Towards a Mobile Temporal Logic of Actions

Júlia Zappe

Dissertation an der Fakultät für Mathematik, Informatik und Statistik der
Ludwig-Maximilians-Universität München

vorgelegt von Júlia Zappe

München, den 8.7.2005

Erstgutachter: Prof. Dr. Fred Kröger
Zweitgutachter: Dr. Stephan Merz
Externer Gutachter: Prof. Dr. Holger Schlingloff
Tag der mündlichen Prüfung: 22.9.2005

Acknowledgements

I would like to thank my supervisor Fred Kröger. He was willing to discuss at any time, and I could always rely on his full support. I am also thankful to him for his encouragement, especially in some of the rather dragging phases of my work. I am particularly grateful to Stephan Merz. Without his constant support and admirable patience throughout the whole period of writing I probably would not have been able to finish this thesis. I have not only benefited from his extraordinary professional competence, but have also taken advantage of his exceptional human qualities. I also would like to express my gratitude towards Martin Wirsing for providing me with a pleasant working environment by taking me into his group. He always has shown much interest in my work. The idea for the subject of this thesis was initiated by him and Stephan Merz.

I feel a need to thank all my friends and my family for not leaving me alone, not even in times when I tended to be almost unbearable... I am aware that I have demanded much of you by asking to share the burden with me. Thank you for not running away!

Contents

Zusammenfassung

Die vorliegende Arbeit stellt einen neuen Ansatz zur Spezifikation von mobilen Systemen vor. Als mobiles System wird hier ein System bezeichnet, das Code verwendet, der zur Laufzeit von einem Rechner auf einen anderen übertragen werden kann, und dessen Ausführung auf dem neuen Rechner fortgesetzt wird. Ein solches System wird gern als eine Hierarchie von "Orten" modelliert, deren Struktur verändert werden kann. Dies ist auch der Ausgangspunkt für unseren Modellbegriff. Es wird eine raum-zeitliche Logik namens MTLA eingeführt, deren temporaler Teil auf Lamports Temporal Logic of Actions (TLA) basiert. Zusätzlich werden räumliche Modalitäten definiert um die Struktur des Systems und ihre Veränderungen zu beschreiben. Geeignete Begriffe für die Verfeinerung solcher Systeme sowie ihre Repräsentierbarkeit in MTLA werden untersucht. Des weiteren wird den theoretischen Fragen der Axiomatisierbarkeit, der Erfüllbarkeit und des Model Checking Problems nachgegangen.

Abstract

In this thesis we present a novel approach to the specification of mobile systems. By mobile system we mean a system that makes use of code that can be transmitted from one computer to another one at runtime, so that the execution is continued on the new computer. Such systems are often modelled as a hierarchy of locations whose structure can be modified. This is also the starting point for our model notion. We introduce a spatio-temporal logic called MTLA whose temporal part is based on Lamport's Temporal Logic of Actions (TLA). In addition to the temporal operators we define spatial modalities to describe the structure of the system and its modifications. We study suitable notions for the refinement of such systems as well as their representability in MTLA. Furthermore, we investigate theoretical questions like axiomatisability, satisfiability and the model checking problem.

Chapter 1

Introduction

With the lightning progress of networking technology and the increasing use of networks the role of systems that make use of mobile code – the term "mobile code" signifying code that can be transmitted to remote sites, even during execution – becomes more and more important. As a particular kind of mobile systems, mobile agent systems have arisen starting from the nineties [5, 8, 53]. A mobile agent is a sort of mobile code with some specific properties. Simultaneously with the development of such mobile systems, formal methods have been investigated to support their design. In the course of these studies it has soon become clear that traditional models of distributed systems are not adequate to capture certain aspects of mobility.

The first formalism handling mobility that has gained wide attention and has achieved recognition is Milner's π-calculus [38]. The π-calculus extends the process algebra Calculus of Communicating Systems (CCS) [37] following an approach proposed by Engberg and Nielsen [17]. The main characteristic of the π-calculus is that names of communication channels can be transmitted as messages. This feature allows to express the modification of the communication structure of the system.

As Cardelli has pointed out in [9], the π-calculus provides a good model to describe mobility of distributed systems as long as only small, local area networks are concerned. However, in the context of wide area networks mobility itself is not

the only issue that has to be taken into account. In contrast to local area networks, mutually distrustful *administrative domains* separated by barriers play a prominent role in the Internet. In order to adequately model mobility in large-scale networks, the crossing of boundaries between such protected domains by mobile code should be explicitely expressible. Following this observation, several novel formalisms (e.g. [11, 19, 15, 52]), mainly process calculi, have been introduced, many of them based upon the π-calculus. A common feature of these formalisms is the assumption that mobile systems have a hierarchical structure, and mobility is modelled by allowing to modify this structure.

One of the best known of these calculi – and the one that has strongly influenced and motivated our work – is the Ambient Calculus [11] by Cardelli and Gordon. The Ambient Calculus is a process calculus where processes may reside at nodes of an edge-labelled tree. By executing some capabilities, the processes can modify the tree structure. The basic primitives of the Ambient Calculus are similar to those of the π-calculus.

For some of the above-mentioned calculi modal logics have been introduced [10, 7, 16] to express properties of mobile systems. The models of these logics are the process terms of the respective calculi. Beside temporal modalities they additionally use spatial modalities to describe modifications of the hierarchical structure of the system. Typically, the formulas of these logics closely reflect the syntactic structure of the process terms. In particular, they can separate terms that only differ in their structure but have the same behaviour (with respect to the operational semantics), in other words, process terms with the same behaviour can satisfy different sets of formulas (cf. [47]). As a consequence, these logics do not seem to be suitable as specification logics.

In the present thesis we suggest a different approach to specify mobile systems. We propose a spatio-temporal logic whose semantics is based on *runs* of mobile systems, instead of a specific process calculus. In our approach – in imitation of most of the mentioned calculi – such a run is (essentially) a sequence of finite trees representing the topological structure of the system. However, there is a local state at every node of the tree instead of a process. Altogether, the semantics of our logic is based on a kind of Kripke structure, where every world has a spatial

(tree) structure.

One of our main goals has been to define a logic that supports the specification of mobile systems by stepwise refinement. Since in the context of the specification of reactive systems this goal has been successfully realised by Lamport's Temporal Logic of Actions [31], we decided to base the temporal part of the logic we introduce upon TLA. In order to describe the system's spatial structure and its modification we extend TLA by spatial modalities that allow to refer to different nodes of the trees.

The thesis is organised as follows. In chapter 2 we introduce the kernel of the logic Mobile Temporal Logic of Actions (MTLA) and show that like TLA, it is invariant under finite (temporal) stuttering. This is important for the logic to support system specification by refinement. However, in the context of a spatio-temporal logic the usual notion of stuttering invariance does not capture aspects of refinement connected with the spatial structure of the system. Hence, we also define a notion of *spatial stuttering invariance* and prove (an important fragment of) the logic to have this property.

Chapter 3 discusses why the traditional notion of refinement does not suffice in the context of mobile systems. Different notions of refinement are suggested and motivated with the help of specification examples. These new notions are connected with *spatial refinement*. They are based on the idea that a high-level location should be allowed to be implemented by several concrete locations. We show that all the presented refinement paradigms are supported by MTLA in the sense that on the logical level refinement can be expressed simply by implication.

Chapter 4 investigates the question of axiomatisability of the propositional fragment of the logic. To keep the proofs simpler, we present a proof system called Σ_{MLTL} for the logic MLTL of which MTLA is a fragment and which is simply LTL extended by the spatial modalities of MTLA. We show this system to be sound and complete with respect to the semantics of MLTL. The completeness proof also provides a kind of finite model property which will be helpful to prove that the satisfiability problem is decidable.

Chapter 5 presents automata theoretical solutions of the model checking and the

satisfiability problems for MLTL. We show how to translate a formula φ of MLTL into a weak alternating automaton that accepts exactly the models of φ. The model checking and the decidability problems are then reduced to the non-emptiness problem of appropriate automata.

Chapter 6 raises certain questions that arise in connection with the dynamic creation of mobile agents. We suggest extensions of the core of the logic that may help to solve these problems – without elaborating the new operators in such detail like the ones belonging to the kernel of MTLA.

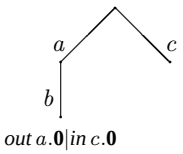
Chapter 2

Mobile TLA

This chapter introduces the logic Mobile Temporal Logic of Actions, MTLA for short. MTLA is intended for the specification of systems that make use of mobile code. It is based on Lamport's Temporal Logic of Actions TLA [31] and extends it by spatial operators. The first step is to fix our model notion for mobile systems. The main features of the logic are presented informally with the help of a simple specification example. In sec. 2.3, the formal definition of the semantics is given. In the remainder of the chapter we study appropriate notions of temporal and spatial stuttering and prove the logic (without one special operator) to be invariant under finite "spatio-temporal" stuttering.

2.1 The Models

When modelling a mobile system, most existing formalisms – like the Ambient Calculus [11], KLAIM [15] or the Join Calculus [19] for instance – assume that the system has a spatial structure (a hierarchy of locations), and mobility is thought of as the ability to modify this structure. As an example, consider the process term $P \equiv a[b[\mathit{out}\, a.\mathbf{0}|\mathit{in}\, c.\mathbf{0}]]|c[\mathbf{0}]$ of the Ambient Calculus. This process can be represented graphically as a tree whose edges are labelled by names:



In this process, ambient b can leave its parent a by "executing" its capability *out* a. Subsequently, this same ambient b can use its capability *in* c and enter ambient c. Such an evolution of the process can be illustrated by a sequence of edge-labelled trees as shown in figure 2.1.

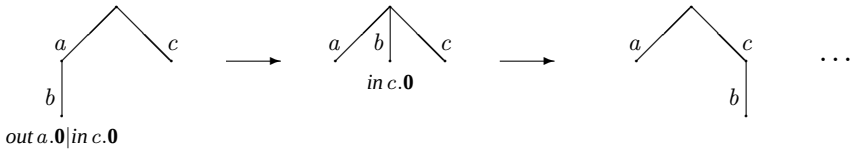


Figure 2.1: A run of $a[b[\mathit{out}\, a.\mathbf{0}|\mathit{in}\, c.\mathbf{0}]|c[\mathbf{0}]]$

This observation suggests to describe runs of mobile systems by sequences of configurations, where a configuration consists of a finite tree representing the topological structure of the system and of an assignment that associates with every node a local state. The nodes of the trees are labelled by unique names of a denumerable set N, the root labelled (implicitly) by the special name $\varepsilon \notin \mathsf{N}$.

Formally, a finite, non-empty tree t is given by a strict partial order $(\mathsf{N}_t, <_t)$ over a finite set $\mathsf{N}_t \subset \mathsf{N}$ of names. In particular, we identify the nodes of the tree with their labels. We define $\mathsf{N}_t^\varepsilon = \mathsf{N}_t \cup \{\varepsilon\}$ and extend the relation $<_t$ to N_t^ε by requiring $a <_t \varepsilon$ for all $a \in \mathsf{N}_t$. Intuitively, the relation $a <_t b$ holds for two nodes if node a is beneath node b. In other words, $<_t$ is the transitive closure of the successor relation of the tree. It has to satisfy the following conditions:

1. The relation $<_t$ is irreflexive, that is, for all $a \in \mathsf{N}_t^\varepsilon$ holds $a \not<_t a$.

2. The relation $<_t$ is transitive, i.e. for all $a, b, c \in \mathsf{N}_t^\varepsilon$ holds: if $a <_t b$ and $b <_t c$, then $a <_t c$.

3. For all $a, b, c \in \mathsf{N}_t^\varepsilon$ holds: if $a <_t b$ and $a <_t c$, then either $b <_t c$ or $c <_t b$.

Conditions 1. and 2. express that $<_t$ is a partial order. The third condition makes sure that the relation gives rise to a tree structure by requiring that two nodes which have a common "descendant" have to be on the same path.

The empty tree, i.e. the tree which does not have any node, is denoted by empty. Note that this tree is different from the tree $t = (\emptyset, <_t)$ since the latter has a node (exactly one), namely ε.

The subtree of a tree $t = (\mathsf{N}_t, <_t)$ rooted at node n is denoted by $t{\downarrow}n$. Formally, for any $n \in \mathsf{N}$ it is defined as

$$t{\downarrow}n = \begin{cases} (\{m \in \mathsf{N} | m <_t n\}, <'_t) & \text{if } n \in \mathsf{N}_t \\ \mathsf{empty} & \text{otherwise} \end{cases}$$

where $<'_t$ denotes the restriction of $<_t$ to $\{m \in \mathsf{N} | m <_t n\} \times \{m \in \mathsf{N} | m <_t n\}$, that is, it equals $<_t \cap (\{m \in \mathsf{N} | m <_t n\} \times \{m \in \mathsf{N} | m <_t n\})$.

We extend this definition to paths: for a sequence $\alpha \in \mathsf{N}^*$ of names, $t{\downarrow}\alpha$ is defined inductively by

$$\begin{aligned} t{\downarrow}\varepsilon &= t \\ t{\downarrow}\alpha.n &= (t{\downarrow}\alpha){\downarrow}n \quad . \end{aligned}$$

A *configuration* is defined with respect to a non-empty universe $|\mathfrak{I}|$ and a set $\mathcal{V}_f$ of (flexible) variables: it is a pair (t, λ) where $t = (\mathsf{N}_t, <_t)$ is a finite, non-empty tree and $\lambda : \mathsf{N}_t^\varepsilon \times \mathcal{V}_f \to |\mathfrak{I}|$ assigns a value to every variable in $\mathcal{V}_f$ at every location $n \in \mathsf{N}_t^\varepsilon$.

Infinite sequences of such configurations will serve as models for MTLA.

Figure 2.2 shows the graphical representation of (the prefix) of a run. An expression like $z =$ "go" at node b in the first configuration indicates that the value of variable z at node b equals "go", or more precisely, that $\lambda_0(b, z) =$ "go", where "go" is an element of the universe $|\mathfrak{I}|$.

Notation By ω we denote the set of the natural numbers including 0.

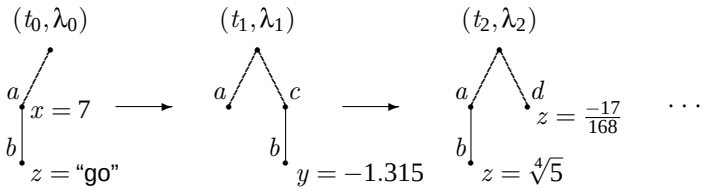


Figure 2.2: Prefix of a run

For sequences σ, τ, the concatenation of σ and τ is denoted by $\sigma \circ \tau$. We write finite sequences as $\langle a_0 \ldots a_n \rangle$, infinite sequences as $a_0 a_1 \ldots$. For an infinite sequence $\sigma = a_0 a_1 \ldots$ and a natural number $i \in \omega$, the suffix $a_i a_{i+1} \ldots$ of σ is denoted by $\sigma|_i$.

2.2 Example of a Mobile Agent

As our first simple specification example, we consider an agent that collects offers for flights in a network. In order to model the network, we assume a finite, fixed set Net of (immobile) network nodes with $home \in Net$ denoting the agent's home location. The (mobile) agent is represented by the name $ag \notin Net$. Its local state is described by the variables $ctl, item$ and $found$. Variable ctl specifies the agents control state, with "idle" and "busy" as possible values. While the agent is active, the variable $item$ contains what the agent is currently looking for: its value is a pair (d, t), d denoting a destination and t some time period. The variable $found$ stores the set of flights collected by the agent.

The MTLA-specification of such an agent is given in fig. 2.3. In order to avoid parenthesis, we follow Lamport's way [30] to write long conjunctions and disjunctions as a list whose items are labelled with $\wedge$ and $\vee$, respectively.

Before turning to the details of the definition of the logic, we give an informal explanation of this specification.

The agent's initial state is described by the formula

$$Init \equiv home.ag\langle \textbf{true} \rangle \wedge ag.ctl = \text{"idle"}$$

$$
\begin{aligned}
\mathit{Init} &\equiv \mathit{home}.\mathit{ag}\langle\textbf{true}\rangle \land \mathit{ag}.\mathit{ctl} = \text{“idle”} \\
\mathit{Network} &\equiv \textstyle\bigwedge_{n,m \in \mathit{Net}} \Box n\langle m[\textbf{false}]\rangle \land \Box\big[\textbf{false}\big]_{n.\mathit{id}} \\
\mathit{Prep}(d,t) &\equiv \land\ \mathit{ag}\langle\textbf{true}\rangle \land \circ \mathit{ag}\langle\textbf{true}\rangle \\
&\quad \land\ \mathit{ag}.\mathit{ctl} = \text{“idle”} \land \mathit{ag}.\mathit{ctl}' = \text{“busy”} \\
&\quad \land\ \mathit{ag}.\mathit{item}' = (d,t) \land \mathit{ag}.\mathit{found}' = \emptyset \\
&\quad \land\ \text{UNCHANGED}\ \mathit{res} \\
\mathit{GetFlight}_n &\equiv \land\ n.\mathit{ag}\langle\textbf{true}\rangle \land \circ n.\mathit{ag}\langle\textbf{true}\rangle \\
&\quad \land\ \mathit{ag}.\mathit{ctl} = \text{“busy”} \land \mathit{ag}.\mathit{item} \in n.\mathit{flights} \\
&\quad \land\ \mathit{ag}.\mathit{found}' = \mathit{ag}.\mathit{found} \cup \mathit{getFlight}(\mathit{ag}.\mathit{item}, n.\mathit{flights}) \\
&\quad \land\ \text{UNCHANGED}\ \mathit{ag}.\mathit{ctl}, \mathit{ag}.\mathit{item}, \mathit{home}.\mathit{res} \\
\mathit{Move}_{n,m} &\equiv \land\ n.\mathit{ag}\langle\textbf{true}\rangle \land \circ m.\mathit{ag}\langle\textbf{true}\rangle \\
&\quad \land\ \mathit{ag}.\mathit{ctl} = \text{“busy”} \land \mathsf{keep}_{\mathit{ag}} \\
&\quad \land\ \text{UNCHANGED}\ \mathit{ag}.\mathit{ctl}, \mathit{ag}.\mathit{item}, \mathit{ag}.\mathit{found}, \mathit{home}.\mathit{res} \\
\mathit{Deliver} &\equiv \land\ \mathit{ag}\langle\textbf{true}\rangle \land \circ \mathit{ag}\langle\textbf{true}\rangle \\
&\quad \land\ \mathit{ag}.\mathit{ctl} = \text{“busy”} \land \mathit{ag}.\mathit{ctl}' = \text{“idle”} \\
&\quad \land\ \mathit{res}' = \mathit{res} \cup \mathit{ag}.\mathit{found} \\
\mathit{HomeActs} &\equiv (\exists d,t : \mathit{Prep}(d,t)) \lor \mathit{Deliver} \\
\mathit{vars} &\equiv \langle \mathit{ag}.\mathit{ctl}, \mathit{ag}.\mathit{item}, \mathit{ag}.\mathit{found}, \mathit{home}.\mathit{res}\rangle \\
\mathit{FlightAgent} &\equiv \land\ \mathit{Init} \\
&\quad \land\ \mathit{Network} \\
&\quad \land\ \Box\big[\mathit{home}[\mathit{HomeActs}] \lor \textstyle\bigvee_{n \in \mathit{Net}} \mathit{GetFlight}_n\big]_{\mathit{vars}} \\
&\quad \land\ \textstyle\bigwedge_{n \in \mathit{Net}} \Box[\bigvee_{m \in \mathit{Net}} \mathit{Move}_{n,m}]_{-n.\mathit{ag}}
\end{aligned}
$$

Figure 2.3: MTLA-Specification of a Flight Agent

which claims that ag initially resides at its home location $home$ and is in its “idle” state. MTLA provides formulas of the form $n[F]$ for every name $n \in \mathsf{N}$; the informal interpretation of such a formula is that F holds at node n provided that such a node exists; $n\langle F\rangle$ abbreviates $\neg n[\neg F]$, that is, it means the same as $n[F]$ but it additionally requires the existence of a node with name n; $\mathit{home}.\mathit{ag}\langle F\rangle$ is an abbreviation of $\mathit{home}\langle \mathit{ag}\langle F\rangle\rangle$. The network is described by

$$\mathit{Network} \equiv \bigwedge_{n,m \in \mathit{Net}} \Box n\langle m[\textbf{false}]\rangle \land \Box\big[\textbf{false}\big]_{n.\mathit{id}} \quad .$$

As in TLA, a formula of the form $\Box[A]_v$, where v is some variable, asserts that

whenever the value of v changes during a transition, the formula A – that describes a transition – holds. Hence, formula $Network$ expresses that the network nodes are present forever, that they are not nested, and that their ids never change.

The third conjunct describes which transitions and in which way can change the system's variables. These are on the one hand the actions that can be performed only at the home location: either the agent is given a task in the form of a destination d and a time period t as expressed by formula $\exists d, t : Prep(d, t)$, or the agent delivers the offers it has found at its home location $home$ as described by $Deliver$. On the other hand, the agent can collect offers for flights at any network location n as expressed by $GetFlight_n$. Note that as in TLA, the value of a term t in the next state is written as the "primed" version t' of t. The value of a term t at a node n different from the root is denoted by $n.t$. Similarly, $n.t'$ denotes the value of the term t at node n in the next state. For terms $t_1, \ldots, t_k$ the formula UNCHANGED $t_1, \ldots, t_k$ says that the value of neither of these terms changes during the transition.

The last conjunct in the formula $FlightAgent$ specifies the agent's possible movements between the network nodes. In MTLA, a formula of the form $\Box[A]_{-S}$ means that A holds whenever formula S becomes false during a transition. In our case, the formula asserts that whenever a formula $n.ag$, which abbreviates $n.ag\langle\textbf{true}\rangle$, becomes false (that is, whenever ag leaves location n), this is due to one of the actions described by the formulas $Move_{n,m}$. The formula $Move_{n,m}$ claims that ag is at location n, it is active and it has already checked the flight offers at this location and that after the transition, it is at location m as expressed by $\circ m.ag\langle\textbf{true}\rangle$ and its local variables and tree structure do not change during the transition: this is expressed by the conjunct UNCHANGED ... and by the formula keep_{ag}.

An important feature of TLA as well as of MTLA is that the same formalism can be used to specify systems and to describe their properties. This enables expressing the assertion that a system specified by formula $Spec$ has the property given by formula $Prop$ by the implication $Spec \Rightarrow Prop$. For example, the property that the agent is always located at one of the network nodes, can be expressed by the

following formula:

$$FlightAgent \Rightarrow \Box \bigvee_{n \in Net} n.ag\langle \mathbf{true} \rangle \quad .$$

The proof of the validity of such a formula would be based on two kinds of steps. First it has to be proven that initially the system has the desired property, i.e. that the implication $Init \Rightarrow \bigvee_{n \in Net} n.ag\langle \mathbf{true} \rangle$ is valid. After that, one has to prove that formula $\bigvee_{n \in Net} n.ag\langle \mathbf{true} \rangle$ is invariant under all possible transitions of the system.

