correct Labelling

The semantics of cLTL formulae is based only on the labelling of a counter system by propositions from  $AP$  while the rest of the labels is not taken into account. The definition of APS demands them only to be faithful with respect to  $\mathcal{S}$  concerning the structure of runs as state sequences. We are, however, interested in APS where the labelling by cLTL formulae coincides with their semantics on the runs of  $\mathcal{P}$  as well as their corresponding runs in  $\mathcal{S}$ .

► **Definition 4.6** (Correct labelling). *Let  $\mathcal{P}$  be an APS in  $\mathcal{S}$  and  $\varphi \in sub(\Phi)$  a cLTL formula. A state  $q$  of  $\mathcal{P}$  is correctly labelled with respect to  $\varphi$  if and only if for all runs  $\rho \in runs(\mathcal{P})$  and all positions  $i \in pos_{\rho}(q)$  of  $q$  on  $\rho$ ,*

$$\varphi \in \lambda_{\mathcal{P}}(q) \quad \Leftrightarrow \quad (\mathcal{P}, \rho, i) \models \varphi$$

*A state  $q$  is correctly labelled with respect to a set  $M \subseteq sub(\Phi)$  if that is the case for each of its elements. The APS  $\mathcal{P}$  is correctly labelled with respect to  $\varphi$  or  $M$ , respectively, if that is the case for all states of  $\mathcal{P}$ .*

Notice that a state  $q$  can only be labelled correctly with respect to some formula  $\varphi$  if it holds, semantically, at all positions on a run where  $q$  occurs or at none of them. With this strict notion of correctness, deciding whether a correctly labelled APS in  $\mathcal{S}$  witnesses

the existence of a run satisfying  $\Phi$  amounts to checking it to be non-empty and verifying that the initial state is labelled by  $\Phi$ .

### 4.2.2 Consistency

While correctness is a semantic property, in the following the syntactic notion called *consistency* is introduced. The intention is to provide a criterion that can be evaluated algorithmically and is strong enough to guarantee semantic properties, more precisely, to imply the correctness of the labelling. On the other hand, it should admit a sufficiently large class of structures to include a representative witness for each satisfied formula.

Consider an APS  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$  in  $\mathcal{S}$ . The essential idea is to identify a syntactic reason for some formula  $\varphi \in \text{sub}(\Phi)$  to either hold or not hold, whenever some state  $q \in Q$  is visited on a valid run of  $\mathcal{P}$ . The simplest case is that of propositions: the labelling by a proposition is always correct because the labelling itself defines the semantics of propositions. The important aspect concerning the labelling by propositions is that it coincides with that in  $\mathcal{S}$ . This is required by Definition 4.4 and since the semantics of any cLTL formula depends only on propositions and the updates of the counters  $C_{\mathcal{S}}$  occurring already in  $\mathcal{S}$ , we can record the following straight-forward, yet crucial observation.

► **Proposition 4.7.** *For every APS  $\mathcal{P}$  in  $\mathcal{S}$ , run  $\rho \in \text{runs}(\mathcal{P})$ , position  $i \in \mathbb{N}$ , and formula  $\varphi \in \text{sub}(\Phi)$*

$$(\mathcal{P}, \rho, i) \models \varphi \quad \Leftrightarrow \quad (\mathcal{S}, \text{org}(\rho), i) \models \varphi.$$

Correctness of Boolean combinations can be verified locally for any state  $q \in Q$  when inductively assuring that  $q$  is labelled correctly by all strict subformulae. For example, a negation  $\neg\varphi$  holds on all runs at all positions of  $q$  if and only if on all runs  $\varphi$  does never hold at  $q$ . The problem of expressing the semantics of constraints on counters of the system can be shifted to the system itself. If all incoming transitions of a state carry a specific guard, it necessarily holds on any valid run when visiting the state and therefore a corresponding formula label can be considered to be correct. Vice versa, if all incoming transitions are guarded by the dual of the constraint, it can by definition not hold at this state on any valid run. Notice that guards may lead to such a semantic conclusion but also to an empty set of runs, in which case the statement would nevertheless hold (although vacuously). Therefore, the emptiness-check in the decision procedure is necessary.

Consider a counted until formula  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi$  and let  $q, q'$  be row states with  $q' \in \text{succ}_{\mathcal{P}}(q)$  where  $q'$  is (correctly) labelled by  $\psi$  and the states in-between are correctly labelled by  $\chi$ . Assume that  $\mathcal{P}$  features a counter  $c_{\tau, q}$  that tracks the value of the term  $\tau$  as a *balance* that starts with zero at  $q$  and is updated according to the local effect a

visited state has on the value of  $\tau$ . If the incoming (forward) transition of  $q'$  is labelled by the guard  $c_{\tau,q} \geq b$ , then  $\varphi$  can be assumed to hold when a valid run visits  $q$ . Dually, if all such states  $q'$  are guarded instead by the dual constraint  $c_{\tau,q} < b$ , then there is no way a valid run could satisfy  $\varphi$  when visiting  $q$ .

Let  $\lambda_{\mathcal{P}}^{\#} : Q^* \rightarrow \mathbb{N}^{sub(\Phi)}$  denote the label accumulation in a multi-set fashion counting the number of occurrences of each label by

$$\lambda_{\mathcal{P}}^{\#}(w) : \varphi \mapsto |\{i \in [0, |w| - 1] \mid \varphi \in \lambda_{\mathcal{P}}(w(i))\}|$$

for all  $w \in Q^*$  and  $\varphi \in sub(\Phi)$ .

► **Definition 4.8** (Balance counter). *Let  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$  be an APS in  $\mathcal{S}$ ,  $\tau$  a constraint term over  $sub(\Phi)$ , and  $q \in Q$  a row state in  $\mathcal{P}$ . A balance counter for  $\tau$  and  $q$  in  $\mathcal{P}$  is a counter  $c_{\tau,q} \in \text{counters}(\mathcal{P})$  that is updated, on all transition  $(q_1, \mu, \Gamma, q_2) \in \Delta_{\mathcal{P}}$ , by*

$$\mu(c_{\tau,q}) = \begin{cases} 0 & \text{if } q_1 \prec_{\mathcal{P}} q \\ \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(q_1)) & \text{if } q_1 \succeq_{\mathcal{P}} q. \end{cases}$$

Notice that  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(q))$  denotes the evaluation of  $\tau$  with respect to the labelling set of  $q$  (cf. Section 2.1.2), i.e., a formula in  $\tau$  is substituted by 1 if  $q$  is labelled by it and otherwise by 0. Let us record the formal correspondence of the value of a balance counter and the semantic evaluation of counting constraints in terms of the following lemma. Let  $\#_{i,j}^{\mathcal{P};\rho} : sub(\Phi) \rightarrow \mathbb{N}$  denote the semantic counting function mapping any formula to the number

$$\#_{i,j}^{\mathcal{P};\rho}(\varphi) := |\{k \in [i, j] \mid (\mathcal{P}, \rho, k) \models \varphi\}|$$

of positions it is satisfied on a run  $\rho \in \text{runs}(\mathcal{P})$  in the interval  $[i, j]$ .

► **Lemma 4.9.** *Let  $\tau$  be a constraint term over  $sub(\Phi)$ ,  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$  be an APS correctly labelled with respect to  $sub(\tau)$ , and  $\rho \in \text{runs}(\mathcal{P})$ . Let  $c_{\tau,q}$  be a balance counter in  $\mathcal{P}$  for  $\tau$  and a row state  $q \in Q$  occurring at  $i \in \text{pos}_{\rho}(q)$ . Then, for all positions  $j \geq i$ ,*

$$\text{val}(\rho(j+1))(c_{\tau,q}) = \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\rho(i) \dots \rho(j))) = \llbracket \tau \rrbracket(\#_{i,j}^{\mathcal{P};\rho}).$$

**Proof.** Due to correctness, counting the presence of formula  $\varphi \in sub(\tau)$  as label at some

position  $k$  on  $\rho$  is equivalent to evaluating it, hence  $\#_{k,k}^{\mathcal{P},\rho}(\varphi) = \lambda_{\mathcal{P}}^{\#}(\rho(k))$ . Thus,

$$\begin{aligned} \#_{i,j}^{\mathcal{P},\rho}(\varphi) &= \#_{i,i}^{\mathcal{P},\rho}(\varphi) + \#_{i+1,i+1}^{\mathcal{P},\rho}(\varphi) + \cdots + \#_{j,j}^{\mathcal{P},\rho}(\varphi) \\ &= \lambda_{\mathcal{P}}^{\#}(\rho(i))(\varphi) + \lambda_{\mathcal{P}}^{\#}(\rho(i+1))(\varphi) + \cdots + \lambda_{\mathcal{P}}^{\#}(\rho(j))(\varphi) \\ &= \lambda_{\mathcal{P}}^{\#}(\rho(i) \dots \rho(j))(\varphi). \end{aligned}$$

Since  $\llbracket \tau \rrbracket(\#_{i,j}^{\mathcal{P},\rho})$  only depends on the values  $\#_{i,j}^{\mathcal{P},\rho}$  assigns to formulae from  $sub(\varphi)$  it follows that  $\llbracket \tau \rrbracket(\#_{i,j}^{\mathcal{P},\rho}) = \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\rho(i) \dots \rho(j)))$ .

By the definition of balance counters, all updates of  $c_{\tau,q}$  before the first and only occurrence of  $q$  at position  $i$  are zero and thus  $val(\rho(i))(c_{\tau,q}) = 0$ . Further, for all positions  $k \geq i$ , the updates of  $c_{\tau,q}$  are defined in terms of the local labelling and therefore

$$val(\rho(j+1))(c_{\tau,q}) = val(\rho(i))(c_{\tau,q}) + \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\rho(i))) + \cdots + \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\rho(j))) = \llbracket \tau \rrbracket(\#_{i,j}^{\mathcal{P},\rho}).$$

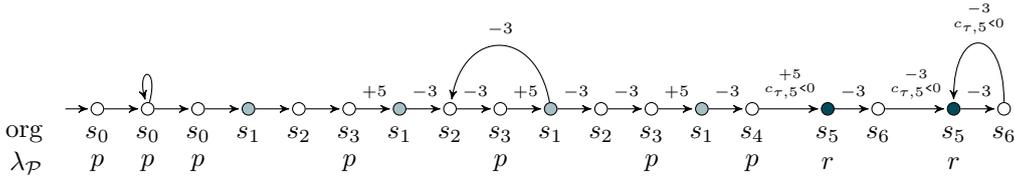
■

In combination with appropriately guarded states, balance counters allow us to reason syntactically about the satisfaction of  $\varphi$ , as long as the labelling is correct with respect to its strict subformulae. Such counters are particularly useful to track the value of a term across an entire loop, even if some runs of  $\mathcal{P}$  iterate it more often than others. For states that are not separated by entire loops such a counter is not necessary, because the effect of simple paths on a term  $\tau$  can be evaluated statically.

► **Example 4.10.** Consider again the counter system  $\mathcal{S}$  and the schema  $\mathcal{P}$  of Example 4.3 and Figure 4.2 as well as a cLTL formula  $\varphi = \neg(p \wedge r) \mathbf{U}_{[5p-3-\neg p \geq 0]} r$  employing a counting constraint  $5p - 3 - \neg p \geq 0$ . The formula states that  $r$  must be reached while observing  $p$  sufficiently often. Notice that the loop  $L_2$  is bad with respect to this constraint since traversing it has a negative effect ( $-1$ ) on the value of the constraint term  $\tau = 5p - 3 - \neg p$ .

Figure 4.3 shows the path schema  $\mathcal{P}$  with updates and guards of a balance counter  $c_{\tau,5}$  for  $\tau$  and its 6th state  $q_5$ . The first (left-most) guard enforces that the second loop must be iterated at least seven times. The updates and guards for counters  $c$  and  $d$  originating from the system  $\mathcal{S}$  impose further implications on how often the first loop has to be taken.

The states  $q_{14}$  and  $q_{16}$  are the only states potentially witnessing the satisfaction of  $\varphi$  at state  $q_5$  since no other is labelled by  $r$ . Since both are guarded by  $c_{\tau,5} < 0$ , no valid run can possibly satisfy  $\varphi$  when visiting state  $q_5$ . It is thus correct to not label the state by  $\varphi$  and, consequently, to label it by  $\neg\varphi$ . Notice that the guards in fact exclude some runs that would otherwise violate the correctness of this labelling.


 Figure 4.3: Sketch of the APS  $\mathcal{P}$  from Figure 4.2 and a balance counter (cf. Example 4.10).

Concerning the satisfaction of  $\varphi$  on loops, an alternative reasoning can be applied: If  $\varphi$  holds at the first occurrence of a state  $q$  on a run and at the last, the formula holds at all occurrences of  $q$  in-between. The reason is, essentially, that the effect of one iteration of a loop on the value of the term  $\tau$  is always the same and thus the worst (i.e., smallest) value of  $\tau$  is encountered either in the first or the last iteration. In APS, each loop is preceded and succeeded by exact copies (unfoldings) and hence, if these are correctly labelled with respect to  $\varphi$ , then the loop labelling inherits their correctness. We will come back to this in Section 4.3. The following definition provides a formalisation of the reasoning sketched above, including further edge cases.

► **Definition 4.11** (Consistency). *Let  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$  be an APS in  $\mathcal{S}$  with  $|Q| = n$ , simple path  $q_0 \dots q_{n-1}$ , and  $\varphi$  a cLTL formula. A state  $q_i \in Q$  is  $\varphi$ -consistent if  $\varphi \in AP$  is a proposition or*

(A)  $\varphi = (\tau \geq b) \in \text{Grad}(C)$ , all incoming transitions  $(q, \mu, \Gamma, q_i) \in \Delta_{\mathcal{P}}$  are guarded by  $\varphi \in \Gamma$  if  $\varphi \in \lambda_{\mathcal{P}}(q_i)$  and by  $\bar{\varphi} \in \Gamma$  otherwise, and if  $i = 0$ , then  $\varphi \in \lambda_{\mathcal{P}}(q_i) \Leftrightarrow \llbracket \tau \rrbracket(\mathbf{0}) \geq b$ .

For non-atomic formulae  $\varphi$ , the state  $q_i$  is  $\varphi$ -consistent if for all  $\psi \in \text{sub}(\varphi) \setminus \{\varphi\}$  all states  $q \in Q$  are  $\psi$ -consistent and one of the following conditions B to D applies.

(B)  $\varphi = \chi \wedge \psi$  and  $\varphi \in \lambda_{\mathcal{P}}(q_i) \Leftrightarrow \chi, \psi \in \lambda_{\mathcal{P}}(q_i)$ ; or  $\varphi = \neg\psi$  and  $\neg\psi \in \lambda_{\mathcal{P}}(q_i) \Leftrightarrow \psi \notin \lambda_{\mathcal{P}}(q_i)$ .

(C)  $\varphi = \mathbf{X}\psi$  and  $\mathbf{X}\psi \in \lambda_{\mathcal{P}}(q_i) \Leftrightarrow \psi \in \lambda_{\mathcal{P}}(q)$ , for all  $q \in \text{succ}_{\mathcal{P}}(q_i)$ .

(D)  $\varphi = \chi \mathbf{U}_{[\tau \geq b]}\psi$  and one of the following holds:

1.  $\varphi \in \lambda_{\mathcal{P}}(q_i)$ ,  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\text{lastl}(\mathcal{P}))) > 0$ ,  $\psi \in \lambda_{\mathcal{P}}(q)$  for some  $q \in \text{lastl}(\mathcal{P})$ , and  $\chi \in \lambda_{\mathcal{P}}(q')$  for all  $q' \in \text{succ}_{\mathcal{P}}^*(q_i)$ .

2. The state  $q_i$  is not part of a loop. If  $\varphi \notin \lambda_{\mathcal{P}}(q_i)$ , then  $\psi \notin \lambda_{\mathcal{P}}(q_i)$  or  $0 < b$ , and

- i) there is  $k \geq i$  such that  $\chi \notin \lambda_{\mathcal{P}}(q_k)$  and, for all  $j \in [i, k]$ ,  $|\text{succ}_{\mathcal{P}}(q_j)| = 1$  and  $\psi \in \lambda_{\mathcal{P}}(q_j) \Rightarrow \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(q_i \dots q_{j-1})) < b$  or
  - ii)  $\mathcal{P}$  contains a balance counter  $c_{\tau,i} \in C_{\mathcal{P}}$  for  $\tau$  and  $q_i$ , and  $(c_{\tau,i} < b) \in \Gamma$  for all  $(q, \mu, \Gamma, q_j) \in \Delta_{\mathcal{P}}$  where  $j > i$ ,  $\psi \in \lambda_{\mathcal{P}}(q_j)$ , and  $\forall_{k \in [i, j-1]} : \chi \in \lambda_{\mathcal{P}}(q_k)$ .
- If  $\varphi \in \lambda_{\mathcal{P}}(q_i)$ , then there is  $k \geq i$  with  $\psi \in \lambda_{\mathcal{P}}(q_k)$ ,  $\forall_{j \in [i, k-1]} : \chi \in \lambda_{\mathcal{P}}(q_j)$ , and
- iii)  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(q_i \dots q_{k-1})) \geq b$  and  $\forall_{j \in [i, k-1]} : |\text{succ}_{\mathcal{P}}(q_j)| = 1$ , or
  - iv)  $k > i$  and  $\mathcal{P}$  contains a balance counter  $c_{\tau,i} \in C_{\mathcal{P}}$  for  $\tau$  and  $q_i$ , and some transition  $(q_{k-1}, \mu, \Gamma \cup \{c_{\tau,i} \geq b\}, q_k) \in \Delta_{\mathcal{P}}$ .

3.  $q_i$  is on some loop  $L$  of  $\mathcal{P}$ , and  $q_{i-|L|}$  and  $q_{i+|L|}$  (if  $L \neq \text{lastl}(\mathcal{P})$ ) are  $\varphi$ -consistent.

The APS  $\mathcal{P}$ , a loop, or a row in  $\mathcal{P}$  are  $\varphi$ -consistent if all their states are  $\varphi$ -consistent, respectively.

Observe that the consistency for a formula  $\varphi$  of a given APS  $\mathcal{P}$  can easily be checked recursively over the structure of  $\varphi$  and the states of  $\mathcal{P}$  in polynomial time in  $\text{size}(\mathcal{P}) \cdot |\varphi|$ .

### 4.3 Consistent APS are Correct

With the introduction of consistent APS as the formal witness object and the description of the necessary checks to be performed, the decision procedure for the model-checking problem is, technically, detailed out exhaustively. However, the crucial part of explaining why and proving that this procedure delivers the intended result is the subject of the remainder of the present chapter.

This section is dedicated to the soundness argument. That is, we prove that *if* the procedure succeeds to provide a non-empty and  $\Phi$ -consistent APS that labels the initial state by  $\Phi$ , then the formula is in fact satisfied. The argument is formulated based on the definition of correctness and the following correspondence to consistency.

► **Theorem 4.12 (Correctness).** *Let  $\Phi$  be a cLTL formula and  $\mathcal{P}$  an APS in a counter system  $\mathcal{S}$ . If  $\mathcal{P}$  is  $\Phi$ -consistent, then it is labelled correctly with respect to  $\Phi$ .*

Consequently, if  $\mathcal{S}$  contains a *non-empty* APS  $\mathcal{P}$  that is  $\Phi$ -consistent and where the initial state is labelled by  $\Phi$ , then all the runs  $\sigma \in \text{runs}(\mathcal{P})$  satisfy  $\Phi$  and so does the corresponding run  $\text{org}(\sigma) \in \text{runs}(\mathcal{S})$ . Therefore, Theorem 4.12 implies that the guess-and-check procedure above is sound. The remainder of this section is dedicated to proving Theorem 4.12. Thus, let in the following  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \preceq, \text{org})$  be a fixed and

$\Phi$ -consistent APS. The presented proof is a structural induction on  $\Phi$  and is organised as follows. The central proof goal is stated as

$$\forall q \in Q \forall \sigma \in \text{runs}(\mathcal{P}) \forall i \in \text{pos}_\sigma(q) : \quad \varphi \in \lambda_{\mathcal{P}}(q) \quad \Leftrightarrow \quad (\mathcal{P}, \sigma, i) \models \varphi \quad (4.1)$$

and to be shown for  $\varphi = \Phi$ .

First, the base cases are established in Section 4.3.1 by showing that the hypothesis holds if  $\Phi$  is either an atomic proposition or a guard. For the induction step, the cases of Boolean combinations and *next* formulae are covered subsequently in Section 4.3.2 and Section 4.3.3 is dedicated to *until* formulae.

### 4.3.1 Atomic Propositions and Guards

Let  $q \in Q$  be an arbitrary state of  $\mathcal{P}$ ,  $\sigma \in \text{runs}(\mathcal{P})$  a run of  $\mathcal{P}$  and  $i \in \mathbb{N}$  a position on  $\sigma$  such that  $\sigma(i) = (q, \theta)$  for some valuation  $\theta \in \mathbb{Z}^C$ . Consider the following cases for the structure of  $\Phi$ .

**Propositions** ( $\Phi = p \in AP$ ). Correctness with respect to propositions is provided unconditionally by definition.

**Guards over system counters** ( $\Phi = (\tau \geq b) \in \text{Grd}(C)$ ). From the definition of consistency (Definition 4.11), only condition A applies for guards. By the definition of runs, if  $i = 0$  then  $(q, \theta) = \sigma(0) = (q_I, \mathbf{0})$ . Consistency demands in this case that  $\Phi \in \lambda_{\mathcal{P}}(q)$  if and only if  $\llbracket \tau \rrbracket(\mathbf{0}) \geq b$  which is the case if and only if  $(\mathcal{P}, \sigma, 0) \models \Phi$ . For  $i > 0$  there is a transition  $(q', \mu, \Gamma, q) \in \Delta_{\mathcal{P}}$  such that  $\text{val}(\sigma(i)) \models_{\text{PA}} \Gamma$ . If  $\Phi \in \lambda_{\mathcal{P}}(q)$  consistency requires that  $\Phi \in \Gamma$  and thus  $(\mathcal{P}, \sigma, i) \models \Phi$ . Otherwise if  $\Phi \notin \lambda_{\mathcal{P}}(q)$ , then  $\bar{\Phi} \in \Gamma$  and thus  $(\mathcal{P}, \sigma, i) \not\models \Phi$ .

**Induction.** This settles the base case for the structure of  $\Phi$ . Henceforth, assume as *induction hypothesis (IH)* that the statement of Equation (4.1) holds for all  $\varphi \in \text{sub}(\Phi) \setminus \{\Phi\}$ .

### 4.3.2 Boolean Combinations and Temporal Next

For the induction step assume  $q \in Q$ ,  $\sigma \in \text{runs}(\mathcal{P})$  and  $i \in \mathbb{N}$  with  $\sigma(i) = (q, \theta)$  as above. Consider the following cases for  $\Phi$  and the corresponding conditions B and C of Definition 4.11.

**Negation** ( $\Phi = \neg\varphi$ ). Considering negation, Equation (4.1) follows by

$$\neg\varphi \in \lambda_{\mathcal{P}}(q) \stackrel{(B)}{\Leftrightarrow} \varphi \notin \lambda_{\mathcal{P}}(q) \stackrel{(IH)}{\Leftrightarrow} (\mathcal{P}, \sigma, i) \not\models \varphi \Leftrightarrow (\mathcal{P}, \sigma, i) \models \neg\varphi$$

**Conjunction** ( $\Phi = \varphi \wedge \psi$ ). Analogously to the case of negation observe that

$$\begin{aligned} \varphi \wedge \psi \in \lambda_{\mathcal{P}}(q) &\stackrel{(B)}{\Leftrightarrow} \varphi, \psi \in \lambda_{\mathcal{P}}(q) \stackrel{(IH)}{\Leftrightarrow} (\mathcal{P}, \sigma, i) \models \varphi \text{ and } (\mathcal{P}, \sigma, i) \models \psi \\ &\Leftrightarrow (\mathcal{P}, \sigma, i) \models \varphi \wedge \psi \end{aligned}$$

**Temporal next** ( $\Phi = \mathbf{X}\varphi$ ). The successor  $q' := st(\sigma(i+1))$  of  $q$  at position  $i+1$  on  $\sigma$  is necessarily also a successor in the graph of  $\mathcal{P}$ , i.e.  $q' \in \text{succ}_{\mathcal{P}}(q)$ , and therefore

$$\mathbf{X}\varphi \in \lambda_{\mathcal{P}}(q) \stackrel{(C)}{\Leftrightarrow} \varphi \in \lambda_{\mathcal{P}}(q') \stackrel{(IH)}{\Leftrightarrow} (\mathcal{P}, \sigma, i+1) \models \varphi \Leftrightarrow (\mathcal{P}, \sigma, i) \models \mathbf{X}\varphi.$$

### 4.3.3 Temporal Until

The remaining case of the formula having the form  $\Phi = \varphi \mathbf{U}_{[\tau \geq b]} \psi$  is more involved because this is where the powerful counting mechanism finally comes into play. As above let  $q \in Q$  be some state occurring at a position  $i \in \mathbb{N}$  of some valid run  $\sigma \in \text{runs}(\mathcal{P})$ . The correctness arguments are given next depending on which sub-case of the consistency criterion applies to  $q$ . Conditions D1 and D2 of Definition 4.11 are exhaustive if  $q$  is not part of a loop and lets us already conclude Equation (4.1) for all such states. Building on this intermediate result, correctness is shown for the remaining case of condition D3.

#### Condition D1: A dominating last loop

If  $q$  is consistent because of the first condition, there is a state  $q' \in Q$  on the last loop that is labelled by  $\psi \in \lambda_{\mathcal{P}}(q')$  reachable from  $q$  in  $\mathcal{P}$ . Thus, there is a position  $j \geq i$  on  $\sigma$  with  $st(\sigma(j)) = q'$ . In fact, since the last loop is repeated infinitely on  $\sigma$ , the state  $q'$  reoccurs at all positions  $j' = j + n \cdot |\text{lastl}(\mathcal{P})|$  for  $n \geq 0$ . The value of the constraint term  $\tau$  on the run segment from  $i$  to such a position  $j'$  is

$$\begin{aligned} \llbracket \tau \rrbracket (\#_{i, j'-1}^{\mathcal{P}, \sigma}) &= \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\sigma(i)\sigma(i+1)\dots\sigma(j'))) && \text{(IH)} \\ &= \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\sigma(i)\dots\sigma(j-1))) + n \cdot \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\text{lastl}(\mathcal{P}))). \end{aligned}$$

Since  $\llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\text{lastl}(\mathcal{P}))) > 0$ , the value  $\llbracket \tau \rrbracket (\#_{i, j'-1}^{\mathcal{P}, \sigma})$  satisfies the counting constraint (is larger or equal to  $b$ ) for sufficiently large  $n$ . Moreover, all states reachable from  $q$  are labelled by  $\varphi$  and hence, by (IH), it holds at all position between  $i$  and  $j'$  on  $\sigma$ .

Consequently,  $(\mathcal{P}, \sigma, i) \models \Phi$ , which is in line with  $\Phi \in \lambda_{\mathcal{P}}(q)$  provided by the consistency criterion.

### Condition D2: Explicit Evaluation of Counting Constraints

For condition D2 to apply,  $q$  must not be on a loop and thus  $i$  is the only position on  $\sigma$  where the state occurs.

**Case  $\Phi \in \lambda_{\mathcal{P}}(q)$ .** If  $q$  is labelled by  $\Phi \in \lambda_{\mathcal{P}}(q)$ , consistency provides that there is a state  $q' \succeq_{\mathcal{P}} q$  that is labelled by  $\psi \in \lambda_{\mathcal{P}}(q')$  and reached (at least for the first time) by a sequence of states labelled by  $\varphi$ . Hence, by induction, for the first position  $j = \min \text{pos}_{\sigma}(q') \geq i$  of  $q'$  on  $\sigma$ , we have  $(\mathcal{P}, \sigma, j) \models \psi$  and  $(\mathcal{P}, \sigma, j') \models \varphi$  for all positions  $j' \in [i, j - 1]$  in between. In case none of the states in between has more than one successor in  $\mathcal{P}$  (condition D(2)iii), the state sequence  $st(\sigma(i) \dots \sigma(j))$  is simple and

$$\llbracket \tau \rrbracket (\#_{i,j-1}^{\mathcal{P};\sigma}) = \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(st(\sigma(i) \dots \sigma(j-1)))) \geq b.$$

Otherwise (condition D(2)iv), the transition taken by  $\sigma$  between position  $j - 1$  and  $j$  is guarded by  $c_{\tau,q} \geq b$  for some balance counter  $c_{\tau,q} \in \text{counters}(\mathcal{P})$  for  $\tau$  and  $q$ . Thus, by correctness for labels from  $\text{sub}(\tau)$  (IH) and Lemma 4.9,

$$\llbracket \tau \rrbracket (\#_{i,j-1}^{\mathcal{P};\sigma}) = \text{val}(\sigma(j))(c_{\tau,q}) \geq b$$

and therefore  $(\mathcal{P}, \sigma, i) \models \Phi$ .

**Case  $\Phi \notin \lambda_{\mathcal{P}}(q)$ .** If  $\Phi \notin \lambda_{\mathcal{P}}(q)$  is not a label of  $q$ , condition D2 provides two alternatives. The first possibility for  $q$  being  $\Phi$ -consistent if  $\Phi \notin \lambda_{\mathcal{P}}(q)$  (condition D(2)i) provides that at some point  $j \geq i$  the formula  $\varphi$  is violated so that all later positions cannot contribute to the satisfaction of  $\Phi$ . It requires moreover, that up to that position all states  $q'$  have only one successor and thus occur only once before position  $j$ . They may be repeated after  $j$  but that does not matter for the satisfaction of  $\Phi$ . For each of these states  $q'$  that do qualify as potential witness by satisfying  $\psi$ , the condition imposes that the simple path in  $\mathcal{P}$  from  $q$  to  $q'$  violates the counting constraint. Since all segments of  $\sigma$  between  $i$  and  $j$  are simple, no position in between can witness satisfaction of  $\Phi$ .

The alternative condition D(2)ii imposes that whenever  $\sigma$  reaches a position  $j$  with  $(\mathcal{P}, \sigma, j) \models \psi$  (identified correctly by  $\psi \in \lambda_{\mathcal{P}}(\sigma(j))$ ) due to IH) the run either already passed a position in between violating  $\varphi$  or the state is guarded by  $c_{\tau,q} < b$  for some balance

counter  $c_{\tau,q}$  for  $\tau$  and  $q$ . Thus, the valuation  $val(\sigma(j))$  satisfies the guard meaning that

$$b > val(\sigma(j))(c_{\tau,q}) = \llbracket \tau \rrbracket (\#_{i,j-1}^{\mathcal{P},\sigma})$$

by Lemma 4.9. Consequently, there is no suitable witness position and therefore  $(\mathcal{P}, \sigma, i) \not\models \Phi$ .

► **Lemma 4.13.** *Let  $q \in Q$  be row state in  $\mathcal{P}$ ,  $\sigma \in runs(\mathcal{P})$  and  $\{i\} = pos_{\sigma}(q)$ . If  $\mathcal{P}$  is correctly labelled with respect to all strict subformulae  $sub(\Phi) \setminus \{\Phi\}$  of  $\Phi$ , then  $(\mathcal{P}, \sigma, i) \models \Phi$  if and only if  $\Phi \in \lambda_{\mathcal{P}}(q)$ .*

Together, conditions D1 and D2 are exhaustive concerning the consistency of row states. We can thus already conclude, given the induction hypothesis, that all  $\Phi$ -consistent row states of  $\mathcal{P}$  are correctly labelled with respect to  $\Phi$ .

### Condition D3: Representative Unfoldings

The remaining alternative of Definition 4.11 (D) concerns only loop states and therefore assume that  $q = st(\sigma(i)) \in L$  for some loop  $L \in loops(\mathcal{P})$ .

**Last loop.** First consider the case that  $q$  occurs on the last loop  $L = lastl(\mathcal{P})$  of  $\mathcal{P}$  and recall that  $L$  is preceded by an identically labelled front row  $F \in Q^+$ . Let  $\ell := |L| = |F|$  be the length of  $L$  and  $h \in \mathbb{N}$  be the position where  $\sigma$  enters  $F$ , i.e., where  $st(\sigma(h)) = F(0)$ . Since  $L$  is repeated infinitely, the sequence of labels of  $\sigma$  starting at position  $h$  has the form

$$\lambda_{\mathcal{P}}(\sigma(h)\sigma(h+1)\dots) = \lambda_{\mathcal{P}}(FL^{\omega}) = \lambda_{\mathcal{P}}(F)^{\omega} = \lambda_{\mathcal{P}}(L)^{\omega}$$

and is invariant under shifting  $h$  by multiples of  $\ell$ . Concerning the satisfaction of  $\Phi$ , this implies for any two positions  $x$  and  $y$  on  $\sigma$  with  $h \leq x \leq y$  and  $n \in \mathbb{N}$

- $\psi \in \lambda_{\mathcal{P}}(\sigma(y)) \Leftrightarrow \psi \in \lambda_{\mathcal{P}}(\sigma(y+n\ell))$ ,
- $\left( \forall_{z \in [x,y-1]} : \varphi \in \lambda_{\mathcal{P}}(\sigma(z)) \right) \Leftrightarrow \left( \forall_{z \in [x+n\ell,y-1+n\ell]} : \varphi \in \lambda_{\mathcal{P}}(\sigma(z)) \right)$ , and
- $\lambda_{\mathcal{P}}^{\#}(\sigma(x)\dots\sigma(y-1)) = \lambda_{\mathcal{P}}^{\#}(\sigma(x+n\ell)\dots\sigma(y-1+n\ell))$ .

By induction, the labelling of  $\mathcal{P}$  is correct for all strict subformulae of  $\Phi$  and thus

- $(\mathcal{P}, \sigma, y) \models \psi \Leftrightarrow (\mathcal{P}, \sigma, y+n\ell) \models \psi$ ,
- $\left( \forall_{z \in [x,y-1]} : (\mathcal{P}, \sigma, z) \models \varphi \right) \Leftrightarrow \left( \forall_{z \in [x+n\ell,y-1+n\ell]} : (\mathcal{P}, \sigma, z) \models \varphi \right)$ , and

- $\#_{x,y-1}^{\mathcal{P},\sigma}(\chi) = \#_{x+n\ell,y-1+n\ell}^{\mathcal{P},\sigma}(\chi)$  for all subformulae  $\chi \in \text{sub}(\tau)$  of the constraint term  $\tau$ .

Therefore,

$$(\mathcal{P}, \sigma, x) \models \Phi \quad \Leftrightarrow \quad (\mathcal{P}, \sigma, x + n\ell) \models \Phi. \quad (4.2)$$

Let  $j \in [0, \ell - 1]$  be the position of  $q = L(j)$  on  $L$ . Recall that  $\lambda_{\mathcal{P}}(L(j)) = \lambda_{\mathcal{P}}(F(j))$  and consistency of  $q$  provides that the row state  $q' := F(j) = \text{st}(\sigma(h + j))$  is  $\Phi$ -consistent. Since  $i = h + j + n\ell$  for some  $n \in \mathbb{N}$  it follows that

$$\Phi \in \lambda_{\mathcal{P}}(q) \Leftrightarrow \Phi \in \lambda_{\mathcal{P}}(\sigma(q')) \stackrel{\text{Lem. 4.13}}{\Leftrightarrow} (\mathcal{P}, \sigma, h + j) \models \Phi \stackrel{\text{Eq. (4.2)}}{\Leftrightarrow} (\mathcal{P}, \sigma, i) \models \Phi.$$

**Intermediate loop.** If  $q$  is not the last loop  $\text{lastl}(\mathcal{P}) \neq L$  of  $\mathcal{P}$ ,  $\Phi$ -consistency of  $q$  provides corresponding  $\Phi$ -consistent states on the rows immediately preceding and succeeding  $L$  in  $\mathcal{P}$ . The following lemma formulates the main argument for deriving correctness of the labelling of  $q$  from the correct labelling of the corresponding states of the rows enclosing  $L$ .

► **Lemma 4.14.** *Let  $L$  be a loop in  $\mathcal{P}$ ,  $\ell := |L|$  and  $\tau_L := \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(L))$ . Let  $\sigma \in \text{runs}(\mathcal{P})$  and  $n, h, h' \in \mathbb{N}$  such that  $\lambda_{\mathcal{P}}(\sigma(h)\sigma(h+1) \dots \sigma(h')) = \lambda_{\mathcal{P}}(L)^n$ .*

*i) If some state of  $L$  is not labelled by  $\varphi$ , then for all  $x \in [h, h' - 2\ell]$*

$$(\mathcal{P}, \sigma, x) \models \Phi \quad \Leftrightarrow \quad (\mathcal{P}, \sigma, x + \ell) \models \Phi.$$

*ii) If  $\tau_L \leq 0$ , then for all  $x \in [h, h' - 2\ell]$*

$$(\mathcal{P}, \sigma, x) \models \Phi \quad \Rightarrow \quad (\mathcal{P}, \sigma, x + \ell) \models \Phi.$$

*iii) If all states of  $L$  are labelled by  $\varphi$  and  $\tau_L \geq 0$ , then for all  $x \in [h + \ell, h']$*

$$(\mathcal{P}, \sigma, x) \models \Phi \quad \Rightarrow \quad (\mathcal{P}, \sigma, x - \ell) \models \Phi.$$

**Proof.** Recall that  $\mathcal{P}$  is consistently labelled with respect to all strict subformulae of  $\Phi = \varphi \mathbf{U}_{[\tau \geq b]} \psi$  and by IH these formulae hold at some position on  $\sigma$  if and only the corresponding state is labelled by it.

*i)* Assume that  $\varphi \notin \lambda_{\mathcal{P}}(q')$  for some state  $q' \in L$  and let  $x \in [h, h' - 2\ell]$  be some position on  $\sigma$  within the periodically labelled part. Due to correctness, the formula  $\Phi$

holds at position  $x$  if and only if all positions  $y \in [x, z - 1]$  are labelled by  $\varphi$  up to some position  $z \geq x$  that is itself labelled by  $\psi$  and where  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x) \dots \sigma(z - 1))) \geq b$ . Since  $L$  is not entirely labelled by  $\varphi$ , the position  $z$  must occur before a whole iteration of  $L$  is completed, i.e.,  $z < x + \ell$ . Otherwise, some state not labelled by  $\varphi$  occurred in between. Thus,  $(\mathcal{P}, \sigma, x) \models \Phi$  if and only if there is such  $z \in [x, x + \ell - 1]$ . The same argument applies to position  $x' = x + \ell$  since  $x' + \ell - 1 = x + 2\ell - 1 \leq h'$  is still on the periodically labelled part of  $\sigma$ . Therefore,  $(\mathcal{P}, \sigma, x + \ell) \models \Phi$  if and only if there is  $z' \in [x', x' + \ell - 1]$  such that  $\psi \in \lambda_{\mathcal{P}}(\sigma(z'))$ , all positions  $y' \in [x', z' - 1]$  are labelled by  $\varphi$ , and  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x') \dots \sigma(z' - 1))) \geq b$ .

Further, we have  $\lambda_{\mathcal{P}}(\sigma(x) \dots \sigma(z)) = \lambda_{\mathcal{P}}(\sigma(x + \ell) \dots \sigma(z + \ell))$  and thus for all  $z \in [x, x + \ell - 1]$  it follows that  $\psi \in \lambda_{\mathcal{P}}(z) \Leftrightarrow \psi \in \lambda_{\mathcal{P}}(z + \ell)$ , all states  $y \in [x, z - 1]$  are labelled by  $\varphi$  if and only if this is the case for all  $y' \in [x + \ell, z + \ell - 1]$ , and

$$\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x + \ell) \dots \sigma(z + \ell - 1))) = \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x) \dots \sigma(z - 1))).$$

Consequently, there is a witness position  $z$  for the satisfaction of  $\Phi$  at  $x$  if and only if there is such a position  $z'$  for  $x' = x + \ell$ .

*ii)* Assume that  $L$  is entirely labelled by  $\varphi$ ,  $\tau_L \leq 0$ , and  $(\mathcal{P}, \sigma, x) \models \Phi$  for  $x \in [h, h' - 2\ell]$ . As above, let  $z \geq x$  be some position witnessing the satisfaction of  $\Phi$  at position  $x$ , i.e.,  $\psi \in \lambda_{\mathcal{P}}(\sigma(z))$ , all positions  $y \in [x, z - 1]$  are labelled by  $\varphi$ , and  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x) \dots \sigma(z - 1))) \geq b$ . If  $z < x + \ell$  let  $z' := z + \ell$  and observe that, as above,  $z'$  is a witness position for the satisfaction of  $\Phi$  at position  $x + \ell$  because  $x + \ell \leq z' < x + 2\ell \leq h'$  and thus  $\lambda_{\mathcal{P}}(\sigma(x) \dots \sigma(z)) = \lambda_{\mathcal{P}}(\sigma(x + \ell) \dots \sigma(z'))$ . If otherwise,  $z \geq x + \ell$ , then  $z$  itself witness the satisfaction of  $\Phi$  at  $x + \ell$  because

$$\begin{aligned} \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x + \ell) \dots \sigma(z - 1))) &\geq \tau_L + \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x + \ell) \dots \sigma(z - 1))) \\ &= \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(\sigma(x) \dots \sigma(z - 1))) \\ &\geq b. \end{aligned}$$

*iii)* Assume finally that  $L$  is entirely labelled by  $\varphi$ ,  $\tau_L \geq 0$ , and  $(\mathcal{P}, \sigma, x) \models \Phi$  for  $x \in [h + \ell, h']$ . Let  $z \geq x$  be a witness position for  $\Phi$  at  $x$ . Since  $x - \ell \geq h$  is still on the

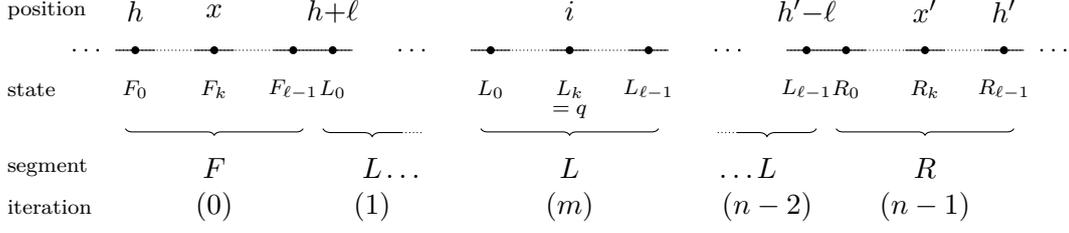


Figure 4.4: Sketch of the run  $\sigma$  and the location of the relevant parts and indices.

periodically labelled part of  $\sigma$  we have

$$\begin{aligned}
 \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\sigma(x-\ell) \dots \sigma(z-1))) &= \tau_L + \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\sigma(x) \dots \sigma(z-1))) \\
 &\geq \llbracket \tau \rrbracket (\lambda_{\mathcal{P}}^{\#}(\sigma(x) \dots \sigma(z-1))) \\
 &\geq b.
 \end{aligned}$$

Moreover, the positions  $x-\ell, \dots, x-1$  are labelled by  $\varphi$  and thus all positions from  $x-\ell$  to  $z-1$ . Therefore,  $z$  also witnesses the satisfaction of  $\Phi$  at position  $x-\ell$  on  $\sigma$ . ■

Using the lemma above, the cases for the state  $q$  being part of an intermediate loop can be analysed individually as follows. Let  $F = F_0 \dots F_{\ell-1}$  and  $R = R_0 \dots R_{\ell-1}$  be the front and rear row, respectively, surrounding  $L$  in  $\mathcal{P}$  and let  $k \in [0, \ell-1]$  be the index of  $q = L_k$  on  $L$ . Further, let  $h, h', x, x' \in \mathbb{N}$  be the (unique) positions on  $\sigma$  of  $F_0, R_{\ell-1}, F_k, R_k$ , respectively. Then, the labelling sequence of  $\sigma$  from  $h$  to  $h'$  has the form

$$\lambda_{\mathcal{P}}(\sigma(h) \dots \sigma(h')) = \lambda_{\mathcal{P}}(FL^{n-2}R) = \lambda_{\mathcal{P}}(L)^n$$

where  $n-2 = |\text{pos}_{\sigma}(L_0)|$ , is the number of iterations of  $L$  on  $\sigma$ . Figure 4.4 shows a sketch of the run  $\sigma$  and the location of the relevant parts and indices. We can now choose  $m \geq 1$  and  $m' \geq 2$  such that

$$h + \ell \leq i = h + k + m\ell = h' + k - m'\ell \leq h' - \ell$$

and use Lemma 4.14 as follows in order to show that  $\Phi \in \lambda_{\mathcal{P}}(\sigma(i))$  if and only if  $(\mathcal{P}, \sigma, i) \models \Phi$ .