The formal definition of the syntax and semantics of MTLA-formulas is given in the next section.

2.3 Simple MTLA

In the following we present the logic Simple MTLA – this is the kernel of MTLA and will later be extended by different quantifiers. For the sake of brevity, we will refer to Simple MTLA as MTLA whenever it is clear from the context that only the quantifier-free part of the logic is meant.

MTLA extends the logic TLA* by spatial operators. TLA* [34] is a variant of Lamport's TLA [31]. It generalises TLA by allowing temporal operators to occur in the description of transitions while preserving TLA's – from the specification point of view crucial – feature of being invariant under finite stuttering.

Assume given a signature of first-order logic with equality and denumerable sets $\mathcal{V}_f$ and $\mathcal{V}_r$ of flexible and rigid variables with $\mathcal{V}_f \cap \mathcal{V}_r = \emptyset$ as well as $\mathsf{N} \cap \mathcal{V}_f = \emptyset$ and $\mathsf{N} \cap \mathcal{V}_r = \emptyset$. The letters $v, w, v_i, w_i, \ldots$ will denote flexible, x, y, x_i, y_i rigid variables. Further, we assume a first-order interpretation $\mathfrak{I}$ of the function- and predicate symbols in a non-empty universe $|\mathfrak{I}|$ containing a special "null" value $d_{\mathfrak{I}} \in |\mathfrak{I}|$. A *configuration* is a pair (t, λ) as described in section 2.1 on page 9, that is, it consists of a tree $t = (\mathsf{N}_t, <_t)$ and a mapping $\lambda : \mathsf{N}_t^{\varepsilon} \times \mathcal{V}_f \to |\mathfrak{I}|$ assigning to every node of the tree a local state. A *run* is an infinite sequence of configurations $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$

The (pure and impure) terms and formulas of MTLA are given by the following grammar:

$$\begin{array}{rcl} t & ::= & x \mid v \mid f(t_1,\ldots,t_k) \mid \iota x : F \\ u & ::= & t \mid f(u_1,\ldots,u_k) \mid \iota x : A \\ F & ::= & P(t_1,\ldots,t_k) \mid \neg F \mid F_0 \Rightarrow F_1 \mid \exists x : F \mid m[F] \mid \Box F \mid \Box[A]_t \mid \Box[A]_S \\ A & ::= & F \mid \neg A \mid A_0 \Rightarrow A_1 \mid \exists x : A \mid m[A] \mid \bigcirc F \mid \mathsf{keep}_m \end{array}$$

where t, t_i denote pure, u, u_i impure terms, F, F_i pure and A, A_i impure formulas, f a function- and P a predicate symbol. Furthermore, S denotes a pure "spatial" formula, that is, a pure formula built without temporal operators. Our term formation rules include the definite description operator $\iota x : F$. Its interpretation is "the unique x for which F holds" if there is exactly one such value and the null value $d_{\mathfrak{I}}$ otherwise (cf. [42]). The precise definition of the semantics of the formulas and terms is given as follows.

Definition 2.1 *Let $\sigma = (t_0,\lambda_0)(t_1,\lambda_1)\ldots$ be a run where $t_i = (\mathsf{N}_i, <_i)$ are finite trees and $\lambda_i : \mathsf{N}_i^{\varepsilon} \times \mathcal{V} \to |\mathfrak{I}|$ valuations, and let $n \in \mathsf{N}^{\varepsilon}$. The semantics of MTLA-formulas is defined as follows:*

- $\sigma^{(n,\xi)}(x) \;=\; \xi(x)$ *for* $x \in \mathcal{V}_r$
- $\sigma^{(n,\xi)}(v) \;=\; \left\{ \begin{array}{ll} \lambda_0(n,v) & \text{if } n \in \mathsf{N}_0^{\varepsilon} \\ d_{\mathfrak{I}} & \text{otherwise} \end{array} \right\}$ *for* $v \in \mathcal{V}_f$
- $\sigma^{(n,\xi)}(f(t_1,\ldots,t_k)) \;=\; \mathfrak{I}(f)(\sigma^{(n,\xi)}(t_1),\ldots,\sigma^{(n,\xi)}(t_k))$
- $\sigma^{(n,\xi)}(\iota x : A) \;=\; \left\{ \begin{array}{ll} d \in |\mathfrak{I}| & \text{if } \sigma,n,\xi[x := d] \models A \text{ and} \\ & \quad \sigma,n,\xi[x := e] \not\models A \text{ for all } e \in |\mathfrak{I}| \setminus \{d\} \\ d_{\mathfrak{I}} & \text{otherwise} \end{array} \right.$
- $\sigma,n,\xi \models P(t_1,\ldots,t_k)$ *iff* $(\sigma^{(n,\xi)}(t_1),\ldots,\sigma^{(n,\xi)}(t_k)) \in \mathfrak{I}(P)$
- $\sigma,n,\xi \models \neg A$ *iff* $\sigma,n,\xi \not\models A$

- $\sigma, n, \xi \models A \Rightarrow B$ *iff* $\sigma, n, \xi \not\models A$ *or* $\sigma, n, \xi \models B$
- $\sigma, n, \xi \models \exists x : A$ *iff* $\sigma, n, \xi[x := d] \models A$ *for some* $d \in |\mathfrak{I}|$
- $\sigma, n, \xi \models m[A]$ *iff* $m \not<_0 n$ *or* $\sigma, m, \xi \models A$
- $\sigma, n, \xi \models \Box F$ *iff for all* $i \in \omega$, $n \notin \mathsf{N}^{\varepsilon}_j$ *for some* $j \leq i$ *or* $\sigma|_i, n, \xi \models F$
- $\sigma, n, \xi \models \circ F$ *iff* $n \notin \mathsf{N}^{\varepsilon}_1$ *or* $\sigma|_1, n, \xi \models F$
- $\sigma, n, \xi \models \Box[A]_t$ *iff for all* $i \in \omega$, $n \notin \mathsf{N}^{\varepsilon}_j$ *for some* $j \leq i$ *or* $\sigma|_i^{(n,\xi)}(t) = \sigma|_{i+1}^{(n,\xi)}(t)$ *or* $\sigma|_i, n, \xi \models A$
- $\sigma, n, \xi \models \mathsf{keep}_m$ *iff* $t_0{\downarrow}n.m = t_1{\downarrow}n.m$
- $\sigma, n, \xi \models \Box[A]_S$ *iff for all* $i \in \omega$, $n \notin \mathsf{N}^{\varepsilon}_j$ *for some* $j \leq i$ *or* $(\sigma|_i, n, \xi \models S$ *iff* $\sigma|_{i+1}, n, \xi \models S)$ *or* $\sigma|_i, n, \xi \models A$

As it is apparent from the definition, the interpretation of a Simple MTLA-formula is relative to a location n. For $\sigma, n, \xi \models A$ we say that *formula* A *holds for the model* σ *at location* n *under the valuation* ξ. Validity is defined with respect to the root of the trees. We say that a formula F is *valid over a run* σ if and only if $\sigma|_i, \varepsilon, \xi \models F$ for all $i \in \omega$ and all valuations ξ. A formula F *follows from* a set $\mathcal{F}$ of formulas, written as $\mathcal{F} \models F$ iff F is valid over all behaviours over which all formulas in $\mathcal{F}$ are valid. A formula F is *valid*, written as $\models F$, iff F is valid over all behaviours.

A formula of the form $m[F]$ means, roughly speaking, that formula F holds at location m, provided that a location with name m exists in the first configuration of the run. In modal logical terms, $m[\_]$ is the box operator with respect to the relation that connects a node n of a tree t_i with another node of the same tree iff the latter has name m and is "below" node n in t_i, that is, if $m <_i n$ holds.

The interpretation of the always modality is more complicated than usual, because locations may disappear, and we want to consider later reappearances of a name as new names. Intuitively, formula $\Box F$ holds for σ at location n iff F holds for every suffix $\sigma|_i$ at location n as long as n exists. Figure 2.4 shows the "lifeline"

of a name n. For the run there, it holds for example $\Box(v \geq 0)$ at node n (and it would hold even if in the next tree n reappeared, and v had a value less than 0 there).

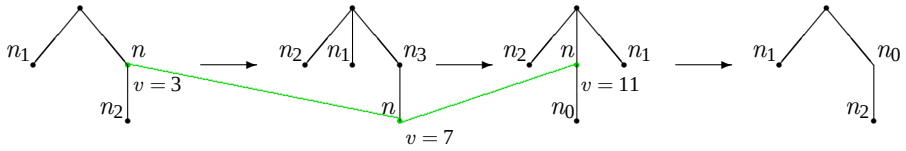


Figure 2.4: Lifeline of a location n

The "keep"-operator keep_m states that the structure of the subtree below location m at the current state does not change during the transition, i.e. this subtree and the subtree below m at the next instant are equal. This kind of transition formulas will be used to describe the movements of agents between different network locations.

For a term t and an (impure) formula A, the semantics of formula $\Box[A]_t$ is defined as for TLA, that is, it asserts that whenever the value of term t changes during a transition, this is due to an "action" described by the (impure) formula A. Therefore, formulas of this form are used to describe the allowed changes of local states. For a spatial formula S and an (impure) formula A, formula $\Box[A]_S$ asserts that whenever the truth value of formula S changes during a transition, formula A has to hold. Such formulas allow to describe structural modifications of trees.

We will use many derived operators. Beside the standard abbreviations like **true**, $\wedge$, $\vee$ and $\forall$ we introduce abbreviations specific to MTLA. We write $m\langle F\rangle$ for $\neg m[\neg F]$. Hence, the operators $m\langle\_\rangle$ can be regarded as the strong counterparts of the modalities $m[\_]$, since the formula $m\langle F\rangle$ asserts that there is a location with name m and that at this location formula F holds. For a name m we sometimes write simply m instead of $m\langle\mathbf{true}\rangle$. In order to save brackets, for names $m_1, \ldots, m_i \in \mathsf{N}$ we usually write $m_1.\cdots.m_i[F]$ for $m_1[\cdots m_i[F]\cdots]$ and $m_1.\cdots.m_i\langle F\rangle$ for $m_1\langle\cdots m_i\langle F\rangle\cdots\rangle$.

For $n \in \mathsf{N}$ and an (im)pure term u, let $n.u$ denote the (im)pure term $\iota x : n[x = u]$, that is, its interpretation is the value of u at node n. For a pure term t, let t' abbreviate the term $\iota x : \bigcirc(t = x)$, that is, its value equals the value of t in the next state. For pure terms $t_1, \ldots, t_k$ we write, in accordance with TLA's convention,

UNCHANGED $t_1, \ldots, t_k$ to abbreviate the (impure) formula $t_1' = t_1 \wedge \ldots \wedge t_k' = t_k$. For an impure formula A and pure terms or spatial formulas $a_1, \ldots, a_k$ we write $\Box[A]_{a_1,\ldots,a_k}$ for $\Box[A]_{a_1} \wedge \ldots \wedge \Box[A]_{a_k}$.

Further useful abbreviations are $\Box[A]_{-S}$ for $\Box[S \Rightarrow A]_S$ as well as $\Box[A]_{+S}$ for $\Box[\neg S \Rightarrow A]_S$. For instance, $\Box[A]_{-b}$ stands for $\Box[b\langle\mathbf{true}\rangle \Rightarrow A]_{b\langle\mathbf{true}\rangle}$, and it states that whenever a node b "disappears" during a transition, this is due to the "action" described by A.

We also adopt the usual abbreviations familiar from LTL like e.g. $\Diamond F$, which is defined as $\neg\Box\neg F$ and asserts the existence of a future state for which F holds. Further, we let $\odot F \equiv \neg\bigcirc\neg F$, that is, $\odot$ is the strong counterpart of the next-time operator. Since runs are infinite and all trees are assumed to be non-empty, the weak and the strong next-time operator coincide at the root, but evaluated at a node $m \in \mathsf{N}$ different from ε the strong operator also asserts the existence of a node with name m in the next state, whereas $\sigma, m, \xi \models \bigcirc F$ holds trivially whenever $m \notin \mathsf{N}_{t_1}$ holds.

2.4 Temporal stuttering

The particular suitability of TLA as a basis for system development by refinement is strongly connected with the fact that TLA-formulas are invariant under finite stuttering. Stuttering invariance means, roughly speaking, that (finite) repetition of the same state in a run has no influence on the set of formulas that hold for the run. The effect of this is that refinement can be expressed in TLA simply by implication: if $Spec$ is an abstract specification of the system and $Impl$ is a finer grained one, then the fact that $Impl$ is a correct implementation of $Spec$ corresponds to the validity of the implication $Impl \Rightarrow Spec$.

We show that (pure) formulas of (Simple) MTLA are also invariant under finite stuttering. For the sake of simplicity, we only consider the propositional fragment of the logic. However, note that all results presented in this chapter can be proved in an analogous way for first order MTLA.

In order to define propositional MTLA, pMTLA for short, we assume a denumer-

able set $\mathcal{V}$ of propositional variables with $\mathcal{V} \cap \mathsf{N} = \emptyset$. The sets of pure and impure formulas of pMTLA are given by

$$\begin{array}{lll} F & ::= & v \in \mathcal{V} \mid \neg F \mid F \Rightarrow G \mid m[F] \mid \Box F \mid \Box[A]_S \quad \text{(pure formulas)} \\ A & ::= & F \mid \neg A \mid A \Rightarrow B \mid m[A] \mid \mathsf{keep}_m \mid \circ F \qquad \text{(impure formulas)} \end{array}$$

where $m \in \mathsf{N}$ is a name and S a pure formula built without any temporal operators. In the context of pMTLA, a run is an infinite sequence $\sigma = (t_0, \lambda)(t_1, \lambda_1)\ldots$ of finite trees $t_i = (\mathsf{N}_i, <_i)$ endowed with valuations $\lambda_i : \mathsf{N}_i^\varepsilon \to 2^{\mathcal{V}}$ that assign to every node a set of propositional variables. The semantics of pMTLA-formulas is defined with respect to such runs and to a node $n \in \mathsf{N}^\varepsilon$.

Definition 2.2 *Let $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ be a run as described above with finite trees $t_i = (\mathsf{N}_i, <_i)$ and valuations $\lambda_i : \mathsf{N}_i^\varepsilon \to 2^{\mathcal{V}}$, and let $n \in \mathsf{N}^\varepsilon$. The semantics of pMTLA-formulas is defined inductively as follows:*

- $\sigma, n \models v$ *iff* $n \in \mathsf{N}_0^\varepsilon$ *and* $v \in \lambda_0(n)$
- $\sigma, n \models \neg A$ *iff* $\sigma, n \not\models A$
- $\sigma, n \models A \Rightarrow B$ *iff* $\sigma, n \not\models A$ *or* $\sigma, n \models B$
- $\sigma, n \models m[A]$ *iff* $m \not<_0 n$ *or* $\sigma, m \models A$
- $\sigma, n \models \Box F$ *iff for all* $i \in \omega$ *either* $n \notin \mathsf{N}_j^\varepsilon$ *for some* $j \leq i$ *or* $\sigma|_i, n \models F$
- $\sigma, n \models \circ F$ *iff* $n \notin \mathsf{N}_1^\varepsilon$ *or* $\sigma|_1, n \models F$
- $\sigma, n \models \mathsf{keep}_m$ *iff* $t_0{\downarrow}n.m = t_1{\downarrow}n.m$
- $\sigma, n \models \Box[A]_S$ *iff for all* $i \in \omega$ *either* $n \notin \mathsf{N}_j^\varepsilon$ *for some* $j \leq i$ *or* $(\sigma|_i, n \models S$ *iff* $\sigma|_{i+1}, n \models S)$ *or* $\sigma|_i, n \models A$

Now we turn to the definition of the notions connected with stuttering invariance. In the next chapter, where we will discuss refinement principles for mobile systems, we will recall the connection between stuttering invariance and system refinement.

First we define *temporal* stuttering equivalence. It is essentially the notion known from TLA [31], the difference is that the locations of the variables play a role as well. Later we also will introduce *spatial* stuttering equivalence.

Definition 2.3 (Stuttering equivalence) *Let* $V \subseteq \{n.v | n \in \mathsf{N}^\varepsilon, v \in \mathcal{V}\}$.

1. *Two configurations* $(s,\lambda),(t,\mu)$ *with* $s = (\mathsf{N}_s, <_s)$ *and* $t = (\mathsf{N}_t, <_t)$ *are called* V-similar, *written* $(s,\lambda) \sim_V (t,\mu)$, *iff the following hold:*

   (a) $s = t$

   (b) for all $n.v \in V$ *with* $n \in \mathsf{N}_s^\varepsilon$ *holds:* $v \in \lambda(n)$ *iff* $v \in \mu(n)$.

   Two runs $\sigma = (s_0,\lambda_0)(s_1,\lambda_1)\ldots$ *and* $\tau = (t_0,\mu_0)(t_1,\mu_1)\ldots$ *are called* V*-similar iff* $(s_i,\lambda_i) \sim_V (t_i,\mu_i)$ *holds for all* $i \in \omega$.

2. V-stuttering equivalence, *written as* $\simeq_V$, *is the smallest equivalence relation on (finite or infinite) sequences of configurations that identifies the sequences* $\rho \circ \langle (s,\lambda) \rangle \circ \sigma$ *and* $\rho \circ \langle (t,\mu)(u,\nu) \rangle \circ \sigma$, *for any finite sequence of configurations* ρ, *finite or infinite sequence of configurations* σ, *and pairwise* V*-similar configurations* $(s,\lambda),(t,\mu),(u,\nu)$.

3. *Stuttering equivalence, written* $\simeq$, *is the smallest equivalence relation on runs that identifies* $\rho \circ \langle (s,\lambda) \rangle \circ \sigma$ *and* $\rho \circ \langle (s,\lambda)(s,\lambda) \rangle \circ \sigma$ *for any finite sequence of configurations* ρ, *infinite sequence of configurations* σ *and configuration* (s,λ).

Figure 2.5 gives an example of (fragments of) two $\{ag.item, ag.ctl\}$-equivalent runs. To see that they are indeed $\{ag.item, ag.ctl\}$-equivalent, observe that

$$(s_0,\lambda_0) = (t_0,\mu_0), \qquad (s_0,\lambda_0) \sim_{\{ag.item,ag.ctl\}} (s_1,\lambda_1)$$
$$(s_2,\lambda_2) = (t_1,\mu_1), \qquad (t_1,\mu_1) \sim_{\{ag.item,ag.ctl\}} (t_2,\mu_2) \quad .$$

An immediate consequence of the above definitions is that $\simeq_V \subseteq \simeq_W$ holds whenever $W \subseteq V$ holds for sets $V, W \subseteq \{n.v | n \in \mathsf{N}^\varepsilon, v \in \mathcal{V}\}$.

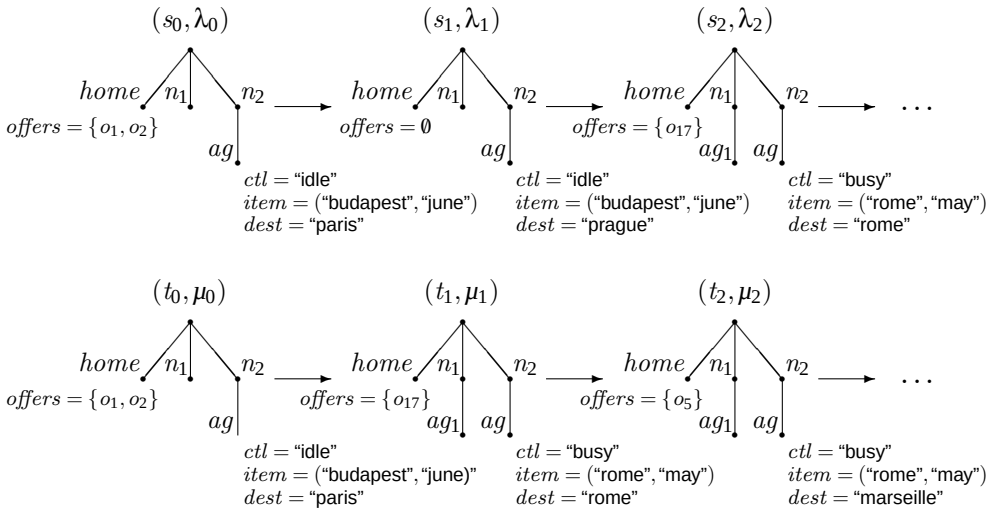


Figure 2.5: Example of $\{ag.item, ag.ctl\}$-equivalent runs

In the following we will refer to the elements of the set $\{n.v | n \in \mathsf{N}^{\varepsilon}, v \in \mathcal{V}\}$ as *localised variables*.

For every (impure) formula A, we define a finite set $\mathrm{FLV}(A)$ of localised variables as follows:

$$\begin{aligned}
\mathrm{FLV}(v) &= \{\varepsilon.v\} \\
\mathrm{FLV}(\neg A) &= \mathrm{FLV}(A) \\
\mathrm{FLV}(A \Rightarrow B) &= \mathrm{FLV}(A) \cup \mathrm{FLV}(B) \\
\mathrm{FLV}(m[A]) &= \{m.v | \varepsilon.v \in \mathrm{FLV}(A)\} \cup \{n.v | n.v \in \mathrm{FLV}(A), n \neq \varepsilon\} \\
\mathrm{FLV}(\Box F) &= \mathrm{FLV}(F) \\
\mathrm{FLV}(\bigcirc F) &= \mathrm{FLV}(F) \\
\mathrm{FLV}(\mathsf{keep}_m) &= \emptyset \\
\mathrm{FLV}(\Box[A]_S) &= \mathrm{FLV}(A) \cup \mathrm{FLV}(S)
\end{aligned}$$

Intuitively, $\mathrm{FLV}(A)$ contains all variables occurring in A "prefixed" by the location the variable belongs to. For example, for $F \equiv m[v \wedge n[\neg v \vee w]] \Rightarrow \bigcirc(v \wedge w)$ we have $\mathrm{FLV}(F) = \{m.v, n.v, n.w, \varepsilon.v, \varepsilon.w\}$.

The following lemma will be useful for the proof of the (temporal) stuttering invariance of propositional MTLA. All the assertions are easy to prove.

Lemma 2.4 *Let* $V \subseteq \{n.v | n \in \mathsf{N}^{\varepsilon}, v \in \mathcal{V}\}$ *and* $\sigma = (s_0, \lambda_0)(s_1, \lambda_1)\ldots$ *as well as* $\tau = (t_0, \mu_0)(t_1, \mu_1)\ldots$ *runs with* $\sigma \simeq_V \tau$*.*

1. $(s_0, \lambda_0) \sim_V (t_0, \mu_0)$
2. *For every* $i \in \omega$ *there exists some* $j \in \omega$ *such that* $\sigma|_i \simeq_V \tau|_j$ *as well as* $\langle (s_0, \lambda_0)\ldots(s_i, \lambda_i)\rangle \simeq_V \langle (t_0, \mu_0)\ldots(t_j, \mu_j)\rangle$*.*
3. *For every* $i \in \omega$ *with* $(s_i, \lambda_i) \not\sim_V (s_{i+1}, \lambda_{i+1})$ *there is some* $j \in \omega$ *such that the following hold:*
   (a) $\sigma|_i \simeq_V \tau|_j$
   (b) $\sigma|_{i+1} \simeq_V \tau|_{j+1}$
   (c) $\langle (s_0, \lambda_0)\ldots(s_i, \lambda_i)\rangle \simeq_V \langle (t_0, \mu_0)\ldots(t_j, \mu_j)\rangle$*.*

Now we are able to prove the main theorem about the stuttering invariance of MTLA-formulas. For the sake of uniformity, we identify $\varepsilon[A]$ with A.

Theorem 2.5 (Stuttering invariance) *Let* F *be a pure,* A *an impure formula,* $n \in \mathsf{N}^{\varepsilon}$*,* $\sigma = (s_0, \lambda_0)(s_1, \lambda_1)\ldots$ *and* $\tau = (t_0, \mu_0)(t_1, \mu_1)\ldots$ *runs.*

1. *If* $\sigma \simeq_{\mathrm{FLV}(n[F])} \tau$*, then* $\sigma, n \models F$ *iff* $\tau, n \models F$*.*
2. *If* $\sigma \simeq_{\mathrm{FLV}(n[A])} \tau$ *and* $\sigma|_1 \simeq_{\mathrm{FLV}(n[A])} \tau|_1$*, then* $\sigma, n \models A$ *iff* $\tau, n \models A$*.*

Proof. We prove the two assertions simultaneously by induction on the structure of (impure) formulas. Since the assertions are symmetrical in σ and τ, it suffices to show one direction of the equivalences.

Case: $v \in \mathcal{V}$. Since in this case 1. implies 2., we only prove the first assertion. Note that $\mathrm{FLV}(n[v]) = \{n.v\}$. Hence, because of $(s_0, \lambda_0) \sim_{\mathrm{FLV}(n[v])} (t_0, \mu_0)$ holds $n \in \mathsf{N}_{s_0}$ iff $n \in \mathsf{N}_{t_0}$ and if $n \in \mathsf{N}_{s_0}$, then $v \in \lambda_0(n)$ iff $v \in \mu_0(n)$, that is, $\sigma, n \models v$ iff $\tau, n \models v$.

Case: $\neg A$. We only show 1., assertion 2. can be shown in an analogous manner. So assume that A is a pure formula and $\sigma \simeq_{\mathrm{FLV}(n[\neg A])} \tau$ and $\sigma, n \models \neg A$, that is,

$\sigma, n \not\models A$. As $\mathrm{FLV}(n[\neg A]) = \mathrm{FLV}(n[A])$, the induction hypothesis for 1. and A can be applied, and we conclude $\tau, n \not\models A$, i.e. $\tau, n \models \neg A$.

Case: $A \Rightarrow B$. Also in this case, the proofs of 1. and 2. use the same arguments, so we only show 1. and assume that A and B are pure formulas and that $\sigma \simeq_{\mathrm{FLV}(n[A \Rightarrow B])} \tau$ holds. By the definition of FLV, it is obvious that $\mathrm{FLV}(n[A]), \mathrm{FLV}(n[B]) \subseteq \mathrm{FLV}(n[A \Rightarrow B])$. Hence, it follows $\sigma \simeq_{\mathrm{FLV}(n[A])} \tau$ and $\sigma \simeq_{\mathrm{FLV}(n[B])} \tau$. This fact and the induction hypothesis imply the assertion.

Case: $m[A]$. Again, we only show 1., the proof of 2. being analogous, and assume that A (and thus also $m[A]$) is a pure formula.
Assume $\sigma \simeq_{\mathrm{FLV}(n[m[A]])} \tau$ and $\sigma, n \models m[A]$, that is $m \not<_{s_0} n$ or $\sigma, m \models A$.

Case: $m \not<_{s_0} n$. By assumption and by lemma 2.4 holds $s_0 = t_0$, in particular $m \not<_{t_0} n$. Hence, $\tau, n \models m[A]$ holds trivially.

Case: $\sigma, m \models A$. As $\mathrm{FLV}(n[m[A]]) = \mathrm{FLV}(m[A])$, by assumption we have $\sigma \simeq_{\mathrm{FLV}(m[A])} \tau$, hence $\tau, m \models A$ by induction hypothesis.

Case: $\Box F$. Assume $\sigma, n \models \Box F$. We want to show $\tau, n \models \Box F$, i.e. for all $i \in \omega$ either $n \notin \mathsf{N}_{t_j}$ for some $j \leq i$ or $\tau|_i, n \models F$. So let $i \in \omega$ be such that $n \in \mathsf{N}_{t_j}$ for all $j \leq i$. Lemma 2.4,2. implies that there is some $k \in \omega$ with $n \in \mathsf{N}_{s_j}$ for all $j \leq k$ and $\sigma|_k \simeq_{\mathrm{FLV}(n[\Box F])} \tau|_i$. Latter is equivalent to $\sigma|_k \simeq_{\mathrm{FLV}(n[F]))} \tau|_i$, as $\mathrm{FLV}(n[\Box F]) = \mathrm{FLV}(n[F])$. Since $n \in \mathsf{N}_{s_j}$ for all $j \leq k$, by assumption holds $\sigma|_k, n \models F$ and therefore $\tau|_i, n \models F$ by induction hypothesis.

Case: $\Box[A]_S$. Assume $\sigma, n \models \Box[A]_S$. We have to show $\tau, n \models \Box[A]_S$, i.e. for every $i \in \omega$ either there is some $j \leq i$ with $n \notin \mathsf{N}_{t_j}$ or $\tau|_i, n \models A$ or $\tau|_i, n \models S$ iff $\tau|_{i+1}, n \models S$. Let $i \in \omega$ such that $n \in \mathsf{N}_{t_j}$ for all $j \leq i$. We distinguish two cases.

Case: $(t_i, \mu_i) \sim_{\mathrm{FLV}(n[\Box[A]_S])} (t_{i+1}, \mu_{i+1})$.
Since $\mathrm{FLV}(n[S]) \subseteq \mathrm{FLV}(n[\Box[A]_S])$, it holds obviously $\tau|_i \simeq_{\mathrm{FLV}(n[S])} \tau|_{i+1}$, and so by induction hypothesis $\tau|_i, n \models S$ iff $\tau|_{i+1}, n \models S$.

Case: $(t_i, \mu_i) \not\sim_{\mathrm{FLV}(n[\Box[A]_S])} (t_{i+1}, \mu_{i+1})$.
By lemma 2.4,3. there exists some $k \in \omega$ such that

1. $\tau|_i \simeq_{\mathrm{FLV}(n[\Box[A]_S])} \sigma|_k$,
2. $\tau|_{i+1} \simeq_{\mathrm{FLV}(n[\Box[A]_S])} \sigma|_{k+1}$ and

3. $\langle (s_0,\lambda_0)\ldots(s_k,\lambda_k)\rangle \simeq_{\mathrm{FLV}(n[\Box[A]_S])} \langle (t_0,\mu_0)\ldots(t_i,\mu_i)\rangle$.

Since $n \in \mathsf{N}_{t_j}$ for all $j \leq i$, condition 3. implies $n \in \mathsf{N}_{s_l}$ for all $l \leq k$. From conditions 1. and 2. together with the induction hypothesis for assertion 2. of the theorem for the impure formula A it follows $\tau|_i, n \models A$. (Note that $\mathrm{FLV}(n[A]) \subseteq \mathrm{FLV}(n[\Box[A]_S])$ and therefore it follows from 1. and 2. that $\sigma|_k \simeq_{\mathrm{FLV}(n[A])} \tau|_i$ and $\sigma|_{k+1} \simeq_{\mathrm{FLV}(n[A])} \tau|_{i+1}$.)

Now we prove the remaining cases of assertion 2.

Case: keep_m. By definition, $\sigma, n \models \mathsf{keep}_m$ holds iff $s_0{\downarrow}n.m = s_1{\downarrow}n.m$. By assumption we have $\sigma \simeq_\emptyset \tau$ and $\sigma|_1 \simeq_\emptyset \tau|_1$, in particular $s_0 = t_0$ and $s_1 = t_1$. The assertion trivially follows from this.

Case: $\circ F$. By assumption holds $\sigma|_1 \simeq_{\mathrm{FLV}(n[\circ F])} \tau|_1$. We distinguish two cases:

Case: $n \notin \mathsf{N}_{s_1}$. Then it also holds $n \notin \mathsf{N}_{t_1}$, and it follows $\tau, n \models \circ F$ by definition.

Case: $\sigma|_1, n \models F$. Because of $\mathrm{FLV}(n[F]) = \mathrm{FLV}(n[\circ F])$ it follows by the induction hypothesis for 1. and for F that $\tau|_1, n \models F$, hence $\tau, n \models \circ F$. ■

2.5 Spatial stuttering

Until now we only have considered a variant of the "usual" notion of stuttering invariance, that is, invariance with respect to temporal stuttering. In the context of a spatio-temporal logic it also makes sense to talk about "spatial" stuttering.

Definition 2.6 (Spatial stuttering equivalence) *Let $M \subseteq \mathsf{N}$ be a set of names. Two configurations (s,λ) and (t,μ) are called* M-equivalent *iff the following conditions are satisfied:*

1. $\mathsf{N}_s \cap M = \mathsf{N}_t \cap M$.
2. *For all $m, n \in \mathsf{N}_s \cap M$: $m <_s n$ iff $m <_t n$.*

3. *For all* $n \in \mathsf{N}_s \cap M$*:* $\lambda(n) = \mu(n)$.

In this case we write $(s,\lambda) \simeq_M (t,\mu)$.
Two runs $\sigma = (s_0,\lambda_0)(s_1,\lambda_1)\ldots$ *and* $\tau = (t_0,\mu)(t_1,\mu_1)\ldots$ *are called* M*-equivalent iff* $(s_i,\lambda_i) \simeq_M (t_i,\mu_i)$ *for all* $i \in \omega$.

It is obvious that $\simeq_M$ is an equivalence relation on the set of runs. It also follows immediately from the definition that for two sets of names M and N with $M \subseteq N$ we have $\simeq_N \subseteq \simeq_M$. Furthermore, if σ and τ are M-equivalent runs, then $\sigma|_i$ and $\tau|_i$ are M-equivalent, too, for every $i \in \omega$.

Intuitively, two configurations (s,λ) and (t,μ) are M-equivalent, if t arises from s by inserting and removing names that do not occur in M, while keeping the order between the names in M, and if the valuations λ and μ agree on all names in M. Figure 2.6 shows an example of two $\{n_0,n_1\}$-equivalent states.

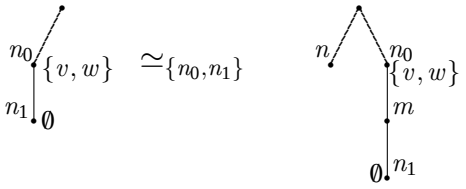


Figure 2.6: Two $\{n_0,n_1\}$-equivalent states

A nice property of the logic would be if a formula F was not able to distinguish between runs that are N-equivalent where N is the set of all names occurring in F. Unfortunately, this is not the case if we allow for the "keep"-operators, because keep_m restricts every name below m although they do not occur in keep_m. However, this "spatial stuttering invariance" property holds for pMTLA without the "keep"-operators.

For a formula A let $\mathsf{nm}(A)$ denote the set of names occurring in A. Again, we identify $\varepsilon[A]$ with A.

Theorem 2.7 (Spatial stuttering invariance) *Let* A *be an impure pMTLA-formula built without any of the formulas* keep_m*. Further, let* $\sigma = (s_0,\lambda_0)(s_1,\lambda_1)\ldots$ *and* $\tau = (t_0,\mu_0)(t_1,\mu_1)\ldots$ *be runs, and let* $n \in \mathsf{N}^\varepsilon$.

If $\sigma \simeq_{\mathsf{nm}(n[A])} \tau$*, then* $\sigma, n \models A$ *iff* $\tau, n \models A$.

Proof. We prove the assertion by induction on the structure of the (impure) formulas A. By symmetry, in all cases, it is enough to show one direction of the equivalence.

Case: v. Assume $\sigma \simeq_{\{n\}} \tau$ and $\sigma, n \models v$, that is, $n \in \mathsf{N}^{\varepsilon}_{s_0}$ and $v \in \lambda_0(n)$. By assumption we have $(s_0, \lambda_0) \simeq_{\{n\}} (t_0, \mu_0)$, hence $n \in \mathsf{N}^{\varepsilon}_{t_0}$ and $\mu_0(n) = \lambda_0(n)$, in particular $v \in \mu_0(n)$, that is, $\tau, n \models v$.

Case: $\neg A$. Assume $\sigma \simeq_{\mathsf{nm}(n[\neg A])} \tau$ and $\sigma, n \models \neg A$, that is, $\sigma, n \not\models A$. Since $\mathsf{nm}(n[\neg A]) = \mathsf{nm}(n[A])$, by induction hypothesis it follows $\tau, n \not\models A$, that is, $\tau, n \models \neg A$.

Case: $A \Rightarrow B$. Assume $\sigma \simeq_{\mathsf{nm}(n[A \Rightarrow B])} \tau$ and $\sigma, n \models A \Rightarrow B$.

Case: $\sigma, n \not\models A$. As $\mathsf{nm}(n[A]) \subseteq \mathsf{nm}(n[A \Rightarrow B])$, it follows from the assumption $\sigma \simeq_{\mathsf{nm}(n[A])} \tau$. Hence, by induction hypothesis we have $\tau, n \not\models A$ and therefore $\tau, n \models A \Rightarrow B$.

Case: $\sigma, n \models B$. Again, we have $\sigma \simeq_{\mathsf{nm}(n[B])} \tau$ by assumption. By induction hypothesis it follows $\tau, n \models B$ and so $\tau, n \models A \Rightarrow B$.

Case: $m[A]$. Assume $\sigma \simeq_{\mathsf{nm}(n[m[A]])} \tau$, and $\sigma, n \models m[A]$.

Case: $m \not\leq_{s_0} n$. Since $m \in \mathsf{nm}(n[m[A]])$ and $(s_0, \lambda_0) \simeq_{\mathsf{nm}(n[m[A]])} (t_0, \mu_0)$ by assumption, the definition of $\mathsf{nm}(n[m[A]])$-equivalence implies $m \not\leq_{t_0} n$, hence $\tau, n \models m[A]$.

Case: $\sigma, m \models A$. Note that $\mathsf{nm}(m[A]) \subseteq \mathsf{nm}(n[m[A]])$ and therefore it holds $\sigma \simeq_{\mathsf{nm}(m[A])} \tau$ by assumption. Hence, by induction hypothesis we conclude $\tau, m \models A$.

Case: $\circ F$. Assume $\sigma, n \models \circ F$.

Case: $n \notin \mathsf{N}_{s_1}$. In this case by assumption holds $n \notin \mathsf{N}_{t_1}$, hence $\tau, n \models \circ F$.

Case: $\sigma|_1, n \models F$. As $\mathsf{nm}(n[\circ F]) = \mathsf{nm}(n[F])$ and $\sigma|_1 \simeq_{\mathsf{nm}(n[\circ F])} \tau|_1$, by induction hypothesis it follows $\tau|_1, n \models F$.

Case: $\Box F$. Assume $\sigma, n \models \Box F$ and let $i \in \omega$ be arbitrary.

Case: There is a $j \leq i$ such that $n \notin \mathsf{N}_{s_j}$. Then it holds by assumption $n \notin \mathsf{N}_{t_j}$, and so $\tau, n \models \Box F$.

Case: $\sigma|_i, n \models F$. Since $\sigma|_i \simeq_{\mathsf{nm}(n[\Box F])} \tau|_i$ and $\mathsf{nm}(n[\Box F]) = \mathsf{nm}(n[F])$, it follows $\tau|_i, n \models F$ by the induction hypothesis.

Case: $\Box[A]_S$. Assume again $\sigma, n \models \Box[A]_S$ and let $i \in \omega$ be an arbitrary natural number.

Case: There is a $j \leq i$ with $n \notin \mathsf{N}_{s_i}$. The assertion follows by exactly the same arguments as in the previous case.

Case: $\sigma|_i, n \models A$. We can conclude by the assumption and by the induction hypothesis that $\tau|_i, n \models A$.

Case: $\sigma|_i, n \models S$ iff $\sigma|_{i+1}, n \models S$. By induction hypothesis, $\sigma|_j, n \models S$ iff $\tau|_j, n \models S$ holds for every $j \in \omega$. This implies $\tau|_i, n \models S$ iff $\tau|_{i+1}, n \models S$. ■

In theorem 2.5 and in theorem 2.7 we have shown that MTLA-formulas which do not contain the keep-operators are invariant under finite temporal and spatial stuttering. These two results can be combined in an obvious way. We first combine the notions of spatial and temporal stuttering equivalence.

Definition 2.8 *Let $M \subseteq \mathsf{N}$ be a set of names and $V \subseteq \{n.v | n \in \mathsf{N}^\varepsilon, v \in \mathcal{V}\}$ a set of localised variables.* (M, V)-equivalence*, written $\simeq_{(M,V)}$, is the smallest equivalence relation on runs that contains both $\simeq_M$ and $\simeq_V$.*

Now we can state a theorem about the "spatio-temporal" stuttering invariance of MTLA without keep_m.

Theorem 2.9 *Let F be a pure formula of propositional MTLA built without any of the formulas keep_m. Let $\sigma = (s_0, \lambda_0)(s_1, \lambda_1)\ldots$ and $\tau = (t_0, \mu_0)(t_1, \mu_1)\ldots$ be runs, and let $n \in \mathsf{N}^\varepsilon$.*

If $\sigma \simeq_{(\mathsf{nm}(n[F]), \mathrm{FLV}(n[F]))} \tau$, then $\sigma, n \models F$ iff $\tau, n \models F$.

Proof. Let M and V abbreviate $\mathsf{nm}(n[F])$ and $\mathrm{FLV}(n[F])$, respectively. By definition, $\sigma \simeq_{(M,V)} \tau$ means that there are runs $\rho_0, \ldots, \rho_k$ with

$$\sigma \simeq_{S_0} \rho_0 \simeq_{S_1} \rho_1 \cdots \rho_{k-1} \simeq_{S_k} \rho_k \simeq_{S_{k+1}} \tau$$

and $S_i \in \{M, V\}$. Hence, the assertion follows immediately from thm. 2.5 and thm. 2.7. ■

Chapter 3

Refinement

One of the reasons to choose TLA as the basis of our logic was its particular suitability for system development by stepwise refinement. As we have shown in the previous chapter, MTLA also has the property of invariance under finite stuttering, which is important in the context of refinement. However, the notion of refinement established in the context of reactive systems, that is operation refinement, does not suffice when systems relying on mobile code are concerned. For instance, in the course of system development, one may decide to implement a single high-level agent by several agents that "imitate" the behaviour of the original agent. Therefore, refinement principles that include the modification of the system's spatial structure are needed.

This chapter attempts to explore this question and to find suitable refinement notions for mobile systems. Besides operation refinement, we will describe the following concepts:

- *Spatial extension:* A single location n can be extended by sub-locations that implement different aspects of the behaviour of n. In general, the local variables at n will be distributed among the new locations. In this case, all these variables have to be hidden from the high-level interface.
- *Virtualisation of locations:* This principle allows to implement a high-level location by an arbitrary location hierarchy, even with a different name. In this

case, the name of the "virtualised" location must be hidden.

The different principles will be illustrated with the aid of our first specification example of a simple flight agent presented in chapter 2. We show that refinement can be expressed in MTLA by implication, that is, the fact that a low-level specification $Impl$ is a refinement of an abstract specification $Spec$ means essentially the validity of the formula $Impl \Rightarrow Spec$.

3.1 Operation refinement

First we consider the usual refinement of operations. Examples of operation refinement are: a high-level operation is implemented by a sequence of low level operations, or a high-level operation is restricted by strengthening its "preconditions". In the case of the flight agent (cf. page 11, fig. 2.3), one could require that the agent is not allowed to go home before it has found a certain number of offers. To express this restriction in the MTLA-specification, we have to modify the formulas $Move_{n,home}$ for every $n \in Net$. The altered formulas of the specification appear in fig. 3.1

$$
\begin{aligned}
MoveHome_n \;\equiv\; & \wedge\, n.ag\langle \textbf{true} \rangle \wedge \circ home.ag\langle \textbf{true} \rangle \\
& \wedge\, ag.ctl = \text{“busy”} \wedge |ag.found| \geq 5 \\
& \wedge\, \text{UNCHANGED}\ ag.ctl, ag.item, ag.found, home.res \\
RestrAgent \;\equiv\; & \wedge \ldots \\
& \wedge \textstyle\bigwedge_{n \in Net} \Box[MoveHome_n \vee \bigvee_{m \in Net \setminus \{home\}} Move_{n,m}]_{-n.ag}
\end{aligned}
$$

Figure 3.1: Flight Agent with restricted moves

It holds obviously that $\models MoveHome_n \Rightarrow Move_{n,home}$ for every name $n \in Net$, so it follows easily from the monotonicity of the operators $\Box[\_]_S$ that

$$RestrAgent \Rightarrow FlightAgent$$

is a valid MTLA-formula. This means that every run of a system satisfying specification $RestrAgent$ is also a run of a system specified by $FlightAgent$. Hence, $RestrAgent$ is a possible implementation of $FlightAgent$.

The restricted agent is an example of strengthening the precondition of an action. As another example of operation refinement we consider a variant of the flight agent that receives the components d and t of the item (d, t) separately. The interesting parts of specification $SepAgent$ are given in fig. 3.2. The formulas whose definitions do not appear in the figure ($SepDeliver$, $SepGetFlight_n$, etc.) are like the formulas $Deliver$, $GetFlight_n$, etc., but complemented with a conjunct UNCHANGED $ag.ctl_d, ag.ctl_t, ag.dest, ag.time$ to make sure that they do not modify the new variables.

Formulas $ChooseDest(d)$ and $ChooseTime(t)$ describe the actions of choosing a destination and a time period, respectively. The agent's control state depends, in addition to ctl, on two further local variables ctl_d and ctl_t; ctl_d has value "idle" as long as no destination is chosen. When destination d is chosen, variable ctl_d is set to "ready", and variable $dest$ to d. The meanings of the variables ctl_t and $time$ are similar.

The overall "preparation" of the agent is given by formula $SepPrep$. This action can be performed as soon as both, a destination d and a time period t are chosen, as indicated by the values of the variables $ag.ctl_d$ and $ag.ctl_t$. Variable $ag.ctl$ is set to "busy", and the agent obtains its task, expressed by $ag.item' = (ag.dest, ag.time)$.

Intuitively, it is clear that specification $SepAgent$ should be a correct implementation of the abstract $FlightAgent$: the steps of choosing the destination d and the time period t are internal steps. These actions together with $SepPrep$ implement the $FlightAgent$'s single "preparation action" $Prep(d, t)$. The internal actions $\exists d : ChooseDest(d)$ and $\exists t : ChooseTime(t)$ are not visible for $FlightAgent$ (they correspond to *stuttering steps*), as they do not modify any of the variables occurring in formula $FlightAgent$ nor the respective order of the locations. On the other hand, $SepPrep$ implies $\exists d, t : Prep(d, t)$, as the variables $ag.dest$ and $ag.time$ supply the witnesses for d and t required by the existential quantifier.