If  $\Phi \in \lambda_{\mathcal{P}}(\sigma(i))$ , then  $(\mathcal{P}, \sigma, x) \models \Phi$  and  $(\mathcal{P}, \sigma, x') \models \Phi$  because  $\lambda_{\mathcal{P}}(L_k) = \lambda_{\mathcal{P}}(F_k) = \lambda_{\mathcal{P}}(R_k)$  and the labellings of  $F$  and  $R$  are correct with respect to  $\Phi$  by Lemma 4.13. In case

$L$  is not entirely labelled by  $\varphi$  or its effect  $\tau_L = \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}^{\#}(L))$  is negative, Lemma 4.14 (i) and (ii), respectively, provides (by recursive application) that  $(\mathcal{P}, \sigma, x) \models \Phi$  implies  $(\mathcal{P}, \sigma, i) \models \Phi$  since  $i = x + m\ell \leq h' - \ell$ . Otherwise, all states of  $L$  are labelled by  $\varphi$  and  $\tau_L \geq 0$  and thus  $(\mathcal{P}, \sigma, x') \models \Phi$  implies  $(\mathcal{P}, \sigma, i) \models \Phi$  by (recursive application of) Lemma 4.14 (iii), since  $i = h' - m'\ell$ .

Similarly if  $\Phi \notin \lambda_{\mathcal{P}}(\sigma(i))$ , Lemma 4.14 (i) applies in case some state of  $L$  is not labelled by  $\varphi$ . Alternatively, if all states of  $L$  are labelled by  $\varphi$  and  $\tau_L \geq 0$ , Lemma 4.14 (iii) can be applied in its dual formulation, i.e., providing that  $(\mathcal{P}, \sigma, x) \not\models \Phi$  implies  $(\mathcal{P}, \sigma, x + \ell) \not\models \Phi$  for  $x \in [h, h' - \ell]$ . For the remaining case where  $\tau_L < 0$  and all states of  $L$  are labelled (correctly) by  $\varphi$  we employ the dual formulation of Lemma 4.14 (ii) providing that

$$(\mathcal{P}, \sigma, y) \not\models \Phi \quad \text{implies} \quad (\mathcal{P}, \sigma, y - \ell) \not\models \Phi \quad (4.3)$$

for all  $y \in [h + \ell, h' - \ell]$ . Notice that this implication cannot be applied directly to the position  $x'$  of  $R_k$  since  $x' > h' - \ell$ . However, we can show that  $(\mathcal{P}, \sigma, x' - \ell) \not\models \Phi$ . Towards contradiction assume that  $(\mathcal{P}, \sigma, x' - \ell) \models \Phi$  and let  $z \geq x' - \ell$  be some position witnessing that on  $\sigma$ . If  $z \geq x'$ , then  $z$  would also witness that  $\Phi$  is satisfied at position  $x'$  because

$$\llbracket \tau \rrbracket(\#_{x', z-1}^{\sigma}) = \llbracket \tau \rrbracket(\#_{x' - \ell, z-1}^{\sigma}) - \tau_L \geq \llbracket \tau \rrbracket(\#_{x' - \ell, z-1}^{\sigma}) \geq b.$$

On the other hand,  $x' - \ell \leq z < x'$  would imply that  $z = x' - \ell + j$  for some distance  $j < \ell$ . Thus, the label sequence

$$\lambda_{\mathcal{P}}(\sigma(x' - \ell) \dots \sigma(x' - \ell + j)) = \lambda_{\mathcal{P}}(\sigma(x) \dots \sigma(x + j))$$

from  $x' - \ell$  up to  $z$  occurs also at position  $x$ , making  $x + j$  a witness that  $\Phi$  holds at position  $x$ , which is not the case. Consequently,  $(\mathcal{P}, \sigma, x' - \ell) \not\models \Phi$  and by recursive application of Equation (4.3) we obtain that  $(\mathcal{P}, \sigma, i) \not\models \Phi$  since  $i = h' - m'\ell$ .

This completes the case that state  $q$  is on an intermediate loop to which condition D3 of Definition 4.11 applies and thereby the induction step of the proof of Theorem 4.12 showing that  $q$  is correctly labelled if it is  $\Phi$ -consistent. The consequence is, that the theoretical non-deterministic procedure outlined in Section 4.1 is *correct* in the sense that if it yields a result, then the formula  $\Phi$  is in fact satisfied by the counter system  $\mathcal{S}$ .

## 4.4 Consistency-preserving Operations

We have seen that consistency is a useful criterion because it implies that an APS is labelled correctly. The next Section 4.5 will be concerned with showing that particular consistent APS exist. The corresponding constructions rely on manipulating these structures while maintaining consistency. Specifically, given an APS  $\mathcal{P}$  with loop  $L \in \text{loops}(\mathcal{P})$  and a run  $\sigma \in \text{runs}(\mathcal{P})$ , the following operations are of interest.

- **Unfolding.** Assuming  $\sigma$  traverses  $L$  at least twice, the first or last iteration is to be represented explicitly by a copy of  $L$  inserted as row right before or after the loop.
- **Cut.** Assuming  $\sigma$  traverses  $L$  exactly once, the backward transition of  $L$  can be removed from  $\mathcal{P}$ .
- **Duplication.** Assuming  $\sigma$  traverses  $L$  at least three times,  $L$  is replaced by two copies of it connected by one unfolding.

Figure 4.5 presents a sketch of the modifications. Apart from staying consistent, the second crucial invariant is that the run  $\sigma$  remains in the system, more precisely, the original run  $\text{org}(\sigma)$  of  $\mathcal{S}$  represented by  $\sigma$  is still represented by some run  $\hat{\sigma}$  after the modification.

The constructions and arguments used for all operations are very similar. Therefore, only the duplication construction is presented in full detail while the properties of unfolding and cut operations are recorded as corollaries.

► **Lemma 4.15 (Duplication).** *Let  $M \subseteq \text{cLTL}$  and  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  be an  $M$ -consistent APS in  $\mathcal{S}$  with  $L \in \text{loops}(\mathcal{P})$  and  $\sigma \in \text{runs}(\mathcal{P})$  such that  $st(\sigma) = uL^n w$  for some  $u \in (Q \setminus L)^*$ ,  $w \in Q^\omega$ , and  $n \geq 3$ . There is an  $M$ -consistent APS  $\hat{\mathcal{P}} = (Q \dot{\cup} U \dot{\cup} K, \hat{\Delta}, \hat{\lambda}, \widehat{\text{org}})$  with additional loop  $K$  and row  $U$  of length  $|K| = |U| = |L|$  and run  $\hat{\sigma}$  such that*

- $\text{loops}(\hat{\mathcal{P}}) = \text{loops}(\mathcal{P}) \cup \{K\}$ ,
- $st(\hat{\sigma}) = uK^{n-2}ULw$ ,
- $\widehat{\text{org}}(\hat{\sigma}) = \text{org}(\sigma)$ , and
- $|\hat{\mathcal{P}}| \leq |\mathcal{P}| + 3 \cdot \text{size}(\Delta_L) + |L| \cdot (|\Delta| + 2|L| + 1) \cdot (\sum_{\varphi' \in M} |\varphi'|)$ .

The construction essentially inserts, right before  $L$  one copy of  $L$  as row  $U$  and another copy as loop  $K$ , as depicted in Figure 4.5. The labelling of the copies is inherited from  $L$  and thus also equal to the former front row of  $L$ , now being the front row of  $K$ .

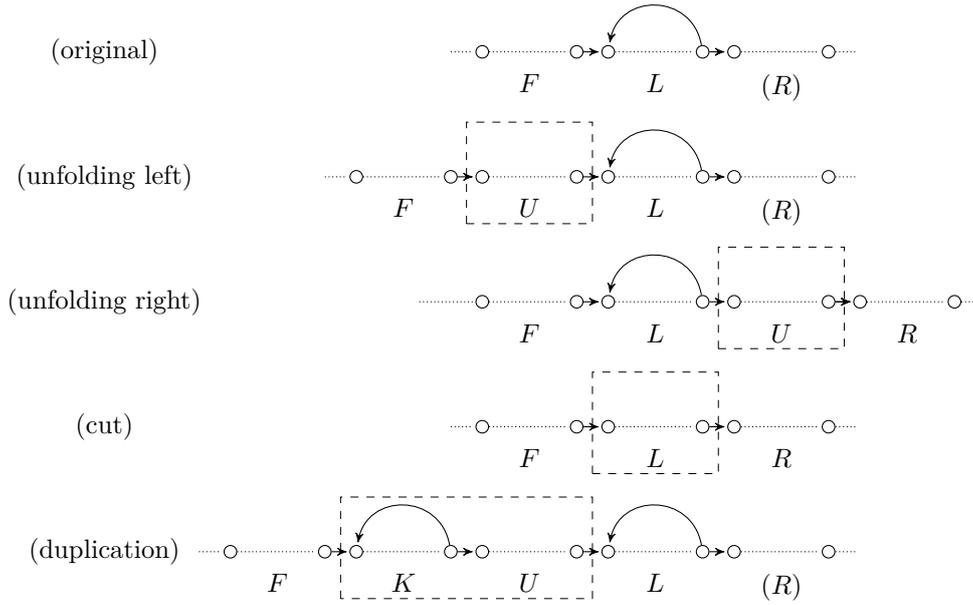


Figure 4.5: Consistency-preserving operations on augmented path schemas. Notice that the row  $R$  may not always exist and then *cut* and *unfolding right* cannot be performed.

Consistency for most formulae relies only on local conditions and we will show that these remain satisfied on the copies of  $L$ . Only, for *until* formulae, it may be necessary to add additional counters and guards in order to establish consistency of the new row  $U$ . Once that is accomplished,  $K$  becomes consistent automatically and consistency of  $L$  is recovered.

#### 4.4.1 Basic Structure and Induction Scheme

**Basic structure.** As basis for the following developments, let  $\mathcal{D} = (Q_{\mathcal{D}}, \Delta_{\mathcal{D}}, \lambda_{\mathcal{D}}, \text{org}_{\mathcal{D}})$  be the APS in  $\mathcal{S}$  derived from  $\mathcal{P}$  by inserting  $U$  and  $K$  into the structure as depicted in Figure 4.5. Technically, let  $U$  and  $K$  be simple (i.e., repetition-free) sequences of length  $|L|$ , comprised of fresh states, pairwise disjoint with  $Q$  and each other, and let  $Q_{\mathcal{D}} := Q \cup U \cup K$ . Let  $\ell := |L| - 1$  be the maximal index on  $L$  (and also  $U$  and  $K$ ) and, for easier reading, let  $L_i$ ,  $U_i$ , and  $K_i$  denote  $L(i)$ ,  $U(i)$ , and  $K(i)$ , respectively, for  $0 \leq i \leq \ell$ . Further, let  $F = \text{front}_{\mathcal{P}}(L) = F_0 \dots F_{\ell}$  denote the row immediately preceding  $L$  in  $\mathcal{P}$  and if  $L \neq \text{lastl}(\mathcal{P})$ , let  $R = \text{rear}_{\mathcal{P}}(L) = R_0 \dots R_{\ell}$  be the rear row of  $L$ . Recall that  $\lambda(F) = \lambda(R) = \lambda(L)$ .

The transition set  $\Delta_{\mathcal{D}}$  is now defined by

$$\begin{aligned} \Delta_{\mathcal{D}} := & (\Delta \setminus \{(F_{\ell}, \mu, \Gamma, L_0) \in \Delta\}) \\ & \cup \{(K_{\ell}, \mu, \Gamma, U_0), (U_{\ell}, \mu, \Gamma, L_0), (K_{\ell}, \mu, \Gamma, K_0) \mid (L_{\ell}, \mu, \Gamma, L_0) \in \Delta\} \\ & \cup \{(F_{\ell}, \mu, \Gamma, K_0) \mid (F_{\ell}, \mu, \Gamma, L_0) \in \Delta\} \\ & \cup \{(U_i, \mu, \Gamma, U_{i+1}), (K_i, \mu, \Gamma, K_{i+1}) \mid 0 \leq i < \ell, (L_i, \mu, \Gamma, L_{i+1}) \in \Delta\}. \end{aligned}$$

Clearly, its cardinality is precisely  $|\Delta_{\mathcal{D}}| = |\Delta| + 2|L| + 1$ . The labelling  $\lambda_{\mathcal{D}}$  and the origin function  $\text{org}_{\mathcal{D}}$  are the extensions of  $\lambda$  and  $\text{org}$ , respectively, to the extended set of states  $Q_{\mathcal{D}} \supset Q$  such that  $\lambda_{\mathcal{D}}(U) = \lambda_{\mathcal{D}}(K) = \lambda(L)$  and  $\text{org}_{\mathcal{D}}(U) = \text{org}_{\mathcal{D}}(K) = \text{org}(L)$ . Since  $\mathcal{P}$  admits the run  $\sigma \in \text{runs}(\mathcal{P})$  that traverses  $L$  at least three times, it is also straightforward to construct a run  $\sigma_{\mathcal{D}} \in \text{runs}(\mathcal{D})$  by replacing the first  $n - 2$  iterations of  $L$  by traversals of  $K$  and the subsequent traversal of  $L$  by  $U$ . The run has thus the state sequence  $st(\sigma_{\mathcal{D}}) = uK^{n-2}ULw$  and exactly the same sequence of valuations  $val(\sigma_{\mathcal{D}}) = val(\sigma)$ . The origin states of  $K$  and  $U$  are those of  $L$  and thus  $\text{org}_{\mathcal{D}}(\sigma_{\mathcal{D}}) = \text{org}(\sigma)$ . The size of  $\mathcal{D}$  can be estimated based on the additional transitions as

$$|\mathcal{D}| \leq |\mathcal{P}| + 3 \cdot \text{size}(\Delta_L)$$

where  $\Delta_L = \{(q, \mu, \Gamma, q') \in \Delta \mid q, q' \in L\}$  is the set of transitions on  $L$ . Notice that the backward transition  $(L_{\ell}, \mu, \Gamma, L_0) \in \Delta$  of  $L$  is copied three times, hence the factor.

**Induction.** Based on this construction we can prove Lemma 4.15 using well-founded induction on the subsets of  $N = \text{sub}(N) \subseteq M$  that are closed under taking subformulae. Observe that if  $\mathcal{P}$  is  $M$ -consistent, then  $\mathcal{P}$  is by definition  $\text{sub}(M)$ -consistent and we can assume without loss of generality that  $M = \text{sub}(M)$ . Further notice that the class of such closed subsets is well-founded with respect to the subset ordering  $\subseteq$  and has the unique minimal element  $\emptyset$ .

Assume  $\Delta_{\mathcal{D}} = \{\delta_0, \dots, \delta_k\}$ . Formally we use induction to show that there is an  $M$ -consistent APS  $\mathcal{D}_M = (Q_{\mathcal{D}}, \Delta_M, \lambda_{\mathcal{D}}, \text{org}_{\mathcal{D}})$  such that

- $\Delta_M = \{\delta'_0, \dots, \delta'_k\}$  where  $\delta'_i = (s, \mu', \Gamma', t)$  for  $\delta_i = (s, \mu, \Gamma, t)$ , some  $\mu \sqsubseteq \mu'$ ,  $\Gamma \subseteq \Gamma'$ , and all  $i \in [0, k]$ ;
- there is  $\sigma_M \in \text{runs}(\mathcal{D}_M)$  with  $st(\sigma_M) = st(\sigma_{\mathcal{D}})$ ; and
- $|\mathcal{D}_M| \leq |\mathcal{D}| + |L| \cdot |\Delta_{\mathcal{D}}| \cdot \sum_{\varphi \in M} |\varphi|$ .

This implies the lemma statement, as is discussed later.

The base case for  $M = \emptyset$  is trivially provided by  $\mathcal{D}_\emptyset = \mathcal{D}$ . Assume henceforth as *induction hypothesis* that for every strict subset  $N = \text{sub}(N) \subset M$  there is an  $N$ -consistent APS  $\mathcal{D}_N$  satisfying the conditions above for the set  $N$ . The *induction step* is now to provide the APS  $\mathcal{D}_M$ . To this end, let  $\varphi \in M$  be a maximal element in  $M$  with respect to the structural subformula ordering and  $N := M \setminus \{\varphi\}$ . Choosing  $\varphi$  maximal guarantees that  $N = \text{sub}(N)$  is closed and let thus  $\mathcal{D}_N = (Q_{\mathcal{D}}, \Delta_N, \lambda_{\mathcal{D}}, \text{org}_{\mathcal{D}})$  be the APS provided by the induction hypothesis with run  $\sigma_N$ . By means of an exhaustive case analysis it is shown in the following that  $\mathcal{D}_N$  is  $\varphi$ -consistent already or (in case of *until* formulae) can be modified to be, only by adding fresh counters, updates, and guards to the transitions of  $\Delta_N$ . In the latter case it will be argued that  $\sigma_N$  constitutes a run  $\sigma_M \in \text{runs}(\mathcal{D}_M)$  as required.

#### 4.4.2 Non-Until Formulae

In the case that  $\varphi$  is not an until formula, we show that  $\mathcal{D}_N$  qualifies immediately as  $\mathcal{D}_M$  because  $\varphi$ -consistency is inherited from  $\mathcal{P}$  without need for modification.

**Atomic propositions and Boolean combinations.** If  $\varphi \in AP$  is an atomic proposition, then  $\mathcal{D}_N$  is  $\varphi$ -consistent by definition. In the case that  $\varphi$  has the form  $\neg\psi$  or  $\chi \wedge \psi$ , consistency depends solely on the labelling of each state  $q \in Q_{\mathcal{D}}$  individually. For all states  $q \in Q_{\mathcal{P}}$  the labellings  $\lambda_{\mathcal{D}}(q) = \lambda_{\mathcal{P}}(q)$  of  $\mathcal{P}$  and  $\mathcal{D}$  are equal,  $\mathcal{P}$  is  $\varphi$ -consistent and by induction  $\mathcal{D}_N$  is consistent with respect to the subformulae  $\chi, \psi \in \text{sub}(\varphi) \setminus \{\varphi\} \subseteq N$ . Therefore, these states are also  $\varphi$ -consistent in  $\mathcal{D}_N$ . The states  $U_i$  and  $K_i$  for  $i \in [0, \ell]$  do not occur in  $\mathcal{P}$  but since  $\lambda_{\mathcal{D}}(U_i) = \lambda_{\mathcal{D}}(K_i) = \lambda_{\mathcal{D}}(L_i) = \lambda_{\mathcal{P}}(L_i)$  they also satisfy the consistency condition for  $\varphi$ . Thus,  $\mathcal{D}_M = \mathcal{D}_N$  being  $M$ -consistent with run  $\sigma_M = \sigma_N$ .

**Guard formulae.** If  $\varphi = (\tau \geq b) \in \text{Grd}(C)$  is a guard formula, we also observe that  $\mathcal{D}_M = \mathcal{D}_N$  is already  $\varphi$ -consistent. Recall that consistency requires that  $\varphi$  or  $\bar{\varphi}$  are guards on every incoming edge of a state  $q \in Q_{\mathcal{D}}$  with  $\varphi \in \lambda_{\mathcal{D}}(q)$  or  $\varphi \notin \lambda_{\mathcal{D}}(q)$ , respectively. All states  $q \in Q_{\mathcal{P}} \setminus \{L_0\}$  that are already present in  $\mathcal{P}$ , except for first state  $L_0$  of  $L$ , are  $\varphi$ -consistent because each incoming transition  $(q', \mu, \Gamma, q) \in \Delta_N$  corresponds to an incoming transition  $(q', \mu', \Gamma', q) \in \Delta_{\mathcal{P}}$  in  $\mathcal{P}$  with  $\Gamma' \subseteq \Gamma$  that is guarded appropriately due to consistency of  $\mathcal{P}$ .

All incoming transitions  $(q', \mu', \Gamma', q) \in \Delta_N$  of a state  $q \in \{U_0, L_0\}$  are  $\varphi$ -consistent because each corresponds to a copy of the backward transition  $(L_\ell, \mu, \Gamma, L_0) \in \Delta_{\mathcal{P}}$  of  $L$

and thus

$$\begin{aligned} \varphi \in \lambda_{\mathcal{D}}(q) &\Rightarrow \varphi \in \lambda_{\mathcal{P}}(L_0) \Rightarrow \varphi \in \Gamma \Rightarrow \varphi \in \Gamma' && \text{and} \\ \varphi \notin \lambda_{\mathcal{D}}(q) &\Rightarrow \varphi \notin \lambda_{\mathcal{P}}(L_0) \Rightarrow \bar{\varphi} \in \Gamma \Rightarrow \bar{\varphi} \in \Gamma'. \end{aligned}$$

The state  $K_0$  also has one incoming (backward) transition that corresponds to the backward transition  $(L_\ell, \mu, \Gamma, L_0) \in \Delta_{\mathcal{P}}$  of  $L$  and obeys the criterion. Further, the state has an incoming forward transition  $(F_\ell, \mu, \Gamma, K_0) \in \Delta_{\mathcal{D}}$  corresponding to  $(F_\ell, \mu, \Gamma, L_0) \in \Delta_{\mathcal{P}}$  to which the same reasoning applies. Similarly, for all  $i \in [1, \ell]$ , the (single) incoming transitions  $(K_{i-1}, \mu'_i, \Gamma'_i, K_i) \in \Delta_N$  and  $(U_{i-1}, \mu''_i, \Gamma''_i, U_i) \in \Delta_N$  of the states  $K_i$  and  $U_i$ , respectively, originate from the incoming transition  $(L_{i-1}, \mu_i, \Gamma_i, L_i) \in \Delta_{\mathcal{P}}$  of  $L_i$  and hence

$$\begin{aligned} \varphi \in \lambda_{\mathcal{D}}(K_i) = \lambda_{\mathcal{D}}(U_i) &\Rightarrow \varphi \in \lambda_{\mathcal{P}}(L_i) \Rightarrow \varphi \in \Gamma_i \Rightarrow \varphi \in \Gamma'_i \cap \Gamma''_i && \text{and} \\ \varphi \notin \lambda_{\mathcal{D}}(K_i) = \lambda_{\mathcal{D}}(U_i) &\Rightarrow \varphi \notin \lambda_{\mathcal{P}}(L_i) \Rightarrow \bar{\varphi} \in \Gamma_i \Rightarrow \bar{\varphi} \in \Gamma'_i \cap \Gamma''_i. \end{aligned}$$

In summary, this shows that  $\mathcal{D}_M = \mathcal{D}_N$  is in fact  $\varphi$ -consistent.

**Temporal next.** For formulae  $\varphi = \mathbf{X}\psi$  the APS  $\mathcal{D}_N$  is also  $\varphi$ -consistent without any modification. The states  $q \in Q_{\mathcal{P}} \setminus \{F_\ell\}$  are  $\varphi$ -consistent because their successors  $\text{succ}_{\mathcal{D}_N}(q) = \text{succ}_{\mathcal{P}}(q) \subseteq Q_{\mathcal{P}}$  are the same in  $\mathcal{D}_N$  and  $\mathcal{P}$ . They are  $\varphi$ -consistent in  $\mathcal{P}$  and that involves only the labelling of their immediate successors. The successors of the states  $F_\ell$  and  $K_\ell$  are  $\text{succ}_{\mathcal{D}_N}(F_\ell) = \{K_0\}$  and  $\text{succ}_{\mathcal{D}_N}(K_\ell) = \{K_0, U_0\}$ , respectively. These are all labelled by  $\lambda_{\mathcal{D}}(K_0) = \lambda_{\mathcal{D}}(U_0) = \lambda_{\mathcal{D}}(L_0) = \lambda_{\mathcal{P}}(L_0)$ . Thus,

$$\mathbf{X}\psi \in \lambda_{\mathcal{D}}(F_\ell) = \lambda_{\mathcal{D}}(K_\ell) \Leftrightarrow \mathbf{X}\psi \in \lambda_{\mathcal{P}}(L_\ell) \Leftrightarrow \psi \in \lambda_{\mathcal{P}}(L_0) = \lambda_{\mathcal{D}}(K_0) = \lambda_{\mathcal{D}}(U_0).$$

The successors of  $U_\ell$  is  $L_0$ . Since  $L_\ell$  is  $\varphi$ -consistent in  $\mathcal{P}$  we have

$$\mathbf{X}\psi \in \lambda_{\mathcal{D}}(U_\ell) \Leftrightarrow \mathbf{X}\psi \in \lambda_{\mathcal{P}}(L_\ell) \Leftrightarrow \psi \in \lambda_{\mathcal{P}}(L_0) \Leftrightarrow \psi \in \lambda_{\mathcal{D}}(L_0).$$

The remaining states  $K_i$  and  $U_i$  for  $i \in [0, \ell - 1]$  inherit  $\varphi$ -consistency from  $L_i$  since they have only one successor  $K_{i+1}$  and  $U_{i+1}$ , respectively, with

$$\mathbf{X}\psi \in \lambda_{\mathcal{D}}(K_i) = \lambda_{\mathcal{D}}(U_i) = \lambda_{\mathcal{D}}(L_i) \Leftrightarrow \psi \in \lambda_{\mathcal{D}}(L_{i+1}) = \lambda_{\mathcal{D}}(K_{i+1}) = \lambda_{\mathcal{D}}(U_{i+1}).$$

### 4.4.3 Until Formulae

For the formulae covered so far, the APS  $\mathcal{D}_N$  is shown to be  $M$ -consistent without any modification, based on how  $\mathcal{D}$  was constructed and  $N$ -consistency. For the case that  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi$  is an *until* formula, however, consistency does not only depend on the local labelling that is inherited from  $\mathcal{P}$  but may demand for balance counters that meet global requirements. Nevertheless, for states that are not directly affected by the duplication of  $L$ ,  $\varphi$ -consistency is preserved as will be shown first. Subsequently, the transitions  $\Delta_N$  of  $\mathcal{D}_N$  are extended by additional balance counters and transition guards such that all states of the new row  $U$  become  $\varphi$ -consistent as well while, at the same time, a run corresponding to  $\sigma_{\mathcal{D}}$  exists. Then,  $\varphi$ -consistency of the states of  $L$  and  $K$  follows by consistency of their surrounding rows.

#### States in $Q_{\mathcal{P}} \setminus L$ are $\varphi$ -consistent in $\mathcal{D}_N$

For a row state to be  $\varphi$ -consistent in  $\mathcal{D}_N$  it may be necessary that the APS includes a balance counter and specific guards. Therefore, an important observation is that the duplication of the loop  $L$  preserves all balance counters of  $\mathcal{P}$ .

► **Lemma 4.16.** *If a counter  $c \in \text{counters}(\mathcal{P})$  is a balance counter for the constraint term  $\tau$  and a state  $q \in Q_{\mathcal{P}}$  in  $\mathcal{P}$ , then it is a balance counter for  $\tau$  and  $q$  in  $\mathcal{D}_N$ .*

**Proof.** Let  $(q_1, \mu, \Gamma, q_2) \in \Delta_N$  be a transition in  $\mathcal{D}_N$ . If  $q_1 \notin \{F_\ell\} \cup K \cup U$ , then  $q_1, q_2 \in Q_{\mathcal{P}}$  and there is a corresponding transition  $(q_1, \mu', \Gamma', q_2) \in \Delta_{\mathcal{P}}$  in  $\mathcal{P}$  with  $\mu' \sqsubseteq \mu$ . Thus,  $\mu(c) = \mu'(c) = 0$  if  $q_1 \prec_{\mathcal{D}} q$  and  $\mu(c) = \mu'(c) = \llbracket \tau \rrbracket(\lambda_{\mathcal{P}}(q_1)) = \llbracket \tau \rrbracket(\lambda_{\mathcal{D}}(q_1))$  if  $q_1 \succeq_{\mathcal{D}} q$ .

Otherwise, if  $q_1 \in \{F_\ell\} \cup K \cup U$ , there is some corresponding transition  $(q'_1, \mu', \Gamma', q'_2) \in \Delta_{\mathcal{P}}$  with  $\mu' \sqsubseteq \mu$  starting at a state  $q'_1 \in L$ . Since balance counters are only defined for row states, we have  $q \in Q \setminus L$ , so either  $q \prec_{\mathcal{P}} L_0$  or  $q \succ_{\mathcal{P}} L_\ell$ . Thus,  $q \succ_{\mathcal{P}} q_1$  implies  $q \succ_{\mathcal{P}} L_\ell$  and in that case all transitions starting in states of  $L$  update  $c$  by 0, in particular  $0 = \mu'(c) = \mu(c)$ . Similarly, if  $q \preceq_{\mathcal{P}} q_1$ , we have even  $q \prec_{\mathcal{P}} L_1$  and thus all transitions starting at some state  $s \in L$  update  $c$  by  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}(s))$ , in particular  $\llbracket \tau \rrbracket(\lambda_{\mathcal{P}}(q'_1)) = \mu'(c) = \mu(c)$ .

Thus, in any case the transition  $(q_1, \mu, \Gamma, q_2) \in \Delta_N$  satisfies the requirements for  $c$  being a balance counter for  $q$  and  $\tau$ . ■

The existence of the balance counters of  $\mathcal{P}$  in  $\mathcal{D}_N$ , allows us now to prove that the conditions for consistency are preserved for the states not directly affected by the duplication of  $L$ . Let  $q \in Q_{\mathcal{P}} \setminus L$  be a state of  $\mathcal{D}_N$ . Since it is contained in  $\mathcal{P}$ , it is

$\varphi$ -consistent there, meaning one of the conditions of Definition 4.11 (D) applies. Let us consider the individual cases and show that they still apply in  $\mathcal{D}_N$ .

**Condition D1.** If condition D1 applies to  $q$  in  $\mathcal{P}$ , this is also the case in  $\mathcal{D}_N$  because  $\lambda_{\mathcal{P}}(q) = \lambda_{\mathcal{D}}(q)$  and  $\lambda_{\mathcal{P}}(\text{lastl}(\mathcal{P})) = \lambda_{\mathcal{D}}(\text{lastl}(\mathcal{D}_N))$ . Moreover, for all  $q' \in \text{succ}_{\mathcal{D}_N}^*(q)$  we find that  $\chi \in \lambda_{\mathcal{D}}(q')$  since either  $q' \in Q_{\mathcal{P}}$  and

$$q' \in Q_{\mathcal{P}} \Rightarrow q' \in \text{succ}_{\mathcal{P}}^*(q) \Rightarrow \chi \in \lambda_{\mathcal{P}}(q') \Rightarrow \chi \in \lambda_{\mathcal{D}}(q'),$$

or  $q' \in \{K_i, U_i\}$  for some  $i \in [0, \ell]$  meaning that  $q \prec_{\mathcal{P}} L_i \prec_{\mathcal{P}} q'$  and then

$$L_i \in \text{succ}_{\mathcal{P}}^*(q) \Rightarrow \chi \in \lambda_{\mathcal{P}}(L_i) \Rightarrow \chi \in \lambda_{\mathcal{D}}(q').$$

**Conditions D(2)i and D(2)ii.** Alternatively, condition D2 may apply to  $q$  in  $\mathcal{P}$ . In case  $\varphi \notin \lambda_{\mathcal{D}}(q)$ , we have  $0 < b$  or  $\psi \notin \lambda_{\mathcal{P}}(q) = \lambda_{\mathcal{D}}(q)$ . Condition D(2)i carries over from  $\mathcal{P}$  to  $\mathcal{D}_N$  because either the simple sequence of states from  $q$  up to the required *defect* state (i.e., a state not labelled by  $\chi$ ) does not intersect with  $L$  at all, or it ends before the last state of  $L$ . In any case, the sequence persists identically in  $\mathcal{D}_N$ .

If condition D(2)ii applies to  $q$  in  $\mathcal{P}$ , let  $c_{\tau, q} \in \text{counters}(\mathcal{P})$  be the required balance counter for  $\tau$  and  $q$  in  $\mathcal{P}$ . By Lemma 4.16 it is also a balance counter in  $\mathcal{D}_N$  so let us show that it qualifies to establish condition D(2)ii for  $q$  in  $\mathcal{D}_N$ . Consider any transition  $(q_1, \mu, \Gamma, q_2) \in \Delta_N$  to a state  $q_2 \succ_{\mathcal{D}} q$  with  $\psi \in \lambda_{\mathcal{D}}(q_2)$  where  $\chi \in \lambda_{\mathcal{D}}(q')$  for all  $q' \in Q_N$  with  $q \preceq_{\mathcal{D}} q' \prec_{\mathcal{D}} q_2$ . These are precisely the transitions that need to be guarded by  $c_{\tau, g} < b$ .

If  $q_2 \notin K \cup U \cup \{L_0\}$ , then  $q_1, q_2 \in Q_{\mathcal{P}}$  and there is a corresponding transition  $(q_1, \mu', \Gamma', q_2) \in \Delta_{\mathcal{P}}$  in  $\mathcal{P}$  with  $\Gamma' \subseteq \Gamma$  and condition D(2)ii applying to  $q$  provides that  $(c_{\tau, q} < b) \in \Gamma'$ . If otherwise  $q_2 \in K \cup U \cup \{L_0\}$ , then  $q_1 \in \{F_{\ell}, L_{\ell}\} \cup K \cup U$  and there is a transition  $(q'_1, \mu', \Gamma', q'_2) \in \Delta_{\mathcal{P}}$  with  $\Gamma' \subseteq \Gamma$ ,  $\lambda_{\mathcal{P}}(q'_1) = \lambda_{\mathcal{D}}(q_1)$ ,  $\lambda_{\mathcal{P}}(q'_2) = \lambda_{\mathcal{D}}(q_2)$ ,  $q'_1 \in \{F_{\ell}\} \cup L$ , and  $q'_2 \in L$ . Observe that

- $\psi \in \lambda_{\mathcal{P}}(q'_2)$  because  $\lambda_{\mathcal{P}}(q'_2) = \lambda_{\mathcal{D}}(q_2)$ ,
- $q \prec_{\mathcal{P}} L_0 \preceq_{\mathcal{P}} q'_2$  because  $q \prec_{\mathcal{D}} q_2 \preceq_{\mathcal{D}} q'_2$  and  $q \notin L$ , and
- all  $q' \in Q_{\mathcal{P}}$  with  $q \preceq_{\mathcal{P}} q' \prec_{\mathcal{P}} q'_2$  are labelled by  $\chi$  because there is a corresponding state  $q''$  with  $q \preceq_{\mathcal{D}} q'' \prec_{\mathcal{D}} q_2$  and  $\lambda_{\mathcal{D}}(q'') = \lambda_{\mathcal{P}}(q')$  and these are all labelled by  $\chi$ .

From the assumption that condition D(2)ii applies to  $q$  in  $\mathcal{P}$  we conclude that  $(c_{\tau, q} < b) \in \Gamma'$  and the condition hence applies in  $\mathcal{D}_N$ .

**Conditions D(2)iii and D(2)iv.** In case  $\varphi \in \lambda_N(q)$ , condition D2 demands the existence of a witness state  $q' \succeq q$  from  $Q_{\mathcal{P}}$ . Inserting  $K$  and  $U$  preserves the conditions applying to it.

If  $q' \prec_{\mathcal{P}} L_0$  or  $q' \succ_{\mathcal{P}} L_\ell$ , the condition is not affected by the insertion at all. All intermediate states remain in place and a potentially required balance counter persists (Lemma 4.16) with the updates and guards unchanged in between  $q$  and  $q'$ . The same holds if  $q' \in L$  because then there is a corresponding witness state  $q'' \in K$  in  $\mathcal{D}_N$  that shares precisely the same properties as  $q'$  in  $\mathcal{P}$ .

Consider thus the remaining case that  $q \prec_{\mathcal{P}} L_0$  (thus  $q \prec_{\mathcal{D}} K_0$ ) and  $q' \succ_{\mathcal{P}} L_\ell$ . Then, the loop  $L$  is necessarily entirely labelled by  $\chi$  and this is hence also the case for  $K$  and  $U$  in  $\mathcal{D}_N$ . Further, only condition D(2)iv can apply to  $q$  in  $\mathcal{P}$  and the required balance counter is still present in  $\mathcal{D}_N$ .

**Condition D3.** The remaining option is that  $q$  is on some loop  $L' \in Q_{\mathcal{P}}^+$  in  $\mathcal{P}$  (and  $\mathcal{D}_N$ ) and condition D3 applies to  $q$  in  $\mathcal{P}$ . Then, also the identically labelled rows preceding and (unless  $L' = \text{lastl}(\mathcal{P})$ ) succeeding  $L'$  in  $\mathcal{P}$  are present in  $\mathcal{D}_N$  because, being rows, they cannot intersect with  $L$  and are not affected by the insertion of  $K$  and  $U$ . Following the previous arguments, the states on these rows are  $\varphi$ -consistent in  $\mathcal{D}_N$  and hence  $q$  is  $\varphi$ -consistent in  $\mathcal{D}_N$ .

### States of the Row $U$

So far, we have considered states not directly affected by the duplication modification and observed that these remain to satisfy the consistency conditions. In the following, let us thus turn to the remaining states, namely those that have been inserted into the structure of  $\mathcal{P}$ . Specifically, the states of the fresh row  $U$  may require us to introduce corresponding balance counters. To this end, we analyse the possible situations of a state of  $U$  and show that in all cases consistency can be established while maintaining a run corresponding to  $\sigma$ .

First of all, recall that  $st(\sigma_N) = st(\sigma_{\mathcal{D}})$  and  $\text{org}(\sigma) = \text{org}_{\mathcal{D}}(\sigma_{\mathcal{D}}) = \text{org}_{\mathcal{D}}(\sigma_N)$ . Therefore, the runs  $\sigma_N$  and  $\sigma$  are equivalent with respect to the satisfaction of any formula  $\varphi \in \text{sub}(\Phi)$  in the sense that for all positions  $i \in \mathbb{N}$

$$(\mathcal{P}, \sigma, i) \models \varphi \iff (\mathcal{D}_N, \sigma_N, i) \models \varphi.$$

From the definition of  $\lambda_{\mathcal{D}}$  and  $\sigma_{\mathcal{D}}$  and the fact that  $\mathcal{P}$  is  $M$ -consistent it follows that for

every formula  $\varphi \in M$

$$\varphi \in \lambda_{\mathcal{D}}(st(\sigma_{\mathcal{D}}(i))) \Leftrightarrow \varphi \in \lambda(st(\sigma(i))) \Leftrightarrow (\mathcal{P}, \sigma, i) \models \varphi.$$

The induction provides the invariant that the state sequence  $st(\sigma_N) = st(\sigma_{\mathcal{D}})$  does not change, so neither does the labelling sequence and we conclude that the labelling of  $\sigma_N$  corresponds to the semantics of formulae from  $\varphi \in M$ , i.e., for all  $i \in \mathbb{N}$

$$\varphi \in \lambda_{\mathcal{D}}(st(\sigma_N(i))) \Leftrightarrow (\mathcal{D}_N, \sigma_N, i) \models \varphi.$$

Given this latter precondition, we can show that any row state of  $\mathcal{D}_N$  can be made consistent by only adding a counter and guards for specific transitions.

► **Lemma 4.17.** *Let  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi \in \text{cLTL}$ ,  $N \supseteq \text{sub}(\varphi) \setminus \{\varphi\}$ ,  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  an  $N$ -consistent APS in  $\mathcal{S}$ ,  $r \in Q$  a row state, and  $\sigma \in \text{runs}(\mathcal{P})$  such that, for all  $i \in \mathbb{N}$ ,  $\varphi \in \lambda(st(\sigma(i)))$  if and only if  $(\mathcal{P}, \sigma, i) \models \varphi$ . Then there is an APS  $\hat{\mathcal{P}} = (Q, \hat{\Delta}, \lambda, \text{org})$  and  $\hat{\sigma} \in \text{runs}(\hat{\mathcal{P}})$  such that*

- $\hat{\Delta} = \{\delta'_0, \dots, \delta'_k\}$  where  $\Delta = \{\delta_0, \dots, \delta_k\}$  and  $\delta'_i = (s, \mu', \Gamma', t)$  for  $\delta_i = (s, \mu, \Gamma, t)$ , some  $\mu \sqsubseteq \mu'$ ,  $\Gamma \subseteq \Gamma'$ , and all  $i \in [0, k]$ ,
- $r$  is  $\varphi$ -consistent in  $\hat{\mathcal{P}}$ ,
- $st(\hat{\sigma}) = st(\sigma)$ , and
- $|\hat{\mathcal{P}}| \leq |\mathcal{P}| + |\Delta| \cdot |\varphi|$ .

**Proof.** Let  $i_r \in \text{pos}_{\sigma}(r)$  be the unique position of the row state  $r$  on  $\sigma$ . Recall that  $(\mathcal{P}, \sigma, i_r) \models \varphi$  if and only if there is a witness position  $i \geq i_r$  such that  $(\mathcal{P}, \sigma, i) \models \psi$ ,  $\llbracket \tau \rrbracket(\#_{i_r, i-1}^{\mathcal{P}, \sigma}) \geq b$ , and  $(\mathcal{P}, \sigma, j) \models \chi$  for all  $j \in [i_r, i-1]$ . Given  $N$ -consistency and  $N \supseteq \text{sub}(\varphi) \setminus \{\varphi\}$ , Theorem 4.12 provides that

$$\begin{aligned} (\mathcal{P}, \sigma, i) \models \psi &\Leftrightarrow \psi \in \lambda(\sigma(i)) \\ \llbracket \tau \rrbracket(\#_{i_r, i-1}^{\mathcal{P}, \sigma}) &= \llbracket \tau \rrbracket(\lambda^{\#}(\sigma(i_r)\sigma(i_r+1)\dots\sigma(i-1))) \\ (\mathcal{P}, \sigma, j) \models \chi &\Leftrightarrow \chi \in \lambda(\sigma(j)) \end{aligned}$$

for  $j \in [i_r, i-1]$ .

*Case 1:*  $(\mathcal{P}, \sigma, i_r) \models \varphi$ . Assume first that  $(\mathcal{P}, \sigma, i_r) \models \varphi$  and consider a corresponding witness position  $i_q \geq i_r$  carrying some state  $q = st(\sigma(i_q))$ . If  $i_q = i_r$ , then

$$b \leq \llbracket \tau \rrbracket(\#_{i_r, i_r-1}^{\mathcal{P}, \sigma}) = 0 = \llbracket \tau \rrbracket(\lambda^{\#}(\varepsilon))$$

and thus, condition D(2)iii of Definition 4.11 applies to  $r$  without any necessary modification. Assume therefore that  $i_q > i_r$  in the following.