Technically speaking, the implication

$$\models SepAgent \Rightarrow FlightAgent$$

holds, due to the stuttering invariance of MTLA-formulas. A proof would be

$$
\begin{aligned}
SepInit \;\equiv\;& \land \ldots \\
& \land\, ag.ctl_d = \text{“idle”} \land ag.ctl_t = \text{“idle”} \\
ChooseDest(d) \;\equiv\;& \land\, ag\langle \textbf{true} \rangle \land \circ ag\langle \textbf{true} \rangle \\
& \land\, ag.ctl = \text{“idle”} \land ag.ctl_d = \text{“idle”} \\
& \land\, ag.dest' = d \land ag.ctl'_d = \text{“ready”} \\
& \land\, \text{UNCHANGED}\ vars, ag.time, ag.ctl_t \\
ChooseTime(t) \;\equiv\;& \land\, ag\langle \textbf{true} \rangle \land \circ ag\langle \textbf{true} \rangle \\
& \land\, ag.ctl = \text{“idle”} \land ag.ctl_t = \text{“idle”} \\
& \land\, ag.time' = t \land ag.ctl'_t = \text{“ready”} \\
& \land\, \text{UNCHANGED}\ vars, ag.dest, ag.ctl_d \\
SepPrep \;\equiv\;& \land\, ag\langle \textbf{true} \rangle \land \circ ag\langle \textbf{true} \rangle \\
& \land\, ag.ctl = \text{“idle”} \land ag.ctl_d = \text{“ready”} \land ag.ctl_t = \text{“ready”} \\
& \land\, ag.item' = (ag.dest, ag.time) \land ag.found' = \emptyset \\
& \land\, ag.ctl' = \text{“busy”} \\
& \land\, \text{UNCHANGED}\ ag.dest, ag.time, home.res \\
SepHomeActs \;\equiv\;& \lor (\exists d : ChooseDest(d)) \lor (\exists t : ChooseTime(t)) \\
& \lor SepPrep \lor SepDeliver \\
sepVars \;\equiv\;& \langle ag.ctl, ag.item, ag.found, ag.ctl_d, ag.ctl_t, \\
& \; ag.time, ag.dest, home.res \rangle \\
SepAgent \;\equiv\;& \land SepInit \\
& \land \ldots \\
& \land \Box[home[SepHomeActs] \lor \bigvee_{n \in Net} SepGetFlight_n]_{sepVars} \\
& \land \ldots
\end{aligned}
$$

Figure 3.2: Flight Agent with separate preparation steps

based mainly on instances of usual TLA rules like

$$\frac{(A \lor x' = x) \Rightarrow (B \lor y' = y)}{\Box[A]_x \Rightarrow \Box[B]_y}$$

as well as on the validity of the formulas

$$home[\exists d : ChooseDest(d)] \Rightarrow \text{UNCHANGED } vars$$
$$home[\exists t : ChooseTime(t)] \Rightarrow \text{UNCHANGED } vars$$
$$home[SepPrep] \Rightarrow home[\exists d, t : Prep(d, t)]$$

and

$$n[A \vee B] \Leftrightarrow n[A] \vee n[B] \quad .$$

3.2 Spatial extension

During the stepwise refinement of a mobile system one may decide to implement a single location of the abstract specification by a whole hierarchy of locations – network nodes may be equipped with sub-locations for different purposes, agents may have sub-nodes to store certain informations etc. When refining a location to several locations, in general the state of the high-level location is distributed among the new sub-locations. In the following we illustrate this refinement principle. We consider the – simpler – case when the variables of the original locations are not distributed separately.

3.2.1 Spatial extension without distribution of variables

In the context of the flight agent specification, one may wish that mobile agents are received inside a specific sub-location instead of directly beneath the network node. The visiting agent could be put first into a sub-node in_n of the location, to go through certain security checks, for example. Then it goes to a location $dock_n$ where the actual interaction takes place and finally, before leaving the network location, it has to visit the out_n sub-location.

Figure 3.3 illustrates the extension of a network node n_1 by the new sub-nodes.

Figure 3.4 shows the relevant parts of specification $DockedAgent$ of such a docked flight agent. The remaining formulas are the same as in specification $FlightAgent$.

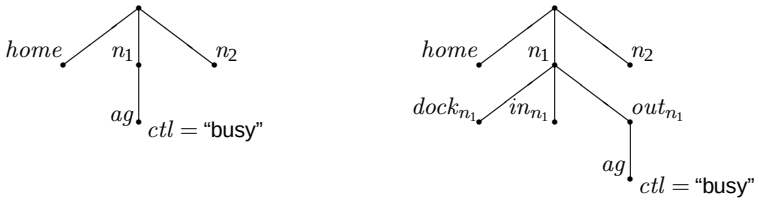


Figure 3.3: Spatial extension of node n_1

$$
\begin{aligned}
DockedInit &\equiv home.dock_{home}.ag\langle \textbf{true} \rangle \wedge ag.ctl = \text{“idle”} \\
DockedNetwork &\equiv \textstyle\bigwedge_{n,m \in Net} \wedge\, \Box n\langle m[\textbf{false}] \rangle \\
&\qquad \wedge\, \Box n\langle in_n\langle \textbf{true} \rangle \wedge dock_n\langle \textbf{true} \rangle \wedge out_n\langle \textbf{true} \rangle \rangle \\
&\qquad \wedge\, \Box[\textbf{false}]_{n.id} \\
SendAgent_n &\equiv \wedge\, n.dock_n.ag\langle \textbf{true} \rangle \wedge \circ n.out_n.ag\langle \textbf{true} \rangle \\
&\quad \wedge \text{ UNCHANGED } vars \\
DockedMove_{n,m} &\equiv \wedge\, n.out_n.ag\langle \textbf{true} \rangle \wedge \circ m.in_m.ag\langle \textbf{true} \rangle \wedge \mathsf{keep}_{ag} \\
&\quad \wedge \text{ UNCHANGED } vars \\
RcvAgent_n &\equiv \wedge\, n.in_n.ag\langle \textbf{true} \rangle \wedge \circ n.dock_n.ag\langle \textbf{true} \rangle \\
&\quad \wedge \text{ UNCHANGED } vars \\
DockedAgent &\equiv \wedge\, DockedInit \\
&\quad \wedge \ldots \\
&\quad \wedge \textstyle\bigwedge_{n \in Net} \wedge\, \Box[SendAgent_n]_{-dock_n.ag} \\
&\qquad \wedge\, \Box[RcvAgent_n]_{-in_n.ag} \\
&\qquad \wedge\, \Box[\textstyle\bigvee_{m \in Net} DockedMove_{n,m}]_{-out_n.ag}
\end{aligned}
$$

Figure 3.4: A docked flight agent

Formula $DockedInit$ says that the agent initially resides at the home location's sub-location $dock_{home}$; formula $DockedNetwork$ requires every network node n to have sub-locations in_n, $dock_n$, and out_n that cannot "disappear". The actions $SendAgent_n$, $RcvAgent_n$ and $DockedMove_n$ control the agent's movements between the different sub-locations and network nodes, respectively.

Observe that $SendAgent_n$ and $RcvAgent_n$ do not change any of the (interesting) variables, and that the agent stays below location n. From this, and from the

fact that the operators $n[\_]$ refer not just to the immediate successors of a node, but look arbitrarily deep inside the tree, it follows that the transitions described by $SendAgent_n$ and $RcvAgent_n$, respectively, cannot be observed by formula $FlightAgent$ (that is, they correspond to stuttering steps of the original specification). Altogether, this specification is again a refinement of the original flight agent specification, and it holds indeed that

$$\models DockedAgent \Rightarrow FlightAgent \quad .$$

Let us emphasise once again that for the validity of this implication it is crucial that the operators $n[\_]$ refer to sub-nodes arbitrarily deep below the root, or more formally, that the following holds:

$$\models n[F] \Rightarrow m[n[F]] \quad .$$

This interpretation of $n[\_]$ enables spatial extension to be represented by implication in MTLA.

3.2.2 Spatial extension with distribution of variables

In the case of the docked agent, refinement could be expressed simply by implication because the new locations in_n, $dock_n$, and out_n did not have any local variables. In general, the refined location's local state will be distributed among the new sub-nodes.

In order to illustrate this form of refinement, we consider again the example of the flight agent. Imagine that the offers at a network node n are kept in a database placed in a sub-location db_n of n. A part of the specification of such an agent – showing only the modified actions – appears in fig. 3.5.

For this specification does not hold that $\models DBAgent \Rightarrow FlightAgent$, because the new agent draws the information about the offers from the variable $db_n.flights$, whereas in the case of the original agent the flights are stored in the variable $n.flights$. However, the implication holds if we "hide" these variables. Hiding of state components is expressed, as in TLA, by existential quantification over

$$
\begin{aligned}
DBNetwork &\equiv \textstyle\bigwedge_{n \in Net} n.db_n \langle \textbf{true} \rangle \wedge \Box[\textbf{false}]_{-n.db_n, n.id} \\
DBGetFlight_n &\equiv \wedge \ldots \\
&\quad \wedge\ ag.ctl = \text{“busy”} \wedge ag.item \in db_n.flights \\
&\quad \wedge\ ag.found' = ag.found \cup getFlight(ag.item, db_n.flights) \\
&\quad \wedge\ \text{UNCHANGED}\ ag.ctl, ag.item, ag.dests, home.res \\
DBAgent &\equiv \wedge\ Init \\
&\quad \wedge\ DBNetwork \\
&\quad \wedge\ \Box\big[home[HomeActs] \vee \textstyle\bigvee_{n \in Net} DBGetFlight_n\big]_{vars} \\
&\quad \wedge\ \textstyle\bigwedge_{n \in Net} \Box[\bigvee_{m \in Net} Move_{n,m}]_{-n.ag}
\end{aligned}
$$

Figure 3.5: Network nodes with database sub-locations

flexible variables. We extend the definition of MTLA-formulas by the following clause:

$$\exists\, m.v : F$$

is a pure formula, where $m \in \mathsf{N}$ denotes a name, $v \in \mathcal{V}_f$ a flexible variable and F a pure MTLA-formula.

The definition of the semantics of existential quantification over flexible variables requires some preparation. The difficulty is to define it in a way that stuttering invariance is preserved.

We first define what it means that two runs are *similar up to v at m* for a flexible variable v and a location $m \in \mathsf{N}^\varepsilon$. Recall the definition of stuttering equivalence $\simeq$ (cf. p.19, def. 2.3).

Definition 3.1 *1. Two runs $\sigma = (s_0, \lambda_0)(s_1, \lambda_1)\ldots$ and $\tau = (t_0, \mu_0)(t_1, \mu_1)\ldots$ are called* equal up to v at m, *written $\sigma =_{m.v} \tau$, iff $s_i = t_i$ for all $i \in \omega$ and $\lambda_i(n, w) = \mu_i(n, w)$ for all $(n, w) \neq (m, v)$.*

2. Similarity up to v at m, *denoted by $\approx_{m.v}$, is the smallest equivalence relation that contains both $=_{m.v}$ and $\simeq$.*

Now we can define the semantics of the flexible existential quantifier:

$$\sigma, n, \xi \models \exists\, m.v : F \quad \text{iff} \quad \text{there exists a run } \tau \approx_{m.v} \sigma \text{ with } \tau, n, \xi \models F \quad .$$

Intuitively, formula $\exists\, m.v : F$ asserts that we can find values for v at m for which F holds (cf. [31]).

For our specification holds

$$\models DBAgent \Rightarrow \exists\, n_1.\mathit{flights} \ldots \exists\, n_k.\mathit{flights} : FlightAgent$$

where $\{n_1, \ldots, n_k\} = Net$. The proof of the validity of this formula would rely on an axiom

$$(\exists\text{-I}) \quad F[t/n.v] \Rightarrow \exists\, n.v : F$$

where $F[t/n.v]$ denotes the "localised" substitution of t for v at n in formula F. Informally, the localised substitution $F[t/n.v]$ replaces all top-level occurrences of v in sub-formulas $n[F]$ (that is, occurrences that are not in the scope of further spatial modalities) by t. The precise inductive definition is given in fig. 3.6, p. 38, which also introduces a corresponding auxiliary notion $r[t/n.v]$ of localised substitution in a term r.

In our case, the "witness" terms are the variables *flights* at the database locations: it is easy to see that the implication

$$DBAgent \Rightarrow FlightAgent[db_{n_1}.\mathit{flights}/n_1.\mathit{flights}, \ldots, db_{n_k}.\mathit{flights}/n_k.\mathit{flights}]$$

is valid.

3.3 Virtualisation of locations

The last and probably most radical refinement principle that we consider is what we call *virtualisation of locations*. This form of refinement allows locations of an abstract specification to be implemented by a structurally different location hierarchy. For example, the flight agent specified by formula $FlightAgent$ in fig. 2.3,

$$
\begin{array}{rcl}
w[t/n.v] & = & \begin{cases} t & \text{if } w = v \text{ and } n = \varepsilon \\ w & \text{otherwise} \end{cases} \\
x[t/n.v] & = & x \\
(\iota x : A)[t/n.v] & = & \iota x : A[t/n.v] \\
f(r_1, \ldots, r_k)[t/n.v] & = & f(r_1[t/n.v], \ldots, r_k[t/n.v]) \\
P(r_1, \ldots, r_k)[t/n.v] & = & P(r_1[t/n.v], \ldots, r_k[t/n.v]) \\
(A \Rightarrow B)[t/n.v] & = & A[t/n.v] \Rightarrow B[t/n.v] \\
(\neg A)[t/n.v] & = & \neg A[t/n.v] \\
(m[A])[t/n.v] & = & \begin{cases} m[A] & \text{if } n = \varepsilon \\ m\big[A[t/\varepsilon.v]\big] & \text{if } m = n \\ m\big[A[t/n.v]\big] & \text{otherwise} \end{cases} \\
(\exists x : A)[t/n.v] & = & \exists x : A[t/n.v] \\
(\Box F)[t/n.v] & = & \Box F[t/n.v] \\
(\circ F)[t/n.v] & = & \circ F[t/n.v] \\
\mathsf{keep}_m[t/n.v] & = & \mathsf{keep}_m \\
(\Box[A]_r)[t/n.v] & = & \Box[A[t/n.v]]_{r[t/n.v]} \\
(\Box[A]_S)[t/n.v] & = & \Box[A[t/n.v]]_{S[t/n.v]}
\end{array}
$$

Figure 3.6: Localised substitution

p. 11, could be implemented by several agents. However, in order to be a correct implementation, the (name of the) original agent has to be hidden from the high-level interface, intuitively meaning that the system with several agents behaves *as if there was* an agent ag satisfying specification $FlightAgent$.

In the following we present the MTLA-specification of such a possible implementation of $FlightAgent$ and use it to discuss the notions of virtualisation and of hiding of locations.

Figure 3.7 shows specification $MultiAgent$, for the sake of brevity with only two agents ag_0 and ag_1.

$$
\begin{aligned}
MAInit &\equiv \textstyle\bigwedge_{i=0}^{1} \wedge\, home.ag_i\langle \mathbf{true}\rangle \wedge ag_i.ctl = \text{“idle”} \\
&\quad \wedge\, home.return_i = \text{“false”} \\
MAPrep(d,t) &\equiv \textstyle\bigwedge_{i=0}^{1} \wedge\, home.ag_i\langle \mathbf{true}\rangle \wedge \circ home.ag_i\langle \mathbf{true}\rangle \\
&\quad \wedge\, ag_i.ctl = \text{“idle”} \wedge ag_i.ctl' = \text{“busy”} \\
&\quad \wedge\, ag_i.item' = (d,t) \wedge ag_i.found' = \emptyset \\
&\quad \wedge\, \text{UNCHANGED } vars_{home} \\
MAMove_{n,m,i} &\equiv \wedge\, n.ag_i\langle \mathbf{true}\rangle \wedge \circ m.ag_i\langle \mathbf{true}\rangle \\
&\quad \wedge\, ag_i.ctl = \text{“busy”} \wedge \mathsf{keep}_{ag_i} \\
&\quad \wedge\, \text{UNCHANGED } vars_0, vars_1, vars_{home} \\
MAGetFlight_{n,i} &\equiv \wedge\, n.ag_i\langle \mathbf{true}\rangle \circ n.ag_i\langle \mathbf{true}\rangle \\
&\quad \wedge\, ag_i.ctl = \text{“busy”} \wedge ag.item \in n.flights \\
&\quad \wedge\, ag_i.found' = ag_i.found \cup getFlight(ag_i.item, n.flights) \\
&\quad \wedge\, \text{UNCHANGED } ag_i.ctl, ag_i.item, vars_{1-i}, vars_{home} \\
MADeliver_i &\equiv \wedge\, home.ag_i\langle \mathbf{true}\rangle \wedge \circ home.ag_i\langle \mathbf{true}\rangle \\
&\quad \wedge\, ag_i.ctl = \text{“busy”} \wedge ag_i.ctl' = \text{“idle”} \\
&\quad \wedge\, home.return_i' = \text{“true”} \wedge home.res_i' = ag_i.found \\
&\quad \wedge\, \text{UNCHANGED } vars_{1-i}, home.res, home.res_{1-i} \\
&\quad \wedge\, \text{UNCHANGED } home.return_{1-i} \\
Merge &\equiv \wedge\, home.return_0 = \text{“true”} \wedge home.return_1 = \text{“true”} \\
&\quad \wedge\, home.res' = home.res_0 \cup home.res_1 \\
&\quad \wedge\, home.return_0' = \text{“false”} \wedge home.return_1' = \text{“false”} \\
&\quad \wedge\, \text{UNCHANGED } vars_0, vars_1 \\
MAHomeActs &\equiv \vee\, (\exists d,t : MAPrep(d,t)) \vee Merge \\
&\quad \vee\, MADeliver_0 \vee MADeliver_1 \\
Next &\equiv MAHomeActs \vee \textstyle\bigvee_{n\in Net}(MAGetFlight_{n,0} \vee MAGetFlight_{n,1}) \\
vars_0 &\equiv \langle ag_0.ctl, ag_0.item, ag_0.found\rangle \\
vars_1 &\equiv \langle ag_1.ctl, ag_1.item, ag_1.found\rangle \\
vars_{home} &\equiv \langle home.res, home.res_0, home.res_1, home.return_0, home.return_1\rangle \\
MultiAgent &\equiv \wedge\, MAInit \\
&\quad \wedge\, Network \\
&\quad \wedge\, \Box[Next]_{vars_{home}, vars_0, vars_1} \\
&\quad \wedge\, \textstyle\bigwedge_{n\in Net} \Box[\bigvee_{m\in Net} MAMove_{n,m,0}]_{-n.ag_0} \\
&\quad \wedge\, \textstyle\bigwedge_{n\in Net} \Box[\bigvee_{m\in Net} MAMove_{n,m,1}]_{-n.ag_1}
\end{aligned}
$$

Figure 3.7: Specification of the “multi agent”

The initial configuration of the system is given by $MAInit$. This formula asserts that in the beginning both agents are at their home location, their control state being "idle" and that they are not returned yet. Action $MAPrep(d,t)$ sets both agents' $item$ to (d,t), the variables $found$ to $\emptyset$, and additionally it requires the $home$ location's local state to remain unaltered.

Formula $MAMove_{n,m,i}$ controls agent ag_i's movements. Action $MADeliver_i$ is a slight modification of $Deliver$, ag_i substituted for ag: it additionally sets variable $home.return_i$ to "true", and instead of changing variable $home.res$ directly, $home.res_i$ is set to $ag_i.found$.

Formula $Merge$ describes the uniting of the offers collected by the different agents. As the results should not be combined before both agents are back, this action has the precondition $home.return_0 =$ "true" $\wedge\, home.return_1 =$ "true".

We claim that this specification is a correct implementation of $FlightAgent$ if agent ag is hidden from the interface.

We discuss now what "hiding" of a location means.

The *hiding* of a location is technically realised by existential quantification over names. We extend the syntax definition of MTLA by the following clause:

$$\exists\, m : F$$

is a pure formula for a pure formula F and a name $m \in \mathsf{N}$.

Intuitively, a run σ satisfies $\exists\, m : F$ iff there is a run τ for which F holds and that arises from σ by extending the trees by name m at every configuration of σ. The precise definition of the semantics of this (flexible) name quantifier is rather involved and requires some preparation.

For finite trees s and t and a name $n \in \mathsf{N}$ the relation $s <_n t$ is defined by

$$s <_n t \quad \text{iff} \quad \mathsf{N}_s = \mathsf{N}_t \setminus \{n\} \text{ and } (a <_s b \text{ iff } a <_t b \text{ for all } a, b \in \mathsf{N}_s) \quad .$$

This relation is extended to configurations by

$$(s,\lambda) <_n (t,\mu) \quad \text{iff} \quad s <_n t \text{ and } \lambda(m,v) = \mu(m,v) \text{ for all } m \in \mathsf{N}_s \text{ and } v \in \mathcal{V}_f \quad .$$

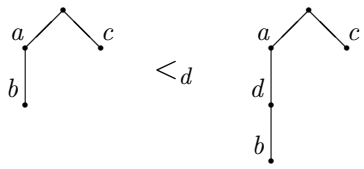


Figure 3.8: Tree extension

Figure 3.8 shows an example of tree extension.

The relation is extended to runs like expected:

$$(s_0,\lambda_0)(s_1,\lambda_1)\ldots <_n (t_0,\mu_0)(t_1,\mu_1)\ldots \text{ iff } (s_i,\lambda_i) <_n (t_i,\mu_i) \text{ for all } i \in \omega \quad .$$

Now we have all ingredients to define the semantics of the name quantifier.

$$\sigma, n, \xi \models \exists\, m : F \quad \text{iff} \quad \text{there exist runs } \rho,\tau \text{ such that } \sigma \simeq \rho,\ \rho <_l \tau, \text{ and } \tau, n, \xi \models F[l/m] \text{ for a name } l \text{ that occurs neither in } F \text{ nor in } \sigma \quad .$$

Specification $MultiAgent$ in fig. 3.7 is indeed a refinement of $FlightAgent$ if location ag is hidden. Logically, this means the validity of the implication

$$MultiAgent \Rightarrow \exists\, ag : FlightAgent \quad . \tag{3.1}$$

In order to see why this formula is valid, consider a run σ satisfying specification $MultiAgent$. At any configuration we have to find the place where to put the "witness" agent – let's call it $witness$ – and to decide how to set its local variables.

First observe that the two agents' actions are never performed simultaneously. So whenever ag_i executes an action at a network node n, agent $witness$ should be put at n and its local state should change in accordance with ag_i's variables.

If $witness$ has to imitate ag_0 at n_0 at some instant and ag_1 at n_1 in the next step, then we have to add a stuttering step in order to move it from n_0 to n_1.

Consider for example the following situation. First, agent ag_1 is at location n_1 and performs action $MAGetFlight_{n_{1,1}}$. At the next moment, ag_0, located at network location n_2, executes $MAGetFlight_{n_{2,0}}$. In this case, $witness$ has to be at n_1 first.