*1a) Witness state not on loop.* Consider the case that  $q$  is not on a loop. To obtain  $\hat{\Delta}$ , we extend  $\mathcal{P}$  by a fresh balance counter  $c_{\tau,r}$  for  $\tau$  and  $r$  and add the guard  $c_{\tau,r} \geq b$  to the guard set  $\Gamma$  of the (unique) incoming transition  $(q', \mu, \Gamma, q) \in \Delta$  of  $q$ . This does not affect consistency for any other formula or state. Moreover,  $r$  is now  $\varphi$ -consistent in  $\hat{\mathcal{P}} = (Q, \hat{\Delta}, \lambda, \text{org})$  by condition D(2)iv of Definition 4.11 since  $q \succeq_{\mathcal{P}} r$ ,  $\psi \in \lambda(q)$ , and  $\chi \in \lambda(q')$  for all  $q'$  with  $r \preceq_{\mathcal{P}} q' \prec_{\mathcal{P}} q$  since  $\text{pos}_{\sigma}(q') \subseteq [i_r, i_q - 1]$ . Let  $\hat{\sigma}$  be the run that is identical to  $\sigma$  but assigns the appropriate value to  $c_{\tau,r}$  at every position, as uniquely determined by the updates of  $c_{\tau,r}$  on each transition. It satisfies the additional constraint at the (only) necessary position  $i_q$  because

$$\begin{aligned} \llbracket \tau \rrbracket (\#_{i_r, i_q-1}^{\hat{\mathcal{P}}, \hat{\sigma}}) &= \llbracket \tau \rrbracket (\lambda^{\#}(\hat{\sigma}(i_r)\hat{\sigma}(i_r+1) \dots \hat{\sigma}(i_q-1))) \\ &= \llbracket \tau \rrbracket (\lambda^{\#}(\hat{\sigma}(i_r)\hat{\sigma}(i_r+1) \dots \hat{\sigma}(i_q-1))) \\ &= \llbracket \tau \rrbracket (\#_{i_r, i_q-1}^{\mathcal{P}, \sigma}) \geq b \end{aligned}$$

by the semantic accuracy of the labelling  $\lambda$  and Lemma 4.9.

*1b) Witness on non-positive loop.* Assume now that  $q \in L$  is part of some loop  $L$  of  $\mathcal{P}$  (and  $\hat{\mathcal{P}}$ ) and let  $\tau_L := \llbracket \tau \rrbracket (\lambda^{\#}(L))$  be the effect of  $L$  on the value of the constraint term  $\tau$ . If  $\tau_L \leq 0$ , consider the first position  $i'_q := \min \text{pos}_{\sigma}(q) > i_r$  of  $q$  on  $\sigma$ . Then,  $i_q = i'_q + n|L|$  for some  $n \geq 0$  and  $i'_q$  witnesses the satisfaction of  $\varphi$  at  $i_r$  just as  $i_q$  because

$$\begin{aligned} \llbracket \tau \rrbracket (\lambda^{\#}(\sigma(i_r) \dots \sigma(i'_q-1))) &= \llbracket \tau \rrbracket (\lambda^{\#}(\sigma(i_r) \dots \sigma(i_q-1))) \\ &\quad - \llbracket \tau \rrbracket (\lambda^{\#}(\sigma(i'_q) \dots \sigma(i'_q + n|L| - 1))) \\ &= \llbracket \tau \rrbracket (\lambda^{\#}(\sigma(i_r) \dots \sigma(i_q-1))) - n \cdot \tau_L \\ &\geq \llbracket \tau \rrbracket (\lambda^{\#}(\sigma(i_r) \dots \sigma(i_q-1))) \\ &\geq b. \end{aligned}$$

Assume therefore without loss of generality that  $i_q = i'_q = \min \text{pos}_{\sigma}(q)$ .

If none of the states at the positions  $i_r, i_r+1, \dots, i_q$  is the end of a loop, condition D(2)iii applies to  $r$  because these states are all labelled by  $\chi$  and the state sequence

$$st(\sigma(i_r)\sigma(i_r+1) \dots \sigma(i_q-1))$$

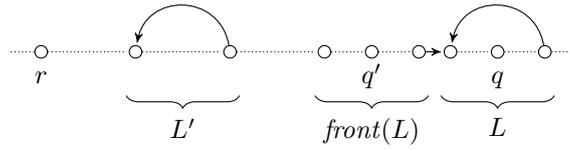


Figure 4.6: Sketch of an augmented path schema with witness state  $q$  situated on a loop  $L$ . If  $L$  has a non-positive effect on the constraint term  $\tau$ , there is also an alternative witness  $q'$  on the front row of  $L$ . Any loop  $L'$  ending between  $r$  and  $q$  must begin and end after  $r$  and before  $q'$ .

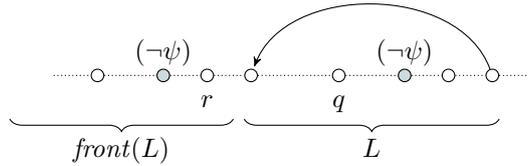


Figure 4.7: Sketch of an augmented path schema with a defect state ( $\odot$ ) on a loop  $L$  and a copy of the state occurring on the preceding row. If the defect cannot occur in-between the row state  $r$  and the loop state  $q$ , then it must precede  $r$ .

is precisely the simple path from  $r$  to  $q$  in the graph of  $\mathcal{P}$ . Otherwise, if one of the states has two successors and is thus the end of some loop  $L'$ , then  $i_q - |L| \geq i_r$  because  $L'$  necessarily starts and ends between the positions  $i_r$  and  $i_q - |L|$ , the latter being situated on the front row of  $L$  (cf. Figure 4.6). Therefore,  $q' := st(\sigma(i_q - |L|))$  is identically labelled as  $q$  by  $\lambda(q') = \lambda(q)$ . Hence, the case for row states discussed above applies to  $q'$ .

*1c) Witness on positive loop.* Assume  $\tau_L > 0$  and some state of  $L$  is not labelled by  $\chi$ . Then,  $i_q = \min pos_\sigma(q)$ , i.e.,  $i_q$  is necessarily on the first iteration of  $L$  and thus the first occurrence of  $q$  on  $\sigma$ . Further,  $i_r > i_q - |L|$  since otherwise a state not labelled by  $\chi$  would occur in between as depicted in Figure 4.7. Hence, the state sequence  $st(\sigma(i_r)\sigma(i_r + 1) \dots \sigma(i_q - 1))$  is again precisely the simple path between  $r$  and  $q$  in the graph of  $\mathcal{P}$  and condition D(2)iii applies to  $r$ .

In case all states of  $L$  are labelled by  $\chi$  and  $L$  is the last loop of  $\mathcal{P}$ , condition D1 applies. Otherwise,  $L$  is traversed finitely often and there exists an identically labelled position  $i'_q := \max pos_\sigma(q) + |L|$  on  $\sigma$  with  $i'_q = i_q + n|L|$  for some  $n \geq 1$ . The state  $q' = st(\sigma(i'_q))$  is the copy of  $q$  on the rear row  $R = rear_{\mathcal{P}}(L)$  following  $L$  in  $\mathcal{P}$  with  $\lambda(R) = \lambda(L)$ . All

the states between  $q$  and  $q'$  are then labelled by  $\chi$ ,  $\psi \in \lambda(q')$  and

$$\begin{aligned}
 \llbracket \tau \rrbracket (\lambda^\#(\sigma)(i_r) \dots \sigma(i'_q - 1)) &= \llbracket \tau \rrbracket (\lambda^\#(\sigma)(i_r) \dots \sigma(i_q - 1)) \\
 &\quad + \llbracket \tau \rrbracket (\lambda^\#(\sigma)(i_q) \dots \sigma(i_q + n|L| - 1)) \\
 &= \llbracket \tau \rrbracket (\lambda^\#(\sigma)(i_r) \dots \sigma(i_q - 1)) + n \cdot \tau_L \\
 &> \llbracket \tau \rrbracket (\lambda^\#(\sigma)(i_r) \dots \sigma(i_q - 1)) \\
 &\geq b.
 \end{aligned}$$

The row position  $i'_q$  and the corresponding row state  $q'$  thus serve as an alternative witness for the satisfaction of  $\varphi$  at position  $i_r$  and state  $r$ , respectively, and the construction for witnessing row states can be applied to establish  $\varphi$ -consistency of  $r$ .

*Case 2:*  $(\mathcal{P}, \sigma, i_r) \not\models \varphi$ . Assume now that  $(\mathcal{P}, \sigma, i_r) \not\models \varphi$  (and thus  $\varphi \notin \lambda(r)$ ). Clearly,  $\psi \notin \lambda(r)$  or  $0 < b$  as required by conditions D(2)i and D(2)ii. If condition D(2)i applies, no further action is required so assume henceforth this is not the case. As above, let us extend  $\mathcal{P}$  by a fresh balance counter  $c_{\tau,r}$  for  $\tau$  and  $r$ . Further, every transition  $(q', \mu, \Gamma, q) \in \Delta$  pointing to any state  $q \succ_{\mathcal{P}} r$  labelled by  $\psi \in \lambda(q)$  is modified by adding the constraint  $(c_{\tau,r} < b)$  to  $\Gamma$ , unless there is some state  $q''$  with  $r \preceq_{\mathcal{P}} q'' \prec_{\mathcal{P}} q$  and  $\chi \notin \lambda(q')$ . This defines the transitions  $\hat{\Delta}$  and implements condition D(2)ii for  $r$  in the obtained APS  $\hat{\mathcal{P}} = (Q, \hat{\Delta}, \lambda, \text{org})$ .

Let again  $\hat{\sigma}$  be identical to  $\sigma$  except for assigning the proper value to  $c_{\tau,r}$  as determined by the updates at every position. That is,  $st(\hat{\sigma}) = st(\sigma)$  and  $val(\sigma(i)) \sqsubseteq val(\hat{\sigma}(i))$  for all  $i \in \mathbb{N}$ . It remains to show that  $\hat{\sigma}$  is still valid in  $\hat{\mathcal{P}}$ , i.e., at every position, all relevant guards are satisfied. While this is the case for guards already present in  $\mathcal{P}$  because these are satisfied by  $\sigma$ , we have to show that for every position  $i \geq 1$  where the corresponding transition  $(st(\hat{\sigma}(i-1)), \mu, \Gamma, st(\hat{\sigma}(i))) \in \Delta'$  contains the new constraint  $(c_{\tau,r} < b) \in \Gamma$ , the valuation  $val(\hat{\sigma}(i))$  satisfies the constraint. Since only states occurring after  $r$  are guarded that way, only the positions  $i > i_r$  are relevant and for those we have by Lemma 4.9 that

$$val(\hat{\sigma}(i))(c_{\tau,r}) = \llbracket \tau \rrbracket (\#_{i_r, i-1}^{\hat{\mathcal{P}}, \hat{\sigma}}) = \llbracket \tau \rrbracket (\lambda^\#(\hat{\sigma}(i_r) \dots \hat{\sigma}(i-1))).$$

Notice further that  $\llbracket \tau \rrbracket (\#_{i_r, i-1}^{\hat{\mathcal{P}}, \hat{\sigma}}) = \llbracket \tau \rrbracket (\#_{i_r, i-1}^{\mathcal{P}, \sigma})$  since  $\tau$  refers only to strict subformulae of  $\varphi$ .

Only states labelled by  $\psi$  have been guarded and thus  $(\mathcal{P}, \sigma, i) \models \psi$ . If now  $(\mathcal{P}, \sigma, j) \models \chi$  for all  $j \in [i_r, i-1]$ , then  $\llbracket \tau \rrbracket (\#_{i_r, i-1}^{\mathcal{P}, \sigma}) < b$  as otherwise  $(\mathcal{P}, \sigma, i_r) \models \varphi$  and this was assumed is not to be the case. The only remaining alternative to be considered is that there

is some defect position  $j \in [i_r, i - 1]$  where  $(\mathcal{P}, \sigma, j) \not\models \chi$  and hence  $\chi \notin \lambda(\sigma(j))$ . It leads, however, to the following contradiction. Assume there is such a position and let  $q := st(\sigma(i))$  and  $q' := st(\sigma(j))$ . Although  $i > j$ , the state  $q$  must come first in  $\mathcal{P}$  and  $\hat{\mathcal{P}}$  ( $q \preceq_{\mathcal{P}} q'$ ) since otherwise  $q$  would not have been guarded in the first place. The only possibility for that to happen is that both states are part of the same loop  $L$ , in a situation as depicted in Figure 4.7. Notice that there cannot be a defect state (one that is not labelled by  $\chi$ ) between  $r$  and  $q$ , so  $r$  must be part of the row immediately preceding  $L$  and  $q'$  cannot be the last state of  $L$ . Consider, thus, the position  $i' := \min pos_{\sigma}(q)$ , being the first occurrence of  $q$  after  $r$ . In between those positions, no defect occurs and therefore  $\llbracket \tau \rrbracket (\#_{i_r, i'-1}^{\mathcal{P}, \sigma}) < b$  as otherwise  $\varphi$  would hold at  $i_r$ . This implies that  $r$  in fact obeys condition D(2)i, but we have excluded this case above because, then, adding the balance counter and the guards would not have been necessary.

*Size of the modification.* Modifying the augmented path schema  $\mathcal{P}$  for one such state  $r$  means adding at most one fresh balance counter  $c_{\tau, r}$  that is to be updated on each transition  $(q, \mu, \Gamma, q') \in \Delta$  succeeding  $r$  by the value (0 or)  $\llbracket \tau \rrbracket (\lambda^{\#}(q))$ . At the same time, the guard set of each such transition is potentially extended by one constraint  $\gamma_{\tau, r}$  of either the form  $c_{\tau, r} \geq b$  or  $c_{\tau, r} < b$ . Recall that  $c_{\tau, r} < b$  abbreviates  $-c_{\tau, r} \geq -b + 1$  and thus  $|\gamma_{\tau, r}| \leq |c_{\tau, r} \geq b| + 1$ . Further, for any transition  $(q, \mu, \Gamma, q') \in \hat{\Delta}$ , the size of the update  $size(\mu(c_{\tau, r}))$  is bounded by the size of  $\tau$  because for  $\tau = a_0\varphi_0 + \dots + a_k\varphi_k$

$$\begin{aligned} size(\mu(c_{\tau, r})) &\leq size(|a_0| + \dots + |a_k|) \\ &\leq size(|a_0|) + \dots + size(|a_k|) \\ &\leq |\tau|. \end{aligned}$$

Hence, the size of  $\mathcal{P}$  increases to at most

$$\begin{aligned} |\hat{\mathcal{P}}| &\leq |\mathcal{P}| + \sum_{(q, \mu, \Gamma, q') \in \hat{\Delta}} |\gamma_{\tau, r}| + size(\mu(c_{\tau, r})) \\ &\leq |\mathcal{P}| + |\Delta| \cdot (|\gamma_{\tau, r}| + |\tau|) \\ &\leq |\mathcal{P}| + |\Delta| \cdot (|c_{\tau, r} \geq b| + 1 + |\tau|) \\ &\leq |\mathcal{P}| + |\Delta| \cdot (|\tau \geq b| + 1) \\ &\leq |\mathcal{P}| + |\Delta| \cdot |\varphi|. \end{aligned}$$

■

### The $\varphi$ -consistent APS $\mathcal{D}_M$

Returning now to the APS  $\mathcal{D}_N$ , observe that Lemma 4.17 above can be applied consecutively to each of the states of  $U$  effectively providing the APS  $\mathcal{D}_M$ . We have seen that all states that do neither belong to  $K$ , nor  $U$ , nor  $L$  are in fact  $\varphi$ -consistent in  $\mathcal{D}_N$  and thus in  $\mathcal{D}_M$  because only counters and guards are added. From the consistency of all row states in  $\mathcal{D}_M$ , also the consistency of all loop states follows by condition D3 of Definition 4.11. The run  $\sigma_N \in \text{runs}(\mathcal{D}_N)$  is preserved in terms of an almost identical run  $\sigma_M \in \text{runs}(\mathcal{D}_M)$  that additionally evaluates the balance counters that have potentially been added. Applying Lemma 4.17 increases the size of the obtained APS by at most  $|\Delta_N| \cdot |\varphi| = |\Delta_{\mathcal{D}}| \cdot |\varphi|$  and thus, by using the induction hypothesis we obtain

$$\begin{aligned} |\mathcal{D}_M| &\leq |\mathcal{D}_N| + |U| \cdot |\Delta_N| \cdot |\varphi| \\ &\leq |\mathcal{D}| + |L| \cdot |\Delta_{\mathcal{D}}| \cdot \left( \sum_{\varphi' \in N} |\varphi'| \right) + |L| \cdot |\Delta_{\mathcal{D}}| \cdot |\varphi| \\ &\leq |\mathcal{D}| + |L| \cdot |\Delta_{\mathcal{D}}| \cdot \left( \sum_{\varphi' \in M} |\varphi'| \right) \end{aligned}$$

and thereby finish the induction.

#### 4.4.4 Conclusion and Corollaries

The induction has shown that for any closed set  $M = \text{sub}(M)$  we can construct an  $M$ -consistent APS  $\mathcal{D}_M$  that only adds additional counters and guards to  $\mathcal{D}$ . Further it preserves a run  $\sigma_M$  that has the same shape as  $\sigma_{\mathcal{D}}$ . Therefore,  $\mathcal{D}_M$  satisfies all claims stated in Lemma 4.15 and constitutes  $\hat{\mathcal{P}}$ . We confirm that its size is bounded by

$$\begin{aligned} |\hat{\mathcal{P}}| = |\mathcal{D}_M| &\leq |\mathcal{D}| + |L| \cdot |\Delta_{\mathcal{D}}| \cdot \left( \sum_{\varphi' \in M} |\varphi'| \right) \\ &\leq |\mathcal{P}| + 3 \cdot \text{size}(\Delta_L) + |L| \cdot (|\Delta_{\mathcal{P}}| + 2|L| + 1) \cdot \left( \sum_{\varphi' \in M} |\varphi'| \right). \end{aligned}$$

Observe that the fact, that the duplication introduced an additional loop  $K$  has not been used for showing that some state is consistent or can be made so. Leaving out the loop  $K$  still allows for using the very same arguments and therefore we can conclude that only *unfolding* a loop  $L$  is possible in the same way. The base schema  $\mathcal{D}$  would only introduce a single row  $U$  as copy of  $L$  and then be precisely of size  $|\mathcal{D}| = |\mathcal{P}| + \text{size}(\Delta)$  with  $|\Delta_{\mathcal{D}}| = |\Delta| + |L|$  transitions.

► **Corollary 4.18.** *Let  $M \subseteq \text{cLTL}$  and  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  be an  $M$ -consistent APS in  $\mathcal{S}$*

with  $L \in \text{loops}(\mathcal{P})$  and  $\sigma \in \text{runs}(\mathcal{P})$  such that  $st(\sigma) = uL^2w$  for some  $u \in (Q \setminus L)^*$  and  $w \in Q^\omega$ . There is an  $M$ -consistent APS  $\hat{\mathcal{P}} = (Q \dot{\cup} U, \hat{\Delta}, \hat{\lambda}, \widehat{\text{org}})$  with additional row  $U$  of length  $|U| = |L|$  and run  $\hat{\sigma}$  such that

- $\text{loops}(\hat{\mathcal{P}}) = \text{loops}(\mathcal{P})$ ,
- $st(\hat{\sigma}) = uULw$ ,
- $\widehat{\text{org}}(\hat{\sigma}) = \text{org}(\sigma)$ , and
- $|\hat{\mathcal{P}}| \leq |\mathcal{P}| + \text{size}(\Delta_L) + |L| \cdot (|\Delta| + |L|) \cdot (\sum_{\varphi' \in M} |\varphi'|)$ .

The construction would not change much either if the additional copy were introduced after the original loop  $L$ . Notice in particular, that Lemma 4.17 does not take the actual location of  $U$  into account. Given that  $L$  is not the last loop of  $\mathcal{P}$  we can therefore conclude that  $L$  can also be unfolded towards the end of the schema.

► **Corollary 4.19.** *Let  $M \subseteq \text{cLTL}$  and  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  be an  $M$ -consistent APS in  $\mathcal{S}$  with  $L \in \text{loops}(\mathcal{P})$  and  $\sigma \in \text{runs}(\mathcal{P})$  such that  $st(\sigma) = uL^2w$  for some  $u \in Q^*$ , and  $w \in (Q \setminus L)^\omega$ . There is an  $M$ -consistent APS  $\hat{\mathcal{P}} = (Q \dot{\cup} U, \hat{\Delta}, \hat{\lambda}, \widehat{\text{org}})$  with additional row  $U$  of length  $|U| = |L|$  and run  $\hat{\sigma}$  such that*

- $\text{loops}(\hat{\mathcal{P}}) = \text{loops}(\mathcal{P})$ ,
- $st(\hat{\sigma}) = uLUw$ ,
- $\widehat{\text{org}}(\hat{\sigma}) = \text{org}(\sigma)$ , and
- $|\hat{\mathcal{P}}| \leq |\mathcal{P}| + \text{size}(\Delta_L) + |L| \cdot (|\Delta| + |L|) \cdot (\sum_{\varphi' \in M} |\varphi'|)$ .

Finally, replacing  $L$  by  $U$  would also only require establishing consistency for  $U$ , which is possible using the same arguments as before. Removing  $L$  does not affect consistency of other states because, as was discussed in the proof of Lemma 4.17, if some state of  $L$  witnesses the satisfaction of some formula at some earlier state, then there is also an alternative witness on either its front or rear row. These remain untouched when  $L$  is cut.

► **Corollary 4.20.** *Let  $M \subseteq \text{cLTL}$  and  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  be an  $M$ -consistent APS in  $\mathcal{S}$  with  $L \in \text{loops}(\mathcal{P})$  and  $\sigma \in \text{runs}(\mathcal{P})$  such that  $st(\sigma) = uLw$  for some  $u \in (Q \setminus L)^*$ , and  $w \in (Q \setminus L)^\omega$ . There is an  $M$ -consistent APS  $\hat{\mathcal{P}} = (Q, \hat{\Delta}, \hat{\lambda}, \widehat{\text{org}})$  run  $\hat{\sigma}$  such that*

- $\text{loops}(\hat{\mathcal{P}}) = \text{loops}(\mathcal{P}) \setminus \{L\}$ ,

- $st(\hat{\sigma}) = uLw$ ,
- $\widehat{\text{org}}(\hat{\sigma}) = \text{org}(\sigma)$ , and
- $|\hat{\mathcal{P}}| \leq |\mathcal{P}| + |L| \cdot |\Delta| \cdot (\sum_{\varphi' \in M} |\varphi'|)$ .

## 4.5 Existence of Consistent APS

Although consistency may appear to impose quite strong restrictions on the shape of witnesses that can be identified, they exist for a significant class of systems. Kutzt and Finkbeiner [KF11] showed that witnesses in terms of path schemas always exist for runs of Kripke structures that satisfy some LTL formula. Demri, Dhar and Sangnier [DDS12; DDS15] extended this result to LTL with past-time operators and flat counter systems. This section is concerned with extending these results to cLTL and augmented path schemas by proving the following result.

► **Theorem 4.21** (Existence). *If a flat counter system  $\mathcal{S}$  satisfies a cLTL formula  $\Phi$ , then there is a  $\Phi$ -consistent APS  $\mathcal{P}_\Phi$  in  $\mathcal{S}$  with  $\mathcal{P}_\Phi \models \Phi$  and  $|\mathcal{P}_\Phi| \in 2^{\mathcal{O}(|\Phi|^2 \cdot |S|)}$ .*

The theorem settles the last remaining aspect of the decision procedure defined at the beginning of this chapter in Section 4.1, namely its completeness. Assuming that  $\mathcal{S} \models \Phi$  there is a run  $\rho \in \text{runs}(\mathcal{S})$  satisfying  $\Phi$  and that run induces a path schema  $\mathcal{P}_\rho = (Q_\rho, \Delta_\rho, \lambda_\rho, \text{org}_\rho)$  in  $\mathcal{S}$  representing it.

It is known that each path in a flat structure can be represented by some path schema of linear size [Bar+05; DDS15]. Since  $\mathcal{S}$  is flat,  $\rho$  traverses a series of disjoint loops  $v_0, \dots, v_m \in S^+$  of  $\mathcal{S}$ , connected by disjoint (and possibly empty) simple paths  $u_1, \dots, u_m \in S^*$ . Thus, the traversed state sequence has the form

$$st(\rho) = u_0 v_0^{n_0} u_1 v_1^{n_1} \dots u_{m-1} v_{m-1}^{n_{m-1}} u_m v_m^\omega$$

where  $u_0 \in S^*$  is the simple path connecting the initial state to the first state that is repeated. Technically, let  $s_0$  be the first state repeated on  $\rho$  and  $v_0$  be the unique simple loop starting in  $s_0$ . Then, let  $s_1$  be the first repeated state on  $\rho$  that does not also belong to  $v_0$  but to a disjoint loop  $v_1$  and so forth. As the path segments are disjoint, their combined lengths  $|u_0 \dots u_m|$  and  $|v_0 \dots v_m|$  are both bounded by the number of states  $|S|$  and thus the sequence  $w := u_0 v_0 \dots u_m v_m$  is of length  $|w| \leq 2|S|$ .

This sequence defines a counter system consisting of one state  $q_i$  for each of the positions  $i \in [0, |w| - 1]$  and where each segment loop  $u_i$  constitutes again a simple loop. As depicted in Figure 4.8, an APS  $\mathcal{P}_\rho$  in  $\mathcal{S}$  can then be constructed that represents the

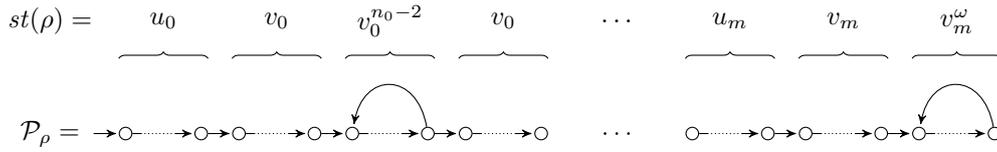


Figure 4.8: An augmented path schema constructed from a run  $\rho$ . The first and last iterations of each loop is represented explicitly. For loops  $v_i$  that are traversed only  $n_i < 3$  times, the corresponding loop in  $\mathcal{P}_\rho$  is left out entirely. If further  $n_i < 2$ , also one of the corresponding rows is skipped. The origin  $\text{org}(q_i)$  of each state  $q_i$  is defined to be the state of  $\mathcal{S}$  at the corresponding position on  $st(\rho)$  and the labelling of  $q_i$  is inherited from  $\text{org}(q_i)$ . The transitions  $(q, \mu, \Gamma, q') \in \Delta_{\mathcal{P}}$  are copies of the transitions  $(\text{org}(q), \mu, \Gamma, \text{org}(q')) \in \Delta_{\mathcal{S}}$  taken by  $\rho$ .

run  $\rho$ . The path schema is essentially the subgraph of  $\mathcal{S}$  traversed by  $\rho$  where loops may be unfolded partially to obtain the required, degenerated shape of the graph (see also Figure 4.2). The number of counters and guards as well as the size of guards and updates are directly inherited from  $\mathcal{S}$ . The APS is comprised of at most  $4|\mathcal{S}|$  states and  $4|\Delta_{\mathcal{S}}|$  transitions: Up to two copies of any state and transition from  $\mathcal{S}$  occur on the counter system defined by  $w$  and two further copies may be introduced while unfolding a loop to provide the front and rear row required by the definition of APS. Guards and updates of transitions in  $\mathcal{S}$  remain untouched and thus the size of  $\mathcal{P}_\rho$ , that is, the representation size of its transition set, is bounded by  $|\mathcal{P}_\rho| \leq 4|\mathcal{S}|$ .

In the following we show, that the labelling of  $\mathcal{P}_\rho$  can be extended incrementally by subformulae of  $\Phi$  until obtaining an APS  $\mathcal{P}_\Phi$  that is  $\Phi$ -consistent. In every step the invariant is maintained that the APS remains consistent with respect to all previously considered formulae and still satisfies  $\Phi$ .

### 4.5.1 Induction Scheme

The existence of  $\mathcal{P}_\Phi$  is shown using well-founded induction over the *downward-closed subsets* of  $\text{sub}(\Phi)$ , that is, the sets  $M = \text{sub}(M) \subseteq \text{sub}(\Phi)$ . Recall that such a class of sets was used for the induction in Section 4.4 and that it is well-founded with respect to the subset ordering  $\subseteq$  and has the unique minimal element  $\emptyset$ . An APS is  $\Phi$ -consistent if and only if it is consistent with respect to all subformulae  $\text{sub}(\Phi)$  and thus the following technical proof goal will be used for the induction and implies the existence of  $\mathcal{P}_\Phi$ : For every set  $M = \text{sub}(M) \subseteq \text{sub}(\Phi)$  there exists an APS  $\mathcal{P}_M = (Q_M, \Delta_M, \lambda_M, \text{org}_M)$  in  $\mathcal{S}$  with the following properties.

1.  $\mathcal{P}_M$  is  $M$ -consistent.
2.  $\mathcal{P}_M \models \Phi$  contains a run satisfying  $\Phi$ .
3. The set of states  $Q_M$  of  $\mathcal{P}_M$  is of bounded cardinality

$$|Q_M| \leq (\hat{n} + 4)^{|M|} \cdot |Q_\rho|.$$

4. The set of transitions  $\Delta_M$  of  $\mathcal{P}_M$  is of bounded cardinality

$$|\Delta_M| \leq (\hat{n} + 4)^{|M|} \cdot |\Delta_\rho|.$$

5. The size of  $\mathcal{P}_M$  is bounded by

$$|\mathcal{P}_M| \leq (\hat{n} + 7)^M \cdot |\mathcal{P}_\rho| + 3 \cdot |\Phi| \cdot |M| \cdot (\hat{n} + 7)^{3|M|} \cdot |\Delta_\rho|^2.$$

The stated bounds employ as parameter the number of (sub-)formulae  $|M|$ , the size of  $\Phi$  and  $\mathcal{S}$ , the number  $|S|$  of states of  $\mathcal{S}$  and the value  $\hat{n} := \hat{a}_\Phi \cdot |\text{sub}(\Phi)| \cdot \hat{u}_\mathcal{S} \cdot |S|$  combining the maximal absolute values  $\hat{a}_\Phi$  and  $\hat{u}_\mathcal{S}$  of integer constants occurring in  $\Phi$  and  $\mathcal{S}$ , respectively. Notice further, that the measure refers to the size of the APS  $\mathcal{P}_\rho$  representing the assumed satisfying run  $\rho \in \text{runs}(\mathcal{S})$  and the number  $|Q_\rho|$  of its states. As discussed above, the latter two correlate linearly with the size of  $\mathcal{S}$ .

**Base case and induction hypothesis.** Since the induction and the proof goal are tailored towards an extension of  $\mathcal{P}_\rho$ , it trivially constitutes the base case  $M = \emptyset$ . For the induction step, we have now to derive that  $\mathcal{P}_M$  exists for an arbitrary  $M \subseteq \text{sub}(\Phi)$  assuming as *induction hypothesis (IH)* that Items 1 to 5 of the proof goal hold for all closed strict subsets  $N = \text{sub}(N) \subset M$  of  $M$ . Let  $\varphi \in M$  be any maximal element of  $M$ , that is, a formula that is not a strict subformula of any other element of  $M$ . If  $M$  is not empty, such a formula exists always and the set  $N := M \setminus \{\varphi\} \subset M$  is closed under taking subformulae because  $M$  is downward-closed and removing a maximal element does not change this property. Therefore, the induction hypothesis provides an APS  $\mathcal{P}_N$  satisfying the conditions above. The remainder of this section is dedicated to constructing  $\mathcal{P}_M$ . To this end, the structural cases of  $\varphi$  (cf. Section 2.2) are analysed individually.

#### 4.5.2 Atomic Propositions and Boolean Combinations

Assume  $\mathcal{P}_N = (Q_N, \Delta_N, \lambda_N, \text{org})$ . If  $\varphi \in AP$  is an atomic proposition,  $\mathcal{P}_N$  is  $\varphi$ -consistent by definition and thus we can choose  $\mathcal{P}_M := \mathcal{P}_N$ , satisfying the claims. For Boolean

combinations, the labelling of  $\mathcal{P}_N$  is easily adjusted to conform to condition B of Definition 4.11. For each state  $q \in Q_N$ , if  $\varphi = \neg\psi$ , let

$$\lambda_M(q) := \begin{cases} \lambda_N(q) \setminus \{\varphi\} & \text{if } \psi \in \lambda_N(q) \\ \lambda_N(q) \cup \{\varphi\} & \text{otherwise} \end{cases}$$

and for the case  $\varphi = \chi \wedge \psi$ , let

$$\lambda_M(q) := \begin{cases} \lambda_N(q) \cup \{\varphi\} & \text{if } \chi, \psi \in \lambda_N(q) \\ \lambda_N(q) \setminus \{\varphi\} & \text{otherwise.} \end{cases}$$

These changes do not modify the set of runs of  $\mathcal{P}_N$ , nor the sets of states and transitions and thus the obtained  $M$ -consistent APS  $\mathcal{P}_M = (Q_N, \Delta_N, \lambda_M, \text{org})$  satisfies  $\Phi$ , and is of the same size.

### 4.5.3 Guard Formulae

A more involved case is that of  $\varphi$  being a guard of the form  $(\tau \geq b) \in \text{Grd}(C)$ . Consistency demands to add the guard or its dual to every transition, depending on whether the target state is labelled by  $\varphi$  or not. Any given labelling thus unambiguously defines the necessary transition guards to be imposed in order to make all states consistent. The difficulty is to find such a labelling that is actually compatible with a run of  $\mathcal{P}_N$  satisfying  $\Phi$ . If the labelling is chosen arbitrarily, the guards can be chosen to formally settle  $\varphi$ -consistency but there would be no guarantee that any run remains. In particular, two contradicting guards at one edge is technically permitted but simply renders the APS empty and thus not satisfying any formula anymore. Assume henceforth  $\mathcal{P}_N = (Q_N, \Delta_N, \lambda_N, \text{org})$  and let  $\sigma \in \text{runs}(\mathcal{P}_N)$  be a run with  $(\mathcal{P}, \sigma) \models \Phi$ .

### Rows and Stable Loops

Every row state  $q \in Q_N$  only occurs at a single position  $i$  on  $\sigma$  and this position can be used to uniquely determine how to label the state: let  $q$  be labelled by  $\varphi$  if and only if  $(\mathcal{P}, \sigma, i) \models \varphi$ . This guarantees that adding  $\varphi$  or  $\bar{\varphi}$ , respectively, as a guard to the incoming transition of  $q$  preserves  $\sigma$  as valid run. At the same time, it makes  $q$  satisfy the consistency criterion. The labelling of a loop is uniquely determined, if it is *stable* with respect the satisfaction of  $\varphi$ , meaning that for each loop state the formula  $\varphi$  is satisfied at all or none of its occurrences. Recall, moreover, that APS are defined to assure that every loop is labelled identically to its front and its rear row, if any. Thus,

the positions before and after a loop, that correspond to some specific state on it, must also be compatible in the sense that either both satisfy or both violate the formula  $\varphi$ , in accordance to the determined labelling of the respective loop state. If these conditions apply for a loop and its enclosing rows, it is called stable.

► **Definition 4.22 (Stability).** *Let  $\mathcal{P}$  be an APS,  $\sigma \in \text{runs}(\mathcal{P})$  and  $\varphi$  a cLTL formula. A loop  $L \in \text{loops}(\mathcal{P})$  with front  $F = \text{front}_{\mathcal{P}}(L)$  is stable on  $\sigma$  with respect to  $\varphi$  if for all  $i \in \text{pos}_{\sigma}(FL)$*

$$(\mathcal{P}, \sigma, i) \models \varphi \quad \Leftrightarrow \quad (\mathcal{P}, \sigma, i + |L|) \models \varphi.$$

For a stable loop, the run again uniquely determines for each of the states whether it is to be labelled by  $\varphi$  or not and obeying this indication guarantees that adding the corresponding guards does not invalidate  $\sigma$ . In general, however,  $\sigma$  does not uniquely determine whether some state  $q \in Q_N$  is supposed to be labelled by  $\varphi$  or not if  $q$  is part of some loop  $L$  in  $\mathcal{P}_N$  that is not stable on  $\sigma$ . Then, there may be two positions  $i \neq j$  on  $\sigma$  both carrying the state  $st(\sigma(i)) = st(\sigma(j)) = q$  but where  $(\mathcal{P}, \sigma, i) \models \varphi$  while  $(\mathcal{P}, \sigma, j) \not\models \varphi$ , or vice versa. For example, a counter referenced in the constraint  $\varphi$  may be decremented during the loop execution and so the constraint may be satisfied first but violated later on along the run. In this case it is not possible to provide a correct labelling of  $\mathcal{P}_N$  for  $\varphi$ , let alone a consistent one, forcing us to carefully *transform* the structure of the APS in order to establish consistency while maintaining the desired properties of being  $N$ -consistent and satisfying  $\Phi$ .

### Loop Elimination

Every non-last loop  $L$  is traversed a specific number  $n = |\text{pos}_{\sigma}(L(0))| \in \mathbb{N}$  of times by  $\sigma$ , i.e., the state sequence of  $\sigma$  has the form  $st(\sigma) = uL^n w$  for some  $u \in (Q \setminus L)^*$  and  $w \in (Q \setminus L)^\omega$ . One approach to provide a consistent labelling for such a loop  $L$  is to represent each of the traversals explicitly in the APS by *unfolding* the loop exactly  $n$  times.

By repeated ( $n - 1$  times) application of Corollary 4.18 and a final application of Corollary 4.20 we obtain from  $\mathcal{P}_N$  an  $N$ -consistent APS  $\hat{\mathcal{P}}_N$  that contains a run  $\hat{\sigma} \in \text{runs}(\hat{\mathcal{P}}_N)$  resembling  $\sigma$  but where each iteration of  $L$  is represented by an individual set of row states. The states of these rows  $R_0, \dots, R_n$  can now be made consistent as described above while preserving the run  $\hat{\sigma}$ . Assuming  $n \geq 2$ , then  $n - 1$  consecutive applications of Corollary 4.18 provide a sequence of schemas  $\mathcal{P}_0, \dots, \mathcal{P}_{n-1}$  with  $\mathcal{P}_0 = \mathcal{P}$ . Observe that introducing the rows  $R_1, \dots, R_i$  increases the number of transitions to

$|\Delta_i| = |\Delta_0| + i|L|$  and for each  $i \in [0, n-2]$

$$|\mathcal{P}_{i+1}| \leq |\mathcal{P}_i| + \text{size}(\Delta_L) + |L| \cdot (|\Delta_i| + |L|) \cdot \left( \sum_{\varphi' \in N} |\varphi'| \right).$$

Therefore,

$$\begin{aligned} |\mathcal{P}_{n-1}| &\leq |\mathcal{P}_0| + \sum_{i=0}^{n-2} \text{size}(\Delta_L) + |L| \cdot (|\Delta_i| + |L|) \cdot \left( \sum_{\varphi' \in N} |\varphi'| \right) \\ &= |\mathcal{P}_0| + (n-1) \cdot \text{size}(\Delta_L) + |L| \cdot |\Phi| \cdot \left( \sum_{i=0}^{n-2} |\Delta_0| + i|L| + |L| \right) \quad (4.4) \\ &= |\mathcal{P}_0| + (n-1) \cdot \text{size}(\Delta_L) + |L| \cdot |\Phi| \cdot \left( (n-1) \cdot |\Delta_0| + |L| \frac{(n-1)n}{2} \right) \\ &\leq |\mathcal{P}_0| + (n-1) \cdot \text{size}(\Delta_L) + |L| \cdot |\Phi| \cdot (n-1)^2 (|\Delta_0| + |L|). \end{aligned}$$

Finally, applying Corollary 4.20 to  $\mathcal{P}_{n-1}$  in order to remove the loop provides an APS  $\hat{\mathcal{P}}$  of size

$$\begin{aligned} |\hat{\mathcal{P}}| &\leq |\mathcal{P}_{n-1}| + |L| \cdot |\Delta_{n-1}| \cdot \left( \sum_{\varphi' \in M} |\varphi'| \right) \\ &\leq |\mathcal{P}_0| + (n-1) \cdot \text{size}(\Delta_L) + |L| \cdot |\Phi| \cdot (n-1)^2 (|\Delta_0| + |L|) \quad (4.5) \\ &\quad + |L| \cdot (|\Delta_0| + (n-1)|L|) \cdot |\Phi| \\ &\leq |\mathcal{P}_0| + (n-1) \cdot \text{size}(\Delta_L) + 2|L| \cdot |\Phi| \cdot (n-1)^2 \cdot (|\Delta_0| + |L|). \end{aligned}$$

To keep the constructed APS small, loop elimination will only be applied if  $n$  is below some fixed bound. In particular, the last loop of  $\mathcal{P}_N$  needs to be handled differently since it is traversed infinitely often.

### Stabilising Loops through Duplication

A second option to transform the APS in order to implement a consistent labelling is based on the following observation. The satisfaction of  $\varphi$  at some state changes at most once during the traversal of a loop  $L$ . For example, let  $i, i', i'' \in \text{pos}_\sigma(q)$  for some state  $q \in L$  with  $i < i' < i''$ . If  $(\mathcal{P}, \sigma, i) \models \varphi$  and  $(\mathcal{P}, \sigma, i') \not\models \varphi$  then the satisfaction of  $\varphi$  cannot switch after position  $i'$  again and thus  $(\mathcal{P}, \sigma, i'') \not\models \varphi$ . Concerning different states  $q \neq q'$  of  $L$ , the validity of  $\varphi$  may change in different iterations of  $L$  along  $\sigma$ . However, all such changes for all the states of  $L$  can be shown to happen within a bounded number  $\hat{n}$  of iterations. The iterations of  $L$  along  $\sigma$  can thus be separated into three consecutive parts, such that in the first and last part, no change in the satisfaction of  $\varphi$  occurs, i.e., these parts are entirely stable. Changes only occur in the middle part, but this will be

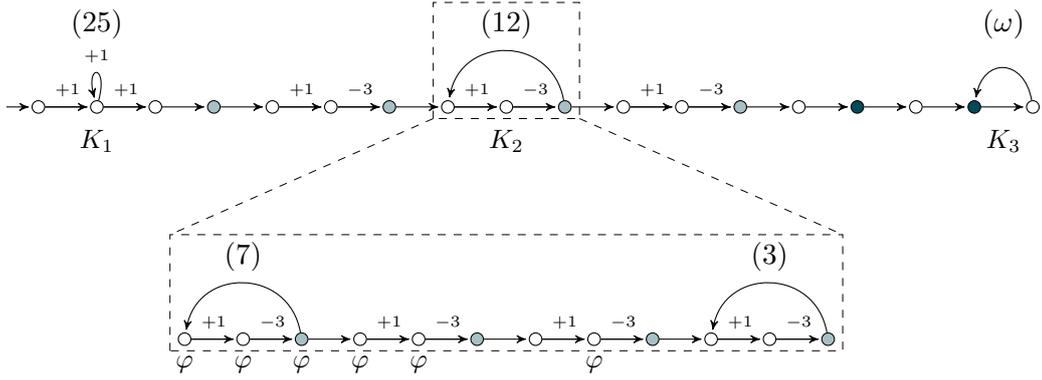


Figure 4.9: Example for stabilising the loop  $K_2$  in  $\mathcal{P}$  (from Figure 4.2) with respect to a formula  $\varphi = c \geq 10$  and a run  $\sigma$  by duplicating and unfolding it. Only updates of the counter  $c$  are shown.

shown to be sufficiently short.

► **Example 4.23.** Consider the APS  $\mathcal{P}$  in Figure 4.2, a run  $\sigma \in \text{runs}(\mathcal{P})$ , the state  $q_9$ , and a guard formula  $\varphi := c \geq 10$ . Let  $K_1 = q_1$ ,  $K_2 = q_7q_8q_9$  and  $K_3 = q_{16}q_{17}$  be the loops of  $\mathcal{P}$  corresponding to the loops  $L_1$ ,  $L_2$ , and  $L_3$  of  $\mathcal{S}$ , respectively. Whether or not  $\varphi$  holds at some position  $i$  with  $st(\sigma(i)) = q_9$  depends on how often  $\sigma$  traverses the good loop  $K_1$  (the more the better) and how often it repeats  $K_2$  before reaching position  $i$  (the more the worse). Assume  $\sigma$  traverses  $K_1$  exactly 25 times and thus enters  $K_2$  with  $c = 25$  (the effect of the front and rear rows of  $K_1$  is cancelled out by the front row of  $K_2$ ).

Let  $\mu_{K_2}$  denote the accumulated updates of  $K_2$ . Each iteration of  $K_2$  decreases the value of  $c$  since  $\mu_{K_2}(c) = -2$  and therefore  $\varphi$  holds at  $q_9$  during the first 7 iterations while it is violated in iteration 8 where the value of  $c$  first drops below to  $25 + 8 \cdot \mu_{K_2}(c) = 9$ . At state  $q_8$ , the effect is delayed because it occurs before the negatively weighted transition. However, in iteration 10, the local difference within the loop is outnumbered by the global effect and thus  $\varphi$  does not hold at any state any more.

While any labelling of  $K_2$  would necessarily be incompatible with  $\sigma$ , there are three phases: the first phase (iteration 1 to 7) where  $\varphi$  always holds, the last phase (iterations 10 and later) where  $\varphi$  never holds, and an unstable transition phase. When  $K_2$  is replaced by two copies of it connected by two unfoldings for the unstable transition phase as indicated in Figure 4.9, a labelling compatible with  $\sigma$  can be provided.

The length of the intermediate unstable phase of the iteration of  $L$  on  $\sigma$  can be bounded by  $\hat{a}_\tau \cdot |\tau| \cdot \hat{u}_L \cdot |L|$  where  $\hat{a}_\tau$  is the maximal absolute value of the coefficients in the term  $\tau$ , and  $\hat{u}_L$  denotes the maximal absolute value of counter updates on the transitions of the

loop  $L$ . For convenience, let us over-approximate this bound using parameters depending only on  $\mathcal{S}$  and  $\Phi$  in order to work with only one value  $\hat{n}$  for all loops and terms that occur in the construction. This allows also for a general size estimation of the constructed APS, depending on  $\mathcal{S}$  and  $\Phi$ . Let, hence, for the rest of this section  $\hat{n} := \hat{a}_\Phi \cdot |\Phi| \cdot \hat{u}_\mathcal{S} \cdot |S|$  where  $\hat{a}_\Phi \geq \hat{a}_\tau$  is the maximal absolute value of all constraint coefficients occurring in (any subformula of)  $\Phi$  and  $\hat{u}_\mathcal{S} \geq \hat{u}_L$  is the maximal absolute value of all updates in  $\mathcal{S}$ . In the following this claim is formalised and proved. Subsequently, duplicating  $L$  as sketched in Figure 4.9 is discussed and used to provide a labelling that is compatible with  $\sigma$  regarding the satisfaction of  $\varphi$ .

► **Definition 4.24 (Pivot).** *Let  $\mathcal{P}$  be an APS,  $\sigma \in \text{runs}(\mathcal{P})$ ,  $L \in \text{loops}(\mathcal{P})$ ,  $F = \text{front}_{\mathcal{P}}(L)$ , and  $\varphi \in \text{sub}(\Phi)$ . A pivot for  $\varphi$  and  $L$  on  $\sigma$  is a position  $h \in \mathbb{N}$  such that for all  $i \in \text{pos}_\sigma(L) \setminus [h, h + \hat{n}|L| - 1]$  where position  $i + |L| \in \text{pos}_\sigma(L)$  is still on  $L$  we have*

$$(\mathcal{P}, \sigma, i) \models \varphi \quad \Leftrightarrow \quad (\mathcal{P}, \sigma, i + |L|) \models \varphi.$$

Intuitively, when  $\sigma$  traverses a loop  $L$ , the pivot describes the starting position of the unstable phase. In Example 4.23 above, the first position of iteration 8 of  $K_2$  would be such a tipping point because all preceding positions (on  $L$ ) satisfy  $\varphi$  and all positions at least  $\hat{n} \geq 2$  iterations later are either not on  $L$  any more or violate  $\varphi$ . Importantly, a given pivot position not only determines the start of an unstable phase but also sets its limit based on  $\hat{n}$  that only depends on  $\mathcal{S}$  and  $\Phi$ . The following observation establishes that the stabilisation procedure based on duplicating a loop can always be applied at a bounded cost in terms of the blow-up.

► **Lemma 4.25.** *Let  $\varphi \in \text{sub}(\Phi) \cap \text{Grd}(C)$  be a guard, and  $\mathcal{P}$  an APS in  $\mathcal{S}$ . For all  $\sigma \in \text{runs}(\mathcal{P})$  and  $L \in \text{loops}(\mathcal{P})$  there is a pivot for  $\varphi$  and  $L$  on  $\sigma$ .*

**Proof.** Assume  $\mathcal{P} = (Q_{\mathcal{P}}, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$ ,  $\varphi = (\tau \geq b)$ ,  $L = L_0 \dots L_\ell$  for  $\ell = |L| - 1$ , and  $F = \text{front}_{\mathcal{P}}(L)$ . Let  $\delta_0, \dots, \delta_\ell$  be the transitions on  $L$  with  $\delta_i = (L_i, \mu_i, \Gamma_i, L_{i+1})$  for  $i \in [0, \ell - 1]$  and  $\delta_\ell = (L_\ell, \mu_\ell, \Gamma_\ell, L_0)$ . The accumulated update function  $\mu_L := \sum_{i \in [0, \ell]} \mu_i$  determines the effect of  $L$  on the value of the term  $\tau$ , denoted by  $\tau_L := \llbracket \tau \rrbracket(\mu_L)$  in the following.

*Case  $\tau_L = 0$ .* If  $\tau_L = 0$ , then  $L$  is stable in the relevant range and the pivot  $h$  can be chosen arbitrarily since for all  $i \in \mathbb{N}$  with  $i, i + |L| \in \text{pos}_\sigma(L)$  we have

$$\llbracket \tau \rrbracket(\text{val}(\sigma(i))) = \llbracket \tau \rrbracket(\text{val}(\sigma(i))) + \tau_L = \llbracket \tau \rrbracket(\text{val}(\sigma(i + |L|))).$$

Case  $\tau_L < 0$ . Assume now that  $\tau_L < 0$  and let  $h_0 \geq \min \text{pos}_\sigma(L)$  be the first (smallest) position of  $L$  such that  $(\mathcal{P}, \sigma, h_0) \not\models (\tau \geq b)$ . If such a position  $h_0$  does not exist,  $h$  can again be chosen arbitrarily, otherwise let  $h = h_0 - |L|$ . By this choice of  $h$ , for all  $i \in \text{pos}_\sigma(L)$  with  $0 \leq i < h$ , if any,  $i + |L| < h + |L| = h_0$  and thus  $(\mathcal{P}, \sigma, i) \models \varphi$  and  $(\mathcal{P}, \sigma, i + |L|) \models \varphi$ .

For the remaining positions beyond  $h + \hat{n}|L|$ , first observe the following for any two positions  $i_1, i_2 \in \text{pos}_\sigma(L)$  having a difference  $r := |i_1 - i_2| < |L|$  of less than one length of  $L$ . In between, exactly  $r$  updates  $\nu_1, \dots, \nu_r$  are applied and thus the difference between the corresponding values of  $\tau$  can be estimated by

$$\begin{aligned} |\llbracket \tau \rrbracket(\text{val}(\sigma(i_1))) - \llbracket \tau \rrbracket(\text{val}(\sigma(i_2)))| &\leq |\llbracket \tau \rrbracket(\nu_1)| + \dots + |\llbracket \tau \rrbracket(\nu_r)| \\ &\leq r \cdot \max\{|\llbracket \tau \rrbracket(\nu_1)|, \dots, |\llbracket \tau \rrbracket(\nu_m)|\} \\ &\leq (L - 1) \cdot \max\{|\llbracket \tau \rrbracket(\nu_1)|, \dots, |\llbracket \tau \rrbracket(\nu_m)|\}. \end{aligned}$$

Since  $\varphi \in \text{sub}(\Phi)$  is a subformula of  $\Phi$ , the length of  $\tau$  (the number of its monomials) is bounded by the size  $|\Phi|$  of  $\Phi$  and the value of its coefficients is bounded by  $\hat{a}_\Phi$ .

Moreover,  $\varphi$  uses only counters from  $C_S$  and those are updated by at most the value  $\hat{u}_S$ . Therefore  $|\llbracket \tau \rrbracket(\mu)| \leq |\Phi| \cdot \hat{a}_\Phi \cdot \hat{d}_S$  for every  $\mu \in \{\nu_1, \dots, \nu_r\}$  and

$$|\llbracket \tau \rrbracket(\text{val}(\sigma(i_1))) - \llbracket \tau \rrbracket(\text{val}(\sigma(i_2)))| \leq (|L| - 1) \cdot |\Phi| \cdot \hat{a}_\Phi \cdot \hat{d}_S < \hat{n}. \quad (4.6)$$

Now, for each position  $i \in \mathbb{N}$  with  $i, i + |L| \in \text{pos}_\sigma(L)$  and  $i \geq h + \hat{n}|L| \geq h_0 + (\hat{n} - 1)|L|$ , there is  $k \geq \hat{n}$  and  $r \in [0, |L| - 1]$  such that  $i = h_0 + k|L| - r$ . By the definition of  $\hat{n}$ , the position  $i$  is distant enough from  $h_0$  such that the maximal possible update effect of the  $r < |L|$  transitions applied between  $i$  and  $i + r = h_0 + k|L| < i + |L|$ , as estimated in Equation (4.6), becomes marginal compared to the (negative) net effect of the  $k \geq \hat{n}$  loop iterations. Specifically, we have

$$\begin{aligned} \llbracket \tau \rrbracket(\text{val}(\sigma(i))) &\leq \llbracket \tau \rrbracket(\text{val}(\sigma(i + r))) \\ &\quad + |\llbracket \tau \rrbracket(\text{val}(\sigma(i + r))) - \llbracket \tau \rrbracket(\text{val}(\sigma(i)))| \\ &\leq \llbracket \tau \rrbracket(\text{val}(\sigma(i + r))) + \hat{n} && \text{(Equation (4.6))} \\ &= \llbracket \tau \rrbracket(\text{val}(\sigma(h_0))) + k \cdot \tau_L + \hat{n} \\ &\leq \llbracket \tau \rrbracket(\text{val}(\sigma(h_0))) && (\tau_L < 0, k \geq \hat{n}) \\ &< b \end{aligned}$$

and thus  $(\mathcal{P}, \sigma, i) \not\models \varphi$  for all positions  $i \geq h + \hat{n}|L|$  on  $L$ .

*Case  $\tau_L > 0$ .* In case  $\tau_L > 0$ , an analogous reasoning can be applied when  $h_0$  is chosen to be the first position  $h_0 \geq \min pos_\sigma(L)$  on  $\sigma$  where  $(\mathcal{P}, \sigma, h_0) \models (\tau \geq b)$ . Again, if  $h_0$  does not exist,  $h$  is arbitrary. Otherwise  $h = h_0 - |L|$  is defined as above providing that the lemma statement holds for all  $i < h$ . For the remaining positions  $i$  with  $i, i+|L| \in pos_\sigma(L)$  and  $i \geq h + \hat{n}|L|$ , we are this time interested in a lower bound for the value of  $\tau$  that is greater or equal to  $b$  in order to show that  $(\mathcal{P}, \sigma, i) \models \varphi$ . For  $r$  and  $k$  as defined above we verify that

$$\begin{aligned}
 \llbracket \tau \rrbracket(\text{val}(\sigma(i))) &\geq \llbracket \tau \rrbracket(\text{val}(\sigma(i+r))) - \hat{n} && (r < |L|, \text{Equation (4.6)}) \\
 &= \llbracket \tau \rrbracket(\text{val}(\sigma(h_0))) + k \cdot \tau_L - \hat{n} \\
 &\geq \llbracket \tau \rrbracket(\text{val}(\sigma(h_0))) && (\tau_L > 0, k \geq \hat{n}) \\
 &\geq b.
 \end{aligned}$$

■

Towards making the APS  $\mathcal{P}_N$   $\varphi$ -consistent, the existence of a pivot position for  $\varphi$  on  $\sigma$  that we have just established can now be used to stabilise the loops in  $\mathcal{P}_N$ . This is a precondition for providing any correct labelling and thus for implementing the consistency criterion. Notice that the following construction is not specific for guards but only depends on the existence of pivot positions. A pivot lemma as above will also be proven for counting until formulae later in order to reuse the result.

► **Lemma 4.26 (Stabilisation).** *Let  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  be an  $N$ -consistent APS,  $\sigma \in \text{runs}(\mathcal{P})$ , and  $\varphi \in \text{sub}(\Phi)$  such that for all  $L \in \text{loops}(\mathcal{P})$  there is a pivot for  $\varphi$  and  $L$  on  $\sigma$ . There is an  $N$ -consistent APS  $\hat{\mathcal{P}} = (\hat{Q}, \hat{\Delta}, \hat{\lambda}, \widehat{\text{org}})$  and  $\hat{\sigma} \in \text{runs}(\hat{\mathcal{P}})$  such that*

- $\widehat{\text{org}}(\hat{\sigma}) = \text{org}(\sigma)$ ,
- all loops  $L \in \text{loops}(\hat{\mathcal{P}})$  are stable on  $\hat{\sigma}$  with respect to  $\varphi$ ,
- $|\hat{Q}| \leq (\hat{n} + 4) \cdot |Q|$
- $|\hat{\Delta}| \leq (\hat{n} + 4) \cdot |\Delta|$
- $|\hat{\mathcal{P}}| \leq (\hat{n} + 7) \cdot |\mathcal{P}| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot |\Delta|^2$

**Proof.** Let  $L$  be a loop of  $\mathcal{P}$  that is not already stable for  $\varphi$  on  $\sigma$ .

*Elimination.* If  $\sigma$  iterates  $L$  not too often, more precisely  $n = |\text{pos}_\sigma(L(0))| < \hat{n} + 6$  times, it can be entirely unfolded within the size constraint. As described above, an APS can be obtained by applying Corollary 4.18 to  $L$  for  $n - 1$  times and subsequently Corollary 4.20. That is,  $L$  is replaced by  $n$  rows  $R_1, \dots, R_n$ . By Equation (4.5), the resulting APS  $\mathcal{P}'$  is of size at most

$$\begin{aligned} |\mathcal{P}'| &\leq |\mathcal{P}| + (n - 1) \cdot \text{size}(\Delta_L) + 2|L| \cdot |\Phi| \cdot (n - 1)^2 \cdot (|\Delta| + |L|) \\ &\leq |\mathcal{P}| + (\hat{n} + 4) \cdot \text{size}(\Delta_L) + 2|L| \cdot |\Phi| \cdot (\hat{n} + 4)^2 \cdot (|\Delta| + |L|). \end{aligned}$$

Assume henceforth that  $L$  is traversed at least  $\hat{n} + 6$  times and let  $h \in \mathbb{N}$  be the corresponding pivot position on  $\sigma$ .

*Unfolding initial iterations.* First, assume that  $h < \min \text{pos}_\sigma(L) + 2|L|$  occurs at latest in the second iteration of  $L$  on  $\sigma$ . The state sequence of  $\sigma$  then has the form  $st(\sigma) = uL^{\hat{n}+5}w$  for some  $u \in (Q \setminus L)^*$ ,  $w \in Q^\omega$ . Thus, repeated application of Corollary 4.18 (precisely  $\hat{n}+3$  times) yields an APS  $\mathcal{P}'$  featuring  $\hat{n}+3$  rows  $F_1, \dots, F_{\hat{n}+3}$  and a run  $\sigma'$  with state sequence  $st(\sigma') = uF_1 \dots F_{\hat{n}+3}L^2w$ . The additional rows accommodate the unstable iterations of  $L$  on  $\sigma$  entirely since there are at most  $\hat{n}$ . We have  $\text{front}_{\mathcal{P}'}(L) = F_{\hat{n}+3}$  and for all  $i$  with  $i, i + |L| \in \text{pos}_{\sigma'}(F_{\hat{n}+3}L)$  that

$$i \geq \min \text{pos}_\sigma(L) + (\hat{n} + 2)|L| > h + \hat{n}|L|.$$

Thus,  $(\mathcal{P}', \sigma', i) \models \varphi$  if and only if  $(\mathcal{P}', \sigma', i + |L|) \models \varphi$  and  $L$  is stable if  $L$  is the last loop of  $\mathcal{P}'$  (and  $\mathcal{P}$ ). Otherwise, satisfaction of  $\varphi$  on the positions of the rear row  $\text{rear}_{\mathcal{P}'}(L)$  may not coincide with those on  $L$  because the pivot guaranties this only for states on the loop. However, since  $L$  is still traversed at least twice by  $\sigma'$ , Corollary 4.19 provides an APS  $\mathcal{P}''$  where  $L$  is unfolded providing a new rear row and a corresponding run  $\sigma''$  on which  $L$  is now certainly stable with respect to  $\varphi$ . The at most  $\hat{n} + 4$  unfolding constructions increase the size of  $\mathcal{P}$  to at most

$$|\mathcal{P}''| \leq |\mathcal{P}| + (\hat{n} + 4) \cdot \text{size}(\Delta_L) + |L| \cdot |\Phi| \cdot (\hat{n} + 4)^2 (|\Delta| + |L|)$$

by application of Equation (4.4).

*Duplication.* Assume now that  $h \geq \min \text{pos}_\sigma(L) + 2|L|$  and let

$$n_1 := \left\lfloor \frac{h - \min \text{pos}_\sigma(L)}{|L|} \right\rfloor + 1 \geq 3$$

denote, intuitively, the iteration of  $L$  on  $\sigma$  in which position  $h$  occurs. Notice, however, that there may not actually exist  $n_1$  iterations of  $L$  on  $\sigma$ , namely if  $h$  exceeds the positions of  $L$  on  $\sigma$ . Yet, assume for now that the number of iterations is large enough such that  $st(\sigma) = uL^{n_1+\hat{n}+3}w$  for some  $u \in (Q \setminus L)^*$  and  $w \in Q^\omega$ . Lemma 4.15 provides an  $N$ -consistent APS  $\mathcal{A}$  derived from  $\mathcal{P}$  containing a loop  $K$  and an intermediate row  $U_0$  that are both copies of  $L$ . Further,  $\mathcal{A}$  admits a run  $\sigma_{\mathcal{A}} \in runs(\mathcal{A})$  with state sequence  $st(\sigma_{\mathcal{A}}) = uK^{n_1-1}U_0L^{\hat{n}+3}w$  and identical labelling  $\lambda_{\mathcal{A}}(\sigma_{\mathcal{A}}) = \lambda(\sigma)$ . Repeated application of Corollary 4.18 (once to  $K$ ,  $\hat{n} + 1$  times to  $L$ ) yields an APS  $\mathcal{B}$  of the desired shape (cf. Figure 4.9) featuring  $\hat{n} + 3$  rows  $F, U_0, \dots, U_{\hat{n}+1}$  and a run  $\sigma_{\mathcal{B}}$  with state sequence  $st(\sigma_{\mathcal{B}}) = uFK^{n_1-2}U_0 \dots U_{\hat{n}+1}L^2w$  and  $\lambda_{\mathcal{B}}(\sigma_{\mathcal{B}}) = \lambda(\sigma)$ . The loop  $K$  is stable for  $\varphi$  on  $\sigma_{\mathcal{B}}$  since for each  $i \in pos_{\sigma_{\mathcal{B}}}(F \cup K)$  we have that  $i < h$  and therefore

$$(\mathcal{B}, \sigma_{\mathcal{B}}, i) \models \varphi \Leftrightarrow (\mathcal{P}, \sigma, i) \models \varphi \Leftrightarrow (\mathcal{P}, \sigma, i + |L|) \models \varphi \Leftrightarrow (\mathcal{B}, \sigma_{\mathcal{B}}, i + |L|) \models \varphi.$$

Similarly, for each position  $i \in pos_{\sigma_{\mathcal{B}}}(U_{\hat{n}+1} \cup L)$  we have  $h + \hat{n}|L| < i$  and, thus,  $(\mathcal{B}, \sigma_{\mathcal{B}}, i) \models \varphi$  if and only if  $(\mathcal{B}, \sigma_{\mathcal{B}}, i + |L|) \models \varphi$  as long as  $i + |L| \in pos_{\sigma_{\mathcal{B}}}(L)$ . If  $L$  is the last loop of  $\mathcal{P}$  (and  $\mathcal{B}$ ), the latter holds for all positions of  $L$ . Otherwise, the satisfaction of  $\varphi$  on the positions of the rear row  $rear_{\mathcal{B}}(L)$  may not coincide with those on  $L$  because the pivot guaranties this only for states on the loop. As above, Corollary 4.19 can be applied to obtain the modified APS  $\mathcal{B}'$  containing an unfolding of  $L$  as new rear row. This provides stability also of  $L$  with respect to  $\varphi$ . With, in sum,  $\hat{n} + 3$  unfoldings and one duplication the size of the constructed APS can be bounded by

$$\begin{aligned} |\mathcal{B}'| &\leq |\mathcal{A}| + (\hat{n} + 3) \cdot size(\Delta_L) + |L| \cdot |\Phi| \cdot (\hat{n} + 3)^2(|\Delta_{\mathcal{A}}| + |L|) && \text{(Equation (4.4))} \\ &\leq |\mathcal{P}| + 3 \cdot size(\Delta_L) + |L| \cdot (|\Delta| + 2|L| + 1) \cdot |\Phi| \\ &\quad + (\hat{n} + 3) \cdot size(\Delta_L) + |L| \cdot |\Phi| \cdot (\hat{n} + 3)^2(|\Delta| + 2|L| + 1 + |L|) \\ &\leq |\mathcal{P}| + (\hat{n} + 6) \cdot size(\Delta_L) + 2|L| \cdot |\Phi| \cdot (\hat{n} + 3)^2(|\Delta| + 3|L| + 1). \end{aligned}$$

*Unfolding ending iterations.* It remains to consider the case that the pivot position  $h$  is close to or beyond the end of the traversal of  $L$  by  $\sigma$ , more precisely, that  $n_1 + \hat{n} + 3 > n$ . Notice, this case cannot apply if  $L$  is the last loop of  $\mathcal{P}$ . Nevertheless, recall that we still assume  $n \geq \hat{n} + 6$ , hence  $n - \hat{n} - 4 \geq 2$ . Therefore, we can apply Corollary 4.18 once and Corollary 4.19 for  $\hat{n} + 2$  times and obtain an APS  $\mathcal{P}'$  with a run  $\sigma'$  of the form  $st(\sigma') = uFL^{n-\hat{n}-3}R_0 \dots R_{\hat{n}+1}w$ . On this run,  $L$  is stable because for each  $i \in pos_{\sigma'}(FL)$

we have that  $i < h$  and  $i + |L| \in \text{pos}_\sigma(L)$ . Hence,

$$(\mathcal{P}', \sigma', i) \models \varphi \Leftrightarrow (\mathcal{P}, \sigma, i) \models \varphi \Leftrightarrow (\mathcal{P}, \sigma, i + |L|) \models \varphi \Leftrightarrow (\mathcal{P}', \sigma', i + |L|) \models \varphi.$$

*Application to all loops.* Each of the described constructions does not interfere with the stability of any other loop in  $\mathcal{P}$ . Therefore, they can be applied consecutively to each of the loops  $\text{loops}(\mathcal{P}) = \{L_1, \dots, L_k\}$  of  $\mathcal{P}$  such that all of them become stable. This provides a series of APS  $\mathcal{P} = \mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_k$  of which  $\mathcal{P}_k$  finally constitutes the APS  $\hat{\mathcal{P}}$  proposed by the lemma statement. The potentially largest number of new states and transitions is added during the duplication construction adding potentially  $(\hat{n} + 5) \cdot |L|$  new states and  $(\hat{n} + 5) \cdot |L| + 1$  new transitions. Therefore, we can estimate the number of states  $Q_i$  and transitions  $\Delta_i$  of  $\mathcal{P}_i$  by

$$|Q_{i+1}| \leq |Q_i| + (\hat{n} + 5) \cdot |L_{i+1}|$$

and

$$|\Delta_{i+1}| \leq |\Delta_i| + (\hat{n} + 5) \cdot |L_{i+1}| + 1.$$

Hence,

$$\begin{aligned} |\hat{Q}| = |Q_k| &\leq |Q_0| + (\hat{n} + 5) \cdot \sum_{i=1}^k |L_i| \\ &\leq |Q_0| + (\hat{n} + 5) \cdot \frac{1}{2} |Q_0| \\ &\leq |Q_0| \cdot (\hat{n} + 4) \end{aligned}$$

and

$$\begin{aligned} |\hat{\Delta}| = |\Delta_k| &\leq |\Delta_0| + (\hat{n} + 5) \cdot \left( \sum_{i=1}^k |L_i| \right) + k \\ &\leq |\Delta_0| + (\hat{n} + 5) \cdot \frac{1}{2} |\Delta_0| + \frac{1}{2} |\Delta_0| \\ &\leq |\Delta_0| \cdot (\hat{n} + 4) \end{aligned}$$

as claimed. Considering all the cases that may apply, the increase of the schema size can be bounded by

$$|\mathcal{P}_{i+1}| \leq |\mathcal{P}_i| + (\hat{n} + 6) \cdot \text{size}(\Delta_{L_{i+1}}) + 2|L_{i+1}| \cdot |\Phi| \cdot (\hat{n} + 4)^2 (|\Delta_i| + 3|L_{i+1}| + 1)$$

and therefore

$$\begin{aligned}
 |\hat{\mathcal{P}}| = |\mathcal{P}_k| &\leq |\mathcal{P}_0| + (\hat{n} + 6) \cdot \left( \sum_{i=1}^k \text{size}(\Delta_{L_i}) \right) \\
 &\quad + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^2 \cdot \sum_{i=0}^{k-1} |L_{i+1}| \cdot (|\Delta_i| + 3|L_{i+1}| + 1) \\
 &\leq (\hat{n} + 7) \cdot |\mathcal{P}_0| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^2 \cdot \sum_{i=0}^{k-1} |L_{i+1}| \cdot (|\Delta_k| + 4|L_{i+1}|) \\
 &\leq (\hat{n} + 7) \cdot |\mathcal{P}_0| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^2 \cdot \sum_{i=0}^{k-1} |L_{i+1}| \cdot (|\Delta_0|(\hat{n} + 4) + 2|\Delta_0|) \\
 &\leq (\hat{n} + 7) \cdot |\mathcal{P}_0| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^2 \cdot \Delta_0(\hat{n} + 6) \cdot \frac{1}{2}|\Delta_0| \\
 &\leq (\hat{n} + 7) \cdot |\mathcal{P}_0| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot |\Delta_0|^2.
 \end{aligned}$$

■

### Establishing Consistency

The previous developments now provide all necessary means to show the existence of the APS  $\mathcal{P}_M$  as desired. Lemma 4.25 provides that there is a pivot for each loop on  $\sigma_N$ . This allows us to apply Lemma 4.26 providing an APS  $\hat{\mathcal{P}}_N$  with run  $\hat{\sigma}_N \in \text{runs}(\hat{\mathcal{P}}_N)$  such that  $(\hat{\mathcal{P}}_N, \hat{\sigma}_N) \models \Phi$  because it represents the same original run as  $\sigma_N$ . Moreover,  $\hat{\mathcal{P}}_N$  is  $N$ -consistent, and all loops are stable with respect to the guard formula  $\varphi \in M$ . This means that for each state  $q \in \hat{Q}_N$  of  $\hat{\mathcal{P}}_N$ ,  $\varphi$  holds at all or none of its occurrences on  $\hat{\sigma}_N$ . We can thus define the labelling based on an arbitrary position  $i_q \in \text{pos}_{\hat{\sigma}_N}(q)$  as

$$\lambda_M(q) := \begin{cases} \hat{\lambda}_N(q) \cup \{\varphi\} & \text{if } (\hat{\mathcal{P}}_N, \hat{\sigma}_N, i_q) \models \varphi \\ \hat{\lambda}_N(q) \setminus \{\varphi\} & \text{otherwise.} \end{cases}$$

It only remains to extend the set of guards  $\Gamma$  on each transition  $(q, \mu, \Gamma, q') \in \hat{\Delta}_N$  to contain  $\varphi$  or  $\bar{\varphi}$  (as transition guard) if  $\varphi \in \lambda_M(q')$  or  $\varphi \notin \lambda_M(q')$ , respectively. This preserves  $\hat{\sigma}_N$  as valid run and renders the resulting APS consistent, constituting  $\mathcal{P}_M$ .

The size of  $\mathcal{P}_M = (Q_M, \Delta_M, \lambda_M, \text{org}_M)$  can be estimated by the size of  $\hat{\mathcal{P}}_N$  and the increase imposed by the additional guards. Neither additional states nor transitions are

added to  $\hat{\mathcal{P}}_N$  and thus

$$\begin{aligned} |\mathcal{Q}_M| &= |\hat{\mathcal{Q}}_N| \leq (\hat{n} + 4) \cdot |\mathcal{Q}_N| && \text{(by Lemma 4.26)} \\ &\leq (\hat{n} + 4) \cdot (\hat{n} + 4)^{|N|} \cdot |\mathcal{Q}_\rho| && \text{(by IH, Item 3)} \\ &= (\hat{n} + 4)^{|M|} \cdot |\mathcal{Q}_\rho| \end{aligned}$$

and

$$\begin{aligned} |\Delta_M| &= |\hat{\Delta}_N| \leq |\Delta_N| \cdot (\hat{n} + 4) && \text{(by Lemma 4.26)} \\ &\leq (\hat{n} + 4)^{|N|} \cdot |\Delta_\rho| \cdot (\hat{n} + 4) && \text{(by IH, Item 4)} \\ &= (\hat{n} + 4)^{|M|} \cdot |\Delta_\rho|. \end{aligned}$$

In  $\mathcal{P}_M$ , every transition is potentially extended by the guard  $\varphi = \tau \geq b$  or  $\bar{\varphi} = \bar{\tau} \geq -b + 1$ . Since  $|\varphi| \leq |\Phi|$  and  $|\bar{\varphi}| \leq |\Phi|$  this increases the size of  $\hat{\mathcal{P}}_N$  such that

$$|\mathcal{P}_M| \leq |\hat{\mathcal{P}}_N| + |\hat{\Delta}_N| \cdot |\Phi| \leq |\hat{\mathcal{P}}_N| + (\hat{n} + 4)^{|M|} |\Delta_\rho| \cdot |\Phi|.$$

With Lemma 4.26 we obtain

$$\begin{aligned} |\mathcal{P}_M| &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot |\Delta_N|^2 + (\hat{n} + 4)^{|M|} |\Delta_\rho| \cdot |\Phi| \\ &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot (|\Delta_\rho| \cdot (\hat{n} + 4)^{|N|})^2 \\ &\quad + (\hat{n} + 4)^{|N|+1} |\Delta_\rho| \cdot |\Phi| \\ &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 3 \cdot |\Phi| \cdot (\hat{n} + 4)^{2|N|+3} \cdot |\Delta_\rho|^2 \end{aligned}$$

and finally, by application of the induction hypothesis,

$$\begin{aligned} |\mathcal{P}_M| &\leq (\hat{n} + 7) \cdot ((\hat{n} + 7)^N \cdot |\mathcal{P}_\rho| + 3 \cdot |\Phi| \cdot |N| \cdot (\hat{n} + 7)^{3|N|} \cdot |\Delta_\rho|^2) \\ &\quad + 3 \cdot |\Phi| \cdot (\hat{n} + 4)^{2|N|+3} \cdot |\Delta_\rho|^2 \\ &\leq (\hat{n} + 7)^{N+1} \cdot |\mathcal{P}_\rho| + 3 \cdot |\Phi| \cdot |N| \cdot (\hat{n} + 7)^{3|N|+1} \cdot |\Delta_\rho|^2 \\ &\quad + 3 \cdot |\Phi| \cdot (\hat{n} + 4)^{2|N|+3} \cdot |\Delta_\rho|^2 \\ &\leq (\hat{n} + 7)^{N+1} \cdot |\mathcal{P}_\rho| + 3 \cdot |\Phi| \cdot (\hat{n} + 7)^{3|N|+3} \cdot |\Delta_\rho|^2 \cdot (|N| + 1) \\ &\leq (\hat{n} + 7)^M \cdot |\mathcal{P}_\rho| + 3 \cdot |\Phi| \cdot |M| \cdot (\hat{n} + 7)^{3|M|} \cdot |\Delta_\rho|^2. \end{aligned} \tag{4.7}$$

This completes the induction step for the case that  $\varphi$  is a guard formula.

#### 4.5.4 Next

Considering temporal *next* formulae  $\varphi = \mathbf{X} \psi$ , consistency is equally easy to establish as for Boolean combinations. The only states  $q$  that have more than one direct successor are those that terminate some non-last loop  $L$ . Recall that each such loop in an augmented path schema is immediately followed by a rear row  $R$  with  $\lambda_{\mathcal{P}}(L) = \lambda_{\mathcal{P}}(R)$ . Therefore, the successors of  $q$  are the identically labelled states  $L(0)$  and  $R(0)$ . In particular, either each or none of them is labelled by  $\psi$ . Consequently, we let all states of  $\mathcal{P}_N$  be labelled by  $\mathbf{X} \psi$  if and only if all successors are labelled by  $\psi$  and this establishes  $\varphi$ -consistency since this modification does not affect the consistency criterion for any other formula, especially not for any  $\psi \in N$ . Modifying only the labelling also preserves  $\sigma$  as a run as well as the size of the thus obtained APS  $\mathcal{P}_M$ .

#### 4.5.5 Until

The case for counted until formulae  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi$  is related to that of guard formulae discussed before in Section 4.5.3. To construct a suitable labelling, loops may have to be stabilised and to this end we will build directly on the construction presented above. Specifically, a pivot lemma is shown for until formulae, complementing Lemma 4.25. This allows us to apply Lemma 4.26 and subsequently  $\varphi$ -consistency is established on the stabilised intermediate APS. This shows that the APS  $\mathcal{P}_M$  exists and provides the induction step for until formulae.

#### Stabilisation

The notion of stability provided by Definition 4.22 and discussed in the context of guard formulae plays a similar role for until formulae. As for constraints, deriving a labelling for an APS from a given run crucially relies on all states to be stable on that run for a formula. The proof of Lemma 4.26 has already provided a construction to stabilise loops. Recall that it builds on the argument that loops have a constant effect on a constraint term, formulated in terms of a pivot position after which a loop stabilises within a bounded number of steps.

Since the constraint terms of until formulae employ again other subformulae, the effect depends on whether they hold or not in some specific iteration and is therefore not constant in general. However, assuming as additional precondition that a loop is labelled correctly by all strict subformulae guarantees a constant loop effect and allows us to prove the existence of pivot positions also for until formulae.

► **Remark 4.27.** Recall that pivots were defined with respect to the number  $\hat{n} := \hat{a}_\Phi \cdot |\Phi| \cdot \hat{u}_S \cdot |S|$  characterising the potential length of an unstable phase of loop iterations. For until formulae the counter updates of the system have no direct influence on that length and  $\hat{u}_S$  could be omitted. However, for ease of presentation this unified value is also used here.

► **Lemma 4.28 (Until pivot).** Let  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi \in \text{sub}(\Phi)$  be a counting until formula and  $\mathcal{P}$  an APS in  $\mathcal{S}$  that is consistent for all strict subformulae of  $\varphi$ . For all  $\sigma \in \text{runs}(\mathcal{P})$  and  $L \in \text{loops}(\mathcal{P})$  there is a pivot for  $\varphi$  and  $L$  on  $\sigma$ .

**Proof.** Assume  $\mathcal{P} = (Q, \Delta, \lambda, \text{org})$  and  $L = L_0 \dots L_\ell \in \text{loops}(\mathcal{P})$  be a loop in  $\mathcal{P}$  with  $\ell = |L| - 1$ . Let  $F_L := \text{front}_{\mathcal{P}}(L)$  and  $\tau_L := \llbracket \tau \rrbracket(\lambda^\#(L))$  denote the effect of the loop on the constraint term  $\tau$ . The latter is well-defined because the labelling provided by  $\lambda$  is correct due to consistency.

*Defect on  $L$  or  $\tau_L = 0$ .* Assume first that not all states of  $L$  are labelled by the formula  $\chi$ . In this case, we show that  $L$  is in fact stable and the pivot  $h$  can be chosen arbitrarily.

Let  $i \in \text{pos}_\sigma(L)$  be some position visiting  $L$  such that  $i + |L| \in \text{pos}_\sigma(L)$ , i.e.,  $i$  is not in the last iteration of  $L$  on  $\sigma$ . We have  $(\mathcal{P}, \sigma, i) \models \varphi$  if and only if there is some position  $j \geq i$  with  $\psi \in \lambda(\sigma(j))$ ,  $\llbracket \tau \rrbracket(\lambda^\#(\sigma(i)\sigma(i+1)\dots\sigma(j-1))) \geq b$ , and  $\chi \in \lambda(\sigma(i)) \cap \lambda(\sigma(i+1)) \cap \dots \cap \lambda(\sigma(j-1))$ . Because of the latter, this can only hold for  $j < i + |L|$ . Otherwise, the state of  $L$  not labelled by  $\chi$  would necessarily appear between  $i$  and  $j$ . Recall that  $L$  is either repeated infinitely or succeeded by a rear row  $R_L = \text{rear}_{\mathcal{P}}(L)$  being labelled identically. We have

$$\lambda(\sigma(i) \dots \sigma(i + |L| - 1)) = \lambda(\sigma(i + |L|) \dots \sigma(i + 2|L| - 1)).$$

Consequently,

$$\lambda(\sigma(i) \dots \sigma(j)) = \lambda(\sigma(i + |L|) \dots \sigma(j + |L|))$$

and, thus,  $(\mathcal{P}, \sigma, i) \models \varphi$  if and only if  $(\mathcal{P}, \sigma, i + |L|) \models \varphi$ .

The situation is similar if  $L$  is entirely labelled by  $\chi$  but has a zero effect  $\tau_L = 0$  on the constraint. For any position  $i$  with  $i, i + |L| \in \text{pos}_\sigma(L)$  we have that  $(\mathcal{P}, \sigma, i) \models \varphi$  implies that there is a witness position  $j \geq i$  with either  $j \in [i, i + |L| - 1]$  or  $j \geq i + |L|$ . In the former case, the position  $j + |L|$  witnesses that  $(\mathcal{P}, \sigma, i + |L|) \models \varphi$  as discussed before. In the latter case, the position  $j$  itself is a witness since  $L$  has no effect and thus

$$\llbracket \tau \rrbracket(\#_{i+|L|, j-1}^{\mathcal{P}, \sigma}) = \llbracket \tau \rrbracket(\#_{i, j-1}^{\mathcal{P}, \sigma}) \geq b. \quad (4.8)$$

Conversely, if  $(\mathcal{P}, \sigma, i + |L|) \models \varphi$ , any corresponding witness position  $j > i$  succeeds  $i$ . Equation (4.8) still applies and thus  $j$  witnesses also that  $(\mathcal{P}, \sigma, i) \models \varphi$ .

*No defect on  $L$  and  $\tau_L > 0$ .* Assume now that all states on  $L$  are labelled by  $\chi$  and  $\tau_L > 0$ . Let  $h_0 \in \text{pos}_\sigma(L)$  be the first (smallest) position such that  $(\mathcal{P}, \sigma, h_0) \not\models \varphi$ . If it does not exist,  $h$  can be chosen arbitrarily. Otherwise, let  $h := h_0 - |L|$ .

For those positions  $i \in \text{pos}_\sigma(L)$  with  $i < h$  we have  $i < i + |L| < h_0$  and  $(\mathcal{P}, \sigma, i) \models \varphi$  by definition. For all positions  $i \in \text{pos}_\sigma(L)$  with  $i \geq h + \hat{n}|L|$  we show that  $(\mathcal{P}, \sigma, i) \not\models \varphi$ . Towards contradiction assume that  $(\mathcal{P}, \sigma, i) \models \varphi$  on account of some witness position  $j \geq i$ . First, this means  $\psi \in \lambda(\sigma(j))$ . Second,  $\chi$  holds at all positions from  $i$  to  $j - 1$  and since  $L$  is entirely labelled by  $\chi$ , the formula holds also at all positions from  $h_0$  to  $i - 1$ . Thus,  $\chi$  holds at each position from  $h_0$  to  $j - 1$ .

Third, let  $h'_0 \in [h, h_0 - 1]$  be the position between  $h$  and  $h_0$  such that  $i = h'_0 + n|L|$  for some  $n \geq \hat{n}$ . The effect of a single step along  $\sigma$  on the value of the term  $\tau$  is bounded by  $a_\tau \cdot \ell_\tau \leq a_\Phi \cdot |\Phi|$  where  $a_\tau$  is the maximal absolute value of the coefficients in  $\tau$  and  $\ell_\tau$  is the number of monomial terms in  $\tau$ . Since there are at most  $|L| - 1$  positions between  $h'_0$  and  $h_0$ , the effect of the path between  $h'_0$  and  $h_0$  is bounded by

$$|\llbracket \tau \rrbracket(\#_{h'_0, h_0-1}^{\mathcal{P}, \sigma})| \leq a_\tau \cdot \ell_\tau \cdot (|L| - 1) \leq a_\Phi \cdot |\Phi| \cdot |Q_S| \leq \hat{n}. \quad (4.9)$$

Hence, we have

$$\begin{aligned} \llbracket \tau \rrbracket(\#_{h_0, j-1}^{\mathcal{P}, \sigma}) &= \llbracket \tau \rrbracket(\#_{h'_0, i-1}^{\mathcal{P}, \sigma}) + \llbracket \tau \rrbracket(\#_{i, j-1}^{\mathcal{P}, \sigma}) - \llbracket \tau \rrbracket(\#_{h'_0, h_0-1}^{\mathcal{P}, \sigma}) \\ &\geq \llbracket \tau \rrbracket(\#_{h'_0, i-1}^{\mathcal{P}, \sigma}) + \llbracket \tau \rrbracket(\#_{i, j-1}^{\mathcal{P}, \sigma}) - \hat{n} && \text{(Equation (4.9))} \\ &\geq n \cdot \tau_L + \llbracket \tau \rrbracket(\#_{i, j-1}^{\mathcal{P}, \sigma}) - \hat{n} && (h'_0 \geq \min \text{pos}_\sigma(L) - |L|) \\ &\geq \llbracket \tau \rrbracket(\#_{i, j-1}^{\mathcal{P}, \sigma}) && (n \geq \hat{n}, \tau_L > 0) \\ &\geq b. \end{aligned}$$

In summary this would mean that  $(\mathcal{P}, \sigma, h_0) \models \varphi$  contradicting the choice of  $h_0$ .