As variable $ag_1.found$ changes by adding a new offer to it, the same should happen to $witness.found$. Now we have to introduce a stuttering step so that the "virtual" agent $witness$ can be moved from n_1 to n_2, by executing action $Move_{n_1,n_2}$ of specification $FlightAgent$. When $MAGetFlight_{n_2,0}$ is performed in the original run, $witness.found$ is set in accordance with the change of $ag_0.found$: the same new offer is added to $witness.found$ as to $ag_0.found$. At any instant, the value of $witness.found$ should be the union of the values $ag_0.found$ and $ag_1.found$; $witness.ctl$ equals "busy" iff at least one of the agents is busy; $witness.item$ is the same as $ag_0.item$ ($= ag_1.item$).

Fig. 3.9 illustrates how to construct from a run satisfying specification $MultiAgent$ another one for which holds $FlightAgent[witness/ag]$. The first line shows a part of a run σ where first $MAGetFlight_{n_1,1}$ and then $MAGetFlight_{n_2,0}$ is executed. First, we add a stuttering step to the original run (we will need this additional step

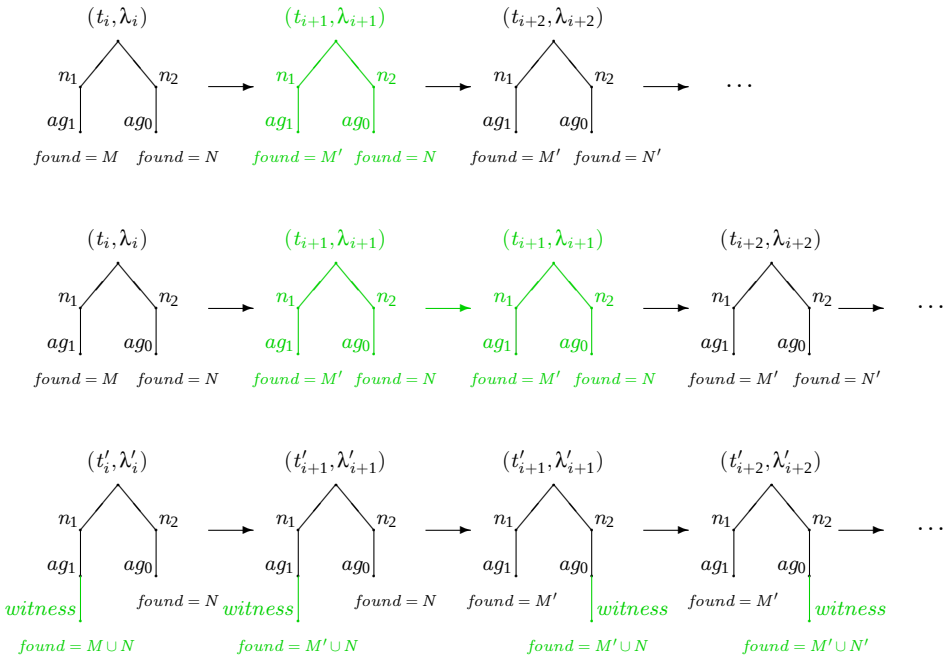


Figure 3.9: Illustration to $MultiAgent$

to move the abstract agent from site n_1 to n_2) and obtain a run ρ with $\rho \simeq \sigma$. Then we extend every configuration by $witness$. We put it below the agent whose action it has to imitate. In this way we get a run τ with $\rho <_{witness} \tau$. The local variables at

$witness$ are set as described above. The (piece of the) run we obtain corresponds to the following sequence of high-level actions:

$$GetFlight_{n_1}[witness/ag], Move_{n_1,n_2}[witness/ag], GetFlight_{n_2}[witness/ag] \quad .$$

Formal proofs of formulas with hidden names make use of introduction axioms of the form

$$(\exists\text{-ref}) \quad F[n/m, t_1/m.v_1, \ldots, t_k/m.v_k] \Rightarrow \exists\, m : F$$
$$(\exists\text{-sub}) \quad n\langle\textbf{true}\rangle \Rightarrow \exists\, m : n.m\langle\textbf{true}\rangle \qquad (m \neq n) \quad .$$

Axiom ($\exists$-ref) corresponds to axiom ($\exists$-I) in sec. 3.2.2. In order to conclude a formula $\exists\, m : F$, it calls for witnesses for m as well as for the local variables $v_1, \ldots, v_k$ at m. The second axiom allows to extend a location n by a "virtual" sub-location.

In our example, we would use axiom ($\exists$-ref) with the substitutions

$$ag.found \leftarrow ag_0.found \cup ag_1.found \qquad ag.item \leftarrow ag_0.item$$
$$ag.ctl \leftarrow \textbf{if}\ home.return_0 = \text{“false”} \wedge home.return_1 = \text{“false”}\ \textbf{then}\ \text{“idle”}\ \textbf{else}\ \text{“busy”}$$

complemented with a "spatial refinement mapping" that returns ag_0 or ag_1, depending on which one of the implementation level agents is performing an action. As this cannot be determined in terms only of the current state, additional auxiliary – in this case prophecy – variables [2] will be needed.

Here, the implication can be proved without using axiom ($\exists$-sub), but it will be needed in many situations, in particular to prove refinement when a high-level specification that uses mobile agents is implemented by other techniques than mobility.

Let us emphasise once again that the above-mentioned rules for the name quantifier are not complete, and that in general additional history and/or prophecy variables will have to be introduced.

Chapter 4

Axiomatisation

In this chapter we examine the axiomatisation of propositional MTLA. This logic can be regarded as a fragment of a spatio-temporal logic whose spatial operators are those of MTLA, but in which there is no distinction between pure and impure formulas and the next-time and always operators can be applied to arbitrary formulas. Since $\Box[A]_S$ is then equivalent with $\Box(A \vee (S \Leftrightarrow \circ S))$, MTLA can be understood as a fragment of the spatial part of the logic together with the temporal operators keep_m, $\circ$ and $\Box$. As this is just LTL extended by MTLA's spatial operators, we will call this logic (propositional) MLTL, or pMLTL for short. Formally, the language of pMLTL is given by the following:

$$v \in \mathcal{V} \mid F \Rightarrow G \mid \neg F \mid m[F] \mid \mathsf{keep}_m \mid \circ F \mid \Box F \quad .$$

The semantics of these formulas is defined as for (propositional) MTLA:

Definition 4.1 *Let $\sigma = (t_0, \lambda_0), (t_1, \lambda_1) \ldots$ be a run with $t_i = (\mathsf{N}_i, <_i)$ as defined in sec. 2.3. The semantics of pMLTL formulas is defined as follows.*

- $\sigma, n \models v$ *iff* $n \in \mathsf{N}_0^\varepsilon$ *and* $v \in \lambda_0(n)$
- $\sigma, n \models \neg F$ *iff* $\sigma, n \not\models F$
- $\sigma, n \models F \Rightarrow G$ *iff* $\sigma, n \not\models F$ *or* $\sigma, n \models G$

- $\sigma, n \models m[F]$ *iff* $m \not<_0 n$ *or* $\sigma, m \models F$
- $\sigma, n \models \mathsf{keep}_m$ *iff* $t_0{\downarrow}n.m = t_1{\downarrow}n.m$
- $\sigma, n \models \bigcirc F$ *iff* $n \notin \mathsf{N}_1$ *or* $\sigma|_1, n \models F$
- $\sigma, n \models \Box F$ *iff* *for all* $i \geq 0$ *either* $n \notin \mathsf{N}_j$ *for some* $j \leq i$ *or* $\sigma|_i, n \models F$

In the following we introduce a proof system Σ_{MLTL} and show that it is sound and complete with respect to the semantics of propositional MLTL. We have decided to present the axiomatisation and the proofs for MLTL instead of MTLA for the sake of simplicity. The proof system can be adapted to MTLA along the lines of [34].

In order to deal with the different difficulties separately, the axiomatisation is divided in three parts. In the first step we only consider the spatial part (called SL, for spatial logic) of the logic and provide a sound and complete axiomatisation Σ_{SL} for it. Completeness is proven by showing how to construct a model – in this case a configuration (t, λ) in the sense of sec. 2.3 – for a given finite and consistent set of formulas in the spatial fragment of pMLTL. The second step is to extend the proof system by axioms and rules that characterise the temporal operators $\bigcirc$ and $\Box$ and their interplay with the spatial operators. The completeness proof consists again in the construction of a model for a given finite and consistent set $\mathcal{F}$ of formulas, that is, of a run σ for which $\sigma, \varepsilon \models F$ holds for every $F \in \mathcal{F}$. The last step is to present axioms specific to the formulas keep_m and to extend the completeness proof accordingly.

4.1 The proof system Σ_{SL}

Let $\mathcal{V}$ be a denumerable set of propositional variables and N a denumerable set of names with $\mathcal{V} \cap \mathsf{N} = \emptyset$. Let SL denote the "spatial part" of propositional MLTL, that is, its language contains the following formulas:

$$v \in \mathcal{V} \mid F \Rightarrow G \mid \neg F \mid m[F] \quad .$$

We will use the abbreviations introduced for MLTL whenever applicable, that is, we will write $m\langle F\rangle$ for $\neg m\langle\neg F\rangle$, $n_1.\cdots.n_i[F]$ for $n_1[\cdots n_i[F]\cdots]$, $n_1.\cdots.n_i\langle F\rangle$ for $n_1\langle\cdots n_i\langle F\rangle\cdots\rangle$ and sometimes simply $n_1.\cdots.n_i$ instead of $n_1.\cdots.n_i\langle\mathbf{true}\rangle$.

The semantics of SL is defined in terms of finite trees whose nodes have unique names and are labelled by sets of propositional variables. Formally, it is defined as follows:

Definition 4.2 *Let $t = (\mathsf{N}_t, <_t)$ be a finite tree, $n \in \mathsf{N}^\varepsilon$ and $\lambda : \mathsf{N}_t^\varepsilon \to 2^{\mathcal{V}}$ a labelling. The semantics of SL is defined inductively:*

- $t,\lambda,n \models v$ *iff* $n \in \mathsf{N}_t^\varepsilon$ *and* $v \in \lambda(n)$
- $t,\lambda,n \models \neg F$ *iff* $t,\lambda,n \not\models F$
- $t,\lambda,n \models F \Rightarrow G$ *iff* $t,\lambda,n \not\models F$ *or* $t,\lambda,n \models G$
- $t,\lambda,n \models m[F]$ *iff* $m \not<_t n$ *or* $t,\lambda,m \models F$

We call a formula F *valid* and write $\models F$ iff for all finite non-empty trees t and all assignments $\lambda : \mathsf{N}_t^\varepsilon \to 2^{\mathcal{V}}$ holds: $t,\lambda,\varepsilon \models F$.

Let the proof system Σ_{SL} be defined as given in figure 4.1.

(ax0)	$\vdash F$ if F is tautological	(axn0)	$\vdash n[F]$ if F is tautological
(ax1)	$\vdash a[F \Rightarrow G] \Rightarrow (a[F] \Rightarrow a[G])$	(axn1)	$\vdash n[a[F \Rightarrow G] \Rightarrow (a[F] \Rightarrow a[G])]$
(ax2)	$\vdash a[F] \Rightarrow b[a[F]]$	(axn2)	$\vdash n[a[F] \Rightarrow b[a[F]]]$
(ax3)	$\vdash \neg a[F] \Rightarrow a[\neg F]$		
(ax4)	$\vdash a[a[\mathbf{false}]]$		
(ax5)	$\vdash (a_1.b \wedge a_2.b) \Rightarrow (a_1.a_2 \vee a_2.a_1)$ (for $a_1 \neq a_2$)		

$$\text{(mp)}\quad \frac{\vdash F \Rightarrow G \qquad \vdash F}{\vdash G}$$

Figure 4.1: The proof system Σ_{SL}.

We write $\vdash F$ iff F is derivable in Σ_{SL}; for a set $\mathcal{F}$ of formulas we write $\mathcal{F} \vdash F$ iff F is derivable in Σ_{SL} possibly using assumptions from the set $\mathcal{F}$. For a finite set $\mathcal{F}$ of formulas we also use $\mathcal{F}$ to denote the conjunction of all formulas in $\mathcal{F}$.

In our derivations we will write (prop) to indicate the use of propositional reasoning. Consistency of a set of formulas is defined as usual:

Definition 4.3 *A finite set $\mathcal{F}$ of formulas in SL is called* consistent *iff $\nvdash \neg\mathcal{F}$ holds. Otherwise it is called* inconsistent.

Let us have a look at the axioms of Σ_{SL}. The first axiom (ax0) is clear; (ax1) is the usual K-axiom for modal logics; (ax2) ensures that the spatial operators $n[\_]$ can look arbitrarily deep inside the tree; axiom (ax3) asserts that there is at most one "a-successor"; (ax4) claims that two nodes with the same name must not occur on the same path. Axiom (ax5) describes the tree structure of the model: whenever node b is below a_1 as well as below a_2, the two nodes have to be on the same path.

The axioms (axn0) - (axn2) are "boxed" versions of axioms (ax0) - (ax2) expressing that the logic works at any node n in the same way as at the root. Note that we do not provide axioms corresponding to the axioms (ax3) - (ax5) put inside the spatial operators $n[\_]$ even though those formulas are valid. The reason is that one can derive them using the axioms listed in figure 4.1. The derivations are given in the appendix. Observe that instead of the axioms (axn0) - (axn2) we could have provided a generalisation rule

$$\frac{\vdash F}{\vdash n[F]} \tag{4.1}$$

for every name n. Although this would be a sound rule for SL, we refrain from taking it into Σ_{SL}, because it is not sound with respect to propositional MLTL. For details on this we refer to sec 4.2. We will show that (4.1) is derivable in Σ_{SL}, but first we derive a few useful formulas.

Proposition 4.4 *Let F, G denote formulas, $a, b, c \in \mathsf{N}$ names and $\alpha, \beta, \gamma \in \mathsf{N}^+$ non-empty sequences of names.*

1. (T0) $\vdash a[F] \Rightarrow \alpha.a[F]$

2. (T1) $\vdash \neg\alpha[F] \Rightarrow \alpha[\neg F]$

3. *(a)* (T2) $\vdash \alpha[F \Rightarrow G] \Rightarrow (\alpha[F] \Rightarrow \alpha[G])$

   (b) (T3) $\vdash (\alpha[F] \Rightarrow \alpha[G]) \Rightarrow \alpha[F \Rightarrow G]$

4. (T4) *If $F \Rightarrow G$ is tautological, then* $\vdash \alpha[F] \Rightarrow \alpha[G]$

5. (T5) $\vdash \alpha\langle\neg F\rangle \Leftrightarrow \neg\alpha[F]$

6. (T6) $\vdash \alpha\langle F\rangle \Leftrightarrow (\alpha[F] \wedge \alpha\langle\mathbf{true}\rangle)$

7. (T7) $\vdash \alpha.\beta\langle\mathbf{true}\rangle \wedge \beta.\gamma\langle F\rangle \Leftrightarrow \alpha.\beta.\gamma\langle F\rangle$

8. (T8) $\vdash a.b.c\langle F\rangle \Rightarrow a.c\langle F\rangle$

9. *(a)* (T9a) $\vdash a[F \vee G] \Leftrightarrow a[F] \vee a[G]$

   (b) (T9b) $\vdash a[F \wedge G] \Leftrightarrow a[F] \wedge a[G]$

Proof.

1. The assertion is easy to prove by induction on the length of α and by using (ax2) and propositional reasoning.

2. If α is of length one, then this is simply (ax3). Otherwise let $\alpha = \beta.a$ with $\beta \in \mathsf{N}^+$. We can derive the formula as follows:

(1)	$\neg\beta.a[F] \Rightarrow \neg a[F]$	(T0),(prop)
(2)	$\neg a[F] \Rightarrow a[\neg F]$	(ax3)
(3)	$a[\neg F] \Rightarrow \beta.a[\neg F]$	(T0)
(4)	$\neg\beta.a[F] \Rightarrow \beta.a[\neg F]$	(1),(2),(3),(prop)

3. *(a):* We prove the assertion by induction on the length of the path α. If α has length 1, then it is (ax1). So we can assume that $\alpha = \beta.b$ with $\beta \in \mathsf{N}^+$ and that the assertion is already proved for β.

(1)	$\beta[b[F \Rightarrow G] \Rightarrow b[F] \Rightarrow b[G]]$	(axn1),(T0),(mp)
(2)	$\beta[b[F \Rightarrow G] \Rightarrow b[F] \Rightarrow b[G]] \Rightarrow$ $(\beta.b[F \Rightarrow G] \Rightarrow \beta[b[F] \Rightarrow b[G]])$	I.H.
(3)	$\beta[b[F] \Rightarrow b[G]] \Rightarrow (\beta.b[F] \Rightarrow \beta.b[G])$	I.H.
(4)	$\beta.b[F \Rightarrow G] \Rightarrow (\beta.b[F] \Rightarrow \beta.b[G])$	(1),(2),(3),(prop)

(b): Using (a), we are able to give a derivation of the other implication, too:

(1)	$\alpha[\neg F \Rightarrow (F \Rightarrow G)]$	(axn0),(T0),(mp)
(2)	$\alpha[G \Rightarrow (F \Rightarrow G)]$	(axn0),(T0),(mp)
(3)	$\alpha[\neg F] \Rightarrow \alpha[F \Rightarrow G]$	(1),(T2),(mp)
(4)	$\alpha[G] \Rightarrow \alpha[F \Rightarrow G]$	(2),(T2),(mp)
(5)	$\neg\alpha[F] \Rightarrow \alpha[F \Rightarrow G]$	(T1),(3),(prop)
(6)	$(\alpha[F] \Rightarrow \alpha[G]) \Rightarrow \alpha[F \Rightarrow G]$	(4),(5),(prop)

4.

(1)	$\alpha[F \Rightarrow G]$	(axn0),(T0),(mp)
(2)	$\alpha[F] \Rightarrow \alpha[G]$	(1),(T2),(mp)

5. Since $\neg\neg F \Leftrightarrow F$ is tautological, this follows directly from (T4) and the definition of $a\langle\_\rangle$.

6.

(1)	$\neg\alpha[\neg F] \Leftrightarrow \neg(\alpha[F \Rightarrow \mathbf{false}])$	(T4),(prop)
(2)	$\alpha[F \Rightarrow \mathbf{false}] \Leftrightarrow (\alpha[F] \Rightarrow \alpha[\mathbf{false}])$	(T2),(T3)
(3)	$(\alpha[F] \Rightarrow \alpha[\mathbf{false}]) \Leftrightarrow (\neg\alpha[F] \vee \alpha[\mathbf{false}])$	(ax0)
(4)	$\alpha\langle F\rangle \Leftrightarrow (\alpha[F] \wedge \alpha\langle\mathbf{true}\rangle)$	(1),(2),(3),(prop)

7.

(1)	$\beta.\gamma\langle F\rangle \Rightarrow \beta[\gamma\langle F\rangle]$	(T1),(T4),(prop)
(2)	$\beta[\gamma\langle F\rangle] \Rightarrow \alpha.\beta[\gamma\langle F\rangle]$	(T0)
(3)	$\alpha.\beta\langle\mathbf{true}\rangle \wedge \alpha.\beta[\gamma\langle F\rangle] \Rightarrow \alpha.\beta.\gamma\langle F\rangle$	(T6),(prop)
(4)	$\alpha.\beta\langle\mathbf{true}\rangle \wedge \beta.\gamma\langle F\rangle \Rightarrow \alpha.\beta.\gamma\langle F\rangle$	(1),(2),(3),(prop)
(5)	$\alpha.\beta[\mathbf{false}] \Rightarrow \alpha.\beta[\gamma[\neg F]]$	(T4)

(6) $\alpha.\beta.\gamma\langle F\rangle \Rightarrow \alpha.\beta\langle \mathbf{true}\rangle$ (5),(prop),(T5)

(7) $\beta.\gamma[\neg F] \Rightarrow \alpha.\beta.\gamma[\neg F]$ (T0)

(8) $\alpha.\beta.\gamma\langle F\rangle \Rightarrow \beta.\gamma\langle F\rangle$ (7),(prop),(T5),(T4)

(9) $\alpha.\beta\langle \mathbf{true}\rangle \wedge \beta.\gamma\langle F\rangle \Leftrightarrow \alpha.\beta.\gamma\langle F\rangle$ (4),(6),(8),(prop)

8. (1) $a[c[\neg F] \Rightarrow b.c[\neg F]]$ (axn2)

(2) $a.c[\neg F] \Rightarrow a.b.c[\neg F]$ (1),(ax1),(mp)

(3) $a.b.c\langle F\rangle \Rightarrow a.c\langle F\rangle$ (2),(T5),(prop)

9. First we derive (T9a):

(1) $a[F \vee G] \Rightarrow a[\neg F \Rightarrow G]$ (T4)

(2) $a[\neg F \Rightarrow G] \Rightarrow (a[\neg F] \Rightarrow a[G])$ (ax1)

(3) $\neg a[F] \Rightarrow a[\neg F]$ (ax3)

(4) $a[\neg F \Rightarrow G] \Rightarrow (\neg a[F] \Rightarrow a[G])$ (2),(3),(prop)

(5) $a[F \vee G] \Rightarrow a[F] \vee a[G]$ (1),(2),(4),(prop)

(6) $a[F] \Rightarrow a[F \vee G]$ (T4)

(7) $a[G] \Rightarrow a[F \vee G]$ (T4)

(8) $a[F] \vee a[G] \Rightarrow a[F \vee G]$ (6),(7),(prop)

(9) $a[F \vee G] \Leftrightarrow a[F] \vee a[G]$ (5),(8),(prop)

This result helps us to derive (T9b):

(1) $a[F \wedge G] \Rightarrow a[F] \wedge a[G]$ (T4),(prop)

(2) $a[F] \wedge a[G] \Rightarrow (a\langle F\rangle \wedge a\langle G\rangle) \vee a[\mathbf{false}]$ (T6),(prop)

(3) $a\langle F\rangle \wedge a\langle G\rangle \Rightarrow \neg(a[\neg F] \vee a[\neg G])$ (ax0)

(4) $\neg(a[\neg F] \vee a[\neg G]) \Rightarrow \neg a[\neg F \vee \neg G]$ (T9a),(prop)

(5) $\neg a[\neg F \vee \neg G] \Rightarrow a[\neg(\neg F \vee \neg G)]$ (ax3)

(6) $a[\neg(\neg F \vee \neg G)] \Rightarrow a[F \wedge G]$ (T4)

(7) $(a\langle F\rangle \wedge a\langle G\rangle) \Rightarrow a[F \wedge G]$ (3),(4),(5),(6),(prop)

(8) $a[\mathbf{false}] \Rightarrow a[F \wedge G]$ (T4)

(9) $a[F \wedge G] \Leftrightarrow a[F] \wedge a[G]$ (1),(2),(7),(8),(prop)

Now we show that the generalisation rules (4.1) are derivable in Σ_{SL}.