*No defect on  $L$  and  $\tau_L < 0$ .* Assume finally that all states of  $L$  are labelled by  $\chi$  and  $\tau_L < 0$ . Observe that for all positions  $i$  with  $i, i + |L| \in \text{pos}_\sigma(L)$  we have that  $(\mathcal{P}, \sigma, i) \models \varphi$  implies  $(\mathcal{P}, \sigma, i + |L|) \models \varphi$ . If  $\varphi$  holds at  $i$  on account of a witness position  $j \geq i + |L|$ ,

then

$$\begin{aligned}
 \llbracket \tau \rrbracket(\#_{i+|L|,j-1}^{\mathcal{P},\sigma}) &= \llbracket \tau \rrbracket(\#_{i,j-1}^{\mathcal{P},\sigma}) - \llbracket \tau \rrbracket(\#_{i,i+|L|-1}^{\mathcal{P},\sigma}) \\
 &= \llbracket \tau \rrbracket(\#_{i,j-1}^{\mathcal{P},\sigma}) - \tau_L \\
 &\geq \llbracket \tau \rrbracket(\#_{i,j-1}^{\mathcal{P},\sigma}) \\
 &\geq b
 \end{aligned}$$

and  $j$  thus witnesses also that  $\varphi$  holds at position  $i + |L|$ . Otherwise,  $j + |L| < i + 2|L|$  is either still a position with a state from  $L$  or a position on the succeeding equally labelled row. In both cases  $\lambda(\sigma(i) \dots \sigma(j)) = \lambda(\sigma(i + |L|) \dots \sigma(j + |L|))$  and therefore  $\psi \in \lambda(\sigma(j + |L|))$ ,  $\chi \in \lambda(\sigma(i + |L|) \cap \dots \cap \lambda(\sigma(j + |L|)))$ , and

$$\begin{aligned}
 \llbracket \tau \rrbracket(\#_{i+|L|,j+|L|-1}^{\mathcal{P},\sigma}) &= \llbracket \tau \rrbracket(\lambda^\#(\sigma(i + |L|) \dots \sigma(j + |L| - 1))) \\
 &= \llbracket \tau \rrbracket(\lambda^\#(\sigma(i) \dots \sigma(j - 1))) \\
 &\geq b.
 \end{aligned}$$

Consequently, if the satisfaction at some state of  $L$  changes from one iteration to the next at all, there is a first such position  $h \in \text{pos}_\sigma(L)$  with  $(\mathcal{P}, \sigma, h) \not\models \varphi$ ,  $h + |L| \in \text{pos}_\sigma(L)$ , and  $(\mathcal{P}, \sigma, h + |L|) \models \varphi$ . With  $h$  being chosen minimal, the lemma statement holds for all  $i \in \text{pos}_\sigma(L)$  with  $i < h$ . For all other such positions  $i \geq h + \hat{n}|L|$ , we show that  $(\mathcal{P}, \sigma, i) \models \varphi$  in the following.

Let  $j \geq h + |L|$  be a position on  $\sigma$  witnessing that  $\varphi$  holds at  $h + |L|$ . This position must be beyond the last position of  $L$  because otherwise it would imply that the position  $j - |L|$  witnesses the satisfaction of  $\varphi$  at  $h$ . More precisely, assume towards contradiction that  $j \in \text{pos}_\sigma(L)$ . Since  $j \geq h + |L|$  and thus  $h \leq j - |L| \in \text{pos}_\sigma(L)$ , we have  $\psi \in \lambda(\sigma(j - |L|)) = \lambda(\sigma(j))$ . Each position of  $L$  is labelled by  $\chi$ , in particular those from  $h$  to  $j - |L|$ , and, as above,

$$\llbracket \tau \rrbracket(\#_{h,j-|L|-1}^{\mathcal{P},\sigma}) = \llbracket \tau \rrbracket(\#_{h+|L|,j-1}^{\mathcal{P},\sigma}) \geq b.$$

Now, knowing  $j \notin \text{pos}_\sigma(L)$  we conclude that  $h < h + \hat{n}|L| \leq i < j$  and hence all positions from  $i$  to  $j - 1$  are labelled by  $\chi$ . Also, there is  $i' \in [i + 1, i + |L|]$  such that  $i' = h + |L| + n|L|$  for some  $n \geq \hat{n}$  and by the same reasoning as for Equation (4.9) we

have  $|\llbracket \tau \rrbracket(\#_{i,i'-1}^{\mathcal{P},\sigma})| \leq \hat{n}$  and thus  $\llbracket \tau \rrbracket(\#_{i,i'-1}^{\mathcal{P},\sigma}) \geq -\hat{n}$ . Hence,

$$\begin{aligned} \llbracket \tau \rrbracket(\#_{i,j-1}^{\mathcal{P},\sigma}) &= \llbracket \tau \rrbracket(\#_{h+|L|,j-1}^{\mathcal{P},\sigma}) - \llbracket \tau \rrbracket(\#_{h+|L|,i'-1}^{\mathcal{P},\sigma}) + \llbracket \tau \rrbracket(\#_{i,i'-1}^{\mathcal{P},\sigma}) \\ &\geq \llbracket \tau \rrbracket(\#_{h+|L|,j-1}^{\mathcal{P},\sigma}) - n \cdot \tau_L - \hat{n} \\ &\geq \llbracket \tau \rrbracket(\#_{h+|L|,j-1}^{\mathcal{P},\sigma}) && \text{(by } -\tau_L > 0, n \geq \hat{n}\text{)} \\ &\geq b \end{aligned}$$

providing that  $(\mathcal{P}, \sigma, i) \models \varphi$ . This shows that  $h$  is a valid pivot.  $\blacksquare$

### Imposing Consistency

Having established that there is a pivot position  $h$  for each loop in  $\mathcal{P}_N$ , we can stabilise all of them with respect to the formula  $\varphi$  reusing the construction developed in the context of guard formulae. Applying Lemma 4.26 to  $\mathcal{P}_N$  provides a derived APS  $\hat{\mathcal{P}}_N$  and a run  $\hat{\sigma}_N \in \text{runs}(\hat{\mathcal{P}}_N)$  still satisfying  $\Phi$  but on which all loops are stable with respect to  $\varphi$ .

In combination with the inherited  $N$ -consistency of  $\hat{\mathcal{P}}_N$  this implies that for every formula  $\psi \in M = N \cup \{\varphi\}$  and every state  $q \in Q$  either  $(\hat{\mathcal{P}}_N, \hat{\sigma}_N, i_q) \models \psi$  holds for all states  $i_q \in \text{pos}_{\hat{\sigma}_N}(q)$  or none of them. Therefore, the labelling function  $\lambda_M : Q \rightarrow 2^{\text{cLTL}}$  given by

$$\lambda_M(q) := \begin{cases} \hat{\lambda}_N(q) \cup \{\varphi\} & \text{if } \forall_{i_q \in \text{pos}_{\hat{\sigma}_N}(q)} : (\hat{\mathcal{P}}_N, \hat{\sigma}_N, i_q) \models \varphi \\ \hat{\lambda}_N(q) \setminus \{\varphi\} & \text{otherwise} \end{cases}$$

induced by  $\hat{\sigma}_N$  is not only well-defined but faithfully represents the semantics of all formulae from  $M$ . Exchanging the labelling function of  $\hat{\mathcal{P}}_N$  by  $\lambda_M$  now allows us to apply Lemma 4.17 consecutively to all row states  $r \in \hat{Q}_N$  of the APS. Since making  $r$  consistent by applying the lemma involves only adding an additional, fresh counter and dedicated guards, there is no interference between the application for different row states. Having made all row states consistent provides that the loop states are also consistent by condition D3 of Definition 4.11. Thus, after at most  $|\hat{Q}_N|$  applications of the lemma the final APS  $\mathcal{P}_M = (\hat{Q}_N, \Delta_M, \lambda_M, \widehat{\text{org}}_N)$  is obtained with size bounded by

$$|\mathcal{P}_M| \leq |\hat{\mathcal{P}}_N| + |Q_N| \cdot |\hat{\Delta}_N| \cdot |\varphi|.$$

### The Size of $\mathcal{P}_M$

Let us sum up the construction of  $\mathcal{P}_M$  for  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi$ :

1. Lemma 4.28 provides the existence of pivots for the loops in  $\mathcal{P}_N$  and allows for

applying Lemma 4.26 to obtain an all-stable version  $\hat{\mathcal{P}}_N$  of  $\mathcal{P}_N$  with

$$\begin{aligned} |\hat{\mathcal{P}}_N| &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot |\Delta_N|^2, \\ |\hat{Q}_N| &\leq (\hat{n} + 4) \cdot |Q_N|, \\ |\hat{\Delta}_N| &\leq (\hat{n} + 4) \cdot |\Delta_N|. \end{aligned} \quad \text{and}$$

2. The run  $\hat{\sigma}_N$  determines a labelling of  $\hat{\mathcal{P}}_N$ . Lemma 4.17, applied to all its row states, then provides the  $M$ -consistent APS  $\mathcal{P}_M$  of size

$$|\mathcal{P}_M| \leq |\hat{\mathcal{P}}_N| + |\hat{Q}_N| \cdot |\hat{\Delta}_N| \cdot |\varphi|.$$

To confirm the induction step let us estimate the size of  $|\mathcal{P}_M|$  by

$$\begin{aligned} |\mathcal{P}_M| &\leq |\hat{\mathcal{P}}_N| + |\hat{\Delta}_N|^2 \cdot |\Phi| \\ &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 2 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot |\Delta_N|^2 \\ &\quad + (\hat{n} + 4)^2 \cdot |Q_N|^2 \cdot |\Phi| \\ &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 3 \cdot |\Phi| \cdot (\hat{n} + 4)^3 \cdot |\Delta_N|^2 \\ &\leq (\hat{n} + 7) \cdot |\mathcal{P}_N| + 3 \cdot |\Phi| \cdot (\hat{n} + 4)^{2|N|+3} \cdot |\Delta_\rho|^2 \end{aligned}$$

using the induction hypothesis. Finally, we confirm the proof goal of the induction in analogy to the estimation for guard formulae in Equation (4.7).

#### 4.5.6 Summary

The base case and the exhaustive analysis of all structural cases of the formula  $\varphi$  have shown that the induction hypothesis holds for all downward-closed subsets of  $sub(\Phi)$  and therefore especially for  $sub(\Phi)$ . We conclude that the APS  $\mathcal{P}_\Phi$  proposed by Theorem 4.21 exists, witnessed by  $\mathcal{P}_{sub(\Phi)}$ . Recall that the base case used the APS  $\mathcal{P}_\rho$  constructed directly from the run  $\rho$  in  $\mathcal{S}$  with  $|\mathcal{P}_\emptyset| = |\mathcal{P}_\rho| \leq 4|\mathcal{S}|$  and  $|\Delta_\emptyset| = |\Delta_\rho| \leq 4|\Delta_\mathcal{S}|$ . Further, notice that  $\log_2(\hat{a}_\Phi \cdot |sub(\Phi)|) \leq size(\hat{a}_\Phi) + |sub(\Phi)| \leq |\Phi|$  and  $\log_2(\hat{u}_\mathcal{S} \cdot |Q_\mathcal{S}|) \leq size(\hat{u}_\mathcal{S}) + |Q_\mathcal{S}| \leq |\mathcal{S}|$ . The size of  $\mathcal{P}_\Phi$  can hence be bounded exponentially in  $|\Phi|^2$  and

$|\mathcal{S}|$  by

$$\begin{aligned}
 |\mathcal{P}_\Phi| &\leq (\hat{n} + 7)^{|\text{sub}(\Phi)|} \cdot |\mathcal{P}_\rho| + 3 \cdot |\Phi| \cdot |\text{sub}(\Phi)| \cdot (\hat{n} + 7)^{3|\text{sub}(\Phi)|} \cdot |\Delta_\rho|^2 \\
 &\leq (\hat{n} + 7)^{|\text{sub}(\Phi)|} \cdot 4|\mathcal{S}| + 3 \cdot |\Phi| \cdot |\text{sub}(\Phi)| \cdot (\hat{n} + 7)^{3|\text{sub}(\Phi)|} \cdot 4^2 \cdot |\mathcal{S}|^2 \\
 &\leq 2^{|\Phi| \cdot (|\Phi| \cdot |\mathcal{S}| + 3)} \cdot 4|\mathcal{S}| + 48 \cdot |\Phi|^2 \cdot |\mathcal{S}|^2 \cdot 2^{3|\Phi| \cdot (|\Phi| \cdot |\mathcal{S}| + 3)} \\
 &\leq 2^{k \cdot |\Phi|^2 \cdot |\mathcal{S}|}
 \end{aligned}$$

for some constant  $k \in \mathbb{N}$ . This concludes the proof of Theorem 4.21.

## 4.6 Discussion

The present chapter has demonstrated that the model-checking problem of cLTL is decidable over flat counter systems. To this end, augmented path schemas were introduced as a compact symbolic representation of runs of counter systems along with additional semantic information. They serve to witness that a cLTL formula  $\Phi$  is satisfied. In fact, obeying the consistency criterion, the labelling by subformulae of  $\Phi$  can be understood as a representation of a proof that the contained runs satisfy the formula. The construction of such a schema from only a satisfying run provides the completeness of the guess-and-check procedure and an upper bound on their size establishes a complexity bound.

The potential estimate appears high in comparison to the PSPACE-bound for LTL over Kripke structures and the NP-bound over flat counter systems. Although counting is an extremely powerful mechanism and it is not uncommon to arrive at such high complexities, unfortunately it remains open whether the blow-up is unavoidable and thereby the apparent question for a precise lower bound of the problem.

One source of exponential growth in the construction is that constants used in constraints can potentially lead to unstable parts on a run of linear size in their value. As demonstrated in Figure 4.10, representing an unstable phase of loop iterations explicitly in a path schema by unfolding the loop may result in a structure of exponential size. Reducing the complexity of the procedure would hence require representing such unstable parts much more concisely.

Even if we consider unary encoding of constants, another source of the exponential growth would have to be considered. The construction handles nested counting operators by potentially duplicating each loop for every additional nesting level. Further investigation would therefore be necessary to understand the effect of nested counting operators on the structure of a run and whether exponentially many copies of a single loop are actually required to represent it.

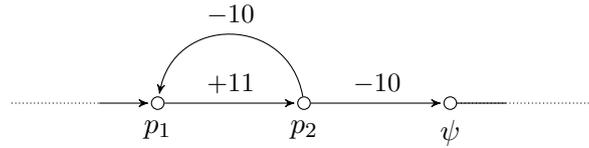


Figure 4.10: Sketch of a loop that can result in a large number of unstable iterations. For example, a formula  $\varphi = true \mathbf{U}_{[11p_1-10p_2 \geq 10]} \psi$  would be violated at both loop states during the last 10 iterations. Before, it holds at the first state but not when visiting the second. Only before the last 20 iterations would  $\varphi$  hold at both states and the unstable part of a corresponding run would thus be of length 10; in general, it is linear in the absolute values used in the constraint and exponential in their representation size. The edge labels indicate the effect of visiting the states on the value of the counting constraints.

Nevertheless, the developed technique provides effective means to reason on counter systems with respect to properties expressed in cLTL. In the next chapter, we will see how this can be used as the basis of a concrete verification approach.

## SMT-based Flat Model Checking

Both counter systems as system model and counting temporal logic as specification language are very powerful means to state verification tasks. As was observed above (Theorem 4.1), the corresponding decision problems are undecidable in general and the decision procedure developed in the previous chapter relies on *flatness* of the system model. The aim of this chapter is to advance from the theoretical developments towards a more concrete approach to program verification. It follows closely the presentation of [DP19].

To maximise the utility of the verification approach, the general restriction to flat systems shall not be imposed on the programs to be verified. Instead, augmented path schemas will be used as *flat under-approximations* in order to employ the power of the unrestricted formalisms while benefiting from the improved complexity. Using flat systems as under-approximation in verification tasks was suggested by Demri, Dhar and Sangnier [DDS15]. In general, using approximations to approach otherwise unsolvable problems has proven to be a successful strategy in software verification. Both under- and over-approximation techniques are described in the literature and implemented in powerful tools. The latter refers to considering a superset of the system’s behaviour, i.e., of its runs. The method can be used to show the *absence* of a particular run in the original system by showing its absence in an even larger superset. The intention is to obtain a more concise representation and thereby a verification speed-up. A prominent technique to obtain an over-approximation is to consider *abstract states* representing sets of original states or configurations, e.g. characterised by particular logical predicates [GS97; CU98] (see also [Cla+18, Chapters 13 and 15]). It can be combined with iterated refinement of the abstraction, as used in *counterexample-guided abstraction refinement* [Cla+03].

The complementary approach is to under-approximate the behaviour of a system in terms of a subset of its runs. While the absence of a particular run does not provide a conclusive result in this case, it allows for proving the *existence* of a witness—and thus to confirm an existential property or refute a universal property. A well-known

method of under-approximation is *bounded model checking* [Bie+99; Bie09] where a given property is evaluated on finite prefixes of system runs. A prefix of length  $k$  provides all necessary information to evaluate lasso-shaped runs of the form  $s_0s_1 \dots (s_\ell \dots s_{k-1})^\omega$  visiting up to  $k$  different states. The transition relation and the semantics of an LTL property is formulated in propositional logic and unfolded  $k$  times. A solution to the resulting formula then represents a satisfying run. An advantage of the method is that improving the under-approximation amounts to simply increasing the parameter  $k$  (as opposed to, e.g., computing suitable predicates for a more precise abstraction). Beginning with small values for  $k$  and dynamically increasing the verification depth provides the potential of finding witnesses early without evaluating the system exhaustively. In result, bounded-model checkers are very competitive when it comes to quickly finding witnesses, as suggested by the results of the Software Verification Competition [Bey17].

Similarly to bounded model checking, the approach proposed here employs a *depth* parameter to control the size of the flat under-approximation and allows the user to flexibly adjust the trade-off between exhaustiveness and computational effort. Essential advantages of APS, however, are that they are more concise and that they can represent infinite subsets of runs instead of the finite number of lassos represented by one or finitely many prefixes. Limiting the size of APS can be understood as bounding the number of loop alternations in the original system, while still admitting any number of iterations.

Figure 5.1 shows an example of a counter system  $\mathcal{S}$  and the sketch of a path schema—a special case of flat systems—representing an infinite subset of the runs of  $\mathcal{S}$ . It demonstrates how a flat system of sufficient depth can accommodate a bounded number of loop alternations. A run through  $\mathcal{S}$  may alternate, e.g., the loops  $L_1$  and  $L_2$  arbitrarily and this is not possible in a single flat system, by the very nature of the concept. However, increasing the number of available states allows for representing an increased number of alternations and, hence, a more complicated shape. Further, more complex but periodic looping patterns can be represented by combining several iterations of different loops in  $\mathcal{S}$  into one larger loop of the approximation.

Notice that, when increasing the approximation depth to include one more alternation, an infinite number of additional runs is represented—and verified—at once. In contrast, the finite prefixes considered in bounded model checking would always limit not only the structure of the represented runs but also the number of times any loop can be iterated. Even for a given number of loop iterations to be represented, the required depth is much larger because they have to be unfolded entirely. For example, the run specified by the APS and the numbers of loop iterations in Figure 5.1 would require an unfolded prefix of length 50 while the APS requires only 24 states.

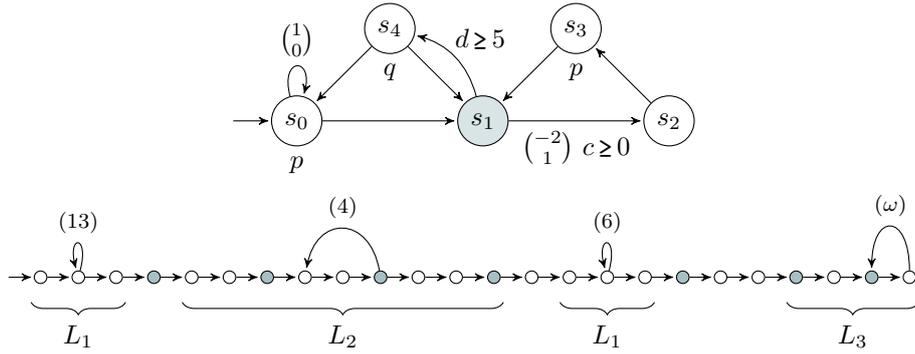


Figure 5.1: A diagram showing a counter system  $\mathcal{S}$  (top) over propositions  $AP = \{p, q\}$  and counters  $\{c, d\}$  as well as a sketch of an APS  $\mathcal{P}$  (bottom) that alternates loops  $L_1 = \{s_0\}$  and  $L_2 = \{s_1, s_2, s_3\}$  of  $\mathcal{S}$  unfolded as required by Definition 4.4. The numbers above loops specify one of the runs represented by  $\mathcal{P}$ . Notice that the original loops in  $\mathcal{S}$  are iterated two times more than the corresponding loops in  $\mathcal{P}$  due to the additional unfoldings, e.g.,  $L_1$  is iterated 15 times consecutively on the represented original run before visiting  $L_2$ .

Considering first a small depth and increasing it only if no witness was found allows for finding “simple” witnesses quickly where they exist, even for complex path properties. Recall that the underlying theory provides a bound on the maximal depth that needs to be considered in the case of a flat system. The method is (necessarily) incomplete in the general case but can nevertheless be directly applied.

Increasing the depth of a flat under-approximation is similar to so-called *loop acceleration* in symbolic verification. It aims at stepping over an arbitrary number of consecutive iterations of a loop during state space exploration by symbolically representing its effect. Since this is particularly effective for simple loops, flatness is a desired property [Bar+05] also in this setting. Unfortunately, acceleration typically concerns the computation of reachability sets [Bar+05; Bey+07; Can+08; KW10; Hoj+12] and is thus insufficient when analysing *path properties* as expressible in (extensions of) LTL. For accelerating the latter, flat systems and path schemas in particular provide a suitable symbolic model since they represent entire runs.

Based on the theory developed in Chapter 4, an explicit formulation of the (approximated) model-checking problem in quantifier-free Presburger arithmetic (**qfPA**) is described in the subsequent chapter. Recall that Presburger arithmetic is first-order logic over the natural numbers with addition. The satisfiability problem is decidable [Pre29] and in the case of the quantifier-free fragment in NP [BT76]. Importantly, the theory of **qfPA** is well-supported by a number of competitive SMT solvers (cf. [CSW15]). The construction

is parametrised by the depth of the flat approximation that is to be verified, and the resulting qfPA formula is linear in the problem size and the chosen depth.

### Flat Model Checking

Recall that, given a counter system  $\mathcal{S}$  and a cLTL formula  $\Phi$  the *existential model-checking problem* for cLTL is to decide whether  $\mathcal{S} \models \Phi$ , i.e., to compute if  $\mathcal{S}$  contains a run satisfying  $\Phi$ . Recall also that the problem is undecidable for two reasons: First, counter systems extend Minsky machines [Min67] and even LTL can express their undecidable (control-state) reachability problem. Second, cLTL extends fLTL and checking a universal Kripke structure encodes its undecidable satisfiability problem (cf. Chapter 3).

Let us therefore approach a parametrised approximation of the problem that we call *flat model checking*. It considers only runs with a specific shape, namely those represented by *path schemas*. Recall that a path schema [LS04; DDS15] is characterised by a (connected) sequence  $u_0v_0u_1v_1 \dots u_mv_m$  of paths  $u_i$  and loops  $v_i$  of  $\mathcal{S}$ . It represents all those runs  $\rho$  of  $\mathcal{S}$  that traverse a state sequence of the form  $u_0v_0^{k_0} \dots u_{m-1}v_{m-1}^{k_{m-1}} u_mv_m^\omega$  for some  $k_0, \dots, k_{m-1} \in \mathbb{N}$ . Restricting the length of such a schema effectively controls how complicated the shape of the considered runs can be. In particular, it bounds the cycle alternation performed by a run.

► **Definition 5.1** (Flat model checking). *Let  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  be a counter system and  $n \in \mathbb{N}$ . The flat approximation of depth  $n$  of  $\mathcal{S}$  is the set*

$$FA(\mathcal{S}, n) := \{ \rho \in runs(\mathcal{S}) \mid \exists_{u_0, v_0, \dots, u_m, v_m \in S^*} \exists_{k_0, \dots, k_{m-1} \in \mathbb{N}} : |u_0v_0u_1v_1 \dots u_mv_m| \leq n \\ \wedge st(\rho) = u_0v_0^{k_0} \dots u_{m-1}v_{m-1}^{k_{m-1}} u_mv_m^\omega \}.$$

The flat model-checking problem is to decide for  $\mathcal{S}$ ,  $n$ , and a cLTL formula  $\varphi$ , whether there is a run  $\rho \in FA(\mathcal{S}, n)$  with  $(\mathcal{S}, \rho) \models \varphi$ , denoted  $FA(\mathcal{S}, n) \models \varphi$ .

A flat approximation  $FA(\mathcal{S}, n)$  induces a flat counter system  $\mathcal{F}$  with  $FA(\mathcal{S}, n) = runs(\mathcal{F})$  and thus a series  $(\mathcal{F}_n)_{n \in \mathbb{N}}$  of flat counter systems representing an increasing number of runs of  $\mathcal{S}$ . Flat model checking can hence be understood as verifying the  $n$ th system in this series providing the computational benefits of flatness in the concrete case. As mentioned earlier, this is similar to bounded model checking, where the approximation is prefix-based and represents only a finite number of runs.

Notice that, even if  $\mathcal{S}$  is *not flat*, each run contained in the flat approximation  $FA(\mathcal{S}, n)$  of  $\mathcal{S}$  can by definition be represented by an APS in  $\mathcal{S}$  of size  $n$ . Therefore,  $FA(\mathcal{S}, n)$  also

yields an exponential witness for non-emptiness on account of the construction proving Theorem 4.21.

► **Corollary 5.2.** *If  $FA(\mathcal{S}, n) \models \Phi$  then there is a non-empty and  $\Phi$ -consistent APS in  $\mathcal{S}$  with initial state labelled by  $\Phi$  and of at most exponential size in  $n$  and  $\Phi$ .*

## 5.1 From Flat Model Checking to Presburger Arithmetic

For solving the flat model-checking problem of a counter system  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  and a formula  $\Phi \in \text{cLTL}(C_S)$ , for  $C_S := \text{counters}(\mathcal{S})$ , the previous developments devise the search for an augmented path schema  $\mathcal{P}$  in  $\mathcal{S}$  that is  $\Phi$ -consistent, labelled initially by  $\Phi$  and non-empty. This section presents a formulation of this search in quantifier-free Presburger arithmetic, aiming for an SMT-based implementation. The idea is to encode an APS of size  $n \in \mathbb{N}$  and a run of it as valuation of a set of first-order variables. A formula  $\text{fmc}(\mathcal{S}, \Phi, n)$  is constructed to be satisfiable if there is a run  $\rho \in FA(\mathcal{S}, n)$  satisfying  $\Phi$  and such that any solution represents a valid witness that  $\mathcal{S} \models \Phi$ .

We only encode such augmented path schemas  $\mathcal{P} = (Q, \Delta_{\mathcal{P}}, \lambda_{\mathcal{P}}, \text{org})$  where the states are natural numbers  $Q = \{0, \dots, n-1\}$ , ordered according to the natural ordering. That is, the forward transitions from  $\Delta_{\mathcal{P}}$  point forward along this relation (cf. Definition 4.4). This is not an essential restriction as the assumption can be imposed on an arbitrary APS by an isomorphic renaming of the states. Also, it is assumed that the labelling  $\lambda_{\mathcal{P}} : Q \rightarrow 2^{\text{sub}(\Phi)}$  consists only of subformulae of  $\Phi$  since other formulae are irrelevant. Under these assumptions, it suffices to encode explicitly only the beginning and end of loops, the origin and labelling of each state, as well as a run in terms of the number of iterations for each loop. Further, the formula expresses the satisfaction of all encountered guards to ensure the validity of the run as well as the consistency criterion.

The presented formula does not only contain first-order variables for integer numbers but also of natural, Boolean, and enumeration types (sorts). They can, theoretically, be encoded into integers but are more readable and directly supported by, e.g., the **z3** SMT solver [MB08]. The notation  $\text{var} : X$  is used to denote that some variable symbol  $\text{var}$  is of some sort  $X$ . A mapping  $f : Y \rightarrow X$  with some finite domain  $Y = \{y_1, \dots, y_{|Y|}\}$  can be represented by (a valuation of) variable symbols  $f_{y_1}, \dots, f_{y_{|Y|}} : X$  that will mostly be denoted concisely by a single variable  $f : X^Y$ . For  $y \in Y$  let  $f(y)$  denote  $f_y$ . Subsets of a finite domain  $Y$ , are represented in terms of mappings holding their characteristic function. That is, variable symbols  $\text{set} : 2^Y$  are represented as those of type  $\{0, 1\}^Y$ . For  $y \in Y$  let  $y \in \text{set}$  and  $y \notin \text{set}$  abbreviate the equality terms  $\text{set}(y) = 1$  and  $\text{set}(y) = 0$ , respectively. The shorthand  $\text{ite}(\text{cond}, \text{prop}, \text{alt})$  represents the *if-then-else* construct

equivalent to  $(cond \rightarrow prop) \wedge (\neg cond \rightarrow alt)$ .

Figure 5.2 depicts an example of an APS  $\mathcal{P}$  and its representation in terms of first-order variables and their valuation. For every state  $i \in Q$ , we encode the positions of loops in terms of a variable  $\mathbf{typ}_i : \{\boxminus, \triangleright, \boxplus, \triangleleft\}$  that indicates whether it is outside ( $\boxminus$ ), inside ( $\boxplus$ ), the beginning ( $\triangleright$ ), or the end ( $\triangleleft$ ) of a loop. For easier reading, expressions of the form  $\mathbf{typ}_i = \diamond$  may be abbreviated by  $\diamond_i$  for  $\diamond \in \{\boxminus, \triangleright, \boxplus, \triangleleft\}$ . The origin is represented by a variable  $\mathbf{org}_i : S$  and the labelling by  $\mathbf{lbl}_i : \{0, 1\}^{sub(\Phi)}$ , describing the set  $\lambda_{\mathcal{P}}(i) \subseteq sub(\Phi)$ . In the following, the formula  $\mathbf{fmc}(\mathcal{S}, \Phi, n)$  is constructed to formulate the flat model-checking problem in **qfPA**. It is parametrised by the counter system  $\mathcal{S}$ , the cLTL formula  $\Phi$  to be checked, and the approximation depth  $n$ . The formula

$$\mathbf{fmc}(\mathcal{S}, \Phi, n) := \mathbf{aps}(\mathcal{S}, n) \wedge \mathbf{run}(\mathcal{S}, n) \wedge \mathbf{consistency}(n, \Phi) \wedge \Phi \in \mathbf{lbl}_0$$

is comprised of different components that are dedicated to specific aspects of the encoding. Further, it expresses that the first state of the encoded APS is labelled by  $\Phi$ .

To allow for a simplified presentation, let us assume that there is at most one transition between every two states of  $\mathcal{S}$ , thus being uniquely identified by  $\mathbf{org}_i$  and  $\mathbf{org}_{i+1}$ . The assumption could be eliminated by adding  $2n$  additional variables determining explicitly which transition is selected for the represented APS.

### 5.1.1 Basic Structure

The basic structure of APS is specified as **qfPA** formula

$$\mathbf{aps}(\mathcal{S}, n) := \mathbf{org}_0 = s_I \wedge \mathbf{typ}(n) \wedge \mathbf{labels}(\mathcal{S}, n) \wedge \mathbf{transitions}(\mathcal{S}, n).$$

It states that  $s_I$  is the origin of the first state and that loops are delimited by  $\triangleright$  and  $\triangleleft$  in terms of the formula

$$\mathbf{typ}(n) := \bigwedge_{i \in [1, n-1]} \mathbf{ite}(\triangleleft_{i-1} \vee \boxminus_{i-1}, \boxminus_i \vee \triangleright_i, \boxplus_i \vee \triangleleft_i).$$

To express that the labelling of states by propositions coincides with that of  $\mathcal{S}$ , the formula

$$\mathbf{labels}(\mathcal{S}, n) := \bigwedge_{\substack{p \in AP, \\ i \in [0, n-1]}} p \in \mathbf{lbl}_i \leftrightarrow \bigvee_{s \in \lambda^{-1}(p)} \mathbf{org}_i = s$$

is used.

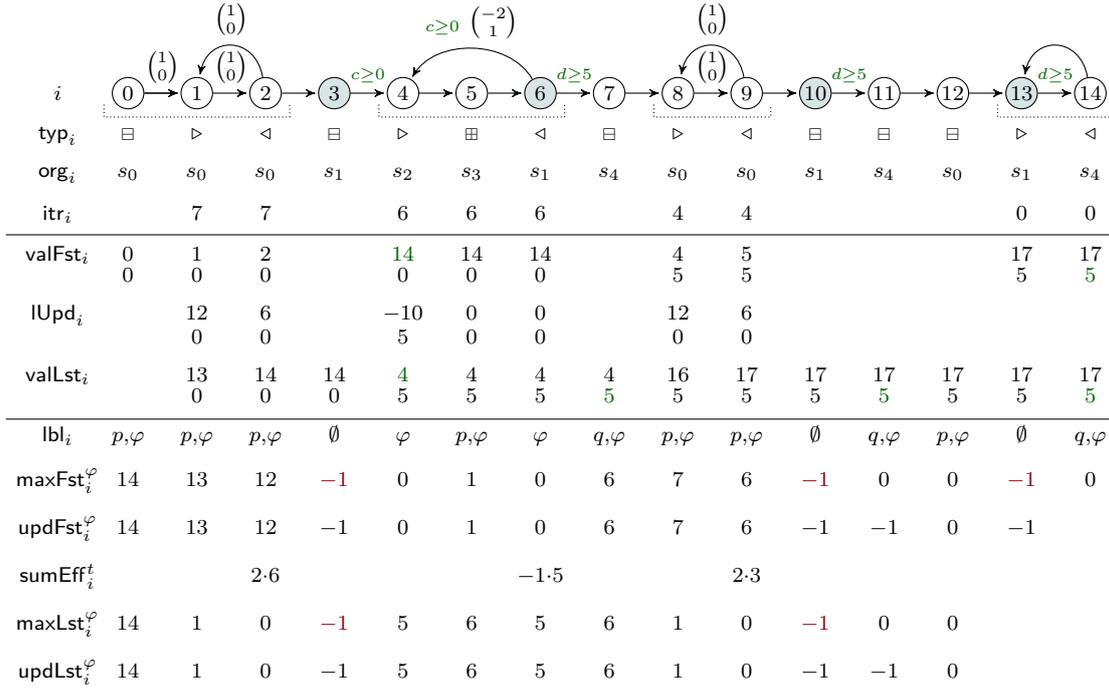


Figure 5.2: Example of the encoding of the run and path schema shown in Figure 5.1 with consistent labelling by  $\varphi = \text{true} \mathbf{U}_{[p \rightarrow \neg p \geq 0]} q$ . It demonstrates propagation of counter values and the best witness position for  $\varphi$ . Some variables are omitted for readability. Notice how the single-state loops are represented by half as many iterations of two-state loops. In case the number of iterations is odd, such as for the first loop representing iterations of  $L_1$  in Figure 5.1, an equivalent APS can be assumed where the first iteration is unfolded.

**Transitions.** One way to express that each backward transition from the last to the first state of a loop has a correspondence in  $\mathcal{S}$  is to build a constraint over all pairs of states from  $Q$ . This is, however, quadratic in  $n$  and we therefore use a propagation scheme introducing  $n$  additional variables  $\text{orgAtEnd}_i : S$ . We let them equal  $\text{org}_i$  where  $\text{typ}_i = \triangleleft$  and otherwise be copied from  $\text{orgAtEnd}_{i+1}$ , thus propagating backward the origin of the last state of every loop. The corresponding formula is

$$\begin{aligned} \text{orgAtEnd}(n) &:= \text{orgAtEnd}_{n-1} = \text{org}_{n-1} \\ &\wedge \bigwedge_{i \in [0, n-2]} \text{ite}(\triangleleft_i, \text{orgAtEnd}_i = \text{org}_i, \text{orgAtEnd}_i = \text{orgAtEnd}_{i+1}). \end{aligned}$$

The formula

$$\text{transitionsBwd}(\mathcal{S}, n) := \bigwedge_{i \in [0, n-1]} \triangleright_i \rightarrow \bigvee_{(s, \mu, \Gamma, s') \in \Delta} \text{orgAtEnd}_i = s \wedge \text{org}_i = s'$$

then guarantees that all backward transitions exist in  $\mathcal{S}$ . Similar propagation chains will be used at other occasions to avoid a quadratic blow-up of the formula size due to information non-locality. Forward transitions are specified similarly, but without the need for propagation, by

$$\text{transitionsFwd}(\mathcal{S}, n) := \bigwedge_{i \in [1, n-1]} \bigvee_{(s, \mu, \Gamma, s') \in \Delta} \text{org}_{i-1} = s \wedge \text{org}_i = s'.$$

The combination of the formulae now defines the formula

$$\text{transitions}(\mathcal{S}, n) := \text{orgAtEnd}(n) \wedge \text{transitionsBwd}(\mathcal{S}, n) \wedge \text{transitionsFwd}(\mathcal{S}, n)$$

used as part of  $\text{aps}(\mathcal{S}, n)$  above. Notice that this encoding assumes a minimal loop length of 2 due to distinct positions for the first ( $\triangleright$ ) and the last ( $\triangleleft$ ) state of each loop. Single-state loops can still be represented as longer (e.g. two-state) loops by combining multiple iterations as one loop that is iterated less often (cf. Figure 5.2). Excluding single-state loops increases the upper bound for the size of path schemas only by one state per loop.

**Front and Rear Rows.** The definition of augmented path schemas demands that loops be surrounded by identical rows. Being identical, these rows are not represented explicitly in the encoding. Instead, runs will be assumed to traverse each representation of a loop at least three times, the first representing the front, the last representing the rear and the remaining representing the actual loop traversals. The construction will distinguish between the first, second and last iteration, where necessary. This is equivalent to representing the states of the front and rear rows individually but allows for a more compact encoding and also provides an efficient way to correlate every loop state to its correspondents on the front and rear.

### 5.1.2 Runs

The formula  $\text{run}(\mathcal{S}, n)$  specifies the shape and constraints of a run in the encoded schema. It has the form

$$\text{run}(\mathcal{S}, n) := \text{itr}(n) \wedge \text{valuations}(\mathcal{S}, n) \wedge \text{guards}(\mathcal{S}, n).$$

Variables  $\text{itr}_i : \mathbb{N}$  are used to indicate how often state  $i \in Q$  is visited and are thus constraint to equal 1 outside loops and to stay constant inside each loop. Since every loop state is used to also represent its counterpart on the front and rear, they are to be repeated at least three times. Infinite iteration of the last loop is represented by the otherwise unused value 0. This is formulated by

$$\text{itr}(n) := \text{itr}_{n-1} = 0 \wedge \bigwedge_{i \in [0, n-2]} (\exists_i \wedge \text{itr}_i = 1) \vee (\triangleleft_i \wedge \text{itr}_i > 2) \vee \text{itr}_i = \text{itr}_{i+1}.$$

The other components of the formula concern the valuation of counters and evaluation of transition guards. They are described in the following.

#### Counter Valuations

The valuation of any counter at any position on a run  $\rho$  of an encoded APS  $\mathcal{P}$  is determined unambiguously by the shape of  $\mathcal{P}$  (in terms of the sequence of states and their origins) and the number of repetitions of every loop. Yet, in order to formulate that guards need to be satisfied, the counter values will be made explicit in terms of variables  $\text{valFst}_i, \text{valSec}_i : \mathbb{Z}^{C_S}$  and  $\text{valLst}_i : \mathbb{Z}_{\infty}^{C_S}$  for every state  $i \in Q$ . They are supposed to hold the counter valuations at the first, second, and last occurrence, respectively, of state  $i$  on the represented run. Naturally, outside loops the first and last valuations are equal and the second does technically not exist, so  $\text{valSec}_i$  does not have a semantically meaningful value. Nevertheless, all variables are introduced for each state as loops may occur anywhere. Recall that the states of a loop also represent those of its front and rear rows, so the first and last iteration corresponds to those. The formula

$$\begin{aligned} \text{valuations}(\mathcal{S}, n) := & \text{valFst}_0 = \mathbf{0} \\ & \wedge \text{valRow}(\mathcal{S}, n) \wedge \text{valLoop}(\mathcal{S}, n) \\ & \wedge \text{valFstSecItr}(\mathcal{S}, n) \wedge \text{valLastItr}(\mathcal{S}, n) \\ & \wedge \text{valPropagation}(n) \wedge \text{loopUpdate}(\mathcal{S}, n) \end{aligned}$$

encodes the semantics of counter updates in terms of the valuations in the represented run. By definition, runs start with the valuation  $\mathbf{0}$ , assigning 0 to the whole domain. For row states  $i$  (of type  $\boxplus$ ) the valuation at its (first and only) occurrence is computed from the valuation at the last occurrence of the previous state  $i - 1$  by adding (elementwise) the update function  $\mu$  of the transition  $(\text{org}(i - 1), \mu, \Gamma, \text{org}(i)) \in \Delta$  from  $i - 1$  to  $i$ . As mentioned earlier, the first and last occurrence are the same and the variable  $\text{valSec}_i$  is deliberately set to equal them as well but could as well be left unconstrained. Hence, let

$$\text{valRow}(\mathcal{S}, n) := \bigwedge_{i \in [1, n-1]} \boxplus_i \rightarrow \bigwedge_{(s, \mu, \Gamma, s') \in \Delta} \text{org}_{i-1} = s \wedge \text{org}_i = s' \rightarrow$$

$$\text{valFst}_i = \text{valSec}_i = \text{valLst}_i = \text{valLst}_{i-1} + \mu.$$

Recall that we assume that there is at most one transition between every two states in  $\mathcal{S}$ . Inside  $(\boxplus)$  and at the end of loops  $(\triangleleft)$ , the counter values are propagated individually for the first, second, and last iteration, expressed by

$$\text{valLoop}(\mathcal{S}, n) := \bigwedge_{i \in [1, n-1]} \boxplus_i \vee \triangleleft_i \rightarrow$$

$$\bigwedge_{(s, \mu, \Gamma, s') \in \Delta} \text{org}_{i-1} = s \wedge \text{org}_i = s' \rightarrow \left( \begin{array}{l} \text{valFst}_i = \text{valFst}_{i-1} + \mu \\ \wedge \text{valSec}_i = \text{valSec}_{i-1} + \mu \\ \wedge \text{valLst}_i = \text{valLst}_{i-1} + \mu \end{array} \right).$$

At the beginning  $(\triangleright)$  of a loop the value in the first iteration is computed from the preceding position. The first value in the second iteration is to be computed from the last value of the first iteration. However, given a state  $i$ , it cannot be determined *a priori* which state exactly constitutes the end of the loop. To obtain the value of the state that happens to be the last on the loop, variables  $\text{valFstAtEnd}_i$  are introduced to hold the valuation at the last state during the first iteration throughout the loop and make it thus directly accessible at the beginning. They are defined using a propagation scheme as above expressed by

$$\text{valPropagation}(n) := \text{valFstAtEnd}_{n-1} = \text{valFst}_{n-1} \wedge$$

$$\bigwedge_{i \in [0, n-2]} \text{ite}(\triangleleft_i, \text{valFstAtEnd}_i = \text{valFst}_i, \text{valFstAtEnd}_i = \text{valFstAtEnd}_{i+1}).$$

Then,  $\text{valSec}_i$  can be set to  $\text{valFstAtEnd}_i + \mu$  where  $\mu$  comes from the incoming *backward* transition of state  $i$ . This is specified by

$$\text{valFstSecItr}(\mathcal{S}, n) := \bigwedge_{\substack{i \in [1, n-1] \\ (s, \mu, \Gamma, s') \in \Delta}} \triangleright_i \rightarrow \left( \begin{array}{l} (\text{org}_{i-1} = s \wedge \text{org}_i = s' \rightarrow \text{valFst}_i = \text{valLst}_{i-1} + \mu) \\ \wedge (\text{orgAtEnd}_i = s \wedge \text{org}_i = s' \rightarrow \text{valSec}_i = \text{valFstAtEnd}_i + \mu) \end{array} \right).$$

Having a direct handle on the valuations in the first and second iteration (in terms of the variables  $\text{valFst}_i$  and  $\text{valSec}_i$ ) as well as the total number of loop iterations ( $\text{itr}_i$ ), it is tempting to specify the valuations in the last iteration simply by

$$\text{valLst}_i = \text{valFst}_i + (\text{valSec}_i - \text{valFst}_i) \cdot (\text{itr}_i - 1).$$

Unfortunately, this formula uses multiplication of variables and hence exceeds Presburger arithmetic. Therefore, we need to specify the value of  $(\text{valSec}_i - \text{valFst}_i) \cdot (\text{itr}_i - 1)$  differently. Instead, the updates over the second to last loop iteration are accumulated in an explicit variable  $\text{IUpd}_i$  such that  $\text{valLst}_i$  can be set to  $\text{valFst}_i + \text{IUpd}_i$ . We express this accumulation by the formula

$$\begin{aligned} \text{loopUpdate}(\mathcal{S}, n) := & \bigwedge_{i \in [1, n-2]} \bigwedge_{(s, \mu, \Gamma, s') \in \Delta} \\ & (\triangleleft_i \wedge \text{org}_{i-1} = s \wedge \text{org}_i = s' \quad \rightarrow \text{IUpd}_i = \mu \cdot \text{itr}_i - \mu) \\ & \wedge (\boxplus_i \wedge \text{org}_{i-1} = s \wedge \text{org}_i = s' \quad \rightarrow \text{IUpd}_i = \mu \cdot \text{itr}_i - \mu + \text{IUpd}_{i+1}) \\ & \wedge (\triangleright_i \wedge \text{orgAtEnd}_i = s \wedge \text{org}_i = s' \quad \rightarrow \text{IUpd}_i = \mu \cdot \text{itr}_i - \mu + \text{IUpd}_{i+1}). \end{aligned}$$

Essentially, the multiplication by  $\text{itr}_i$  is distributed over the individual transition updates along the loop. This is admissible because the individual updates  $\mu$  appear in the formula not as variables but as constants. Notice that this formulation deliberately multiplies functions with integers, which is to be understood as point-wise application. Further, the choice of using 0 to mark the infinite iteration of the last loop (as opposed to, e.g.,  $\infty$ ) is useful here because otherwise the equation would not be well defined, a negative and a positive update could result in having to add  $-\infty$  and  $\infty$ . In the formulation above,  $\text{IUpd}_i$  is always zero for states  $i$  on the last loop but this is no problem because this particular situation can be handled using  $\text{valFst}_i$  and  $\text{valSec}_i$ . Observe also that the variable  $\text{IUpd}_i$  holds only intermediate results inside and at the end of loops and is undefined outside. Only for states  $i$  that are the beginning of a loop, it holds the precise

accumulated loop effect but this suffices since this value is propagated as specified by the formula  $\text{valLoop}(\mathcal{S}, n)$  above.

Using  $\text{IUpd}_i$ , the calculation of the valuations in the last iteration of a loop is now formulated as

$$\begin{aligned} \text{valLastItr}(\mathcal{S}, n) := & \bigwedge_{i \in [1, n-1]} \triangleright_i \rightarrow \bigwedge_{(s, \mu, \Gamma, s') \in \Delta} \text{orgAtEnd}_i = s \wedge \text{org}_i = s' \rightarrow \\ & \text{ite} \left( \text{itr}_i > 0, \text{valLst}_i = \text{valFst}_i + \text{IUpd}_i, \bigwedge_{c \in C_S} (\text{valFst}_i(c) = \text{valSec}_i(c) = \text{valLst}_i(c)) \right. \\ & \quad \vee (\text{valFst}_i(c) > \text{valSec}_i(c) \wedge \text{valLst}_i(c) = -\infty) \\ & \quad \left. \vee (\text{valFst}_i(c) < \text{valSec}_i(c) \wedge \text{valLst}_i(c) = \infty) \right). \end{aligned}$$

### Transition Guards

To ensure that the represented run is valid it must satisfy all the transition guards at any time. The formula  $\text{valuations}(\mathcal{S}, n)$  developed above ensures that the variables  $\text{valFst}_i$ ,  $\text{valSec}_i$ , and  $\text{valLst}_i$  faithfully provide the counter valuations when reaching the state  $i \in Q$  for the first, the second and the last time, respectively. Recall that, due to flatness, each loop is entered and left only once. Since every guard of the counter system is a linear inequality and the effect of the updates of any specific loop is constant, it suffices to check the guard in the first and last traversal in order to guarantee that it is satisfied throughout all repetitions of a particular loop state.

For a constraint term over  $C_S$  of the form  $\tau = \sum_{j=0}^{\ell} a_j c_j$  and a variable symbol  $\text{var} : \mathbb{Z}^{C_S}$ , let  $\tau[\text{var}] := \sum_{j=0}^{\ell} a_j \cdot \text{var}(c_j)$  denote the syntactic substitution of the counter names by the variable symbol (representing the value of)  $\text{var}(c_j)$ , in analogy to the evaluation of constraint terms using valuations (cf. Section 2.1.2). The formula

$$\begin{aligned} \text{guardsFwd}(\mathcal{S}, n) := & \bigwedge_{\substack{i \in [1, n-1], \\ (s, \mu, \Gamma, s') \in \Delta}} \text{org}_{i-1} = s \wedge \text{org}_i = s' \rightarrow \\ & \bigwedge_{(\tau \geq b) \in \Gamma} \tau[\text{valFst}_i] \geq b \wedge (\neg \triangleright_i \rightarrow \tau[\text{valLst}_i] \geq b) \end{aligned}$$

then specifies that the encoded run satisfies the guards whenever taking a forward transition. Recall that for some counter  $c \in C_S$  the variable  $\text{valLst}_i(c)$  may be assigned a symbolic value. Thus, a proper interpretation (or expansion) of  $\geq$  is assumed such that  $\infty \geq b$  holds for every  $b \in \mathbb{Z}$  while  $-\infty \geq b$  holds for none. Notice that the (forward)

transition from state  $i - 1$  to state  $i$  is not taken at the beginning of the last iteration of a loop and thus, its guard must not be checked for the corresponding valuation. Instead, the guard of the backward transition pointing to  $i$  must be verified. This transition is taken by the encoded run for the first time when entering the second loop iteration. The guards of backward transitions are thus reflected exhaustively by

$$\text{guardsBwd} := \bigwedge_{\substack{i \in [1, n-1], \\ (s, \mu, \Gamma, s') \in \Delta}} \triangleright_i \wedge \text{orgAtEnd}_i = s \wedge \text{org}_i = s' \rightarrow \bigwedge_{(\tau \geq b) \in \Gamma} \tau[\text{valSec}_i] \geq b \wedge \tau[\text{valLst}_i] \geq b.$$

Thereby we complete the definition of the formula

$$\text{guards}(\mathcal{S}, n) := \text{guardsFwd}(\mathcal{S}, n) \wedge \text{guardsBwd}(\mathcal{S}, n)$$

and the specification of proper runs in terms of the formula  $\text{run}(\mathcal{S}, n)$ .

### 5.1.3 Consistency

The formulae constructed above describe the fact that there is some non-empty augmented path schema in the counter system  $\mathcal{S}$  of which the first state is labelled by  $\Phi$ . In the following, we develop the components of the formula

$$\begin{aligned} \text{consistency}(n, \Phi) := & \bigwedge_{(-\varphi) \in \text{sub}(\Phi)} \text{consistencyNeg}(n, \varphi) \\ & \wedge \bigwedge_{\varphi \wedge \psi \in \text{sub}(\Phi)} \text{consistencyAnd}(n, \varphi, \psi) \\ & \wedge \bigwedge_{(\tau \geq b) \in \text{sub}(\Phi)} \text{consistencyGrd}(n, \tau, b) \\ & \wedge \bigwedge_{\varphi \in \text{sub}(\Phi)} \text{consistencyX}(\varphi) \\ & \wedge \bigwedge_{\chi \text{U}_{[\tau \geq b]} \psi \in \text{sub}(\Phi)} \text{consistencyU}(n, \chi, \psi, \tau, b) \end{aligned}$$

stating that this APS is consistent. Recall that  $\Phi$ -consistency requires all states of an APS to be consistent with respect to all subformulae of  $\Phi$ . Definition 4.11 discriminates the structural cases of a cLTL formula and therefore the components of the qfPA formulation cover one case each and impose consistency of all states for one subformula of  $\Phi$  at a time.

### Propositions and Boolean Combinations

Condition B of Definition 4.11 can almost literally be translated to **qfPA** formulae

$$\text{consistencyNeg}(n, \varphi) := \bigwedge_{i \in [0, n-1]} (\neg \varphi) \in \text{lbl}_i \leftrightarrow \varphi \notin \text{lbl}_i$$

and

$$\text{consistencyAnd}(n, \varphi, \psi) := \bigwedge_{i \in [0, n-1]} (\varphi \wedge \psi) \in \text{lbl}_i \leftrightarrow \varphi \in \text{lbl}_i \wedge \psi \in \text{lbl}_i.$$

### Guard Formulae

Concerning condition A, counter guard formulae of the form  $\tau \geq b$  are not modelled explicitly. Rather, the formula

$$\text{consistencyGrd}(n, \tau, b) := \bigwedge_{i \in [0, n-1]} (\tau \geq b) \in \text{lbl}_i \leftrightarrow \tau[\text{valFst}_i] \geq b \wedge \tau[\text{valLst}_i] \geq b.$$

imposes that the represented run satisfies the constraints as if they were transition guards on all incoming transitions on any state labelled by a guard formula. Recall that it suffices to assert that the guard is satisfied at the first and last occurrence of a state.

### Temporal Next

To express condition C of the consistency definition, concerning *temporal next* formulae, variables  $\text{lblAtBeg}_i : 2^{\text{sub}(\Phi)}$  are used to propagate labelling information from the first state of a loop forward towards its end. Similar to the backward propagation of the origin, let

$$\text{propagateX}(n, \varphi) := \bigwedge_{i \in [1, n-1]} \text{ite}(\triangleright_i, \varphi \in \text{lblAtBeg}_i \leftrightarrow \varphi \in \text{lbl}_i, \varphi \in \text{lblAtBeg}_i \leftrightarrow \varphi \in \text{lblAtBeg}_{i-1}).$$

Notice that it is not necessary to determine the propagation value at the first ( $i = 0$ ) state because it is never part of a loop. The condition is now specified by

$$\begin{aligned} \text{consistencyX}(n, \varphi) := & \text{propagateX}(n, \varphi) \wedge (\mathbf{X} \varphi \in \text{lbl}_{n-1} \leftrightarrow \varphi \in \text{lblAtBeg}_{n-1}) \\ & \wedge \bigwedge_{i \in [0, n-2]} \text{ite}(\mathbf{X} \varphi \in \text{lbl}_i, \varphi \in \text{lbl}_{i+1} \wedge (\triangleleft_i \rightarrow \varphi \in \text{lblAtBeg}_i), \\ & \varphi \notin \text{lbl}_{i+1} \wedge (\triangleleft_i \rightarrow \varphi \notin \text{lblAtBeg}_i)). \end{aligned}$$

**Temporal Until: Condition D1**

Consider a formula  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi \in \text{sub}(\Phi)$ . The consistency criterion considers three conditions for until formulae of that form. Towards defining the corresponding **qfPA** formula  $\text{consistencyU}(n, \chi, \psi, \tau, b)$  consider first condition D1 stating, essentially, that the last loop exhibits a positive effect that eventually proves the formula to hold. To express the requirements of that condition, the following information is required. Given a state  $i \in [0, n - 1]$ , first of all, it must be labelled by  $\varphi$  and that information is available in terms of the value of the variable  $\text{lbl}_i$ . Second, assume a variable  $\text{acc}_0^\tau : \mathbb{Z}$  holding the accumulated effect of the last loop on the value of  $\tau$ . Third, let  $\text{onLast}^\psi : \mathbb{B}$  be set to true if and only if  $\psi$  occurs as label on some state of the last loop and  $\text{glob}_i^\chi : \mathbb{B}$  hold if and only if  $\chi$  holds globally from state  $i$  on. Then, condition D1 is expressed by

$$\text{conD1}(\varphi, i) := \varphi \in \text{lbl}_i \wedge \text{acc}_0^\tau > 0 \wedge \text{onLast}^\psi \wedge \text{glob}_i^\chi.$$

It remains to formulate the side conditions guaranteeing that the variables actually hold the assumed value.

**Accumulated effect of the last loop.** To describe the accumulated value of  $\tau$  on a single iteration of the last loop we introduce  $\text{acc}_i^\tau$  not only for  $i = 0$  but for each  $i \in [0, n - 1]$ . The idea is now to accumulate backwards from  $\text{acc}_{n-1}^\tau$  to  $\text{acc}_0^\tau$  the effects  $\tau[\text{lbl}_i]$  as long as  $i$  is part of the last loop (identified by  $\text{itr}_i$  being equal 0). Let

$$\begin{aligned} \text{accu}(n, \tau) := \\ \text{acc}_{n-1}^\tau = \tau[\text{lbl}_{n-1}] \wedge \bigwedge_{i \in [0, n-2]} \text{ite}(\text{itr}_i = 0, \text{acc}_i^\tau = \text{acc}_{i+1}^\tau + \tau[\text{lbl}_i], \text{acc}_i^\tau = \text{acc}_{i+1}^\tau). \end{aligned}$$

It implies, as intended, that  $\text{acc}_0^\tau$  holds the effect of the last loop on the value of  $\tau$ .

**Reachability of defect- and witness states.** Consider the evaluation of whether  $\chi$  holds globally at all *reachable* states. For loop states  $i \in Q$ , this means that not only the successors  $j \geq i$  must be labelled by  $\chi$  but the whole loop. Therefore, we employ a propagation scheme with two passes. First, a backward propagation imposes that variables  $\text{prpg}_i^\chi$  hold if and only if all states  $j \geq i$  are labelled by  $\chi$ . Based on this information, the intended valuation for  $\text{glob}_i^\chi$  is enforced by a forward propagation. The

formula

$$\begin{aligned} \text{glob}(n, \chi) := & (\text{prpg}_{n-1}^{\chi} \leftrightarrow \chi \in \text{lbl}_{n-1}) \wedge \left( \bigwedge_{i \in [0, n-2]} \text{prpg}_i^{\chi} \leftrightarrow \text{prpg}_{i+1}^{\chi} \wedge \chi \in \text{lbl}_i \right) \\ & \wedge (\text{glob}_0^{\chi} \leftrightarrow \text{prpg}_0^{\chi}) \wedge \bigwedge_{i \in [1, n-1]} \text{glob}_i^{\chi} \leftrightarrow \text{ite}(\exists_i \vee \triangleright_i, \text{prpg}_i^{\chi}, \text{glob}_{i-1}^{\chi}) \end{aligned}$$

implies that each variable  $\text{glob}_i^{\chi}$  is true if and only if  $\chi$  is labelled at all states reachable from  $i$ . The information whether  $\psi$  holds somewhere on the last loop is made available in terms of the variable  $\text{onLast}^{\psi}$  by

$$\text{fin}(n, \psi) := \text{onLast}^{\psi} \leftrightarrow \bigvee_{i \in [0, n-1]} \text{itr}_i = 0 \wedge \psi \in \text{lbl}_i.$$

### Temporal Until: Condition D2

Condition D2 demands the existence or absence of a witness state proving that a formula  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi \in \text{sub}(\Phi)$  holds. As before, it would be inefficient to model balance counters and the guards required by the criterion explicitly. Instead, a formulation is developed that assures that the encoded APS can be assumed to have the necessary counters and guards.

For example, assume some state  $i$  is to be labelled by  $\varphi$  and consider the best (maximal) value of the term  $\tau$  on a path starting at state  $i$  and leading to some state satisfying  $\psi$ , without violating  $\chi$  in between. If that value is at least  $b$ , then there is a state at which a balance counter  $c_{\tau, i}$  for  $\tau$  and  $i$  would have precisely this value and checking the constraint  $c_{\tau, i} \geq b$  would succeed. On the other hand, if the best value is below  $b$ , then there is no such state. Even, the dual constraint could be added to any potential witness state and the encoded run would still be valid.

Consider an APS  $\mathcal{P}$  in  $\mathcal{S}$  with states  $Q = [0, n-1]$  and assume it is consistent with respect to all strict subformulae of  $\varphi$  and admits a run  $\sigma \in \text{runs}(\mathcal{P})$ . Let  $x_{\text{last}} := \min \text{pos}_{\sigma}(n-1)$  be the first position of state  $n-1$  on  $\sigma$  and let  $\text{maxWit}_{\varphi}^{\mathcal{P}, \sigma} : \mathbb{N} \rightarrow \mathbb{Z}_{\infty}$  denote the discussed function defined for  $x \in \mathbb{N}$  by

$$\begin{aligned} \text{maxWit}_{\varphi}^{\mathcal{P}, \sigma}(x) := & \\ \max(\{ \llbracket \tau \rrbracket (\#_{x, y-1}^{\mathcal{P}, \sigma}) \mid x \leq y \leq x_{\text{last}}, (\mathcal{P}, \sigma, y) \models \psi, \forall y' \in [x, y-1] : (\mathcal{P}, \sigma, y') \models \chi \} \cup \{-\infty\}). & \end{aligned}$$

We make three essential observations regarding  $\text{maxWit}_{\varphi}^{\mathcal{P}, \sigma}$ .

First, consider the positions  $x \leq x_{\text{last}} - |\text{lastl}(\mathcal{P})|$  preceding the last loop. For those,

$\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(x)$  accurately determines the maximal value for  $\tau$  (the symbolic value  $-\infty$  expressing non-existence of a witness position) unless condition D1 applies to  $st(\sigma(x))$ . Assuming that there is a witness position  $z \geq x_{\text{last}}$ , the last loop must be entirely labelled by  $\chi$  and if  $\llbracket \tau \rrbracket(\#_{x,z-1}^{\mathcal{P},\sigma}) > \max\text{Wit}_\varphi^{\mathcal{P},\sigma}(x)$ , the effect of the final loop on  $\tau$  must be positive.

Second, if one of conditions D(2)i and D(2)ii applies to a (row) state  $i \in Q$ , then there cannot be a witness position for  $\varphi$  holding at  $i$ , especially not before  $x_{\text{last}}$ , and thus  $\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(x_i) < b$  for  $\text{pos}_\sigma(i) = \{x_i\}$  (cf. Section 4.3.3). On the other hand, if  $\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(x_i) < b$ , then one of conditions D1 and D(2)i applies or any balance counter  $c_{\tau,i}$  for  $i$  and  $\tau$  would satisfy the guard  $c_{\tau,i} < b$  at any witness position  $j \geq i$  for  $\varphi$ . In the latter case it can thus be assumed that state  $i$  obeys condition D(2)ii in  $\mathcal{P}$ —recall the construction in the proof of Lemma 4.17, Case 2, that adds a balance counter and corresponding constraints under these conditions without relevant side effects.

Third, a similar point can be made for conditions D(2)iii and D(2)iv given that  $\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(x_i) \geq b$ . These conditions imply that there, in fact, is a witness position for  $\varphi$  before the end of the second iteration of the last loop. Vice versa, the definition of  $\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(x_i)$  demands for some witness position  $y \geq x_i$ . Again, the construction of a consistent APS in the proof of Lemma 4.17, Case 1, has shown that if this witness exists, then one of conditions D1 and D(2)iii holds or condition D(2)iv can be established without adding extra states. Hence,  $i$  can be assumed to obey one of the conditions in  $\mathcal{P}$ .

Based on these considerations, we introduce variables  $\max\text{Fst}_i^\varphi$  and  $\max\text{Lst}_i^\varphi$  for each state  $i \in [0, n-1]$  and *until* formula  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi \in \text{sub}(\Phi)$  that are supposed to represent the value  $\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(\min \text{pos}_\sigma(i))$  at the first occurrence of  $i$  and the value  $\max\text{Wit}_\varphi^{\mathcal{P},\sigma}(\max \text{pos}_\sigma(i))$  at the last position of  $i$ , respectively. Recall that these positions cover only rows as the first and last iteration of loops represent their front and rear, respectively. Notice also that the latter value is not defined for positions belonging to the last loop. Then, condition D2 is formulated for a state  $i$  as

$$\begin{aligned} \text{conD2}(\varphi, i) := & \quad (\varphi \in \text{lbl}_i \leftrightarrow \max\text{Fst}_i^\varphi \geq b) \\ & \wedge ((\varphi \in \text{lbl}_i \leftrightarrow \max\text{Lst}_i^\varphi \geq b) \vee \text{itr}_i = 0). \end{aligned}$$

### Temporal Until: Maximal Value to Witness

The intended value for these variables is specified using a suffix-optimum backward propagation scheme initiated at the end of the represented schema. We can characterise

the values  $\max\text{Wit}_{\varphi}^{\mathcal{P},\sigma}(x)$  by

$$\max\text{Wit}_{\varphi}^{\mathcal{P},\sigma}(x_{\text{last}}) = \begin{cases} 0 & \text{if } \psi \in \lambda(\sigma(x_{\text{last}})) \\ -\infty & \text{otherwise} \end{cases}$$

and for  $x \in [0, x_{\text{last}} - 1]$  by

$$\max\text{Wit}_{\varphi}^{\mathcal{P},\sigma}(x) = \begin{cases} -\infty & \text{if } \chi, \psi \notin \lambda(\sigma(x)) \\ 0 & \text{if } \chi \notin \lambda(\sigma(x)) \text{ and} \\ & \psi \in \lambda(\sigma(x)) \\ \max\text{Wit}_{\varphi}^{\mathcal{P},\sigma}(x+1) + \llbracket \tau \rrbracket(\lambda(\sigma(x))) & \text{if } \chi \in \lambda(\sigma(x)) \text{ and} \\ & \psi \notin \lambda(\sigma(x)) \\ \max\{\max\text{Wit}_{\varphi}^{\mathcal{P},\sigma}(x+1) + \llbracket \tau \rrbracket(\lambda(\sigma(x))), 0\} & \text{if } \chi, \psi \in \lambda(\sigma(x)). \end{cases}$$

As long as  $\chi$  holds, the maximal value is propagated backwards. When the chain breaks at some defect state, no witness position is properly reachable, and the maximal value is set to  $-\infty$ . Each state of the schema where  $\psi$  holds is a potential witness for preceding states. Thus, if the propagated value at this point is less than 0, this state will generally provide a better value for  $\tau$  than any of its successors. In the **qfPA** formulation, the above definition is split into the *computation* of the updated value  $\max\text{Wit}_{\varphi}^{\mathcal{P},\sigma}(x+1) + \llbracket \tau \rrbracket(\lambda(\sigma(x)))$  potentially propagated to its predecessor and the actual *selection* of the appropriate value depending on the case. To express the update across a loop, it is further necessary to express its effect. These aspects are reflected in the components of the formula

$$\text{witnessMax}(n, \varphi) := \text{selectMax}(n, \varphi) \wedge \text{calcUpdated}(n, \varphi) \wedge \text{loopEffect}(n, \tau).$$

**Selection and auxiliary iteration.** Recall that the encoding does not represent every position of the run and not even every state of the path schema explicitly, namely those situated on loops. However, the values at the front and rear row of a loop are represented and the propagation scheme hence needs to “jump” from the rear to the front, that is, extrapolate the calculated value over the iterations of the loop. For that purpose, an additional set of auxiliary variables  $\text{maxAux}_i^{\varphi}$  are introduced representing, intuitively, the first actual iteration of a loop—similarly to the variables  $\text{valSec}_i$  above. Thus, the auxiliary variables complement the variables  $\text{maxFst}_i^{\varphi}$  and  $\text{maxLst}_i^{\varphi}$  representing the front and rear rows, respectively.

The case selection is expressed for all three variants by the formula

$$\text{selectMax}(n, \varphi) := \bigwedge_{i \in [0, n-1]} \left( \begin{array}{l} (\chi \notin \text{lbl}_i \wedge \psi \notin \text{lbl}_i \rightarrow \max\text{Fst}_i = \max\text{Aux}_i = \max\text{Lst}_i = -\infty) \\ \wedge (\chi \notin \text{lbl}_i \wedge \psi \in \text{lbl}_i \rightarrow \max\text{Fst}_i^\varphi = \max\text{Aux}_i = \max\text{Lst}_i^\varphi = 0) \\ \wedge (\chi \in \text{lbl}_i \wedge \psi \notin \text{lbl}_i \rightarrow \left( \begin{array}{l} \max\text{Fst}_i^\varphi = \text{updFst}_i^\varphi \\ \wedge \max\text{Aux}_i^\varphi = \text{updAux}_i^\varphi \\ \wedge \max\text{Lst}_i^\varphi = \text{updLst}_i^\varphi \end{array} \right)) \\ \wedge (\chi \in \text{lbl}_i \wedge \psi \in \text{lbl}_i \rightarrow \left( \begin{array}{l} \max\text{Lst}_i^\varphi = \max(\text{updLst}_i^\varphi, 0) \\ \wedge \max\text{Aux}_i^\varphi = \max(\text{updAux}_i^\varphi, 0) \\ \wedge \max\text{Fst}_i^\varphi = \max(\text{updFst}_i^\varphi, 0) \end{array} \right)) \end{array} \right)$$

where the variables  $\text{updFst}_i^\varphi$ ,  $\text{updLst}_i^\varphi$ , and  $\text{updAux}_i^\varphi$  are assumed to hold the value from the state  $i + 1$  updated according to the labelling (or the respective initialisation). For easier reading, expressions of the form  $\text{var1} = \max(\text{var2}, a)$  are used to abbreviate  $\text{ite}(\text{var2} > a, \text{var1} = \text{var2}, \text{var1} = a)$ .

**Modelling loop effects.** The overall effect of (all iterations of) a loop on the value of  $\tau$  is made accessible in terms of variables  $\text{sumEff}_i^\tau$  where  $i$  is the first state of a loop. It is obtained by summing up the individual contribution  $\tau[\text{lbl}_i] \cdot (\text{itr}_i - 3)$  of each loop state  $i$  bound to variables  $\text{eff}_i^\tau$ . The effect is multiplied only by  $\text{itr}_i - 3$  since the first (front), second (auxiliary), and last (rear) iteration is already accounted for explicitly. To circumvent multiplication of variables in the formula, the variables  $\text{eff}_i^\tau$  are themselves defined by distributing the factor  $(\text{itr}_i - 3)$  over the sum of monomials of the term  $\tau$ , as was necessary also for the accumulation of counter updates. The term is assumed to have the form  $\tau = \sum_{k=0}^m a_k \chi_k$  and the effect is hence specified by

$$\text{loopEffect}(n, \tau) := \left( \bigwedge_{i \in [1, n-2]} (\triangleright_i \rightarrow \text{sumEff}_i^\tau = \text{eff}_i^\tau) \wedge (\boxplus_i \vee \triangleleft_i \rightarrow \text{sumEff}_i^\tau = \text{sumEff}_{i-1} + \text{eff}_i^\tau) \right) \wedge \bigwedge_{i \in [0, n-1]} \left( \begin{array}{l} \text{ite}(\chi_0 \in \text{lbl}_i, \text{eff}_i^{\tau,0} = a_0 \cdot \text{itr}_i - 3a_0, \text{eff}_i^{\tau,0} = 0) \\ \wedge \bigwedge_{k \in [1, m]} \text{ite}(\chi_k \in \text{lbl}_i, \text{eff}_i^{\tau,k} = \text{eff}_i^{\tau,k-1} + a_k \cdot \text{itr}_i - 3a_k, \text{eff}_i^{\tau,k} = \text{eff}_i^{\tau,k-1}) \end{array} \right)$$

where the variables  $\text{eff}_i^\tau = \text{eff}_i^{\tau,m}$  are to be considered identical.

**Calculating values to propagate.** Using the summed-up loop effect, we can now formulate the actual computation of the (potentially) propagated optimum by

$$\begin{aligned}
 \text{calcUpdated}(n, \varphi) := & \\
 & \bigwedge_{i \in [0, n-2]} \left( \begin{array}{l}
 (\boxplus_i \rightarrow \text{updFst}_i^\varphi = \text{updLst}_i^\varphi = \text{maxFst}_{i+1}^\varphi + \tau[|bl_i|]) \\
 \wedge (\triangleleft_i \rightarrow \text{updLst}_i^\varphi = \text{maxFst}_{i+1}^\varphi + \tau[|bl_i|] \\
 \quad \wedge \text{updFst}_i^\varphi = \text{maxAuxAtBeg}_i^\varphi + \tau[|bl_i|] \\
 \quad \wedge \text{updAux}_i^\varphi = \text{maxLst}_i^\varphi + \text{sumEff}_i^\tau) \\
 \wedge (\triangleright_i \vee \boxplus_i \rightarrow \text{updLst}_i^\varphi = \text{maxLst}_{i+1}^\varphi + \tau[|bl_i|] \\
 \quad \wedge \text{updFst}_i^\varphi = \text{maxFst}_{i+1}^\varphi + \tau[|bl_i|] \\
 \quad \wedge \text{updAux}_i^\varphi = \text{maxAux}_{i+1}^\varphi + \tau[|bl_i|])
 \end{array} \right) \\
 & \wedge \text{ite}(\psi \in |bl_{n-1}|, \text{updAux}_{n-1}^\varphi = 0, \text{updAux}_{n-1}^\varphi = -\infty) \\
 & \quad \wedge \text{updFst}_{n-1}^\varphi = \text{maxAuxAtBeg}_{n-1}^\varphi + \tau[|bl_{n-1}|] \\
 & \wedge \text{maxAuxAtBeg}_0^\varphi = \text{maxAux}_0^\varphi \\
 & \quad \wedge \bigwedge_{i \in [1, n-1]} \text{ite}(\triangleright_i, \text{maxAuxAtBeg}_i^\varphi = \text{maxAux}_i^\varphi, \text{maxAuxAtBeg}_i^\varphi = \text{maxAuxAtBeg}_{i-1}^\varphi)
 \end{aligned}$$

where  $\varphi = \chi \mathbf{U}_{[\tau \geq b]} \psi$  is assumed.

The formula  $\text{calcUpdated}(n, \varphi)$  consists of three parts: the first specifies the updated value, depending on the type of state, the second sets the starting value for the propagation at state  $n - 1$  for the auxiliary track on which all others depend, and the third makes the value of the auxiliary variables at the begin of each loop available at the corresponding end.

Consider the first part. Outside of loops (type  $\boxplus$ ), the first and last encounter of any state fall together, and the updated value is simply calculated from the succeeding position, being the first occurrence of the succeeding state. A state of type  $\triangleleft$  marks the end of a loop where the variable  $\text{maxLst}_i^\varphi$  represent the very last state of its rear row and is hence treated just as other row states. As mentioned earlier, the auxiliary track can be considered as the first actual iteration of the loop, thus immediately following the front row. The value of its last state is determined by extrapolating the value at the start of the rear over all iterations by adding the effect of all loop iterations in between. This may in fact be the correct value of  $\text{maxWit}_{\varphi}^{\mathcal{P}, \sigma}$  at this point. However, in case there is a defect on the loop or the effect of the loop is negative, the witness assumed by the extrapolation is not reachable without violating  $\chi$  in between or may not provide the maximal value for  $\tau$ , respectively. Nevertheless, as the value is passed along it traverses

all positions of the loop. Then, if the loop does have a defect, the selection determined by the formula `selectMax` would necessarily reset that value to either 0 or  $-\infty$  and provide a correct value from that point on. Similarly, if the overall effect of the loop is negative and there is a witness providing a higher value of  $\tau$ , this witness would be found on the first iteration and the selection would again promote this one as soon as it is encountered. Hence, upon reaching the first state of the loop, the propagated value on the auxiliary track is in fact correct. It is transferred back to the end of the loop by the last part of the formula (by variables `maxAuxAtBegiφ`) and then used to correctly determine the value of `maxFstiφ`.

Therefore, assessing consistency condition D2 as stated by the formula `conD2` is appropriate, at least for those states to which condition D1 does not apply. Note that, if the latter does apply to some state, the evaluation of the other criterion is irrelevant.

### Temporal Until: Consistency

Based on the developments above, the cases for the consistency criterion are combined to express consistency for temporal until formulae  $\chi \mathbf{U}_{[\tau \geq b]} \psi$  by

$$\begin{aligned} \text{consistencyU}(n, \chi, \psi, \tau, b) := & \text{glob}(n, \chi) \wedge \text{accu}(n, \tau) \wedge \text{fin}(n, \psi) \\ & \wedge \text{witnessMax}(n, \chi \mathbf{U}_{[\tau \geq b]} \psi) \\ & \wedge \bigwedge_{i \in [0, n-1]} \text{conD1}(\chi \mathbf{U}_{[\tau \geq b]} \psi, i) \vee \text{conD2}(\chi \mathbf{U}_{[\tau \geq b]} \psi, i). \end{aligned}$$

The structure of the encoding assures that the actual loops are always identically labelled to their front and rear rows. Thus, assuring those are consistent, all loops automatically satisfy condition D3.

This completes the construction of the formula `consistency(S, n, Φ)` and thereby that of `fmc(S, n, Φ)`.

## 5.2 Properties of the Encoding

A solution to `fmc(S, Φ, n)` yields a  $\Phi$ -consistent APS in  $\mathcal{S}$  and a run. By Theorem 4.12 this implies that  $\mathcal{S}$  satisfies  $\Phi$ . On the other hand, Corollary 5.2 provides that if the flat approximation  $FA(\mathcal{S}, n)$  contains a run satisfying  $\Phi$ , then `fmc(S, Φ, 2p(n))` is satisfiable (for a fixed polynomial  $p$ ), and a smaller depth may suffice.

► **Theorem 5.3.** (i) If `fmc(S, Φ, n)` is satisfiable, then  $\mathcal{S} \models \Phi$ . (ii) If  $FA(\mathcal{S}, n) \models \Phi$ , then `fmc(S, Φ, 2p(n))` is satisfiable.

The encoding hence provides effective means to solve the flat model-checking problem based on qfPA satisfiability checking.

**Variables.** A major concern of the construction is to keep the formula as small as possible. Examining the indexing scheme of variables, observe that their number is linear in  $|\Phi| + |C_S|$  and  $n$ . Notice in particular that the number of variables  $\text{eff}_i^{\tau,k}$  is linear because  $0 \leq i < n$  and each pair  $(\tau, k)$  corresponds to one specific monomial of some constraint in  $\Phi$ . The number of variables required to represent a mapping (e.g.,  $\text{lbl}_i, \text{valFst}_i$ ) is linear in their domain ( $\text{sub}(\Phi)$  and  $C_S$ , respectively). The finite domain of variables  $\text{org}_i : S$  can be represented using naturals or explicit enumeration. The latter case would require  $n \cdot \lceil \log_2(|S|) \rceil$  Boolean variables.

**Formula length.** Recall that the size of  $\mathcal{S}$  is considered to be

$$|\mathcal{S}| = \text{size}(\Delta) = \sum_{(s,\mu,\Gamma,s') \in \Delta} 1 + \left( \sum_{\gamma \in \Gamma} |\gamma| \right) + \sum_{c \in \text{dom}(\mu)} \text{size}(\mu(c)).$$

The length of most parts of the formula  $\text{fmc}(\mathcal{S}, \Phi, n)$  only depend linearly on  $n$  or  $n \cdot |\Delta| \leq n \cdot |\mathcal{S}|$ . The parts encoding the guards in  $\mathcal{S}$  ( $\text{guardsFwd}(\mathcal{S}, n)$  and  $\text{guardsBwd}(\mathcal{S}, n)$ ) further depend (linearly) on the size of the guard sets associated to the transitions, more precisely, linearly on the total number  $\sum_{(s,\mu,\Gamma,s') \in \Delta} |\Gamma|$  of constraints present in  $\mathcal{S}$ . The constraint terms  $\tau$  are instantiated in substitution terms, such as  $\tau[\text{valFst}_i]$ , a bounded number of times per variable. Therefore, the formula size is linear in

$$\sum_{\substack{(s,\mu,\Gamma,s') \in \Delta, \\ \tau \geq b \in \Gamma}} |\tau| \leq |\mathcal{S}|.$$

The components of  $\text{consistency}(n, \Phi)$  cover the different types of cLTL formulae and express the corresponding consistency condition. The number of Boolean combinations and temporal-next formulae can be estimated by  $|\text{sub}(\Phi)|$  and the corresponding formulae are thus of linear size in  $n \cdot |\text{sub}(\Phi)|$ . The remaining part of  $\text{consistency}(n, \Phi)$  concerns guard- and until formulae and its size depends on the size of the constraint terms present in  $\Phi$ . More precisely, the size is linear in

$$\left( \sum_{\tau \geq b \in \text{sub}(\Phi)} |\tau| \right) + \left( \sum_{\chi \mathbf{U}_{[\tau \geq b]} \psi \in \text{sub}(\Phi)} |\tau| \right) \leq |\Phi|.$$

This sums up to a characterisation of the length of the formula  $|\text{fmc}(\mathcal{S}, n, \Phi)|$  to be

dominated by a function that is linear in  $n \cdot |\mathcal{S}| + n \cdot |\Phi|$  and thus  $|\text{fmc}(\mathcal{S}, n, \Phi)| \in \mathcal{O}(n \cdot (|\mathcal{S}| + |\Phi|))$ .

► **Theorem 5.4** (Formula size). *The length of  $\text{fmc}(\mathcal{S}, \Phi, n)$  is in  $\mathcal{O}(n(|\mathcal{S}| + |\Phi|))$ .*

### 5.3 Evaluation

To evaluate whether flat model checking and the **qfPA**-based encoding can be used to perform verification tasks, the procedure was implemented and applied to a set of problems provided by the RERS Challenge [How+14].

The tool `flat-checker`<sup>1</sup>, developed by Pirogov [Pir17] for this purpose, takes a **cLTL** specification, a counter system to be verified in DOT format [GN00] and the approximation depth (schema size) and performs the translation of the verification problem to a linear arithmetic formula. The SMT solver **z3** is used to compute a solution of the formula, if possible, that is subsequently interpreted as satisfying run and presented adequately to the user. The tool is developed in Haskell and provides a search mode that automatically increases the depth up to a given bound, in order to potentially find a small witness quickly, before investing computation time in large depths. A successful search can be continued to find a witness of smallest depth.

The RERS Challenge 2017<sup>2</sup> poses problems as C99 and Java programs that provide output depending on read input symbols and internal state. The programs have a regular structure but are inconceivable with reasonable effort. It features a track comprising 100 LTL formulae to be checked on a program (Problem 1) that is representable as a counter system by treating integer variables as counters. The counting mechanism of **cLTL** admits a more specific formulation of a correctness property, making it more restrictive or permissive than a plain LTL formula. For example, a typical pattern in the RERS problem set has the form  $\neg p \mathbf{U} q$ , stating  $q$  occurs before  $p$ . It can be relaxed to state, e.g.,  $p$  occurs at most 5 times ( $\mathbf{F}_{[p \leq 5]} q$ ) or less often than  $r$  ( $\mathbf{F}_{[p-r < 0]} q$ ). A stronger formulation would be that  $q$  must occur more often before  $p$  ( $\neg p \mathbf{U}_{[q \geq 5]} q$  or  $\neg p \mathbf{U}_{[r-q \geq 5]} q$ ). To evaluate the procedure on counting properties, variations of formulae from the LTL track were constructed to express relaxed or strengthened versions of the properties.

By checking negated properties, counterexamples were found at an approximation depth of at most 128 for all violated formulae, while most formulae could be falsified quickly. From the original 52 falsifiable LTL formulae, 43 were falsified after less than

<sup>1</sup><https://github.com/apirogov/flat-checker>

<sup>2</sup><http://www.rers-challenge.org/2017/>

200 seconds per formula at depth at most 64, the remaining 9 took at most 32 minutes per formula and depth 128. A batch analysis of the whole set of 100 formulae at depth 200 took a total of four days running time (Desktop PC, Intel i5-750 CPU, 4GB RAM). Some derived cLTL formulae took significantly longer to be evaluated than the original LTL formulation. However, in most cases, the introduction of counting constraints did not increase the evaluation effort significantly.

## 5.4 Conclusion

Characterising cLTL model checking over flat systems in Presburger arithmetic fills a gap between corresponding results for temporal logics with and without counting [DDS15; DDS14]. It can also be used as (semi-algorithmic) approach to the satisfiability and synthesis problems of cLTL.

**SMT-based model checking.** The model-checking procedure in Chapter 4 was developed in order to study the model-checking problem of cLTL and to prove that it can, in principle, be solved over flat systems. It is non-deterministic in nature and therefore hardly suitable for a literal implementation. At this point, logic programming serves well as prototyping methodology conveniently bridging the gap between the existential arguments in proof constructions and an executable implementation. The theory provides a framework including systematic vocabulary, mathematical objects as well as essential properties and guarantees that allow for reasoning about the problem. Specifically the notion of augmented path schemas and properties like consistency, as well as the constructions provide a basis for an implementation in general and for an alternative formulation of the non-deterministic procedure in terms of an arithmetic formula in particular.

Formulating a problem in terms of a general logical formalism and applying a generic solving procedure seems, at first, unlikely to provide performance comparable to a domain-specific algorithm. However, modern implementation frameworks, specifically constraint- and satisfiability solvers supporting additional first-order theories such as **z3** [MB08], **CVC4** [Bar+11], or **MathSAT5** [Cim+13] are well-engineered and optimised over years. The formalisms are well studied, and SMT-based verification tools are very effective (see [BDW18] for a unified survey).

**Depth-bounded approximation.** As for bounded model checking, the developed flat model-checking procedure encodes the verification of an under-approximation into the satisfiability problem of a classical logic. Instead of propositional logic, however, linear

arithmetic is used to encode the counting aspect of the system model and the specification language. Further, augmented path schemas as underlying structure provide a more general representation of runs than finite prefixes. The latter incorporate only one terminating loop while APS admit an arbitrary number of loops. Thereby, they represent a potentially infinite class of similarly-shaped runs and can be significantly more succinct. Nevertheless, the iterated refinement process is equally simple as it consists merely of increasing the depth parameter.

**Outlook.** It is clearly only the first, but a necessary step to investigate how to formulate the problem and also to study the relation of the model-checking and the decision problem. A concrete implementation provides first experience of how the method performs and can show promising advantages as well as bottle necks and engineering challenges. Although it may eventually hinder problem-specific optimisations, the SMT-based implementation benefits from the engineering effort put into solvers. The configurability of, e.g., `z3` using specific tactics, provides potential for future improvements. It remains to develop and compare different encoding variants. Especially, formulations that admit incremental solving could speed up the verification process.

The RERS Challenge strives to compare verification techniques and as such uses common standards such as LTL for specification. Although the primary ambition of the approach is to verify the more general class of cLTL properties, the first evaluation suggests that flat model checking is well applicable in a general verification context. Towards developing improved and alternative implementations, a specific and comprehensive suite of cLTL formulae remains to be defined in order to obtain comparable and reproducible performance measures.

The concise representation of runs allows for an accelerated evaluation of complex path properties and therefore flat approximation appears as a promising technique that deserves further investigation. The underlying theory provides that the procedure is complete on flat systems and, practically, an existing witness will be found eventually unless all of them have an infinitely aperiodic shape.

## Model-checking cCTL over Kripke Structures

As observed earlier in Chapter 3, satisfiability of cLTL formulae is undecidable. This implies the same for model-checking cLTL over Kripke structures and consequently LTL<sub>#</sub>, as well as cCTL\* and CTL\*<sub>#</sub> (Theorem 4.1). Moreover, the problems are also undecidable for CTL<sub>#</sub>, as shown by Laroussinie, Meyer and Pettonnet [LMP12] for a very similar logic. In contrast, we prove in this chapter that MC(cCTL, KS) is decidable in polynomial time. This developments are partly based upon [Dec+17, Section 3]. The published results are henceforth extended from fCTL to cCTL and improved from an exponential-time bound to an (optimal) polynomial bound.

To decide MC(cCTL,KS), we adapt the well-known labelling algorithm for CTL, first proposed by Clarke and Emerson [CE81] (see also the later article by Clarke, Emerson and Sistla [CES86] or the more recent textbook by Baier and Katoen [BK08]). Recall that it computes for each state of a Kripke structure inductively the set of satisfied subformulae. To compute if a state satisfies a counting formula of the form  $\varphi \mathbf{EU}_{[\tau \geq b]} \psi$ , we are going to translate the Kripke structure into a *one-counter system (1-CS)* in which a specific control state is reachable if and only if the formula holds. For formulae of the form  $\varphi \mathbf{AU}_{[\tau \geq b]} \psi$ , a similar system is constructed that is empty if and only if the formula holds.