Lemma 4.5 *Let F be a formula derivable in Σ_{SL}. Then $n[F]$ can also be derived in Σ_{SL}.*

Proof. Assume that $\vdash F$. The proof runs by induction on the assumed derivation of F.

Case: F is one of the axioms (ax0),...,(ax5). The axioms (axn0) - (axn2) correspond to (ax0) - (ax2). All the other derivations appear in the appendix.

Case: $F \equiv m[G]$ is one of the axioms (axn0), (axn1) and (axn2). Because of $\vdash m[G] \Rightarrow n[m[G]]$ by (ax2), we obtain $\vdash n[F]$ by (mp).

Case: $\vdash F$ is a conclusion of (mp) with premises $G \Rightarrow F$ and G, that is, we have $\vdash G \Rightarrow F$ and $\vdash G$. By the induction hypothesis follows $\vdash n[G \Rightarrow F]$ as well as $\vdash n[G]$. Hence, we can derive $n[F]$ as follows:

(1)	$n[G \Rightarrow F]$	I.H.
(2)	$n[G]$	I.H.
(3)	$n[G \Rightarrow F] \Rightarrow (n[G] \Rightarrow n[F])$	(ax1)
(4)	$n[G] \Rightarrow n[F]$	(mp),(3),(1)
(5)	$n[F]$	(mp),(4),(2)

■

Note that the rule is only correct without any premises, that is, it fails to hold

$$\frac{\mathcal{F} \vdash F}{\mathcal{F} \vdash n[F]}$$

as one can see by letting $\mathcal{F} = \{v\}$ and $F \equiv v$, for example. However, the result of lemma 4.5 can be generalised to

$$\frac{\mathcal{F} \vdash F}{n[\mathcal{F}] \vdash n[F]}$$

where $n[\mathcal{F}] = \{n[F] | F \in \mathcal{F}\}$ for a set $\mathcal{F}$ of formulas. An instance of this is a rule $n[F \Rightarrow G], n[F] \vdash n[G]$ corresponding to modus ponens.

Before turning to the completeness proof we show that Σ_{SL} is sound with respect to the semantics of pMLTL.

Lemma 4.6 (Soundness of $\Sigma_{\mathbf{SL}}$) *Let $\mathcal{F}$ be a set of formulas in $\mathcal{L}_{SL}$ and let F be a formula. If $\mathcal{F} \vdash F$, then $\mathcal{F} \models F$.*

Proof. The assertion is proven by induction on the derivation $\mathcal{F} \vdash F$. We only consider a few cases.

Case: $F \equiv a[G \Rightarrow H] \Rightarrow (a[G] \Rightarrow a[H])$ and $\mathcal{F} \vdash F$ by (ax1). Let $t = (\mathsf{N}_t, <_t)$ be a tree and λ a labelling such that $t, \lambda, \varepsilon \models a[G \Rightarrow H]$, that is either $a \notin \mathsf{N}_t$ or $t, \lambda, a \models G \Rightarrow H$. We have to show $t, \lambda, \varepsilon \models a[G] \Rightarrow a[H]$. Assume that $t, \lambda, \varepsilon \models a[G]$, i.e. that $a \notin \mathsf{N}_t$ or $t, \lambda, a \models G$. If $a \notin \mathsf{N}_t$, then $t, \lambda, \varepsilon \models a[H]$ by definition. If $a \in \mathsf{N}_t$, then $t, \lambda, a \models G \Rightarrow H$ and $t, \lambda, a \models G$. Hence, by definition we obtain $t, \lambda, a \models H$, that is $t, \lambda, \varepsilon \models a[H]$.

Case: $F \equiv a[G] \Rightarrow b[a[G]]$ and $\mathcal{F} \vdash F$ by (ax2). Let $t = (\mathsf{N}_t, <_t)$ be a tree and λ a labelling with $t, \lambda, \varepsilon \models a[G]$, i.e. $a \notin \mathsf{N}_t$ or $t, \lambda, a \models G$. We have to show that $t, \lambda, \varepsilon \models b[a[G]]$. Only the case that $a <_t b$ is interesting. Then it holds in particular $a \in \mathsf{N}_t$, hence we have $t, \lambda, a \models G$ by assumption and since $a <_t b$, we obtain $t, \lambda, b \models a[G]$ and hence $t, \lambda, \varepsilon \models b[a[G]]$.

Case: $F \equiv a[a[\mathbf{false}]]$ and $\mathcal{F} \vdash F$ by (ax4). Let t be an arbitrary tree and λ a labelling. If $a \notin \mathsf{N}_t$, then $t, \lambda, \varepsilon \models a[a[\mathbf{false}]]$ holds trivially.
If $a \in \mathsf{N}_t$, then $t, \lambda, \varepsilon \models a[a[\mathbf{false}]]$ holds again as it holds $t, \lambda, a \models a[\mathbf{false}]$ because of the uniqueness of the names.

Case: $F \equiv (a_1.b \wedge a_2.b) \Rightarrow (a_1.a_2 \vee a_2.a_1)$ with $a_1 \neq a_2$ and $\mathcal{F} \vdash F$ by (ax5). Let $t = (\mathsf{N}_t, <_t)$ be a tree and λ a labelling with $t, \lambda, \varepsilon \models a_1.b \wedge a_2.b$, that is, $b <_t a_1$ and $b <_t a_2$. Since the names in t are unique, $a_1 \neq a_2$ and since t is a tree it must hold either $a_2 <_t a_1$ or else $a_1 <_t a_2$, i.e. $t, \lambda, \varepsilon \models a_1.a_2 \vee a_2.a_1$. ■

For SL holds the following deduction theorem:

Theorem 4.7 (Deduction theorem) *For a set $\mathcal{F}$ of formulas and formulas F and G we have $\mathcal{F} \cup \{F\} \vdash G$ iff $\mathcal{F} \vdash F \Rightarrow G$.*

Proof. *Only if:* We prove the assertion by induction on the assumed derivation of $\mathcal{F} \cup \{F\} \vdash G$.

Case: If G is an axiom or $G \in \mathcal{F}$, then the claim holds obviously.

Case: If $G \equiv F$, then $F \Rightarrow G$ is tautological, hence the assertion follows by (ax0).

Case: If the last step in the derivation is an application of (mp) to some formulas $H \Rightarrow G$ and H with $\mathcal{F} \cup \{F\} \vdash H \Rightarrow G$ and $\mathcal{F} \cup \{F\} \vdash H$, then we obtain by induction hypothesis that $\mathcal{F} \vdash F \Rightarrow (H \Rightarrow G)$ as well as $\mathcal{F} \vdash F \Rightarrow H$ hold. From this it follows by propositional reasoning that $\mathcal{F} \vdash F \Rightarrow G$ holds.

If: Assume $\mathcal{F} \vdash F \Rightarrow G$. Then we also have $\mathcal{F} \cup \{F\} \vdash F \Rightarrow G$. On the other it holds trivially that $\mathcal{F} \cup \{F\} \vdash F$, hence, by (mp) it follows $\mathcal{F} \cup \{F\} \vdash G$. ■

Our goal is to show that Σ_{SL} is a complete proof system for SL, that is, that every valid SL-formula can be derived in Σ_{SL}. In order to prove this, we follow the classical way: we show that every finite and consistent set of formulas has a model.

In the present case this means to construct a finite tree $t = (\mathsf{N}_t, <_t)$ and assign to every node a set of propositional variables, that is, to find a mapping $\lambda : \mathsf{N}_t^{\varepsilon} \to 2^{\mathcal{V}}$, for a given finite and consistent set $\mathcal{F}$ of formulas, such that the resulting tree is a model of $\mathcal{F}$, that is, that $t, \lambda, \varepsilon \models \mathcal{F}$ holds. First we give an informal explanation of how the construction works.

In order to define the structure of the model we proceed as follows: for every set $\{a_1, \ldots, a_n\}$ of pairwise distinct names occurring in $\mathcal{F}$ we decide whether $a_1.\cdots.a_n\langle\mathbf{true}\rangle$ should or should not hold for our model – in other words, whether $a_n <_t \ldots <_t a_1$ holds for the tree that we want to construct – by completing $\mathcal{F}$ in the sense that for every such set $\{a_1, \ldots, a_n\}$ we add either $a_1.\cdots.a_n\langle\mathbf{true}\rangle$ or $a_1.\cdots.a_n[\mathbf{false}]$ to the set, paying attention that consistency is preserved. In order to decide which propositions should hold at a given node a of our model, we look, roughly speaking, at the formulas of the form $a[F]$ and decide for all sub-formulas G of F whether $a[G]$ or $\neg a[G]$ should hold, by "completing" the

set $\mathcal{F}$ by the corresponding sub-formulas. We assign to a node a exactly those propositions $v \in \mathcal{V}$ for which $a[v]$ belongs to the completed set.

Now we turn to the formalisation of this construction.

For every formula F we define a set $\tau(F)$ of formulas inductively as follows:

$$\begin{aligned} \tau(v) &= \{v\} \\ \tau(\neg F) &= \{\neg F\} \cup \tau(F) \\ \tau(F \Rightarrow G) &= \{F \Rightarrow G\} \cup \tau(F) \cup \tau(G) \\ \tau(m[F]) &= \{m[G] | G \in \tau(F)\} \cup \tau(F) \quad . \end{aligned}$$

For a set $\mathcal{F}$ of formulas $\tau(\mathcal{F})$ denotes the set $\bigcup_{F \in \mathcal{F}} \tau(F)$. For a formula F let $\mathsf{nm}(F)$ denote the set of names occurring in F. Formally, this set is defined as

$$\begin{aligned} &\mathsf{nm}(v) = \emptyset && \mathsf{nm}(\neg F) = \mathsf{nm}(F) \\ &\mathsf{nm}(F \Rightarrow G) = \mathsf{nm}(F) \cup \mathsf{nm}(G) && \mathsf{nm}(a[F]) = \{a\} \cup \mathsf{nm}(F) \quad . \end{aligned}$$

Accordingly, for a set $\mathcal{F}$ of formulas $\mathsf{nm}(\mathcal{F})$ denotes the set $\bigcup_{F \in \mathcal{F}} \mathsf{nm}(F)$.

Now we define the set $\kappa(\mathcal{F})$ as follows:

$$\kappa(\mathcal{F}) := \mathcal{F} \cup \{a_1.\cdots.a_n\langle \mathbf{true} \rangle | a_i \in \mathsf{nm}(\mathcal{F}), a_i \neq a_j \text{ for } i \neq j\} \quad .$$

This set will help us to decide which paths should be contained in the model.

Observe that neither of the mappings τ and κ produces new names, hence for every set $\mathcal{F}$ of formulas holds that

$$\mathsf{nm}(\mathcal{F}) = \mathsf{nm}(\tau\kappa(\mathcal{F})) \quad . \tag{4.2}$$

Definition 4.8 *Let $\mathcal{F}$ be a set of SL-formulas.*

1. *$\mathcal{F}$ is called* complete *iff for all $F \in \tau\kappa(\mathcal{F})$ either $F \in \mathcal{F}$ or $\neg F \in \mathcal{F}$ holds.*

2. *Let $\mathcal{F}$ be finite and consistent. A set $\mathcal{G}$ of SL-formulas is called a* completion *of $\mathcal{F}$ iff*

   (a) $\mathcal{F} \subseteq \mathcal{G}$,

   (b) $\mathcal{G}$ *is complete and consistent, and*

   (c) $\mathcal{G} \subseteq \tau\kappa(\mathcal{F}) \cup \neg\tau\kappa(\mathcal{F})$.

   where $\neg\mathcal{F} = \{\neg F | F \in \mathcal{F}\}$ for any set $\mathcal{F}$ of formulas.

Proposition 4.9 *Let $\mathcal{F}$ be a finite and consistent set of SL-formulas. Then $\mathcal{F}$ has at least one, and only finitely many completions.*

Proof. Observe first that for a consistent set $\mathcal{F}$ and $A \in \tau\kappa(\mathcal{F})$ either $\mathcal{F} \cup \{A\}$ or $\mathcal{F} \cup \{\neg A\}$ is consistent. As neither τ nor κ produce new names (cf. (4.2)), it also holds that $\kappa\tau\kappa(\mathcal{F}) = \tau\kappa(\mathcal{F})$. Further, it is easily shown by structural induction that the mapping τ is idempotent, thus it follows $\tau\kappa\tau\kappa(\mathcal{F}) = \tau\kappa(\mathcal{F})$. From these facts it follows that $\mathcal{F}$ has a completion.
Since for a finite set $\mathcal{F}$, the set $\tau\kappa(\mathcal{F})$ is also finite, it is clear that $\mathcal{F}$ has only finitely many completions. ■

Remark 4.10 *Let $\alpha, \beta \in \mathsf{N}^+$ denote non-empty sequences of names, F, G formulas and $\mathcal{F}$ a set of formulas. We make the following observations about the mapping τ:*

1. *If $\alpha[\neg F] \in \tau(\mathcal{F})$, then $\alpha[F] \in \tau(\mathcal{F})$.*
2. *If $\alpha[F \Rightarrow G] \in \tau(\mathcal{F})$, then $\alpha[F], \alpha[G] \in \tau(\mathcal{F})$.*
3. *If $\alpha.\beta[F] \in \tau(\mathcal{F})$, then $\beta[F] \in \tau(\mathcal{F})$.*

Proof. The first three claims follow immediately from:

1. If $\alpha[\neg F] \in \tau(H)$, then $\alpha[F] \in \tau(H)$.
2. If $\alpha[F \Rightarrow G] \in \tau(H)$, then $\alpha[F], \alpha[G] \in \tau(H)$.

3. If $\alpha.\beta[F] \in \tau(H)$, then $\beta[F] \in \tau(H)$.

Each of these assertions can be easily shown by induction on the structure of the formula H. ■

Proposition 4.11 *Let $\mathcal{F}$ be a finite, consistent and complete set of formulas, F, G formulas, $a, b, c \in \mathsf{N}$ names and $\alpha \in \mathsf{N}^*$ a finite sequence of names.*

1. *If $F \in \tau\kappa(\mathcal{F})$ and $\vdash \mathcal{F} \Rightarrow F$, then $F \in \mathcal{F}$.*
2. *If $a[F] \in \tau\kappa(\mathcal{F})$ and $a\langle\mathbf{true}\rangle \notin \mathcal{F}$, then $a[F] \in \mathcal{F}$.*
3. *If $a.b\langle\mathbf{true}\rangle \in \mathcal{F}$ and $b.c\langle\mathbf{true}\rangle \in \mathcal{F}$ then $a.c\langle\mathbf{true}\rangle \in \mathcal{F}$.*
4. *If $\alpha.a\langle\mathbf{true}\rangle \in \mathcal{F}$, then $a\langle\mathbf{true}\rangle \in \mathcal{F}$.*

Proof. *To 1:* The assertion follows immediately from the consistency and the completeness of $\mathcal{F}$.

To 2: Assume $a[F] \in \tau\kappa(\mathcal{F})$ and $a\langle\mathbf{true}\rangle \notin \mathcal{F}$. Since $a \in \mathsf{nm}(\mathcal{F})$ as stated in (4.2), it follows that $a[\mathbf{false}] \in \mathcal{F}$, hence $\vdash \mathcal{F} \Rightarrow a[F]$ by (T4). The assertion follows now by 1.

To 3: Assume that $a.b\langle\mathbf{true}\rangle, b.c\langle\mathbf{true}\rangle \in \mathcal{F}$. By using (T7) and (T8) we can conclude $\vdash \mathcal{F} \Rightarrow a.c\langle\mathbf{true}\rangle$. Since $\mathcal{F}$ is consistent, it holds $a \neq c$ by (ax4). Hence, $a.c\langle\mathbf{true}\rangle \in \tau\kappa(\mathcal{F})$. By 1. follows $a.c\langle\mathbf{true}\rangle \in \mathcal{F}$.

To 4: Assume $\alpha.a\langle\mathbf{true}\rangle \in \mathcal{F}$. Since $a \in \mathsf{nm}(\mathcal{F})$, it follows that $a\langle\mathbf{true}\rangle \in \tau\kappa(\mathcal{F})$. Furthermore, by (T0) we have $\vdash \mathcal{F} \Rightarrow a\langle\mathbf{true}\rangle$. By 1., we obtain $a\langle\mathbf{true}\rangle \in \mathcal{F}$. ■

Lemma 4.12 *Let $\mathcal{F}$ be a complete and consistent set of formulas, F, G formulas and $a \in \mathsf{N}$ a name.*

1. *If $F \Rightarrow G \in \tau\kappa(\mathcal{F})$, then $F \Rightarrow G \in \mathcal{F}$ iff $F \notin \mathcal{F}$ or $G \in \mathcal{F}$.*
2. *If $a[F \Rightarrow G] \in \tau\kappa(\mathcal{F})$, then $a[F \Rightarrow G] \in \mathcal{F}$ iff $a[F] \notin \mathcal{F}$ or $a[G] \in \mathcal{F}$.*

3. *If* $a.b[F] \in \tau\kappa(\mathcal{F})$, *then* $a.b[F] \in \mathcal{F}$ *iff* $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}$ *or* $b[F] \in \mathcal{F}$.

4. *If* $a[\neg F] \in \tau\kappa(\mathcal{F})$, *then* $a[\neg F] \in \mathcal{F}$ *iff* $a\langle\mathbf{true}\rangle \notin \mathcal{F}$ *or* $a[F] \notin \mathcal{F}$.

Proof. The proof of 1. is standard.

To 2:

If: Assume that $a[F] \notin \mathcal{F}$ or $a[G] \in \mathcal{F}$, which means $\neg a[F] \in \mathcal{F}$ or $a[G] \in \mathcal{F}$ (since $\mathcal{F}$ is complete and $a[F] \in \tau\kappa(\mathcal{F})$ by the definition of τ). In particular, it holds $\vdash \mathcal{F} \Rightarrow (a[F] \Rightarrow a[G])$. Due to (T3), this implies $\vdash \mathcal{F} \Rightarrow a[F \Rightarrow G]$. By prop. 4.11,1. we obtain the assertion.

Only if: This implication follows using (T2) and from the assumption that $\mathcal{F}$ is complete and consistent.

To 3:

If: We first consider the case that $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}$. There are two possibilities: either $a = b$ or $a \neq b$. In both cases follows $\vdash \mathcal{F} \Rightarrow a.b[\mathbf{false}]$ – in the first case by axiom (ax4), in the second case $a.b[\mathbf{false}] \in \mathcal{F}$ by the definition of a complete set of formulas and since $a, b \in \mathsf{nm}(\mathcal{F})$. Using (T4), we conclude $\vdash \mathcal{F} \Rightarrow a.b[F]$ and so by proposition 4.11,1. we obtain $a.b[F] \in \mathcal{F}$.
Now we assume that $b[F] \in \mathcal{F}$. By axiom (ax2) follows $\vdash \mathcal{F} \Rightarrow a.b[F]$, hence $a.b[F] \in \mathcal{F}$ by proposition 4.11,1.

Only if: Assume that $a.b[F] \in \mathcal{F}$ and that $a.b\langle\mathbf{true}\rangle \in \mathcal{F}$ hold. By (T6), we obtain $\vdash \mathcal{F} \Rightarrow a.b\langle F\rangle$. On the other hand we can conclude using (ax2) and (ax3) that $\vdash a.b\langle F\rangle \Rightarrow b[F]$, hence $\vdash \mathcal{F} \Rightarrow b[F]$. Since $b[F] \in \tau\kappa(\mathcal{F})$ by the definition of τ, the claim follows by proposition 4.11,1.

To 4:

If: If $a\langle\mathbf{true}\rangle \notin \mathcal{F}$, then $a[\mathbf{false}] \in \mathcal{F}$, as $a \in \mathsf{nm}(\mathcal{F})$ and since $\mathcal{F}$ is complete. Hence, by (T4) we conclude $\vdash \mathcal{F} \Rightarrow a[\neg F]$ and by proposition 4.11,1. we obtain $a[\neg F] \in \mathcal{F}$. So we assume $a[F] \notin \mathcal{F}$ which implies, since $a[F] \in \tau\kappa(\mathcal{F})$ by the definition of τ, that $\neg a[F] \in \mathcal{F}$. By axiom (ax3) we obtain $\vdash \mathcal{F} \Rightarrow a[\neg F]$, hence it follows that $a[\neg F] \in \mathcal{F}$ by proposition 4.11,1.

Only if: Assume $a[\neg F] \in \mathcal{F}$ and $a\langle\mathbf{true}\rangle \in \mathcal{F}$. It follows $\vdash \mathcal{F} \Rightarrow a\langle\neg F\rangle$ by (T6), hence $\vdash \mathcal{F} \Rightarrow \neg a[F]$ by (T5). As $\mathcal{F}$ is consistent, this implies $a[F] \notin \mathcal{F}$.■

Let $\mathcal{F}$ be a finite, consistent and complete set of formulas. Now we show how to construct a model (t, λ) of $\mathcal{F}$. Let $(\mathsf{N}_t, <_t)$ be defined as

$$\mathsf{N}_t := \{a \in \mathsf{N} | a\langle\mathbf{true}\rangle \in \mathcal{F}\} \quad .$$

and

$$a <_t b \quad \text{iff} \quad b.a\langle\mathbf{true}\rangle \in \mathcal{F}$$

(Note that $<_t$ is indeed a binary relation on N_t as from $b.a\langle\mathbf{true}\rangle \in \mathcal{F}$ it follows $a\langle\mathbf{true}\rangle, b\langle\mathbf{true}\rangle \in \mathcal{F}$.)

The assignment $\lambda : \mathsf{N}_t^{\varepsilon} \to 2^{\mathcal{V}}$ is defined by

$$\begin{aligned} \lambda(\varepsilon) &:= \mathcal{F} \cap \mathcal{V} \\ \lambda(a) &:= \{v \in \mathcal{V} | a[v] \in \mathcal{F}\} \quad . \end{aligned}$$

Before proving that (t, λ) is a model of $\mathcal{F}$, we show that t is a tree.

Lemma 4.13 *Let $\mathcal{F}$ be a finite, consistent and complete set of formulas and let $t = (\mathsf{N}_t, <_t)$ be defined as above. Then t is a tree.*

Proof. In order to show that t is a tree, we have to prove the following three properties:

1. The relation $<_t$ is irreflexive.
2. The relation $<_t$ is transitive.
3. For all $a, b, c \in \mathsf{N}_t^{\varepsilon}$: if $a \neq b$, $c <_t a$ and $c <_t b$, then either $a <_t b$ or $b <_t a$ holds.

To 1.: Since $\vdash \neg a.a\langle\mathbf{true}\rangle$ by (ax4) and because of the consistency of $\mathcal{F}$ we obtain $a.a\langle\mathbf{true}\rangle \notin \mathcal{F}$ for all $a \in \mathsf{N}_t$, hence $a \not<_t a$ for all $a \in \mathsf{N}_t$. By the definition of $<_t$, it obviously holds that $\varepsilon \not<_t \varepsilon$.

To 2.: Let $a <_t b$ and $b <_t c$. It follows that $a, b \neq \varepsilon$. If $c = \varepsilon$, then the relation $a <_t c$ follows immediately from the definition of $<_t$. So let $a, b, c \neq \varepsilon$. It follows by the definition of $<_t$ that $b.a\langle\mathbf{true}\rangle \in \mathcal{F}$ and $c.b\langle\mathbf{true}\rangle \in \mathcal{F}$. By proposition 4.11,3. we obtain that $c.a\langle\mathbf{true}\rangle \in \mathcal{F}$, that is, $a <_t c$.

To 3.: The case if $a = \varepsilon$ or $b = \varepsilon$ is trivial. So we assume that $a, b \in \mathsf{N}_t$. By the definition of $<_t$ we know that $a.c\langle\mathbf{true}\rangle \in \mathcal{F}$ and $b.c\langle\mathbf{true}\rangle \in \mathcal{F}$. Assume that $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}$ and $b.a\langle\mathbf{true}\rangle \notin \mathcal{F}$, i.e. (since $a \neq b$ by assumption) $a.b[\mathbf{false}] \in \mathcal{F}$ and $b.a[\mathbf{false}] \in \mathcal{F}$ because of the completeness of $\mathcal{F}$. Together with (ax5) this produces a contradiction to the consistency of $\mathcal{F}$. ■

Theorem 4.14 *Let $\mathcal{F}$ and (t, λ) be defined as above. Then for all names $a \in \mathsf{N}$ and all SL-formulas F it holds the following:*

1. *If $F \in \tau\kappa(\mathcal{F})$, then $F \in \mathcal{F}$ iff $t, \lambda, \varepsilon \models F$.*
2. *If $a[F] \in \tau\kappa(\mathcal{F})$, then $a[F] \in \mathcal{F}$ iff $a \notin \mathsf{N}_t$ or $t, \lambda, a \models F$.*

Proof. We prove the two assertions simultaneously by induction on F.

Case: $v \in \mathcal{V}$.

1. The assertion follows immediately from the definition of (t, λ).
2. This follows from the definition of (t, λ) and from proposition 4.11,2.

Case: $F \Rightarrow G$.

1. Let $F \Rightarrow G \in \tau\kappa(\mathcal{F})$.

   By lemma 4.12,1., $F \Rightarrow G \in \mathcal{F}$ is equivalent with $F \notin \mathcal{F}$ or $G \in \mathcal{F}$. As $F, G \in \tau\kappa(\mathcal{F})$, we can use the induction hypothesis and conclude that this is equivalent with ($t, \lambda, \varepsilon \not\models F$ or $t, \lambda, \varepsilon \models G$), hence, with $t, \lambda, \varepsilon \models F \Rightarrow G$.
2. By lemma 4.12,2. we have $a[F \Rightarrow G] \in \mathcal{F}$ iff $a[F] \notin \mathcal{F}$ or $a[G] \in \mathcal{F}$. Hence, the induction hypothesis for $a[F]$ and $a[G]$ implies the assertion.

Case: $\neg F$.

1. Let $\neg F \in \tau\kappa(\mathcal{F})$.

   Since $\mathcal{F}$ is complete, $\neg F \in \mathcal{F}$ iff $F \notin \mathcal{F}$. By the induction hypothesis, this is equivalent with $t, \lambda, \varepsilon \not\models F$, that is, with $t, \lambda, \varepsilon \models \neg F$.

2. Let $a[\neg F] \in \tau\kappa(\mathcal{F})$.

   By lemma 4.12,4. and by the definition of t, it holds $a[\neg F] \in \mathcal{F}$ iff $a \notin \mathsf{N}_t$ or $a[F] \notin \mathcal{F}$. By induction hypothesis, this is equivalent with $a \notin \mathsf{N}_t$ or $t, \lambda, a \not\models F$, that is, with $a \notin \mathsf{N}_t$ or $t, \lambda, a \models \neg F$.

Case: $b[F]$.

1. Let $b[F] \in \tau\kappa(\mathcal{F})$.

   By the induction hypothesis for 2., $b[F] \in \mathcal{F}$ iff $b \notin \mathsf{N}_t$ or $t, \lambda, b \models F$. By definition, this is equivalent with $t, \lambda, \varepsilon \models b[F]$.

2. Let $a[b[F]] \in \tau\kappa(\mathcal{F})$.

   Only if: Assume that $a[b[F]] \in \mathcal{F}$ and that $a \in \mathsf{N}_t$. We have to show $t, \lambda, a \models b[F]$, that is, $b \not<_t a$ or $t, \lambda, b \models F$. Assume $b <_t a$, that is, $a.b\langle\mathbf{true}\rangle \in \mathcal{F}$. By lemma 4.12,3., it follows that $b[F] \in \mathcal{F}$. By induction hypothesis, this implies $t, \lambda, b \models F$, as we have $b \in \mathsf{N}_t$ by assumption.

   If: If $a \notin \mathsf{N}_t$, then $a\langle\mathbf{true}\rangle \notin \mathcal{F}$ by definition. Hence, $a[b[F]] \in \mathcal{F}$ by proposition 4.11,2. So assume $a \in \mathsf{N}_t$ and $t, \lambda, a \models b[F]$.

   Case: $b \not<_t a$. Then by the definition of $<_t$ it holds $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}$, hence it follows $a[b[F]] \in \mathcal{F}$ by lemma 4.12,3.
   Case: $b <_t a$ and $t, \lambda, b \models F$. Since $b[F] \in \tau\kappa(\mathcal{F})$, by induction hypothesis we obtain $b[F] \in \mathcal{F}$ and therefore $a[b[F]] \in \mathcal{F}$ by lemma 4.12,3. ■

With the aid of this theorem we can prove the weak completeness of Σ_{SL} in the usual way:

Theorem 4.15 *Let F be an SL-formula. If $\models F$, then $\vdash F$.*

Proof. Assume that $\nvdash F$. Then it holds also $\nvdash \neg\neg F$, that is, the set $\{\neg F\}$ is consistent. By theorem 4.14,1., there is a model (t, λ) with $t, \lambda, \varepsilon \models \neg F$, hence $t, \lambda, \varepsilon \not\models F$ and this proves $\not\models F$. ■

Note that strong completeness does not hold for SL. In order to see why this is the case, consider the following infinite set of formulas: $\mathcal{F} := \{a\langle\mathbf{true}\rangle | a \in \mathsf{N}\}$. Since the models of the logic are finite trees, it is clear that $\mathcal{F}$ can not have any model, that is, that $\mathcal{F} \models \mathbf{false}$ holds. On the other hand it is easy to show that we can not derive **false** from $\mathcal{F}$: assume $\mathcal{F} \vdash \mathbf{false}$. Then there is a finite subset $\{a_1\langle\mathbf{true}\rangle, \ldots, a_n\langle\mathbf{true}\rangle\}$ of $\mathcal{F}$ such that $a_1\langle\mathbf{true}\rangle, \ldots, a_n\langle\mathbf{true}\rangle \vdash \mathbf{false}$ holds. By the deduction theorem (thm. 4.7), this implies $\vdash \neg(a_1\langle\mathbf{true}\rangle \wedge \ldots \wedge a_n\langle\mathbf{true}\rangle)$. However, since the formula $a_1\langle\mathbf{true}\rangle \wedge \ldots \wedge a_n\langle\mathbf{true}\rangle$ obviously has a model, this is a contradiction to the soundness of Σ_{SL}.

4.2 Axiomatisation of propositional MLTL

4.2.1 The proof system Σ_{MLTL^-}

In sec. 4.1 we have introduced the proof system Σ_{SL} and proved it to be sound and complete with respect to the spatial part of propositional MLTL. Now we are going to extend this system in order to obtain a complete axiomatisation for pMLTL. First we only consider the logic pMLTL without the operators keep_m and define a proof system that we will call Σ_{MLTL^-}. Later on in this chapter we will extend this system to a system Σ_{MLTL} which will provide axioms also for the "keep"-operators. The axioms and rules of Σ_{MLTL^-} are collected in figure 4.2. Note that all the formulas (T0) - (T9b) can still be used as Σ_{MLTL^-} extends Σ_{SL}.

Intuitively, (ax6) means that time is linear ("$\Rightarrow$") and infinite ("$\Leftarrow$"). Axiom (ax7) is the usual K-axiom of modal logics, (ax8) and (axn8) are the fix-point characterisations of the always operator.

Note that the "boxed" version of the axiom (ax7) and $n[\neg\bigcirc F \Rightarrow \bigcirc\neg F]$ (boxed version of one direction of (ax6)) can be derived in Σ_{MLTL^-}. The derivations are given in the appendix.

(ax0) $\vdash F$ if F tautological (axn0) $\vdash n[F]$ if F tautological

(ax1) $\vdash a[F \Rightarrow G] \Rightarrow a[F] \Rightarrow a[G]$ (axn1) $\vdash n[a[F \Rightarrow G] \Rightarrow a[F] \Rightarrow a[G]]$

(ax2) $\vdash a[F] \Rightarrow b[a[F]]$ (axn2) $\vdash n[a[F] \Rightarrow b[a[F]]]$

(ax3) $\vdash \neg a[F] \Rightarrow a[\neg F]$

(ax4) $\vdash a[a[\mathbf{false}]]$

(ax5) $\vdash (a_1.b \wedge a_2.b) \Rightarrow (a_1.a_2 \vee a_2.a_1)$ (for $a_1 \neq a_2$)

(ax6) $\vdash \neg\circ F \Leftrightarrow \circ\neg F$

(ax7) $\vdash \circ(F \Rightarrow G) \Rightarrow (\circ F \Rightarrow \circ G)$

(ax8) $\vdash \Box F \Rightarrow F \wedge \circ\Box F$ (axn8) $\vdash n[\Box F \Leftrightarrow F \wedge \circ\Box F]$

(ax9) $\vdash \neg m[\circ F] \Rightarrow \circ\neg m[F]$

(ax10) $\vdash \neg m[\neg\circ F] \Rightarrow \circ m[F]$

(ax11) $\vdash \Box m[F] \Rightarrow m[\Box F]$

(mp) $\dfrac{\vdash F \Rightarrow G \qquad \vdash F}{\vdash G}$ (nex) $\dfrac{\vdash F}{\vdash \circ F}$

(ind) $\dfrac{\vdash F \Rightarrow \circ F \qquad \vdash F \Rightarrow G}{\vdash F \Rightarrow \Box G}$

Figure 4.2: The proof system Σ_{MLTL^-}

As already mentioned in the previous section, the generalisation rules of the form

$$\frac{\vdash F}{\vdash n[F]}$$

are not sound with respect to the semantics of pMLTL. One counterexample is (ax6), which is, as we will show in the soundness theorem, a valid pMLTL-formula, but $n[\circ\neg F \Rightarrow \neg\circ F]$ is not. For a counterexample, consider a run σ as given in fig. 4.3.

As the name n does not appear in the second tree, it holds $\sigma, n \models \circ\neg F$ as well as $\sigma, n \models \circ F$ for an arbitrary formula F. As the latter just means $\sigma, n \not\models \neg\circ F$, we have $\sigma, n \not\models \circ\neg F \Rightarrow \neg\circ F$. Since node n occurs in the first tree, this means $\sigma, \varepsilon \not\models n[\circ\neg F \Rightarrow \neg\circ F]$.

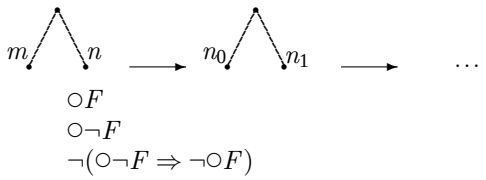


Figure 4.3: Counterexample to the generalisation rule

We list some useful theorems of pMLTL. Their derivations are collected in the appendix.

(T10) $\vdash \bigcirc(F \wedge G) \Leftrightarrow (\bigcirc F \wedge \bigcirc G)$ (T11) $\vdash F \wedge \bigcirc\Box F \Rightarrow \Box F$

(T12) $\vdash n[\Box F] \Leftrightarrow (n[F] \wedge n[\bigcirc\Box F])$ (alw) $\dfrac{\vdash F}{\vdash \Box F}$

For Σ_{MLTL^-} we have the following deduction theorem:

Theorem 4.16 (Deduction Theorem) *Let $\mathcal{F}$ be a set of formulas and F, G formulas. Then $\mathcal{F} \cup \{F\} \vdash G$ if and only if $\mathcal{F} \vdash \Box F \Rightarrow G$.*

Proof. *If:* Assume that $\mathcal{F} \vdash \Box F \Rightarrow G$. Then it hold $\mathcal{F} \cup \{F\} \vdash \Box F \Rightarrow G$ and $\mathcal{F} \cup \{F\} \vdash F$. By the derived rule (alw) it follows that $\mathcal{F} \cup \{F\} \vdash \Box F$, hence $\mathcal{F} \cup \{F\} \vdash G$ by (mp).

Only if: The assertion is shown by induction on the assumed derivation of G from assumptions from the set $\mathcal{F} \cup \{F\}$.

Case: $G \in \mathcal{F}$ or G is one of the axioms of Σ_{MLTL^-}. Then we have $\mathcal{F} \vdash G$ and $\mathcal{F} \vdash G \Rightarrow (\Box F \Rightarrow G)$ by (ax0), hence $\mathcal{F} \vdash \Box F \Rightarrow G$ by (mp).

Case: $G \equiv F$. Then $\mathcal{F} \vdash \Box G \Rightarrow G$ by (ax8) and (prop).

Case: G is a conclusion of (mp) with premises $H \Rightarrow G$ and H, that is, we have $\mathcal{F} \cup \{F\} \vdash H \Rightarrow G$ and $\mathcal{F} \cup \{F\} \vdash H$. Since these derivations are shorter, the induction hypothesis can be applied and we conclude $\mathcal{F} \vdash \Box F \Rightarrow (H \Rightarrow G)$ and $\mathcal{F} \vdash \Box F \Rightarrow H$. By (prop), we obtain $\mathcal{F} \vdash \Box F \Rightarrow G$.

Case: $G \equiv \bigcirc H$ is a conclusion of (nex) with premise H, i.e. $\mathcal{F} \cup \{F\} \vdash H$, hence by the induction hypothesis it follows $\mathcal{F} \vdash \Box F \Rightarrow H$. Now we give a

derivation of $\Box F \Rightarrow \bigcirc H$ from this:

(1)	$\Box F \Rightarrow H$	assumption
(2)	$\bigcirc(\Box F \Rightarrow H)$	(nex),(1)
(3)	$\bigcirc\Box F \Rightarrow \bigcirc H$	(ax7),(2),(mp)
(4)	$\Box F \Rightarrow \bigcirc\Box F$	(ax8),(prop)
(5)	$\Box F \Rightarrow \bigcirc H$	(3),(4),(prop)

Case: $G \equiv G_1 \Rightarrow \Box G_2$ is a conclusion of (ind) with premises $G_1 \Rightarrow G_2$ and $G_1 \Rightarrow \bigcirc G_1$. Then $\mathcal{F} \vdash \Box F \Rightarrow (G_1 \Rightarrow G_2)$ and $\mathcal{F} \vdash \Box F \Rightarrow (G_1 \Rightarrow \bigcirc G_1)$ by induction hypothesis. We give a derivation of $\Box F \Rightarrow (G_1 \Rightarrow \Box G_2)$:

(1)	$\Box F \Rightarrow (G_1 \Rightarrow G_2)$	assumption
(2)	$\Box F \Rightarrow (G_1 \Rightarrow \bigcirc G_1)$	assumption
(3)	$\Box F \wedge G_1 \Rightarrow G_2$	(1),(prop)
(4)	$\Box F \wedge G_1 \Rightarrow \bigcirc G_1$	(2),(prop)
(5)	$\Box F \Rightarrow \bigcirc\Box F$	(ax8),(prop)
(6)	$\Box F \wedge G_1 \Rightarrow \bigcirc\Box F \wedge \bigcirc G_1$	(4),(5),(prop)
(7)	$\bigcirc\Box F \wedge \bigcirc G_1 \Rightarrow \bigcirc(\Box F \wedge G_1)$	(T10),(prop)
(8)	$\Box F \wedge G_1 \Rightarrow \bigcirc(\Box F \wedge G_1)$	(6),(7),(prop)
(9)	$\Box F \wedge G_1 \Rightarrow \Box G_2$	(ind),(3),(8)
(10)	$\Box F \Rightarrow (G_1 \Rightarrow \Box G_2)$	(prop),(9) ■

With the aid of the deduction theorem, the following theorems can be derived. For the proofs we refer again to the appendix.

(T13) $\vdash \Box(F \Rightarrow G) \Rightarrow (\Box F \Rightarrow \Box G)$

(T14) $\vdash \Box(F \wedge G) \Leftrightarrow \Box F \wedge \Box G$

(T15) $\vdash \bigcirc\Box F \Leftrightarrow \Box\bigcirc F$

Now we show that all axioms and rules of Σ_{MLTL^-} are sound with respect to the semantics of pMLTL.

Lemma 4.17 (Soundness) *Let F be a formula and $\mathcal{F}$ a set of formulas such that $\mathcal{F} \vdash F$. Then $\mathcal{F} \models F$.*

Proof. As usual, soundness is proved by induction on the derivation of F from $\mathcal{F}$. We only consider a few cases.

Case: $F \equiv n[\Box G \Leftrightarrow G \wedge \circ\Box G]$ and $\mathcal{F} \vdash F$ by (axn8). Let σ be a run and $n \in \mathsf{N}_0$. We have to show $\sigma, n \models \Box G \Leftrightarrow G \wedge \circ\Box G$.
"$\Rightarrow$": Assume $\sigma, n \models \Box G$, i.e. for all $i \geq 0$ holds that either there is a $k \leq i$ with $n \notin \mathsf{N}_k$ or $\sigma|_i, n \models G$. Since $n \in \mathsf{N}_0$ it follows in particular $\sigma, n \models G$. If $n \notin \mathsf{N}_1$, then $\sigma, n \models \circ\Box G$ by the definition of the semantics of the next-time operator. If $n \in \mathsf{N}_1$ then $\sigma|_1, n \models \Box G$ by the definition of the semantics of $\Box$.
"$\Leftarrow$": Assume that $\sigma, n \models G \wedge \circ\Box G$, that is, $\sigma, n \models G$ and either $n \notin \mathsf{N}_1$ or $\sigma|_1, n \models \Box G$. By the semantics of $\Box$ follows $\sigma, n \models \Box G$.

Case: $F \equiv \neg m[\circ G] \Rightarrow \circ\neg m[G]$ and $\mathcal{F} \vdash F$ by (ax9). Let σ be a run for which it holds $\sigma, \varepsilon \models \neg m[\circ G]$, that is, $m \in \mathsf{N}_0$ and $\sigma, m \not\models \circ G$, which implies $m \in \mathsf{N}_1$ and $\sigma|_1, m \models \neg G$. By definition, this is equivalent with $\sigma, \varepsilon \models \circ\neg m[G]$.

Case: $F \equiv \neg m[\neg\circ G] \Rightarrow \circ m[G]$ and $\mathcal{F} \vdash F$ by (ax10). Let σ be a run with and $\sigma, \varepsilon \models \neg m[\neg\circ G]$, that is, $m \in \mathsf{N}_0$ and $\sigma, m \models \circ G$, that is, $m \in \mathsf{N}_0$ and $m \notin \mathsf{N}_1$ or $\sigma|_1, m \models G$. The latter means exactly $\sigma|_1, \varepsilon \models m[G]$, hence it follows that $\sigma, \varepsilon \models \circ m[G]$.

Case: $F \equiv \Box m[G] \Rightarrow m[\Box G]$ and $\mathcal{F} \vdash F$ by (ax11). Let σ be a run with $\sigma, \varepsilon \models \Box m[G]$, that is, for all $i \in \omega$ it holds $\sigma|_i, \varepsilon \models m[G]$, that is, $\sigma|_i, m \models G$ for all $i \in \omega$ with $m \in \mathsf{N}_i$. This implies $\sigma, m \models \Box G$ by the semantics of $\Box$ and so we have $\sigma, \varepsilon \models m[\Box G]$. ■

In order to prove that Σ_{MLTL^-} is a complete axiomatisation of pMLTL without the "keep"-operators we follow again the traditional method of constructing a model for a finite and consistent set of formulas. We first have to extend the mapping τ of the previous section to the formulas of the whole logic. (Here we also consider the operators keep_m.)

For every formula F we define a set $\tau(F)$ of formulas inductively as given in figure 4.4.

$$\begin{aligned}
\tau(v) &= \{v\} \\
\tau(\neg F) &= \{\neg F\} \cup \tau(F) \\
\tau(F \Rightarrow G) &= \{F \Rightarrow G\} \cup \tau(F) \cup \tau(G) \\
\tau(m[F]) &= \{m[G] | G \in \tau(F)\} \cup \tau(F) \\
\tau(\mathsf{keep}_m) &= \{\mathsf{keep}_m\} \\
\tau(\circ F) &= \{\circ F\} \\
\tau(\Box F) &= \{\Box F\} \cup \tau(F)
\end{aligned}$$

Figure 4.4: Definition of τ

For a set $\mathcal{F}$ of formulas, the set $\kappa(\mathcal{F})$ is defined in the same way as in the case of the spatial logic, that is

$$\kappa(\mathcal{F}) := \mathcal{F} \cup \{a_1.\cdots.a_n\langle\mathbf{true}\rangle | a_i \in \mathsf{nm}(\mathcal{F}), a_i \neq a_j \text{ for } i \neq j\}$$

where the function nm is extended to pMLTL as expected:

$$\mathsf{nm}(\circ F) = \mathsf{nm}(\Box F) = \mathsf{nm}(F) \qquad \mathsf{nm}(\mathsf{keep}_m) = \{m\} \quad .$$

The notions of complete set of formulas and completion are defined exactly as for SL. In the same way as for SL it can be shown that every finite and consistent set of formulas has a completion and that it has only finitely many different completions. All the propositions and lemmas proved in sec. 4.1 hold also for Σ_{MLTL^-} and for τ extended to formulas of pMLTL.

Lemma 4.18 *Let $\mathcal{F}$ be a finite set of formulas and let $\mathcal{F}_1, \ldots, \mathcal{F}_n$ be all its different completions. Then $\vdash \mathcal{F} \Rightarrow \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n$.*

Proof. The assertion can be shown by simple propositional reasoning. One proof can be found in [29], there for LTL. ■

For a set $\mathcal{F}$ of formulas, the set $\theta(\mathcal{F})$ is defined by the following:

$$\theta(\mathcal{F}) = \bigcup_{i=1}^{8} \theta_i(\mathcal{F})$$

where θ_i is defined as given in figure 4.5.