The constructed system resembles the Kripke structure and uses the integer counter to track the value of the constraint term  $\tau$  along a run. Similar to the balance counters used in Chapter 4 (Definition 4.8), it is updated on every transition by the effect of the target state on the value of the term. For example, at some state labelled by the proposition  $p_1$  but not by  $p_2$ , a term  $\tau = p_1 - 2p_2$  evaluates to  $-2$ . If  $\tau$  contains non-atomic formulae instead of only propositions, it can still be evaluated, assuming that their satisfaction at the state has been computed before. The sum of such updates along a path in the structure provides the evaluation of the counting term on it. Now, a cCTL formula, e.g.,  $\varphi \mathbf{EU}_{[\tau \geq b]} \psi$  holds at some state if and only if a state satisfying  $\psi$  is reachable by a path

using only states satisfying  $\varphi$  and with a net effect on the counter of at least  $b$ .

Before assembling these steps formally to a model-checking procedure, let us settle the complexity of the reachability questions that need to be solved. The important observation is that if a 1-CS  $\mathcal{S}$  contains only guards of the form  $c \geq b$  for the unique counter  $c \in \text{counters}(\mathcal{S})$ , it can be computed in polynomial time whether a specific control state is (repeatedly) reachable from a given configuration. Let us call such counter systems 1-CS<sub>≥</sub> in the following.

## 6.1 Reachability and Emptiness in a Subclass of One-counter Systems

The (control-state) reachability problem for counter systems with one counter is already NP-hard, as observed by Haase et al. [Haa+09]. Hardness relies, however, on using *equality* in constraints. For solving our model-checking problem it suffices to restrict the use of guards to the specific form  $c \geq b$  and in that subclass, control-state reachability and emptiness can be decided *deterministically* in polynomial time.

In the remainder of this chapter, let us identify valuations  $\theta : \{c\} \rightarrow \mathbb{Z}$  with the value  $\theta(c)$ . Thus, we conveniently denote configurations of 1-CS by tuples from  $S \times \mathbb{Z}$  and transition updates by integer values. Let a *computation* in a 1-CS  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  be a finite or infinite sequence  $\delta_0 \delta_1 \dots \in \Delta^\infty$  of transitions with  $\delta_i = (s_i, u_i, \Gamma_i, s_{i+1})$  for all its positions  $i \in \mathbb{N}$ . It is *executable* from a configuration  $(s, z)$  of  $\mathcal{S}$  if  $s_0 = s$ , and  $z + \sum_{j=0}^i u_j \models_{\text{PA}} \Gamma_i$  for each position  $i$ . Let a target configuration  $(s', z')$  be called *reachable* from a source configuration  $(s, z)$  if there is a finite computation  $(s_0, u_0, \Gamma_0, s_1) \dots (s_n, u_n, \Gamma_n, s_{n+1}) \in \Delta^*$  executable from  $(s, z)$  such that  $s_{n+1} = s'$  and  $z' = z + \sum_{i=0}^n u_i$ . Let the (least upper) *reachable-value bound* for a configuration  $(s, z)$  and a state  $s' \in S$  be the supremum

$$rvb_{\mathcal{S}}(s, z, s') := \sup(\{z' \mid (s', z') \text{ is reachable from } (s, z) \text{ in } \mathcal{S}\} \cup \{-\infty\}) \in \mathbb{Z}_\infty.$$

It indicates the maximal counter value a state can be reached with or whether there is no such value because the state is not reachable at all ( $-\infty$ ) or reachable with an arbitrarily large counter value ( $\infty$ ).

The essential observation is that the function  $rvb_{\mathcal{S}}(s, z, s')$  can be computed in polynomial time if  $\mathcal{S}$  is a 1-CS<sub>≥</sub>. It can be used to decide (repeated) control-state reachability and thus emptiness.

► **Lemma 6.1.** *The reachable-value bound can be computed for 1-CS<sub>≥</sub> in polynomial time.*

**Proof.** Let  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  be a  $1\text{-CS}_\geq$  and  $(s_0, z_0)$  be the source configuration. To compute the reachable-value bound for all states  $s \in S$  we use an accelerated Bellman-Ford-like forward-propagation algorithm. Initially, every state is assigned the value  $-\infty$  (indicating non-reachability), except for the source state  $s_0$  that is assigned the starting value  $z_0$ . The algorithm first performs  $|S| - 1$  propagation rounds that, for each transition  $(s, u, \Gamma, s') \in \Delta$ , propagates the value assigned to  $s$  along the transition to  $s'$  while applying the update  $u$ . If this value satisfies the guards  $\Gamma$  and is greater than the value already assigned to  $s'$ , it replaces the previously assigned value. After  $|S| - 1$  rounds, like for the classic Bellman-Ford algorithm (but using greatest instead of smallest values), every state has been assigned a value that is (at least) as large as the greatest value with which it can be reached by a simple computation. Consequently, any value that still changes in a subsequent round is obtained by a computation path that contains a loop with positive net effect on the counter value. The values of all these states are thus accelerated, i.e., assigned the symbolic value  $\infty$  because, by iterating the positive loop, an arbitrarily large counter value can be realised. This acceleration may now “unlock” transitions that were previously unused because the corresponding thresholds required more iterations of positive loops. Therefore, another  $|S| - 1$  rounds of the forward propagation is performed subsequently. From any accelerated configuration, every reachable state can be reached by a direct, i.e. simple, computation. Therefore, this number of rounds suffices to reach a fixed point of the propagation procedure. Hence, the algorithm performs in summary  $2|S| - 1$  rounds to compute  $rvb_{\mathcal{S}}(s_0, z_0, s)$  (for each  $s \in S$ ). With each round touching each transition exactly once the algorithm uses only  $(2|S| - 1) \cdot |\Delta|$  steps. The procedure is summarised as pseudo code in Algorithm 1. ■

Using the computation of reachable-value bounds, we can now decide whether a given state is *repeatedly reachable*.

► **Theorem 6.2.** *The repeated control-state reachability problem for  $1\text{-CS}_\geq$  is in  $P$ .*

**Proof.** Let  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  be a  $1\text{-CS}_\geq$  and  $s \in S$ . To decide whether there is some run  $\rho \in \text{runs}(\mathcal{S})$  such that  $\text{pos}_\rho(s)$  is infinite, we first check if and with what counter value  $s$  is reachable from the initial configuration  $(s_I, 0)$  and, subsequently, if  $s$  is reachable from itself. Depending on the counter values realisable on a path from  $s$  to  $s$  we can determine whether any such path is repeatable arbitrarily often. Specifically, we proceed as follows.

Initially, compute  $m := rvb_{\mathcal{S}}(s_I, 0, s)$ . If  $m = -\infty$ , then  $s$  is not reachable and in particular not repeatedly. Otherwise, if  $m > -\infty$ , we check whether  $s$  is reachable from itself without taking any guarded transition. This requires a standard polynomial-time search on the control graph of  $\mathcal{S}$  ignoring the counter and all guarded transitions. Any

---

**Algorithm 1** Compute the reachable-value bound for a 1-CS<sub>2</sub>  $\mathcal{S} = (S, \Delta, s_I, \lambda)$ .
 

---

```

1  function  $rv_{\mathcal{S}}(s_0: S, z_0: \mathbb{Z}): S \rightarrow \mathbb{Z}_{\infty} := \{$ 
2      // initialise reachable values
3      var  $rv: S \rightarrow \mathbb{Z}_{\infty} := \{s \mapsto \text{if } (s == s_0) z_0 \text{ else } -\infty\}$ 

5      // Propagation phase 1
6      loop  $(|S|-1)$  {
7          foreach  $((s, u, \Gamma, s') \text{ in } \Delta \text{ if } rv(s)+u \models_{PA} \Gamma)$  {
8               $rv(s') := \text{if } (rv(s)+u > rv(s')) rv(s)+u \text{ else } rv(s')$ 
9          }
10     }

12     // Acceleration
13     foreach  $((s, u, \Gamma, s') \text{ in } \Delta \text{ if } rv(s)+u \models_{PA} \Gamma)$  {
14          $rv(s') := \text{if } (rv(s)+u > rv(s')) \infty \text{ else } rv(s')$ 
15     }

17     // Propagation phase 2
18     loop  $(|S|-1)$  {
19         foreach  $((s, u, \Gamma, s') \text{ in } \Delta \text{ if } rv(s)+u \models_{PA} \Gamma)$  {
20              $rv(s') := \text{if } (rv(s)+u > rv(s')) rv(s)+u \text{ else } rv(s')$ 
21         }
22     }

24      $rv$ 
25 }
    
```

---

non-empty, guard-free path from  $s$  to  $s$  can be repeated, independently of the value with that  $s$  is reached for the first time and how the counter value evolves. To this end, let  $\hat{\mathcal{S}}$  be an exact copy of  $\mathcal{S}$  but with an additional fresh copy  $\hat{s}$  of  $s$  that has no incoming transitions but the same outgoing transitions as  $s$ . That is,  $\hat{\mathcal{S}}$  contains for each transition  $(s, u, \Gamma, s') \in \Delta$  additionally the transition  $(\hat{s}, u, \Gamma, s')$ . In  $\hat{\mathcal{S}}$  each path from  $s$  to  $s$  still exists, except for the trivial, empty path.

If the guard-free reachability check fails, we can conclude that all (non-empty) paths from  $s$  to  $s$  use at least one guarded transition and we henceforth distinguish the cases of  $m$  being finitely bounded and being unbounded. If  $m \in \mathbb{Z}$  is finite, compute  $\hat{m} := rv_{\hat{\mathcal{S}}}(\hat{s}, m, s)$  in order to determine how  $s$  is reachable from itself. Again, if  $\hat{m} = -\infty$ , then  $s$  is not reachable from  $(s, m)$  and thus not reachable even twice from  $(s_I, 0)$ . On the other hand, if  $s$  is repeatedly reached from  $(s_I, 0)$  by some run, then this run reaches  $s$  for the first time with a value at most  $m$  and from there again with some value. Hence,  $\hat{m} > -\infty$  in that case. Then,  $\hat{m} < m$  means that even the “best” path from  $s$  to  $s$  has a negative net effect on the counter value. Thus, the value will decrease strictly with

each repetition and must eventually fall below the thresholds imposed by the guards. On the other hand, if  $\hat{m} \geq m$ , then there is some computation from configuration  $(q, m)$  to the configuration  $(q, \hat{m})$  that can be repeated infinitely because every guard will still be satisfied as long as the counter value did not strictly decrease.

The remaining case is that  $m = \infty$ , i.e., that  $s$  can be reached with an arbitrarily large counter value. Therefore, we can assume that it is large enough to satisfy every guard, for an arbitrarily long subsequent computation. Again, we determine the net effect of the best path from  $s$  to  $s$  in this situation by computing the reachable-value bound. To this end, we use  $\hat{S}_\top$ , being  $\hat{S}$  with all transition guards removed. Then,  $\hat{m}' := rvb_{\hat{S}_\top}(\hat{s}, 0, s)$  precisely determines whether there is a run repeating  $s$  infinitely, namely if and only if  $\hat{m}' \geq 0$ . Given a computation from  $s$  to  $s$  with non-negative net effect, it can be repeated over and over again. In case each such computation has a negative effect, the value will again fall below even the lowest threshold. Notice that  $m = \infty$  does not indicate that an infinite value can be reached but merely an arbitrarily large, finite value. Any such value will be depleted after sufficiently many cycles with negative effect on the counter. The procedure is summarised as pseudo code<sup>1</sup> in Algorithm 2. ■

---

**Algorithm 2** Decide repeated control-state reachability for a state  $\mathbf{s}$  in  $1\text{-CS}_2$   $\mathcal{S} = (S, \Delta, s_I, \lambda)$  from an initial configuration  $(s_0, z_0)$ .

---

```

1  function repeatedReach(s0: S, z0: ℤ, s: S): ℬ := {
2    val m := rvbS(s0, z0, s)
3    if (m == -∞) {
4      false
5    }
6    else if (reachS⊥( $\hat{s}$ , s)) { //  $\hat{s}$  is reachable from s in  $\hat{S}$  with all guarded transitions removed
7      true
8    }
9    else if (m < ∞) {
10     rvbS( $\hat{s}$ , m, s) >= m
11   }
12   else {
13     rvbS⊥( $\hat{s}$ , 0, s) >= 0
14   }
15 }
```

---

With Theorem 6.2 we obtain immediately a procedure to check whether a given  $1\text{-CS}_2$  is empty because this is the case if and only if at least one of its states is repeatedly reachable from the initial configuration.

---

<sup>1</sup>Notice that in Algorithm 2,  $\hat{S}$  depends implicitly on the variable  $\mathbf{s}$ .

► **Theorem 6.3.** *Emptiness of 1-CS<sub>≥</sub> is in P.*

It remains to decide (one-time) reachability in 1-CS<sub>≥</sub>. Notice that the definition we use here requires that there be a proper, infinite run passing through a state in order to consider it reachable. Therefore, the decision procedure presented in the following also relies on Theorem 6.2.

► **Theorem 6.4.** *Control-state reachability in 1-CS<sub>≥</sub> is in P.*

**Proof.** Let  $\mathcal{S} = (Q, \Delta, s_I, \lambda)$  be a 1-CS<sub>≥</sub> and  $s \in S$  be a target state. Observe that  $s$  is reachable by a run of  $\mathcal{S}$  if and only if there is a counter value  $m \in \mathbb{Z}$  such that the configuration  $(s, m)$  is reachable from the configuration  $(s_I, 0)$  and there is some state  $s' \in S$  that is repeatedly reachable from  $(s, m)$ . This is precisely the case if either

- i)*  $m := rvb(s_I, 0, s) \in \mathbb{Z}$  is finite and some state  $s' \in S$  is repeatedly reachable from the maximal configuration  $(s, m)$  or
- ii)*  $rvb(s_I, 0, s) = \infty$  is unbounded and some state  $s' \in S$  is repeatedly reachable from  $(s, m)$  for some  $m \in \mathbb{Z}$ .

Observe that the latter case implies that

- iii)*  $rvb(s_I, 0, s) = \infty$  and there is a simple path from  $s$  to some state  $s' \in S$  that is repeatedly reachable from the initial configuration  $(s_I, 0)$ .

In fact, the second case (*ii*) above is also *implied by* statement (*iii*): If  $s'$  is repeatedly reachable from  $(s_I, 0)$ , then it must, in particular, be repeatedly reachable from some configuration  $(s', m)$ . Since  $rvb(s_I, 0, s) = \infty$  is unbounded, there is a large enough counter value  $m'$  such that  $(s, m')$  can be reached from  $(s_I, 0)$  and all guards on the simple path from  $s$  to  $s'$  are satisfied when traversing it to reach a configuration  $(s', m'')$  where  $m'' \geq m$ . Then, the computation repeating  $s'$  infinitely can be executed also starting from the configuration  $(s, m'')$ , completing the run reaching  $s$  and continuing infinitely. The characterisation of reachability in terms of the cases (*i*) and (*iii*) gives rise to the polynomial-time procedure presented in Algorithm 3. ■

## 6.2 Deciding MC(cCTL,KS) in Polynomial Time

With the auxiliary results above, we can now show that model-checking of cCTL formulae over Kripke structures can be done in polynomial time.

---

**Algorithm 3** Decide control-state reachability for a state  $s$  in 1-CS<sub>≥</sub>  $\mathcal{S} = (S, \Delta, s_I, \lambda)$  from an initial configuration  $(s_0, z_0)$ .

---

```

1  function reach(s0: S, z0: ℤ, s: S): ℬ := {
2      val m := rwbS(s0, z0, s)

4      // Case (i)
5      if (m ∈ ℤ) {
6          foreach (s' in S if repeatedReach(s, m, s')) {
7              return true
8          }
9      }

11     // Case (iii)
12     if (m == ∞) {
13         foreach (s' in S if reachS⊤(s, s') // there is a path from s to s' in S
14                 and repeatedReach(s0, z0, s')) {
15             return true
16         }
17     }

19     false
20 }
    
```

---

► **Theorem 6.5.** *The problem MC(cCTL,KS) is in P.*

**Proof.** Let  $\mathcal{K} = (S, \Delta, s_I, \lambda)$  be a Kripke structure and  $\Phi$  a cCTL formula. Similar to the standard model-checking algorithm for CTL, we compute subsets  $S_\varphi \subseteq S$  of the states of  $\mathcal{K}$  for every subformula  $\varphi \in \text{sub}(\Phi)$  of  $\Phi$  such that for all  $s \in S$  we have  $s \in S_\varphi$  if and only if  $\mathcal{K}_s \models \varphi$ , where  $\mathcal{K}_s := (S, \Delta, s, \lambda)$  equals  $\mathcal{K}$  with initial state  $s$  instead of  $s_I$ . Checking whether the initial state  $s_I$  is contained in  $S_\Phi$  then solves the problem.

The set  $S_\Phi$  is computed recursively over the structure of  $\Phi$  as follows. Propositions ( $p \in AP$ ), negation ( $\neg\varphi$ ), conjunction ( $\varphi \wedge \psi$ ) and *temporal next* (**EX**  $\varphi$ , **AX**  $\varphi$ ) are handled as usual, that is

$$\begin{aligned}
 S_p &:= \{q \in S \mid p \in \lambda(q)\}, \\
 S_{\neg\varphi} &:= S \setminus S_\varphi, \\
 S_{\varphi \wedge \psi} &:= S_\varphi \cap S_\psi, \\
 S_{\mathbf{EX}\varphi} &:= \{q \in S \mid \text{succ}_{\mathcal{K}}(q) \cap S_\varphi \neq \emptyset\} \text{ and} \\
 S_{\mathbf{AX}\varphi} &:= \{q \in S \mid \text{succ}_{\mathcal{K}}(q) \subseteq S_\varphi\}.
 \end{aligned}$$

For the cases  $\Phi = \varphi \mathbf{EU}_{[\tau \geq b]} \psi$  and  $\Phi = \varphi \mathbf{AU}_{[\tau \geq b]} \psi$  assume  $\tau$  has the form  $a_0\chi_0 +$

$\dots + a_n \chi_n$  for  $n \in \mathbb{N}$ ,  $a_0, \dots, a_n \in \mathbb{Z}$ , and subformulae  $\chi_0, \dots, \chi_n$ . Further, let the sets  $S_\varphi, S_\psi, S_{\chi_0}, \dots, S_{\chi_n}$  be given inductively.

While the algorithm for *CTL* would construct the state sets for temporal operators, such as  $S_{\varphi \cup \psi}$ , by a global fixed point computation, we deal with each state from  $S$  individually here (at the cost of the linear factor  $|S|$  for the running time). To decide for any state  $s \in S$  whether it is supposed to be included in  $S_{\varphi \mathbf{EU}_{[\tau \geq b]} \psi}$  we construct a one-counter system  $\mathcal{K}'$  as follows from  $\mathcal{K}$ .

1. Remove all outgoing transitions from states violating  $\varphi$  and thus precisely the paths exhibiting a defect position with respect to  $\varphi$  before the last one.
2. Introduce the counter  $c$  and define its update on any outgoing transition of any state  $q \in S$  to be its effect

$$\mu_{\tau, q} : c \mapsto \sum_{i \in [0, n] \mid q \in S_{\chi_i}} a_i$$

on the value of the constraint term  $\tau = a_0 \chi_0 + \dots + a_n \chi_n$ .

3. Introduce a fresh sink state  $q_t \notin S$  reachable from any state  $q \in S_\psi$  if the value accumulated in  $c$  is at least  $b$ .

Formally, let us thus define  $\mathcal{K}' := (S \cup \{q_t\}, \Delta', s, \lambda)$  with

$$\begin{aligned} \Delta' := & \{(q, \mu_{\tau, q}, \emptyset, q') \mid (q, \varepsilon, \emptyset, q') \in \Delta, q \in S_\varphi\} \\ & \cup \{(q, 0, \{c \geq b\}, q_t) \mid q \in S_\psi\} \\ & \cup \{(q_t, 0, \emptyset, q_t)\}. \end{aligned}$$

Then, the state  $q_t$  is reachable in  $\mathcal{K}'$  from a configuration  $(s, 0)$  if and only if  $\mathcal{K}_s \models \varphi \mathbf{EU}_{[\tau \geq b]} \psi$ . It is thus decidable in polynomial time by Theorem 6.4.

A state  $s \in S$  satisfies the formula  $\Phi$  having the form  $\varphi \mathbf{AU}_{[\tau \geq b]} \psi$  if there is no run starting in state  $s$  and violating the formula  $\varphi \mathbf{U}_{[\tau \geq b]} \psi$ . This is the case on some path if at every position where  $\psi$  holds the accumulated effect on the value of  $\tau$  up to this position is less than  $b$  or a state violating  $\varphi$  occurred before. Thus, we construct a one-counter system  $\mathcal{K}'$  updating a fresh counter  $c$  to hold the value of  $\tau$  as above. In addition, all states satisfying  $\psi$  are now guarded by the constraint  $c < b$  and for every state violating  $\varphi$  we also include a transition leading to a fresh sink state  $q_t \notin S$  (featuring a loop).

Technically, for  $\Phi = \varphi \mathbf{AU}_{[\tau \geq b]} \psi$  let  $\mathcal{K}' := (S \cup \{q_t\}, \Delta', s, \lambda)$  with

$$\begin{aligned} \Delta' := & \{(q, \mathbf{0}, \emptyset, q_t) \mid q \notin S_\varphi\} \cup \{(q_t, \mathbf{0}, \emptyset, q_t)\} \\ & \cup \{(q, \mu_{\tau, q}, \emptyset, q') \mid (q, \perp, \emptyset, q') \in \Delta, q' \notin S_\psi\} \\ & \cup \{(q, \mu_{\tau, q}, \{c < b\}, q') \mid (q, \perp, \emptyset, q') \in \Delta, q' \in S_\psi\}. \end{aligned}$$

Then, all runs of  $\mathcal{K}'$  correspond to a run of  $\mathcal{K}$  violating  $\varphi \mathbf{U}_{[\tau \geq b]} \psi$ . On the other hand, if there is such a run in  $\mathcal{K}$  it can be followed in  $\mathcal{K}'$  entirely, because the obligation  $\psi$  is never reached while satisfying the constraint  $\tau \geq b$  or  $\varphi$  is violated at some point at which the state  $q_t$  can be entered in  $\mathcal{K}'$ . Consequently,  $\text{runs}(\mathcal{K}') \neq \emptyset$  if and only if  $\mathcal{K}_s \models \varphi \mathbf{AU}_{[\tau \geq b]} \psi$ . This is decidable in polynomial time by Theorem 6.3 since  $\mathcal{K}'$  uses only constraints of the form  $c < b$ . They can be equivalently replaced by  $c \geq -b + 1$  after negating all counter updates in  $\mathcal{K}'$ .  $\blacksquare$

► **Remark 6.6.** Laroussinie, Meyer and Petonnet [LMP12] describe how to evaluate a similar type of  $\mathbf{EU}$  formulae on so called durational Kripke structures (DKS). Conceptually, this problem corresponds to checking an  $\mathbf{EU}_{[.]}$  formula where all constraint coefficients are either 1 or  $-1$ . They use a method procedure where no guards need to be considered and that is based on a Floyd-Warshall algorithm. The approach could be applied to DKS with arbitrary weights and then used alternatively to check  $\mathbf{EU}_{[.]}$  formulae. However, their procedure for universal-type operators, such as  $\mathbf{AU}$ , relies on the fact that the weights are only 1,  $-1$ , or 0 and does therefore not apply here without exponential blow-up in complexity.

Finally, let us record that, for flat Kripke structures, the labelling algorithm above can be extended to handle also any cLTL formula using the decision procedure for  $\text{MC}(\text{cLTL}, \text{FCS})$  presented in Chapter 4. Invoking the NEXP procedure is necessary at most for each subformula and each state and can therefore be performed in exponential space. Thereby we obtain a model-checking procedure for cCTL\*.

► **Theorem 6.7.** *The problem  $\text{MC}(\text{cCTL}^*, \text{FKS})$  is in EXPSpace.*

Notice that this argument cannot be applied for flat counter systems because the decision procedure for cLTL checks whether a formula holds for a specific initial *configuration*. The latter is determined not only by a state but, in presence of counters, also by a counter valuation. Validity of a formula therefore not only depends on the state but also on the (effect of) the path used to reach it while the labelling algorithm assumes that a property can be evaluated equivalently with all counters set to zero.

The problems  $\text{MC}(\text{CTL}_\#, \text{KS})$  as well as  $\text{MC}(\text{CTL}_\#^*, \text{KS})$  are undecidable because  $\text{CTL}_\#$  admits Boolean combinations of counting constraints and therefore in particular equalities of the form  $\tau = b$ . Given this capability, the undecidability results by Laroussinie, Meyer and Petonnet [LMP12] apply to  $\text{CTL}_\#$ .

## ■ CHAPTER 7

# CTL<sub>#</sub><sup>\*</sup> and Presburger Arithmetic with Counting

The present chapter is dedicated to characterising the model-checking problem of CTL<sub>#</sub><sup>\*</sup> over flat counter systems. The logic subsumes all fragments studied in the previous chapters and represents undoubtedly the most flexible specification formalism as it admits linear- and branching-time modalities as well as the powerful bookmark-based counting mechanism. As observed earlier in Theorem 3.3, its satisfiability problem is undecidable over all the considered classes of counter system. Undecidability is also established for the general model-checking problems MC(CTL<sub>#</sub><sup>\*</sup>, CS) and MC(CTL<sub>#</sub><sup>\*</sup>, KS) as stated by Theorem 4.1. In the following, let us therefore focus on model-checking flat structures. For those, the problem will be shown to be decidable.

► **Theorem 7.1.** *The problem MC(CTL<sub>#</sub><sup>\*</sup>, FCS) is decidable.*

To show decidability, a polynomial encoding into the satisfiability problem of a decidable extension of Presburger arithmetic, called PH, is provided. To this end, the construction described in Chapter 5 will be reused. The extension PH of PA features a first-order quantifier  $\exists_y^= x$  for counting the solutions of a formula that is based on an equicardinality quantifier suggested by Härtig [Här62]. Subsequently, for the reverse direction an exponential reduction provides a corresponding hardness result for the model checking problems of LTL<sub>#</sub>, CTL<sub>#</sub>, and thus CTL<sub>#</sub><sup>\*</sup>.

An earlier, condensed version of this chapter restricted to flat Kripke structures was published in [Dec+17, Section 5].

## 7.1 Presburger Arithmetic with Counting Quantifier

Recall that Presburger arithmetic (PA) consists of linear constraints over integer variables and admits Boolean combinations and existential quantification. Let us consider its extension by the counting quantifier  $\exists_y^= x$  stating that the variable  $x$  holds precisely the

number of distinct values for  $y$  making the formula within its scope hold. For example, the formula  $\varphi(x) = \exists_y^x . 5 \leq y \wedge y \leq 10$  states that the (natural) interval from 5 to 10 contains  $x$  elements and is hence satisfied by precisely those valuations that assign the number 6 to  $x$ . That is, the quantifier binds the variable  $y$  and expresses a property of the value of  $x$ , the latter being free in that formula. Since  $x$  ranges over integers, the formulae  $\exists_y^x . y \leq 5$  as well as  $\exists_y^x . true$  are not satisfiable because the number of admissible values for  $y$  is not finite.

Formally, for a set of first-order variables  $X$ , the set  $\text{PH}(X)$  of *Presburger-Härtig arithmetic formulae*  $\varphi$  is defined by the grammar

$$\varphi ::= \psi \mid \exists_y^x . \varphi$$

for variables  $x, y \in X$  and Presburger arithmetic formulae  $\psi \in \text{PA}(X)$ . The semantics is defined as an extension  $\models_{\text{PH}}$  of the satisfaction relation  $\models_{\text{PA}}$  of  $\text{PA}$  by

$$\begin{aligned} \theta \models_{\text{PH}} \psi & \quad :\Leftrightarrow \quad \theta \models_{\text{PA}} \psi \\ \theta \models_{\text{PH}} \exists_y^x . \varphi & \quad :\Leftrightarrow \quad |\{b \in \mathbb{Z} \mid \theta[y \mapsto b] \models_{\text{PH}} \varphi\}| = \theta(x) \end{aligned}$$

for valuations  $\theta : X \rightarrow \mathbb{Z}$ , variables  $x, y \in X$ , and formulae  $\psi \in \text{PA}(X)$  and  $\varphi \in \text{PH}(X)$ . The satisfiability problem is defined in terms of the set  $\text{SAT}(\text{PH})$  of satisfiable formulae in analogy to—and in fact extending— $\text{SAT}(\text{PA})$  (cf. Section 2.1.2).

The counting quantifier originates in the binary equicardinality quantifier  $I$  for first-order logic suggested by Härtig [Här62] (see also the comprehensive survey by Herre et al. [Her+91]). For two first-order formulae  $\varphi$  and  $\psi$  with free variable  $x$  the quantifier expresses by  $I_x(\varphi, \psi)$  that the two sets of valuations of  $x$  satisfying  $\varphi$  and  $\psi$ , respectively, are of the same cardinality. Apelt [Ape66] studied the addition of this quantifier to Presburger arithmetic and showed that it can be eliminated. The satisfiability problem is thus reducible to that of  $\text{PA}$  and therefore decidable.

Notice that the notation  $\exists_y^x$  explicitly binding the number of solutions of a formula to a variable  $x$  defined above is expressible by the original quantifier as used by Apelt since  $\exists_y^x \varphi$  is equivalent to  $I_y(\varphi, 1 \leq y \wedge y \leq x)$ . In fact, also  $I_y(\varphi, \psi)$  can be expressed by

$$(\exists_x . (\exists_y^x . \varphi) \wedge (\exists_y^x . \psi)) \quad \vee \quad ((\neg \exists_{y_{\max}} . \forall y . y > y_{\max} \rightarrow \neg \varphi) \wedge (\neg \exists_{y_{\max}} . \forall y . y > y_{\max} \rightarrow \neg \psi))$$

stating that either both formulae  $\varphi$  and  $\psi$  have the same finite number of solutions or both have an infinite number of solutions. The result of Apelt was later shown again by Schweikardt [Sch05] explicitly for the unary counting quantifier  $\exists_y^x$ .

The elimination procedures by Apelt and Schweikardt both introduce a non-elementary blow-up of the formula and it is not known whether this can be avoided. In this regard, the reduction of the satisfiability problem of PH to the model-checking problem of  $CTL_{\#}^*$  provides an alternative view on the arithmetic theory.

## 7.2 A Reduction from SAT(PH) to MC(LTL<sub>#</sub>, FKS)

Satisfiability of an arithmetic formula  $\Phi \in \text{PH}(X)$  is encoded by constructing an LTL<sub>#</sub> formula. To this end, let us assume that  $\Phi$  is satisfiable if and only if it is satisfiable by a valuation  $\theta : X \rightarrow \mathbb{N}$  assigning only *natural* solutions. This is not an essential restriction because natural-sorted variables can represent integer-sorted variables with linear overhead.

Let  $z_1, z_2, \dots \notin X$  be bookmarks distinct from the variables used in  $\Phi$ . In the following, let us construct an LTL<sub>#</sub> formula  $\text{enc}(\Phi, \eta, i)$  depending on the PH formula  $\Phi$ , a mapping  $\eta : X \rightarrow \mathbb{N}_+$  assigning an index to each arithmetic variable in  $\Phi$ , and a current index  $i$ . It encodes the arithmetic constraints imposed by  $\Phi$  on variables  $x \in X$  into constraints on the distance between two specific positions on a run marked by two bookmarks  $z_j$  and  $z_{j-1}$  where  $j = \eta(x) > 0$ . The formula is defined recursively over the structure of  $\Phi$  by

$$\begin{aligned} \text{enc}(a, \eta, i) &= a \\ \text{enc}(a \cdot x, \eta, i) &= a \cdot \#_{z_{\eta(x)-1}}(\text{true}) - a \cdot \#_{z_{\eta(x)}}(\text{true}) \\ \text{enc}(\tau_1 + \tau_2, \eta, i) &= \text{enc}(\tau_1, \eta, i) + \text{enc}(\tau_2, \eta, i) \\ \text{enc}(\tau_1 \geq b, \eta, i) &= \text{enc}(\tau_1, \eta, i) \geq b \\ \text{enc}(\neg\varphi, \eta, i) &= \neg\text{enc}(\varphi, \eta, i) \\ \text{enc}(\varphi_1 \wedge \varphi_2, \eta, i) &= \text{enc}(\varphi_1, \eta, i) \wedge \text{enc}(\varphi_2, \eta, i) \\ \text{enc}(\exists_x.\varphi, \eta, i) &= \mathbf{F} z_i.\text{enc}(\varphi, \eta[x \mapsto i], i + 1) \\ \text{enc}(\exists_y^{\neq x}.\varphi, \eta, i) &= \mathbf{F} \left( \text{enc}(x, \eta, i) = \#_{z_{i-1}}(z_i.\text{enc}(\varphi, \eta[y \mapsto i], i + 1)) \right. \\ &\quad \left. \wedge \mathbf{XG} z_i.\neg\text{enc}(\varphi, \eta[y \mapsto i], i + 1) \right) \end{aligned}$$

for terms  $\tau_1, \tau_2$ , formulae  $\varphi_1, \varphi_2$ ,  $x \in X$ ,  $i \in \mathbb{N}$  and  $a, b \in \mathbb{Z}$ . For the flat Kripke structure  $\mathcal{K} = (\{q\}, \{(q, \varepsilon, \emptyset, q)\}, q, \lambda : q \mapsto \emptyset)$ , that consist of a single loop of length one,  $\Phi$  is satisfiable if and only if  $\mathcal{K} \models z_0.\text{enc}(\Phi, \mathbf{1}, 1)$ .

The last parameter  $i$  of the transformation is used to enumerate the variables of  $\Phi$  in order of appearance. It holds the largest index number that has not been assigned to a first-order variable yet, so  $z_i$  is assumed to be unused so far. The quantification of a

first-order variable  $x$  in a formula  $\exists_x.\varphi$  thus leads the transformation to introduce a fresh bookmark  $z_i$  and increment  $i$ .

The existence of a value for  $x$  satisfying  $\varphi$ , as expressed by  $\exists_x.\varphi$ , is formulated in  $LTL_{\#}$  as the existence of a position  $z_i$  such that the distance between  $z_{i-1}$  and  $z_i$  satisfies  $\varphi$ . This is precisely expressed by  $\mathbf{F} z_i.\hat{\varphi}$ , assuming a formula  $\hat{\varphi}$  properly encodes the constraint  $\varphi$  on  $x$  in terms of a constraint on  $z_i - z_{i+1}$ . Thus, the set of solutions for  $x$  is encoded in the set of positions that can be assigned to  $z_i$  in terms of their distance to  $z_{i-1}$ .

The temporal logic  $LTL_{\#}$  does not provide explicit access to the position  $\beta(z)$  assigned to a bookmark  $z \in B$  by a valuation  $\beta : B \rightarrow \mathbb{N}$  but counting the number of positions satisfying *true* can be used instead. For any position  $k \geq \beta(z_i) \geq \beta(z_{i-1})$  on the run  $\rho \in runs(\mathcal{K})$ , the positional distance between the bookmarks  $z_{i-1}$  and  $z_i$  is precisely

$$\begin{aligned} \beta(z_i) - \beta(z_{i-1}) &= (k - \beta(z_{i-1})) - (k - \beta(z_i)) \\ &= \#_{\beta(z_{i-1}),k}^{\mathcal{K},\rho}(true) - \#_{\beta(z_i),k}^{\mathcal{K},\rho}(true). \end{aligned}$$

Therefore, arithmetical first-order constraints over  $x$  can be translated literally to constraints on the  $LTL_{\#}$  term  $\#_{z_{i-1}}(true) - \#_{z_i}(true)$  since

$$\llbracket \#_z(true) \rrbracket(\mathcal{K}, \rho, k, \beta) = \#_{\beta(z),k}^{\mathcal{K},\rho}(true)$$

for any bookmark  $z$  and valuation  $\beta$ .

A PH formula  $\exists_y^x.\varphi$  determines that there is a finite number  $n$  of solutions  $y$  to the formula  $\varphi$  and that  $x$  holds that value. The idea of the translation is to formulate that there be a future position such that its distance represents the largest of these solutions. At that position, the translation imposes that no future position also encodes a solution and that the number of solutions up this point is precisely the value assigned to  $x$  (i.e., encoded into the distance between  $z_{\eta(x)-1}$  and  $z_{\eta(x)}$ ).

► **Example 7.2.** Consider the PH formula  $\exists_{x_1}.\exists_{x_2}.x_1 = 2x_2 \wedge \exists_{x_3}^{x_1}.x_3 \leq 3$ . The variables  $x_1$ ,  $x_2$ , and  $x_3$  are represented using bookmarks  $z_0, \dots, z_3$ . The first bookmark  $z_0$  serves as anchor placed at the beginning of the run and the modality  $\mathbf{F}$  expresses the existence of values for  $x_1$  and  $x_2$  in terms of the positions of  $z_1$  and  $z_2$  in the formula

$$z_0.\mathbf{F} z_1.\mathbf{F} z_2.\text{enc}(x_1 = 2x_2, \eta, 3) \wedge \text{enc}(\exists_{x_3}^{x_1}.x_3 \leq 3, \eta, 3)$$

where, thus,  $\eta(x_1) = 1$ ,  $\eta(x_2) = 2$ , and the next index to be used is 3. Figure 7.1 sketches a word model and the placement of bookmarks as expressed by the formula. At the first

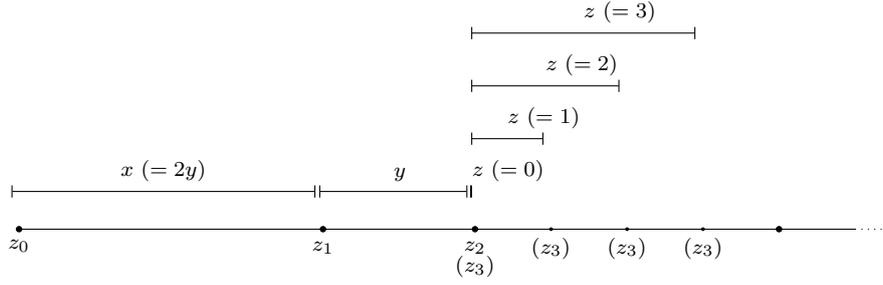


Figure 7.1: Sketch of the bookmark placement on a run as expressed by the encoding of the PH formula  $\exists_{x_1}. \exists_{x_2}. x_1 = 2x_2 \wedge \exists_{x_3}^{=x_1}. x_3 \leq 3$  discussed in Example 7.2.

position, the anchor bookmark  $z_0$  is placed and the distance to the position where  $z_1$  is placed represents the value of  $x_1$ . The two **F** operators impose that two distances need to be chosen to place  $z_1$  and  $z_2$ , thereby encoding the choice of the values for  $x_1$  and  $x_2$ , respectively.

Where  $z_2$  is placed, the formula

$$\text{enc}(x_1 = 2x_2, \eta, 3) = \#_{z_0}(\text{true}) - \#_{z_1}(\text{true}) = 2 \cdot \#_{z_1}(\text{true}) - 2 \cdot \#_{z_2}(\text{true})$$

is supposed to hold. It expresses the constraint that  $x_1 = 2x_2$  in terms of the respective distances. Finally,

$$\begin{aligned} \text{enc}(\exists_{x_3}^{=x_1}. x_3 \leq 3, \eta, 3) &= \mathbf{F} \#_{z_0}(\text{true}) - \#_{z_1}(\text{true}) = \#_{z_2} \left( z_3. \#_{z_2}(\text{true}) - \#_{z_3}(\text{true}) \leq 3 \right) \\ &\wedge \mathbf{XG} z_3. \neg \left( \#_{z_2}(\text{true}) - \#_{z_3}(\text{true}) \leq 3 \right). \end{aligned}$$

imposes that there are precisely as many possibilities to place bookmark  $z_3$  (i.e., to choose a value for the variable  $x_3$ ) as the value of  $x_1$ . The first part of the formula counts the number of positions after  $z_2$  where  $z_3$  can be placed such that the difference between  $z_2$  and  $z_3$  is at most 3. This number is bounded by the distance chosen to “jump” to by the **F** modality. The second part imposes that this distance is in fact maximal by stating that there is no future position where  $z_3$  can possibly be placed while still satisfying the constraint. Hence, the positions counted in the first part are exhaustive for the whole run.

The only valid solution in this example is to place  $z_1$  at distance 4 to  $z_0$  (encoding  $x_1 = 4$ ) and consequently placing  $z_2$  at distance 2 to  $z_1$  (encoding  $x_2 = 2$ ). That provides exactly 4 positions to place  $z_3$  at, corresponding to the values  $0, \dots, 3$  for  $x_3$ .

Observe that  $\mathcal{K}$  exhibits precisely one run and this run is identical to any of its suffixes.

Therefore, path quantification does not have an effect on the semantics of any formula and may deliberately be added. In particular, the LTL operators **F**, **X**, and **G** can be exchanged for their CTL variants **EF**, **EX**, and **EG**, yielding a  $CTL_{\#}$  formula instead.

The translation duplicates each formula that is prefixed by the counting quantifier. In the worst case, this can lead to an exponentially increased formula size, if measured as defined before, although the number of subformulae remains linear. Representing formulae as directed acyclic graph instead of linear strings, would avoid the exponential blow-up and only require linear time and space to perform the reduction. However, this seems less significant in the light that the best-known decision procedure for PH is non-elementary.

► **Theorem 7.3.** *There are exponential-time reductions from  $SAT(PH)$  to  $MC(LTL_{\#}, FKS)$  and  $MC(CTL_{\#}, FKS)$ .*

Notice that the encoding is linear for formulae that do not use the counting quantifier, providing in fact a reduction from  $SAT(PA)$ .

► **Corollary 7.4.** *There are linear-time reductions from  $SAT(PA)$  to  $MC(LTL_{\#}, FKS)$  and  $MC(CTL_{\#}, FKS)$ .*

Consequently, 2NEXP-hardness<sup>1</sup> established by Berman [Ber80] for  $SAT(PA)$  carries over to the model-checking problem.

► **Theorem 7.5.** *The problems  $MC(LTL_{\#}, FKS)$  and  $MC(CTL_{\#}, FKS)$  are 2NEXP-hard.*

### 7.3 A Reduction from $MC(CTL_{\#}^*, FCS)$ to $SAT(PH)$

This section develops a polynomial reduction of the  $CTL_{\#}^*$  model-checking problem over flat counter systems to the satisfiability problem of PH. Thereby the following result is established.

► **Theorem 7.6.** *There is a polynomial-time reduction from  $MC(CTL_{\#}^*, FCS)$  to  $SAT(PH)$ .*

Essentially, the reduction consists of formulating the  $CTL_{\#}^*$  semantics in PH. To this end, the necessary objects and functions need to be defined in terms of valuations of arithmetic variables. Recalling the definition in Section 2.2, the primary objects are the runs of a given system together with functions for accessing the state and valuation

---

<sup>1</sup>The precise complexity of  $SAT(PA)$  is characterised by Berman in terms of *alternating* Turing machines restricted by a double-exponential time bound. Thus, 2NEXP-hardness is an underestimation used here to ease comparison.

at any given position. Constructions for representing runs of flat counter systems were provided earlier by Demri et al. [Dem+10; DDS14; DDS18]. In fact, the present chapter lifts the inter-reducibility result of [DDS14; DDS18] from  $CTL^*$  and PA to  $CTL_{\#}^*$  and PH. For the sake of completeness, let us briefly describe the representation of runs and how their positions can be accessed based on the construction developed in Chapter 5.

### 7.3.1 Representing Runs

Recall that the set of runs of a flat counter system  $\mathcal{S}$  is equal to the set of flat approximations  $FA(\mathcal{S}, n) = runs(\mathcal{S})$  for a sufficiently large approximation depth  $n$ . Thus, let us use the encoding from Chapter 5 developed to represent sets of runs as augmented path schemas. Choosing  $n \geq 4|S|$  guarantees that the encoding represents all runs of  $\mathcal{S}$ . The components of the formula  $fmc(\mathcal{S}, n, \Phi)$  for representing an APS and thus used to represent runs in the following are  $aps(\mathcal{S}, n)$  and  $run(\mathcal{S}, n)$  defined in Sections 5.1.1 and 5.1.2. They expose a number of free variables and a valuation of them satisfying the two formulae represents a run  $\rho \in runs(\mathcal{S})$ . Therefore, they can serve as representation for a sort of variables  $\mathbf{rho} : runs(\mathcal{S}) \cup \{\perp\}$  holding a run of  $\mathcal{S}$  as values. Technically, not every valuation properly encodes a run and let us therefore consider all such unapt valuations to represent an additional value  $\perp$ .

► **Corollary 7.7.** *Variables of sort  $runs(\mathcal{S}) \cup \{\perp\}$  are effectively representable by a number of integer sorted variables linear in  $|S|$ .*

To account for the void value  $\perp$ , let  $Run(\mathbf{rho})$  be a predicate (of corresponding sort) that identifies the valuations representing a proper run. Importantly, it is definable by the formula

$$aps(\mathcal{S}, n) \wedge run(\mathcal{S}, n)$$

where it is assumed that the variable  $\mathbf{rho}$  is represented literally by the free variables in  $aps(\mathcal{S}, n)$  and  $run(\mathcal{S}, n)$ . If  $\mathbf{rho}$  is represented by other arithmetic variables, the free variables in the formulae can be substituted accordingly.

The definition of the predicate  $Run$  depends on  $\mathcal{S}$  and  $n$ . To avoid unnecessarily complicated notation, the flat counter system and the depth are fixed for the remainder of this section.

### 7.3.2 Accessing Positions

To express the semantics of a  $CTL_{\#}^*$  formula, we need not only to represent arbitrary runs but also to access the configuration at any arbitrary position. For this purpose, let

$\text{Conf}(\text{rho}, \text{pos}, \text{st}, \text{val})$  be a predicate over variables  $\text{rho} : \text{runs}(\mathcal{S}) \cup \{\perp\}$ ,  $\text{pos} : \mathbb{N}$ ,  $\text{st} : S$ , and  $\text{val} : \mathbb{Z}^{C_S}$ . Assuming that the variables are assigned a valid run  $\rho \in \text{runs}(\mathcal{S})$ , a position  $x \in \mathbb{N}$ , a state  $s \in S$ , and a valuation  $\theta : C_S \rightarrow \mathbb{Z}$ , the predicate is supposed to hold if and only if  $\rho(x) = (s, \theta)$ .

In the following, a PA formula is constructed to express such a predicate. Again, the variable  $\text{rho}$  is assumed to be represented by a set of variables corresponding to those that occur free in the formulae  $\text{aps}(\mathcal{S}, n)$  and  $\text{run}(\mathcal{S}, n)$ , as discussed above. Recall that this includes for each state  $i \in [0, n - 1]$  of the represented path schema especially the variables

- $\text{typ}_i : \{\boxplus, \triangleright, \boxminus, \triangleleft\}$  for the loop type,
- $\text{org}_i : S$  for the origin, and
- $\text{itr}_i : \mathbb{N}$  for the number of occurrences along the run.

Intuitively, to access the configuration at some position  $x$  on the run assigned to  $\text{rho}$ , the corresponding prefix is modelled. For every state  $i \in [0, n - 1]$ , a variable  $\text{end}_i : \mathbb{B}$  is supposed to indicate whether it occurs on the prefix or not. Further, variables  $\text{itrPfx}_i : \mathbb{N}$  hold the number of repetitions on the prefix. Let the predicate

$$\text{PfxEnd}(\text{end}_0, \dots, \text{end}_{n-1}) : \Leftrightarrow \neg \text{end}_0 \wedge \bigwedge_{i \in [1, n-1]} \text{end}_i \rightarrow \text{end}_{i+1}$$

expresses that there is precisely one last state on the prefix, identified as the largest position  $i$  such that  $\neg \text{end}_i$  holds. The iterations on the prefix must coincide with those on the run up to the point where the prefix ends. If the prefix ends at some state on a loop, the state and its predecessors are taken once more than its successors. After the loop, the iteration count is zero until the end of the schema. This is expressed by the

formula defining the predicate

$$\begin{aligned}
 & \text{Pfxltr}(\text{itr}_0, \dots, \text{itr}_{n-1}, \text{end}_0, \dots, \text{end}_{n-1}, \text{itrPfx}_0, \dots, \text{itrPfx}_{n-1}) :\Leftrightarrow \\
 & \text{itrPfx}_0 = 1 \\
 & \wedge \text{ite}(\text{end}_{n-1} \wedge \neg \text{end}_{n-2}, \text{itrPfx}_{n-1} = \text{itrPfx}_{n-2} - 1, \text{itrPfx}_{n-1} = \text{itrPfx}_{n-2}) \\
 & \wedge \bigwedge_{i \in [1, n-2]} \left( \begin{array}{l}
 (\boxplus_i \rightarrow \text{ite}(\text{end}_i, \text{itrPfx}_i = 0, \text{itrPfx}_i = \text{itr}_i)) \\
 \wedge (\boxtriangleright_i \rightarrow \text{ite}(\text{end}_i, \text{itrPfx}_i = 0, \text{itrPfx}_i > 0)) \\
 \wedge (\boxplus_i \rightarrow \text{ite}(\text{end}_i \wedge \neg \text{end}_{i-1}, \text{itrPfx}_i = \text{itrPfx}_{i-1} - 1, \\
 \text{itrPfx}_i = \text{itrPfx}_{i-1})) \\
 \wedge (\boxless_i \wedge \neg \text{end}_i \rightarrow \text{itrPfx}_i = \text{itrPfx}_{i-1} \wedge (\neg \text{end}_{i+1} \rightarrow \text{itrPfx}_i = \text{itr}_i)) \\
 \wedge (\boxless_i \wedge \text{end}_i \rightarrow \text{ite}(\text{end}_{i-1}, \text{itrPfx}_i = \text{itrPfx}_{i-1}, \text{itrPfx}_i = \text{itrPfx}_{i-1} - 1))
 \end{array} \right).
 \end{aligned}$$

In combination, the predicates allow for expressing that a variable  $\text{pfx}$  holds the prefix of the run assigned to  $\text{rho}$  at the position  $\text{st}$  by

$$\begin{aligned}
 \text{PrefixOf}(\text{pfx}, \text{rho}, \text{pos}) :\Leftrightarrow & \text{PfxEnd}(\text{end}_0, \dots, \text{end}_{n-1}) \wedge \left( \sum_{i \in [0, n-1]} \text{itrPfx}_i \right) = \text{pos} + 1 \\
 & \wedge \text{Pfxltr}(\text{itr}_0, \dots, \text{itr}_{n-1}, \text{end}_0, \dots, \text{end}_{n-1}, \text{itrPfx}_0, \dots, \text{itrPfx}_{n-1}).
 \end{aligned}$$

For improved readability, the formulation assumes a sort of prefixes of runs represented by variables  $\text{end}_0, \dots, \text{end}_{n-1}$  and  $\text{itrPfx}_0, \dots, \text{itrPfx}_{n-1}$ . The variable  $\text{pfx}$  is of that sort and assumed to be represented literally by these variables. The state at the end of the represented prefix is identified by

$$\text{FinalState}(\text{pfx}, \text{rho}, \text{st}) :\Leftrightarrow (\neg \text{end}_{n-1} \wedge \text{org}_{n-1} = \text{st}) \vee \bigvee_{i \in [0, n-2]} \neg \text{end}_i \wedge \text{end}_{i+1} \wedge \text{org}_i = \text{st},$$

under the same assumption on the names of variables representing  $\text{pfx}$ .

**Counter valuation.** The counter valuation at the end of the modelled prefix can be computed in the same way as the valuations along the whole run, as expressed by the formula  $\text{valuations}(\mathcal{S}, n)$  defined in Section 5.1.2. However, since the intermediate valuations on the prefix are not of interest, the propagation schema is simpler. Effectively, the updates can be summed up along the whole schema weighted by the iteration count on the prefix. Then, the result at position  $n - 1$  is precisely the valuation at the end of the prefix since the iteration counts after the prefix ended are all zero. Therefore, let the

predicate

$\text{FinalValuation}(\text{pfx}, \text{rho}, \text{val}) :\Leftrightarrow$

$$\begin{aligned} & \exists \text{val}_0, \dots, \text{val}_{n-1}, \text{valAux}_0, \dots, \text{valAux}_{n-1}. \text{val}_0 = \mathbf{0} \wedge \text{val}_{n-1} = \text{val} \\ & \wedge \bigwedge_{i \in [1, n-1]} \left( \begin{array}{l} (\triangleright_i \wedge \text{itrPfx}_i = 0 \rightarrow \text{val}_i = \text{val}_{i-1}) \\ \wedge (\triangleright_i \wedge \text{itrPfx}_i > 0 \rightarrow \text{accumulateVal}(\mathcal{S}, n, i)) \\ \wedge (\neg \triangleright_i \rightarrow \bigwedge_{(s, \mu, \Gamma, s') \in \Delta} \text{org}_{i-1} = s \wedge \text{org}_i = s' \rightarrow \text{val}_i = \text{val}_{i-1} + \text{itrPfx}_i \cdot \mu) \end{array} \right) \end{aligned}$$

associate the representations of a run and a prefix with the valuation at the end of the prefix. The valuations on prefix parts that are repeated multiple times are accumulated as expressed by the formula

$\text{accumulateVal}(\mathcal{S}, n, i) :=$

$$\bigwedge_{(s, \mu, \Gamma, s') \in \Delta} \left( \text{org}_{i-1} = s \wedge \text{org}_i = s' \rightarrow \text{valAux}_i = \text{val}_{i-1} + \mu \right) \wedge (\text{orgAtEnd}_i = s \wedge \text{org}_i = s' \rightarrow \text{val}_i = \text{valAux}_i + \text{itrPfx}_i \cdot \mu).$$

Based on the predicates defined above, the predicate to access the configuration of a run at some specific position can be defined as

$\text{Conf}(\text{rho}, \text{pos}, \text{st}, \text{val}) :\Leftrightarrow$

$$\exists \text{pfx}. \text{PrefixOf}(\text{pfx}, \text{rho}, \text{pos}) \wedge \text{FinalValuation}(\text{pfx}, \text{rho}, \text{val}) \wedge \text{FinalState}(\text{pfx}, \text{rho}, \text{st}).$$

► **Lemma 7.8.** *The predicates  $\text{Run}(\text{rho})$  and  $\text{Conf}(\text{rho}, \text{pos}, \text{st}, \text{val})$  over variables  $\text{rho} : \text{runs}(\mathcal{S}) \cup \{\perp\}$ ,  $\text{pos} : \mathbb{N}$ ,  $\text{st} : S$ , and  $\text{val} : \mathbb{Z}^{C_S}$  are effectively definable by PA formulae of polynomial size in  $|\mathcal{S}|$  such that for every valuation  $\theta$*

$$\theta \models_{PA} \text{Run}(\text{rho}) \quad \Leftrightarrow \quad \theta(\text{rho}) \in \text{runs}(\mathcal{S})$$

and if  $\theta(\text{rho}) \in \text{runs}(\mathcal{S})$ ,

$$\theta \models_{PA} \text{Conf}(\text{rho}, \text{pos}, \text{st}, \text{val}) \quad \Leftrightarrow \quad \theta(\text{rho})(\theta(\text{pos})) = (\theta(\text{st}), \theta(\text{val})).$$

### 7.3.3 Formulating $CTL_{\#}^*$ Semantics in PH

Finally, using the predicates  $\text{Conf}$  and  $\text{Run}$ , we construct for a formula  $\varphi \in CTL_{\#}^*(C_S)$  a PH formula that is satisfiable if and only if  $\mathcal{S} \models \varphi$ . Given the representation of runs of  $\mathcal{S}$

in terms of a fixed number of first-order variables, path quantifiers can be expressed with first-order quantification. Recall that variables for representing a run are summarised by one sorted variable for ease of presentation.

The semantics of temporal operators can be expressed almost literally by using  $\text{Conf}$  to access specific positions. Storing the current position in a bookmark  $x$  is done explicitly by assigning the current position to a corresponding first-order variable  $\text{pos}_x$ . For a counting constraint of the form  $a_1 \cdot \#_{x_1}(\chi_1) + \dots + a_m \cdot \#_{x_m}(\chi_m) \geq b$  variables  $z_1, \dots, z_m$  are introduced to hold the value of the counting terms  $\#_{x_1}(\chi_1), \dots, \#_{x_m}(\chi_m)$ . For example, the intended value for  $z_1$  can be specified by

$$\exists_{\text{pos}'}^{\text{z}_1} \cdot \text{pos}_{x_1} \leq \text{pos}' \leq \text{pos} \wedge \hat{\chi}_1$$

where  $\text{pos}$  holds the position the constraint is to be evaluated at and  $\hat{\chi}_1$  is the translation expressing that  $\chi_1$  holds at position  $\text{pos}'$  of the current run. The actual constraint such as, e.g.,  $\#_x(\chi_1) - \#_x(\chi_2) \geq -1$  can now directly be translated to  $z_1 - z_2 \geq -1$ .

The syntactic translation is now specified by the following recursive definition of the PH formula  $\text{chk}(\varphi, \text{rho}, \text{pos})$  depending on a  $CTL_{\#}^*$  formula and (the names of) first-order variables representing a run and a position on it. Let

$$\begin{aligned} \text{chk}(p, \text{rho}, \text{pos}) &:= \exists_{\text{st}} \cdot \exists_{\text{val}} \cdot \text{Conf}(\text{rho}, \text{pos}, \text{st}, \text{val}) \wedge \bigvee_{s | p \in \lambda(s)} \text{st} = s \\ \text{chk}(\tau_1 \geq b, \text{rho}, \text{pos}) &:= \exists_{\text{st}} \cdot \exists_{\text{val}} \cdot \text{Conf}(\text{rho}, \text{pos}, \text{st}, \text{val}) \wedge \tau[\text{val}] \geq b \\ \text{chk}(\mathbf{X} \psi, \text{rho}, \text{pos}) &:= \exists_{\text{pos}'} \cdot \text{pos}' = \text{pos} + 1 \wedge \text{chk}(\psi, \text{rho}, \text{pos}') \\ \text{chk}(\chi \mathbf{U} \psi, \text{rho}, \text{pos}) &:= \exists_{\text{pos}''} \cdot \text{pos} \leq \text{pos}'' \wedge \text{chk}(\psi, \text{rho}, \text{pos}'') \wedge \\ &\quad \forall_{\text{pos}'} \cdot (\text{pos} \leq \text{pos}' \wedge \text{pos}' < \text{pos}'') \rightarrow \text{chk}(\psi, \text{rho}, \text{pos}') \\ \text{chk}(\mathbf{E} \varphi, \text{rho}, \text{pos}) &:= \exists_{\text{rho}'} \cdot \text{Run}(\text{rho}') \wedge \text{chk}(\varphi, \text{rho}', \text{pos}) \\ &\quad \wedge \forall_{\text{pos}'} \cdot (\text{pos}' \leq \text{pos} \rightarrow \exists_{\text{st}} \cdot \exists_{\text{val}} \cdot \text{Conf}(\text{rho}, \text{pos}', \text{st}, \text{val}) \\ &\quad \quad \wedge \text{Conf}(\text{rho}', \text{pos}', \text{st}, \text{val})) \\ \text{chk}(x.\psi, \text{rho}, \text{pos}) &:= \exists_{\text{pos}_x} \cdot \text{pos}_x = \text{pos} \wedge \text{chk}(\psi, \text{rho}, \text{pos}) \\ \text{chk}(\tau_2 \geq b, \text{rho}, \text{pos}) &:= \exists_{z_1} \dots \exists_{z_m} \cdot \left( \bigwedge_{\ell=1}^m \exists_{\text{pos}'}^{\text{z}_\ell} \cdot \text{pos}_{x_\ell} \leq \text{pos}' \leq \text{pos} \wedge \text{chk}(\chi_\ell, \text{rho}, \text{pos}') \right) \\ &\quad \wedge a_1 \cdot z_1 + \dots + a_m \cdot z_m \geq b \end{aligned}$$

for any  $p \in AP$ ,  $\tau_1 \geq b \in \text{Grd}(C_S)$ , and  $\tau_2 = a_1 \cdot \#_{x_1}(\chi_1) + \dots + a_m \cdot \#_{x_m}(\chi_m)$  where  $m \in \mathbb{N}$ ,  $a_1, \dots, a_m, b \in \mathbb{Z}$ , and  $\chi_1, \dots, \chi_m \in CTL_{\#}^*(C_S)$ . Primed variables denote fresh

copies of the corresponding input variables, e.g.  $\text{pos}'$  becomes  $(\text{pos}')' = \text{pos}''$  and  $\text{pos}''$  becomes  $\text{pos}'''$ . Now,  $\varphi \models \mathcal{S}$  if and only if the formula

$$\exists_{\text{rho}}. \exists_{\text{pos}}. \text{Run}(\text{rho}) \wedge \text{pos} = 0 \wedge \text{chk}(\varphi, \text{rho}, \text{pos})$$

is satisfiable. This concludes the reduction showing that  $\text{MC}(CTL_{\#}^*, \text{FCS})$  is reducible in polynomial time to  $\text{SAT}(\text{PH})$ . By the decidability of the latter problem it completes the proof of Theorem 7.1.

## Conclusion

Stepping back from the detailed technical discussions carried out in the previous chapters, let us finally review the bigger picture. The present chapter briefly recalls the results and used techniques as well as questions left open to future investigations.

### 8.1 Summary

To study the concept of counting in the context of temporal logic specifications we have started by defining an extension of temporal logic. Linear inequations allow the user to formulate constraints on the number of positions along a program execution where given formulae hold. They are evaluated subject to temporal scopes specified using a bookmarking mechanism in the style of the freeze quantifier. Various fragments were studied, restricting the temporal navigation to that available in LTL or CTL. Concerning the counting capabilities, we have mainly considered the restrictions of the scoping mechanism to that of a single temporal operator.

The main focus of the investigations lay on the model-checking problem of the resulting fragments with respect to counter systems. We have argued that the problem is undecidable in this combination for different reasons, namely the computational power of counter systems themselves and the inherent hardness to analyse combinations of temporal and arithmetic constraints. By means of the different logic fragments we examined the latter aspect and realised that decidability is only recovered in cCTL and wLTL<sub>#</sub> under absence of counters in the system model. Apart from omitting counters entirely, the former dimension was studied by means of flatness which proved to have significant impact as it recovers decidability even for the most general combination of system and logic features.

We have shown correspondences between model-checking flat counter systems with respect to counting temporal logic on one hand and the satisfiability problem of variants of Presburger arithmetic on the other. Model-checking CTL<sub>#</sub><sup>\*</sup>, CTL<sub>#</sub>, and LTL<sub>#</sub> over flat counter systems was found to be closely related to Presburger arithmetic with the Härtig

counting quantifier. For the fragment  $\text{cLTL}$ , we developed a formulation in  $\text{qfPA}$  that provides also an approximation approach for the (undecidable) problem of verifying arbitrary counter systems. Also notice that reachability problems in  $\mathbb{Z}$ - $\text{VASS}$ —employed for solving the satisfiability problem of  $\text{wLTL}_\#$ —can be reduced to  $\text{qfPA}$  satisfiability.

The decision procedure for model-checking  $\text{cLTL}$  builds on and extends techniques based on path schemas. The augmented variant developed here serves not only as a finite representation of a set of runs but also of a labelling. Together with the notion of correctness and consistency, they can be considered as a proof that the labelling corresponds to the semantics and hence APS can be used as certificate for the existence of a satisfying run. Their bounded size provides a significantly improved complexity estimation in comparison to the more generic method based on PH. Further, APS serve as a backbone for the flat model-checking approach proposed as approximating verification technique for counter systems and  $\text{cLTL}$ . Their structure guides the flexible  $\text{qfPA}$  formulation and thereby the SMT-based implementation that was used to demonstrate the effectiveness of the procedure.

The results provide a systematic overview of a powerful family of temporal logics with counting and their verification problems on counter systems. The developed techniques draw from and combine different lines of research, most importantly those of counting temporal logics, model-checking flat systems, and Presburger arithmetic with counting.

## 8.2 Outlook

While we were able to provide a characterisation for each combination in Table 1.1, there is clearly much more to investigate. Many results constitute a first step but leave room for more precision. Further, many more aspects demand for investigation, as even subtle differences can impact the results. Some prominent directions are the following.

- As discussed earlier in Section 4.6, the precise complexity of  $\text{cLTL}$  model-checking remains open because the lower bound (NP) originates already from  $\text{LTL}$  over flat Kripke structures and leaves a wide gap to the upper bound (NEXP). The same applies to  $\text{cCTL}^*$ .
- Similarly, the lower bound for PH and thus for model-checking  $\text{LTL}_\#$ ,  $\text{CTL}_\#$ , and  $\text{CTL}_\#^*$  are those of standard Presburger arithmetic. Yet, decidability relies on a non-elementary elimination procedure that is not known to be optimal. Further, an exponential blow-up is observed in the encoding of  $\text{MC}(\text{CTL}_\#^*, \text{FCS})$  into PH, while encoding  $\text{SAT}(\text{PH})$  into the corresponding problems of  $\text{LTL}_\#$  and  $\text{CTL}_\#$  is linear. The

specific choice of the counting quantifier shifts this gap: a quantifier studied by Rescher [Res62] formulates an inequality relation between the number of solutions of two given formulae, as opposed to the equality of the Härtig quantifier. While this would eliminate the exponential blow up in the reduction from  $\text{CTL}_\#^*$  to first-order arithmetic, it would reoccur in the reverse direction. It would therefore be interesting to study the precise relation between the  $\text{LTL}_\#$ ,  $\text{CTL}_\#$  and  $\text{CTL}_\#^*$ .

- Verifying cCTL over Kripke structures in polynomial time is optimal, since the lower bound holds already for CTL. However, the result relies on the choice of inequality constraints. Including equality raises the complexity [LMP12]. However, the best known upper bound in this case is exponential. Further results on suitable classes of one-counter systems may allow improving this bound.
- We have only briefly touched upon satisfiability. There may be further interesting fragments that admit a decidable satisfiability problem. Further, other problems such as realisability and synthesis are of general interest. The formulation in first-order arithmetic could be the starting point for developing corresponding methods.
- The developments focused on the computational complexity in various fragments. However, their distinction is, a priori, merely syntactical. It remains to study the semantic relation between the logic fragments in terms of expressiveness and conciseness.
- A crucial step is turning from theory to application. The developed approximation scheme has shown potential for practical use, but it remains to optimise the SMT encoding. To obtain a competitive tool, much more engineering effort is necessary. Comparison of different formulations would be valuable and the use of dedicated features of the solvers, such as solving tactics or incremental solving could significantly increase performance. The construction used in the completeness proof (Section 4.5) provide more domain knowledge than what is used in the encoding so far. It may therefore be possible to develop a more specific procedure to construct witnesses following these constructions more closely.

While each of the points raises new questions and issues of technical or theoretical nature it may be worth pursuing them towards the common goal: to better understand the fundamental concept of counting and specifically its interplay with temporal behaviour and application in specification and verification.

## Bibliography

- [AH94] Rajeev Alur and Thomas A. Henzinger. ‘A Really Temporal Logic’. In: *J. ACM* 41.1 (1994), pp. 181–204 (cit. on p. 21).
- [Alu+08] Rajeev Alur, Marcelo Arenas, Pablo Barceló, Kousha Etessami, Neil Immerman and Leonid Libkin. ‘First-Order and Temporal Logics for Nested Words’. In: *Logical Methods in Computer Science* 4.4 (2008) (cit. on p. 3).
- [Ape66] H. Apelt. ‘Axiomatische Untersuchungen über einige mit der Presburgerschen Arithmetik verwandten Systeme’. In: *Zeitschr. f. math. Logik und Grundlagen d. Math.* 12 (1966), pp. 131–168 (cit. on p. 135).
- [Bar+05] Sébastien Bardin, Alain Finkel, Jérôme Leroux and Philippe Schnoebelen. ‘Flat Acceleration in Symbolic Model Checking’. In: *ATVA*. Vol. 3707. Lecture Notes in Computer Science. Springer, 2005, pp. 474–488 (cit. on pp. 9, 76, 101).
- [Bar+11] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds and Cesare Tinelli. ‘CVC4’. In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 171–177 (cit. on p. 122).
- [BBW06] Patrick Blackburn, Johan van Benthem and Frank Wolter, eds. *Handbook of Modal Logic, Volume 3*. Elsevier Science, 2006 (cit. on p. 2).
- [BDL12] Benedikt Bollig, Normann Decker and Martin Leucker. ‘Frequency Linear-time Temporal Logic’. In: *TASE*. IEEE Computer Society, 2012, pp. 85–92 (cit. on pp. 3, 11, 15, 33, 34).
- [BDW18] Dirk Beyer, Matthias Dangl and Philipp Wendler. ‘A Unifying View on SMT-Based Software Verification’. In: *J. Autom. Reasoning* 60.3 (2018), pp. 299–335 (cit. on p. 122).
- [BEH95] Ahmed Bouajjani, Rachid Echahed and Peter Habermehl. ‘On the Verification Problem of Nonregular Properties for Nonregular Processes’. In: *LICS*. IEEE Computer Society, 1995, pp. 123–133 (cit. on pp. 3, 4, 9, 36).

## Bibliography

- [Ber80] Leonard Berman. ‘The Complexity of Logical Theories’. In: *Theor. Comput. Sci.* 11 (1980), pp. 71–77 (cit. on p. 139).
- [BER94] Ahmed Bouajjani, Rachid Echahed and Riadh Robbana. ‘Verification of Nonregular Temporal Properties for Context-Free Processes’. In: *CONCUR*. Vol. 836. Lecture Notes in Computer Science. Springer, 1994, pp. 81–97 (cit. on pp. 3, 4, 9, 36).
- [Bey+07] Dirk Beyer, Thomas A. Henzinger, Rupak Majumdar and Andrey Rybalchenko. ‘Path invariants’. In: *PLDI*. ACM, 2007, pp. 300–309 (cit. on p. 101).
- [Bey17] Dirk Beyer. ‘Software Verification with Validation of Results - (Report on SV-COMP 2017)’. In: *TACAS (2)*. Vol. 10206. Lecture Notes in Computer Science. 2017, pp. 331–349 (cit. on p. 100).
- [Bie+99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke and Yunshan Zhu. ‘Symbolic Model Checking without BDDs’. In: *TACAS*. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 193–207 (cit. on p. 100).
- [Bie09] Armin Biere. ‘Bounded Model Checking’. In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 457–481 (cit. on pp. 8, 100).
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008 (cit. on pp. 6, 31, 124).
- [BKP86] Howard Barringer, Ruard Kuiper and Amir Pnueli. ‘A Really Abstract Concurrent Model and its Temporal Logic’. In: *POPL*. ACM Press, 1986, pp. 173–183 (cit. on p. 3).
- [Bol11] Benedikt Bollig. ‘An Automaton over Data Words That Captures EMSO Logic’. In: *CONCUR*. Vol. 6901. Lecture Notes in Computer Science. Springer, 2011, pp. 171–186 (cit. on p. 30).
- [Bou+17] Patricia Bouyer, François Laroussinie, Nicolas Markey, Joël Ouaknine and James Worrell. ‘Timed Temporal Logics’. In: *Models, Algorithms, Logics and Tools*. Vol. 10460. Lecture Notes in Computer Science. Springer, 2017, pp. 211–230 (cit. on p. 3).
- [BT76] Itshak Borosh and Leon B. Treybing. ‘Bounds on positive integral solutions of linear Diophantine equations’. In: *Proc. Amer. Math. Soc.* 55.2 (1976), pp. 299–304 (cit. on pp. 31, 101).

## Bibliography

- [Can+08] Nicolas Caniart, Emmanuel Fleury, Jérôme Leroux and Marc Zeitoun. ‘Accelerating Interpolation-Based Model-Checking’. In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 428–442 (cit. on p. 101).
- [CC00] Hubert Comon and Véronique Cortier. ‘Flatness Is Not a Weakness’. In: *CSL*. Vol. 1862. Lecture Notes in Computer Science. Springer, 2000, pp. 262–276 (cit. on pp. 5, 9).
- [CE81] Edmund M. Clarke and E. Allen Emerson. ‘Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic’. In: *Logic of Programs*. Vol. 131. Lecture Notes in Computer Science. Springer, 1981, pp. 52–71 (cit. on pp. 2, 5, 124).
- [Čer94] Kārlis Čerāns. ‘Deciding Properties of Integral Relational Automata’. In: *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*. Ed. by Serge Abiteboul and Eli Shamir. Vol. 820. Lecture Notes in Computer Science. Springer, 1994, pp. 35–46 (cit. on p. 5).
- [CES86] Edmund M. Clarke, E. Allen Emerson and A. Prasad Sistla. ‘Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications’. In: *ACM Trans. Program. Lang. Syst.* 8.2 (1986), pp. 244–263 (cit. on p. 124).
- [CGP01] Edmund M. Clarke, Orna Grumberg and Doron A. Peled. *Model checking*. MIT Press, 2001 (cit. on p. 6).
- [Cim+13] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma and Roberto Sebastiani. ‘The MathSAT5 SMT Solver’. In: *TACAS*. Vol. 7795. Lecture Notes in Computer Science. Springer, 2013, pp. 93–107 (cit. on p. 122).
- [CJ98] Hubert Comon and Yan Jurski. ‘Multiple Counters Automata, Safety Analysis and Presburger Arithmetic’. In: *CAV*. Vol. 1427. Lecture Notes in Computer Science. Springer, 1998, pp. 268–279 (cit. on p. 8).
- [Cla+03] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu and Helmut Veith. ‘Counterexample-guided abstraction refinement for symbolic model checking’. In: *J. ACM* 50.5 (2003), pp. 752–794 (cit. on pp. 7, 99).
- [Cla+18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith and Roderick Bloem, eds. *Handbook of Model Checking*. Springer, 2018 (cit. on pp. 6, 99).

## Bibliography

- [CSW15] David R. Cok, Aaron Stump and Tjark Weber. ‘The 2013 Evaluation of SMT-COMP and SMT-LIB’. In: *J. Autom. Reasoning* 55.1 (2015), pp. 61–90 (cit. on p. 101).
- [CU98] Michael Colón and Tomás E. Uribe. ‘Generating Finite-State Abstractions of Reactive Systems Using Decision Procedures’. In: *CAV*. Vol. 1427. Lecture Notes in Computer Science. Springer, 1998, pp. 293–304 (cit. on p. 99).
- [DD07] Stéphane Demri and Deepak D’Souza. ‘An automata-theoretic approach to constraint LTL’. In: *Inf. Comput.* 205.3 (2007), pp. 380–415 (cit. on p. 5).
- [DDS12] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. ‘Taming Past LTL and Flat Counter Systems’. In: *IJCAR*. Vol. 7364. Lecture Notes in Computer Science. Springer, 2012, pp. 179–193 (cit. on pp. 9, 21, 76).
- [DDS14] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. ‘Equivalence Between Model-Checking Flat Counter Systems and Presburger Arithmetic’. In: *RP*. Vol. 8762. Lecture Notes in Computer Science. Springer, 2014, pp. 85–97 (cit. on pp. 122, 140).
- [DDS15] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. ‘Taming past LTL and flat counter systems’. In: *Inf. Comput.* 242 (2015), pp. 306–339 (cit. on pp. 9, 11, 21, 44, 47, 76, 99, 102, 122).
- [DDS18] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. ‘Equivalence between model-checking flat counter systems and Presburger arithmetic’. In: *Theor. Comput. Sci.* 735 (2018), pp. 2–23 (cit. on pp. 11, 140).
- [Dec+17] Normann Decker, Peter Habermehl, Martin Leucker, Arnaud Sangnier and Daniel Thoma. ‘Model-Checking Counting Temporal Logics on Flat Structures’. In: *CONCUR*. Vol. 85. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 29:1–29:17 (cit. on pp. 15, 25, 43, 124, 134).
- [Dec11] Normann Decker. ‘Temporal Logic for Properties with Relative Frequency’. English. Diploma Thesis. University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, July 2011, p. 84 (cit. on p. 11).
- [Dem+10] Stéphane Demri, Alain Finkel, Valentin Goranko and Govert van Drimmelen. ‘Model-checking CTL\* over flat Presburger counter systems’. In: *J. Appl. Non Class. Logics* 20.4 (2010), pp. 313–344 (cit. on pp. 11, 140).

## Bibliography

- [Dem06] Stéphane Demri. ‘Linear-time temporal logics with Presburger constraints: an overview’. In: *J. Appl. Non Class. Logics* 16.3-4 (2006), pp. 311–348 (cit. on p. 5).
- [Dha14] Amit Kumar Dhar. ‘Algorithms for model-checking flat counter systems’. PhD thesis. Université Paris Diderot, 2014 (cit. on p. 9).
- [DKL10] Christian Dax, Felix Klaedtke and Martin Lange. ‘On regular temporal logics with past’. In: *Acta Inf.* 47.4 (2010), pp. 251–277 (cit. on p. 3).
- [DP19] Normann Decker and Anton Pirogov. ‘Flat Model Checking for Counting LTL Using Quantifier-Free Presburger Arithmetic’. In: *VMCAI*. Vol. 11388. Lecture Notes in Computer Science. Springer, 2019, pp. 513–534 (cit. on pp. 15, 99).
- [DV13] Giuseppe De Giacomo and Moshe Y. Vardi. ‘Linear Temporal Logic and Linear Dynamic Logic on Finite Traces’. In: *IJCAI*. IJCAI/AAAI, 2013, pp. 854–860 (cit. on p. 3).
- [EC80] E. Allen Emerson and Edmund M. Clarke. ‘Characterizing Correctness Properties of Parallel Programs Using Fixpoints’. In: *ICALP*. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 169–181 (cit. on p. 42).
- [EH83] E. Allen Emerson and Joseph Y. Halpern. ‘"Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time’. In: *POPL*. ACM Press, 1983, pp. 127–140 (cit. on p. 2).
- [Gab+80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah and Jonathan Stavi. ‘On the Temporal Basis of Fairness’. In: *POPL*. ACM Press, 1980, pp. 163–173 (cit. on p. 2).
- [GN00] Emden R. Gansner and Stephen C. North. ‘An open graph visualization system and its applications to software engineering’. In: *Softw. Pract. Exp.* 30.11 (2000), pp. 1203–1233 (cit. on p. 121).
- [GS97] Susanne Graf and Hassen Saïdi. ‘Construction of Abstract State Graphs with PVS’. In: *CAV*. Vol. 1254. Lecture Notes in Computer Science. Springer, 1997, pp. 72–83 (cit. on p. 99).
- [Haa+09] Christoph Haase, Stephan Kreutzer, Joël Ouaknine and James Worrell. ‘Reachability in Succinct and Parametric One-Counter Automata’. In: *CONCUR*. Vol. 5710. Lecture Notes in Computer Science. Springer, 2009, pp. 369–383 (cit. on pp. 21, 125).

## Bibliography

- [Här62] Klaus Härtig. ‘Über einen Quantifikator mit zwei Wirkungsbereichen’. In: *Colloquium on the foundations of mathematics, mathematical machines and their applications*. Ed. by L. Kalmár. Akadémiai Kiadó, Budapest, 1962, pp. 31–36 (cit. on pp. 134, 135).
- [Hen90] Thomas A. Henzinger. ‘Half-Order Modal Logic: How to Prove Real-Time Properties’. In: *PODC*. ACM, 1990, pp. 281–296 (cit. on p. 21).
- [Her+91] Heinrich Herre, Michal Krynicki, Alexander G. Pinus and Jouko A. Väänänen. ‘The Härtig Quantifier: A Survey’. In: *J. Symb. Log.* 56.4 (1991), pp. 1153–1183 (cit. on p. 135).
- [Hoj+12] Hossein Hojjat, Radu Iosif, Filip Konečný, Viktor Kuncak and Philipp Rümmer. ‘Accelerating Interpolants’. In: *ATVA*. Vol. 7561. Lecture Notes in Computer Science. Springer, 2012, pp. 187–202 (cit. on p. 101).
- [How+14] Falk Howar, Malte Isberner, Maik Merten, Bernhard Steffen, Dirk Beyer and Corina S. Pasareanu. ‘Rigorous examination of reactive systems - The RERS challenges 2012 and 2013’. In: *Int. J. Softw. Tools Technol. Transf.* 16.5 (2014), pp. 457–464 (cit. on pp. 12, 121).
- [HR81] Joseph Y. Halpern and John H. Reif. ‘The Propositional Dynamic Logic of Deterministic, Well-Structured Programs (Extended Abstract)’. In: *FOCS*. IEEE Computer Society, 1981, pp. 322–334 (cit. on p. 30).
- [HT99] Jesper G. Henriksen and P. S. Thiagarajan. ‘Dynamic Linear Time Temporal Logic’. In: *Ann. Pure Appl. Log.* 96.1-3 (1999), pp. 187–207 (cit. on p. 3).
- [Joh90] David S. Johnson. ‘A Catalog of Complexity Classes’. In: *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*. Elsevier and MIT Press, 1990, pp. 67–161 (cit. on p. 17).
- [Kam68] Johan Anthony Willem Hans Kamp. ‘Tense Logic and the Theory of Linear Order’. PhD thesis. Computer Science Department, University of California at Los Angeles, USA, 1968 (cit. on p. 2).
- [KF11] Lars Kuhtz and Bernd Finkbeiner. ‘Weak Kripke Structures and LTL’. In: *CONCUR*. Vol. 6901. Lecture Notes in Computer Science. Springer, 2011, pp. 419–433 (cit. on pp. 8, 11, 13, 76).
- [Kon+07] Roman Kontchakov, Agi Kurucz, Frank Wolter and Michael Zakharyashev. ‘Spatial Logic + Temporal Logic = ?’ In: *Handbook of Spatial Logics*. Springer, 2007, pp. 497–564 (cit. on p. 3).

## Bibliography

- [Kri59] Saul Kripke. ‘A Completeness Theorem in Modal Logic’. In: *J. Symb. Log.* 24.1 (1959), pp. 1–14 (cit. on p. 2).
- [Kri63] Saul Aaron Kripke. ‘Semantical Analysis of Modal Logic I Normal Modal Propositional Calculi’. In: *Zeitschr. f. math. Logik und Grundlagen d. Math.* 9.5–6 (1963), pp. 67–96 (cit. on p. 2).
- [KW10] Daniel Kroening and Georg Weissenbacher. ‘Verification and falsification of programs with loops using predicate abstraction’. In: *Formal Asp. Comput.* 22.2 (2010), pp. 105–128 (cit. on p. 101).
- [Lam77] Leslie Lamport. ‘Proving the Correctness of Multiprocess Programs’. In: *IEEE Trans. Software Eng.* 3.2 (1977), pp. 125–143 (cit. on p. 1).
- [Lam80] Leslie Lamport. ‘"Sometime" is Sometimes "Not Never" - On the Temporal Logic of Programs’. In: *POPL*. ACM Press, 1980, pp. 174–185 (cit. on p. 2).
- [LMP10] François Laroussinie, Antoine Meyer and Eudes Petonnet. ‘Counting LTL’. In: *TIME*. IEEE Computer Society, 2010, pp. 51–58 (cit. on pp. 3, 4, 9, 11).
- [LMP12] François Laroussinie, Antoine Meyer and Eudes Petonnet. ‘Counting CTL’. In: *Logical Methods in Computer Science* 9.1 (2012) (cit. on pp. 3, 4, 9, 11, 13, 27, 33, 34, 36, 124, 132, 133, 148).
- [LP85] Orna Lichtenstein and Amir Pnueli. ‘Checking That Finite State Concurrent Programs Satisfy Their Linear Specification’. In: *POPL*. ACM Press, 1985, pp. 97–107 (cit. on pp. 5, 39).
- [LS04] Jérôme Leroux and Grégoire Sutre. ‘On Flatness for 2-Dimensional Vector Addition Systems with States’. In: *CONCUR*. Vol. 3170. Lecture Notes in Computer Science. Springer, 2004, pp. 402–416 (cit. on pp. 44, 45, 102).
- [LS05] Jérôme Leroux and Grégoire Sutre. ‘Flat Counter Automata Almost Everywhere!’ In: *ATVA*. Vol. 3707. Lecture Notes in Computer Science. Springer, 2005, pp. 489–503 (cit. on p. 9).
- [LS06] Jérôme Leroux and Grégoire Sutre. ‘Flat counter automata almost everywhere!’ In: *Software Verification: Infinite-State Model Checking and Static Program Analysis*. Vol. 06081. Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006 (cit. on p. 9).
- [LS10] Martin Leucker and César Sánchez. ‘Regular Linear-Time Temporal Logic’. In: *TIME*. IEEE Computer Society, 2010, pp. 3–5 (cit. on p. 3).

## Bibliography

- [MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. ‘Z3: An Efficient SMT Solver’. In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340 (cit. on pp. 103, 122).
- [Min67] Marvin Lee Minsky. *Computation: finite and infinite machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1967. ISBN: 0-13-165563-9 (cit. on pp. 8, 12, 13, 19, 21, 33, 42, 102).
- [Pir17] Anton Pirogov. ‘SMT-based flat model-checking for LTL with Counting’. English. M.Sc. thesis. Universität zu Lübeck, Institute for Software Engineering and Programming Languages, Lübeck, Germany, 2017 (cit. on pp. 12, 121).
- [Plo04] Gordon D. Plotkin. ‘A structural approach to operational semantics’. In: *J. Log. Algebr. Program.* 60-61 (2004), pp. 17–139 (cit. on p. 1).
- [Pnu77] Amir Pnueli. ‘The Temporal Logic of Programs’. In: *FOCS*. IEEE Computer Society, 1977, pp. 46–57 (cit. on pp. 1, 42).
- [Pre29] Mojżesz Presburger. ‘Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt’. In: *Comptes Rendus du premier congrès de mathématiciens des Pays Slaves, Warszawa*. 1929, pp. 92–101 (cit. on pp. 5, 101).
- [Pri57] Arthur Norman Prior. *Time and Modality*. Clarendon Press: Oxford University Press, 1957 (cit. on p. 1).
- [QS82] Jean-Pierre Queille and Joseph Sifakis. ‘Specification and verification of concurrent systems in CESAR’. In: *Symposium on Programming*. Vol. 137. Lecture Notes in Computer Science. Springer, 1982, pp. 337–351 (cit. on p. 5).
- [Rei16] Julien Reichert. ‘On The Complexity of Counter Reachability Games’. In: *Fundam. Inform.* 143.3-4 (2016), pp. 415–436 (cit. on p. 35).
- [Res62] Nicholas Rescher. ‘Plurality-quantification’. In: *J. Symb. Log.* 27.3 (1962), pp. 373–374 (cit. on p. 148).
- [SC82] A. Prasad Sistla and Edmund M. Clarke. ‘The Complexity of Propositional Linear Temporal Logics’. In: *STOC*. ACM, 1982, pp. 159–168 (cit. on p. 30).
- [SC85] A. Prasad Sistla and Edmund M. Clarke. ‘The Complexity of Propositional Linear Temporal Logics’. In: *J. ACM* 32.3 (1985), pp. 733–749 (cit. on pp. 8, 30, 32).

## Bibliography

- [Sch05] Nicole Schweikardt. ‘Arithmetic, first-order logic, and counting quantifiers’. In: *ACM Trans. Comput. Log.* 6.3 (2005), pp. 634–671 (cit. on p. 135).
- [Var88] Moshe Y. Vardi. ‘A Temporal Fixpoint Calculus’. In: *POPL*. ACM Press, 1988, pp. 250–259 (cit. on p. 3).
- [Wol81] Pierre Wolper. ‘Temporal Logic Can Be More Expressive’. In: *FOCS*. IEEE Computer Society, 1981, pp. 340–348 (cit. on p. 3).