$\theta_1(\mathcal{F}) = \{F | \circ F \in \mathcal{F}\}$ $\quad$ $\theta_2(\mathcal{F}) = \{\neg F | \neg\circ F \in \mathcal{F}\}$

$\theta_3(\mathcal{F}) = \{\Box F | \Box F \in \mathcal{F}\}$ $\quad$ $\theta_4(\mathcal{F}) = \{\neg\Box F | \neg\Box F \in \mathcal{F} \text{ and } F \in \mathcal{F}\}$

$\theta_5(\mathcal{F}) = \{a[F] | a[\circ F], a\langle\textbf{true}\rangle \in \mathcal{F}\}$ $\quad$ $\theta_6(\mathcal{F}) = \{\neg a[F] | \neg a[\circ F] \in \mathcal{F}\}$

$\theta_7(\mathcal{F}) = \{a[\Box F] | a[\Box F], a\langle\textbf{true}\rangle \in \mathcal{F}\}$

$\theta_8(\mathcal{F}) = \{\neg a[\Box F] | \neg a[\Box F], a[F] \in \mathcal{F}\}$

Figure 4.5: Definition of $\theta_1, \ldots, \theta_8$

The mapping θ has the following properties:

Proposition 4.19 *Let $\mathcal{F}$ be a finite set of formulas of pMLTL.*

1. $\vdash \mathcal{F} \Rightarrow \circ\theta(\mathcal{F})$.
2. *If $\mathcal{F}$ is consistent, then so is $\theta(\mathcal{F})$.*

Proof.

1. Since $\mathcal{F}$ is finite and since $\vdash \circ(F \wedge G) \Leftrightarrow (\circ F \wedge \circ G)$ by (T10), it is enough to show that $\vdash \mathcal{F} \Rightarrow \circ F$ for every $F \in \theta(\mathcal{F})$. To prove this we have to distinguish the different cases in the definition of $\theta(\mathcal{F})$.
   - $F \in \theta_1(\mathcal{F})$: by definition we have $\circ F \in \mathcal{F}$, hence the assertion follows by (ax0).
   - $F \equiv \neg G \in \theta_2(\mathcal{F})$: then $\neg\circ G \in \mathcal{F}$ by definition, so the assertion follows by (ax6).
   - $F \equiv \Box G \in \theta_3(\mathcal{F})$: by definition we have $\Box G \in \mathcal{F}$. By (ax8) and (prop) this implies $\vdash \mathcal{F} \Rightarrow \circ\Box G$.
   - $F \equiv \neg\Box G \in \theta_4(\mathcal{F})$: by the definition of θ_4 we have $\vdash \mathcal{F} \Rightarrow \neg\Box G \wedge G$, hence the assertion follows from $\vdash \Box G \Leftrightarrow G \wedge \circ\Box G$, (ax6) and propositional logic.

- $F \equiv a[G] \in \theta_5(\mathcal{F})$: by definition we have $\vdash \mathcal{F} \Rightarrow a[\circ G] \wedge a\langle \mathbf{true} \rangle$. (T6) implies $\vdash \mathcal{F} \Rightarrow a\langle \circ G \rangle$, so the assertion follows by (ax10).
- $F \equiv \neg a[G] \in \theta_6(\mathcal{F})$: By the definition of θ_6 we have $\neg a[\circ G] \in \mathcal{F}$. The assertion follows immediately from (ax9).
- $F \equiv a[\Box G] \in \theta_7(\mathcal{F})$: by (T6) we have $\vdash \mathcal{F} \Rightarrow a\langle \Box G \rangle$, hence the assertion follows by $\vdash a\langle \Box G \rangle \Leftrightarrow (a\langle G \rangle \wedge a\langle \circ \Box G \rangle)$ (cf. (T6)) and (ax10).
- $F \equiv \neg a[\Box G] \in \theta_8(\mathcal{F})$: by definition we have $\neg a[\Box G], a[G] \in \mathcal{F}$. Since $\vdash a[\Box G] \Leftrightarrow a[G] \wedge a[\circ \Box G]$ by (T12), we obtain by propositional reasoning $\vdash \mathcal{F} \Rightarrow \neg a[\circ \Box G]$. The assertion follows now by (ax9).

2. Assume that $\theta(\mathcal{F})$ is not consistent, that is, $\vdash \neg\theta(\mathcal{F})$. With (nex) we obtain $\vdash \circ\neg\theta(\mathcal{F})$, hence with (ax6) $\vdash \neg\circ\theta(\mathcal{F})$. By 1. we have $\vdash \neg\mathcal{F}$, that is, $\mathcal{F}$ is inconsistent. ■

Given a finite and consistent set $\mathcal{F}$ of formulas, let the graph $\mathcal{T}(\mathcal{F})$ be defined as follows:

- The roots of $\mathcal{T}(\mathcal{F})$ are the different completions of $\mathcal{F}$.
- If $\mathcal{G}$ is a node of $\mathcal{T}(\mathcal{F})$, then its successors are the completions of $\theta(\mathcal{G})$.

Obviously, the nodes of the graph $\mathcal{T}(\mathcal{F})$ are finite, consistent and complete sets of formulas. Note that the sub-graph of $\mathcal{T}(\mathcal{F})$ that consists of all nodes reachable from the successors of a node $\mathcal{G}$ is exactly the same as $\mathcal{T}(\theta(\mathcal{G}))$.

Lemma 4.20 *Assume that $\mathcal{F}$ is a finite and consistent set of formulas.*

1. *The graph $\mathcal{T}(\mathcal{F})$ has only finitely many different nodes.*
2. *Assume that $\mathcal{F}_1, \ldots, \mathcal{F}_n$ are all different nodes of $\mathcal{T}(\mathcal{F})$.*
   (a) $\vdash \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n \Rightarrow \circ(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$.
   (b) $\vdash \mathcal{F} \Rightarrow \Box(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$.

Proof.

1. The completions of a finite set $\mathcal{G}$ are subsets of $\tau\kappa(\mathcal{G})$ which is a finite set. Further, if F is a formula in $\theta(\mathcal{G})$ that does not occur in $\mathcal{G}$, then there are the following possibilities: $\circ F \in \mathcal{G}$, $F \equiv \neg G$ and $\neg\circ G \in \mathcal{G}$, $F \equiv a[G]$ and $a[\circ G] \in \mathcal{G}$ or $F \equiv \neg a[G]$ and $\neg a[\circ G] \in \mathcal{G}$. In each of these cases holds that the number of the next-time operators decreases which is only possible finitely many times. Hence, there are only finitely many formulas that occur in the graph $\mathcal{T}(\mathcal{G})$ and $\mathcal{T}(\mathcal{G})$ can contain only finitely many different nodes.

2. (a) Let $i \in \{1,\ldots,n\}$. Since all completions of $\theta(\mathcal{F}_i)$ are among the sets $\mathcal{F}_1,\ldots,\mathcal{F}_n$, we obtain by lemma 4.18 and by simple propositional reasoning that $\vdash \theta(\mathcal{F}_i) \Rightarrow \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n$. Using (nex), (ax7) and (mp) it follows that $\vdash \circ\theta(\mathcal{F}_i) \Rightarrow \circ(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$. On the other hand, by proposition 4.19 we have $\vdash \mathcal{F}_i \Rightarrow \circ\theta(\mathcal{F}_i)$ for $i \in \{1,\ldots,n\}$. Altogether, we obtain $\vdash \mathcal{F}_i \Rightarrow \circ(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$ for every $i \in \{1,\ldots,n\}$, hence $\vdash \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n \Rightarrow \circ(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$ by (prop).

   (b) By (ind) and (a) we have $\vdash \mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n \Rightarrow \Box(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$. Since the completions of $\mathcal{F}$ are among $\mathcal{F}_1,\ldots,\mathcal{F}_n$, the assertion follows with lemma 4.18. ■

A path $\mathcal{F}_1, \mathcal{F}_2, \ldots$ in the graph $\mathcal{T}(\mathcal{F})$ is called *complete* iff it satisfies the following conditions:

- If $\neg\Box F \in \mathcal{F}_i$, then $\neg F \in \mathcal{F}_j$ for some $j \geq i$.

- If $\neg a[\Box F] \in \mathcal{F}_i$, then there is a $j \geq i$ s.t. $\neg a[F] \in \mathcal{F}_j$ and for all $k \in \{i,\ldots,j\}$ holds $a\langle\textbf{true}\rangle \in \mathcal{F}_k$.

Lemma 4.21 *Let $\mathcal{F}$ be a finite and consistent set of formulas. Then $\mathcal{T}(\mathcal{F})$ contains a complete path starting at some root.*

Proof. We first prove that for every node $\mathcal{G}$ in $\mathcal{T}(\mathcal{F})$ and every formula F with $\{\neg\Box F, F\} \subseteq \mathcal{G}$ there is a node $\mathcal{H}$ in $\mathcal{T}(\mathcal{F})$ accessible from $\mathcal{G}$ such that $\neg F \in \mathcal{H}$. Suppose that this is not the case. Since $\theta(\mathcal{G})$ contains $\neg\Box F$ by definition and since $F \in \tau\kappa\theta(\mathcal{G})$, it follows in particular that $F \in \mathcal{H}$ for every successor $\mathcal{H}$ of $\mathcal{G}$, hence $\{\neg\Box F, F\} \subseteq \mathcal{H}$ for every successor $\mathcal{H}$ of $\mathcal{G}$. Inductively, it follows that $\{\neg\Box F, F\} \subseteq \mathcal{H}$ for every node accessible from $\mathcal{G}$. Let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ denote all nodes accessible from $\mathcal{G}$ – that is, all nodes of the graph $\mathcal{T}(\theta(\mathcal{G}))$ – and let $\mathfrak{I}$ denote the formula $\mathcal{G}_1 \vee \ldots \vee \mathcal{G}_n$. By (ax0) holds obviously $\vdash \mathfrak{I} \Rightarrow F$, hence $\vdash \Box\mathfrak{I} \Rightarrow \Box F$ by (alw) and (T13). Since $\vdash \theta(\mathcal{G}) \Rightarrow \Box\mathfrak{I}$ by lemma 4.20,2(b), we obtain $\vdash \theta(\mathcal{G}) \Rightarrow \Box F$. Since $\neg\Box F \in \theta(\mathcal{G})$, we also have $\vdash \theta(\mathcal{G}) \Rightarrow \neg\Box F$. This is a contradiction to the consistency of $\theta(\mathcal{G})$.

Next we show that for every node $\mathcal{H}_0$ with $\neg a[\Box F] \in \mathcal{H}_0$ there is a finite path $\mathcal{H}_0, \ldots, \mathcal{H}_j$ in $\mathcal{T}(\mathcal{F})$ such that $a\langle\mathbf{true}\rangle \in \mathcal{H}_i$ for $i \leq j$ and $\neg a[F] \in \mathcal{H}_j$. Suppose that there is no such path. Since $\neg a[\Box F] \in \mathcal{H}_0$, we also have $a\langle\mathbf{true}\rangle \in \mathcal{H}_0$ (cf. proposition 4.11,2.), hence $\neg a[F] \notin \mathcal{H}_0$ by assumption (otherwise the path $\mathcal{H}_0$ of length 1 would satisfy the required condition). Since $a[F] \in \tau\kappa(\mathcal{H}_0)$, this implies $a[F] \in \mathcal{H}_0$. By the definition of θ it follows that $\neg a[\Box F] \in \theta(\mathcal{H}_0)$, hence $\{\neg a[\Box F], a\langle\mathbf{true}\rangle\} \subseteq \mathcal{G}$ for all successors $\mathcal{G}$ of $\mathcal{H}_0$. Using the same arguments as above, we obtain again that $a[F] \in \mathcal{G}$. In this way we show that $\{\neg a[\Box F], a[F]\} \subseteq \mathcal{H}$ holds for all nodes $\mathcal{H}$ accessible from $\mathcal{H}_0$. Let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ denote all nodes in the subgraph $\mathcal{T}(\mathcal{H}_0)$. We have just shown that $\neg a[\Box F] \in \mathcal{G}_i$ and that $a[F] \in \mathcal{G}_i$ hold for $i \in \{1, \ldots, n\}$. By (ax11) and (prop) it follows $\vdash \mathcal{G}_1 \vee \ldots \vee \mathcal{G}_n \Rightarrow \neg\Box a[F]$, hence also $\vdash \theta(\mathcal{H}_0) \Rightarrow \neg\Box a[F]$. On the other hand, by lemma 4.20,2(b) it follows $\vdash \theta(\mathcal{H}_0) \Rightarrow \Box(\mathcal{G}_1 \vee \ldots \vee \mathcal{G}_n)$, hence $\vdash \theta(\mathcal{H}_0) \Rightarrow \Box a[F]$. This is a contradiction to the consistency of $\theta(\mathcal{H}_0)$.

Observe that for every finite path $\mathcal{F}_0, \ldots, \mathcal{F}_n$ in $\mathcal{T}(\mathcal{F})$ with $\neg a[\Box F] \in \mathcal{F}_0$ it holds that if $\neg a[F] \notin \mathcal{F}_i$ for all $i \leq n$, then $\{a[F], \neg a[\Box F], a\langle\mathbf{true}\rangle\} \subseteq \mathcal{F}_i$ for all $i \leq n$. In other words, it holds on every path that as long as the condition "$\neg a[F] \in \mathcal{F}_j$" for a complete path is not satisfied, the condition that "$a\langle\mathbf{true}\rangle \in \mathcal{F}_j$ for all $j \leq i$" is not violated, that is, $\mathcal{F}_0, \ldots, \mathcal{F}_n$ is a "potential prefix" of a complete path.

A complete path can be built now in the same way as in the case of LTL, described for example in [29]. ■

Lemma 4.22 *Assume that $\mathcal{F}$ is a finite and consistent set of formulas and that $\mathcal{F}_0, \mathcal{F}_1, \ldots$ is a complete path in $\mathcal{T}(\mathcal{F})$. For every $i \in \omega$, the following hold:*

1. *If $\circ F \in \tau\kappa(\mathcal{F}_i)$, then $\circ F \in \mathcal{F}_i$ iff $F \in \mathcal{F}_{i+1}$.*
2. *If $\Box F \in \tau\kappa(\mathcal{F}_i)$, then $\Box F \in \mathcal{F}_i$ iff $F \in \mathcal{F}_j$ for all $j \geq i$.*
3. *If $a[\circ F] \in \tau\kappa(\mathcal{F}_i)$, then $a[\circ F] \in \mathcal{F}_i$ iff $a\langle\mathbf{true}\rangle \notin \mathcal{F}_i$ or $a[F] \in \mathcal{F}_{i+1}$.*
4. *If $a[\Box F] \in \tau\kappa(\mathcal{F}_i)$, then $a[\Box F] \in \mathcal{F}_i$ iff for all $j \geq i$: either $a\langle\mathbf{true}\rangle \notin \mathcal{F}_k$ for some $k \in \{i, \ldots, j\}$ or $a[F] \in \mathcal{F}_j$.*

Proof.

1. If $\circ F \in \mathcal{F}_i$, then $F \in \theta(\mathcal{F}_i)$ by definition, hence $F \in \mathcal{F}_{i+1}$. Assume that $F \in \mathcal{F}_{i+1}$. Because of $\circ F \in \tau\kappa(\mathcal{F}_i)$, we have either $\circ F \in \mathcal{F}_i$ or $\neg\circ F \in \mathcal{F}_i$. By the definition of θ, $\neg\circ F \in \mathcal{F}_i$ would imply $\neg F \in \mathcal{F}_{i+1}$, in particular the inconsistency of $\mathcal{F}_{i+1}$. Hence, we have $\circ F \in \mathcal{F}_i$.

2. Assume first that $\Box F \in \mathcal{F}_i$. Since $\mathcal{F}_i$ is consistent and $F \in \tau\kappa(\mathcal{F}_i)$, it follows by (ax8) that $F \in \mathcal{F}_i$. By the definition of θ it follows that $\Box F \in \mathcal{F}_{i+1}$. Using the same arguments, we obtain that $F \in \mathcal{F}_{i+1}$ and $\Box F \in \mathcal{F}_{i+2}$. Inductively, we obtain $F \in \mathcal{F}_j$ for all $j \geq i$.

   Conversely, assume that $\Box F \notin \mathcal{F}_i$, that is, $\neg\Box F \in \mathcal{F}_i$. By the definition of a complete path it follows the existence of an index $j \geq i$ for which $\neg F \in \mathcal{F}_j$ holds.

3. If $a[\circ F] \in \mathcal{F}_i$ and $a\langle\mathbf{true}\rangle \in \mathcal{F}_i$, then by the definition of θ_5 follows that $a[F] \in \mathcal{F}_{i+1}$, as $\mathcal{F}_{i+1}$ is a completion of $\theta(\mathcal{F}_i)$. For the converse, assume $a[\circ F] \notin \mathcal{F}_i$. Then $\neg a[\circ F] \in \mathcal{F}_i$ as $\mathcal{F}_i$ is complete, hence $a\langle\mathbf{true}\rangle \in \mathcal{F}_i$ by proposition 4.11,2. and $\neg a[F] \in \mathcal{F}_{i+1}$ by the definition of θ. Since $\mathcal{F}_{i+1}$ is consistent, this implies the assertion.

4. *Only if:* Assume that $a[\Box F] \in \mathcal{F}_i$ and let $j \geq i$ be a natural number such that $a\langle\mathbf{true}\rangle \in \mathcal{F}_k$ for all $k \in \{i, \ldots, j\}$. We have to show that $a[F] \in \mathcal{F}_j$. Observe first that $a[\Box F], a\langle\mathbf{true}\rangle \in \mathcal{F}_l$ implies $a[\Box F] \in \mathcal{F}_{l+1}$ by the definition of θ,

hence it follows that $a[\Box F] \in \mathcal{F}_k$ for all $k \in \{i, \ldots, j\}$. On the other hand, we have $\vdash a[\Box F] \Rightarrow a[F]$ by (T12), hence $a[F] \in \mathcal{F}_j$ by proposition 4.11,1.

If: Assume $a[\Box F] \notin \mathcal{F}_i$, i.e. $\neg a[\Box F] \in \mathcal{F}_i$. The assertion follows from the definition of a complete path. ■

Now we are able to construct a model of $\mathcal{F}$ based on a complete path. Let $\mathcal{F}_0, \mathcal{F}_1, \ldots$ be a complete path in the graph $\mathcal{T}(\mathcal{F})$. Let $\sigma := (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ where for every $i \in \omega$ the pair (t_i, λ_i) is defined as described in sec. 4.1 starting with the set $\mathcal{F}_i$ (cf. page 59). Note that this construction is independent of the presence of temporal operators. In the following theorem we show that σ is a model for $\mathcal{F}$.

Theorem 4.23 *Let $\mathcal{F}$ be a finite, consistent and complete set of formulas. Let $\mathcal{F}_0, \mathcal{F}_1, \ldots$ be a complete path in $\mathcal{T}(\mathcal{F})$. Further, let $\sigma := (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ be the run constructed from the path as described above with $t_i = (\mathsf{N}_i, <_i)$. Then the following hold for all $i \in \omega$:*

1. *If $F \in \tau\kappa(\mathcal{F}_i)$, then $F \in \mathcal{F}_i$ iff $\sigma|_i, \varepsilon \models F$.*
2. *If $a[F] \in \tau\kappa(\mathcal{F}_i)$, then $a[F] \in \mathcal{F}_i$ iff $a \notin \mathsf{N}_i$ or $\sigma|_i, a \models F$.*

Proof. As in theorem 4.14, we prove the assertions simultaneously by induction on the formula F. The first cases are proven in the same way as in the proof of theorem 4.14. Now we consider the temporal operators.

Case: $\circ F$.

1. By lemma 4.22,1. we have $\circ F \in \mathcal{F}_i$ iff $F \in \mathcal{F}_{i+1}$. By induction hypothesis, this is equivalent to $\sigma|_{i+1}, \varepsilon \models F$, hence, by definition, to $\sigma|_i, \varepsilon \models \circ F$.
2. By lemma 4.22,3. it holds $a[\circ F] \in \mathcal{F}_i$ iff $a\langle \mathbf{true} \rangle \notin \mathcal{F}_i$ or $a[F] \in \mathcal{F}_{i+1}$. By induction hypothesis and by the definition of t_i, this is equivalent with $a \notin \mathsf{N}_i$ or $a \notin \mathsf{N}_{i+1}$ or $\sigma|_{i+1}, a \models F$. By the definition of the semantics of pMLTL, this means $a \notin \mathsf{N}_i$ or $\sigma|_i, a \models \circ F$.

Case: $\Box F$.

1. This follows easily from lemma 4.22,2.
2. By lemma 4.22,4. and the definition of t_i, $a[\Box F] \in \mathcal{F}_i$ iff $a \notin \mathsf{N}_i$ or for all $j \geq i$: either $a \notin \mathsf{N}_k$ for some $k \in \{i, \ldots, j\}$ or $a[F] \in \mathcal{F}_j$. By induction hypothesis, this is equivalent to $a \notin \mathsf{N}_i$ or (for all $j \geq i$: either $a \notin \mathsf{N}_k$ for some $k \in \{i, \ldots, j\}$ or $\sigma|_j, a \models F$), i.e. to $a \notin \mathsf{N}_i$ or $\sigma|_i, a \models \Box F$. ■

4.2.2 The proof system $\Sigma_{\mathbf{MLTL}}$

Now we extend the proof system Σ_{MLTL^-} by axioms that characterise the "keep"-operators and call the new system Σ_{MLTL}. The additional axioms are drawn together in figure 4.6. We already have defined in the previous section the mappings τ and κ for all pMLTL-formulas. All the lemmas proved there concerning these mappings hold also for pMLTL.

All axioms and rules of Σ_{MLTL^-}

(ax12) $\vdash \mathsf{keep}_a \Rightarrow (a.\alpha \Leftrightarrow \circ a.\alpha)$ for $\alpha \in \mathsf{N}^*$

(ax13) $\vdash \mathsf{keep}_a \wedge a.b \Rightarrow \mathsf{keep}_b$

(ax14) $\vdash a[\mathbf{false}] \wedge \circ a[\mathbf{false}] \Rightarrow \mathsf{keep}_a$

(ax15) $\vdash a[\mathsf{keep}_b] \wedge a.b \Rightarrow \mathsf{keep}_b \wedge \circ a.b$

(ax16) $\vdash \mathsf{keep}_b \wedge a.b \wedge \circ a.b \Rightarrow a[\mathsf{keep}_b]$

(ax17) $\vdash a.b[\mathbf{false}] \wedge \circ a.b[\mathbf{false}] \Rightarrow a[\mathsf{keep}_b]$

(ax18) $\vdash a\langle \mathsf{keep}_b \rangle \wedge a.b[\mathbf{false}] \Rightarrow \circ a.b[\mathbf{false}]$

Figure 4.6: The system Σ_{MLTL}

We want to show that Σ_{MLTL} is a complete axiomatisation for pMLTL. Again, we assume a finite and consistent set of formulas and construct a model for it. First we extend the mapping θ in the following way:

$$\theta(\mathcal{F}) := \bigcup_{i=1}^{15} \theta_i(\mathcal{F})$$

where $\theta_1, \ldots, \theta_8$ are defined as before and $\theta_9, \ldots, \theta_{15}$ as given in fig. 4.7.

$$
\begin{aligned}
\theta_9(\mathcal{F}) &= \{a.\alpha\langle\mathbf{true}\rangle | \mathsf{keep}_a \in \mathcal{F} \text{ and } a.\alpha\langle\mathbf{true}\rangle \in \mathcal{F}\} \\
\theta_{10}(\mathcal{F}) &= \{a.\alpha[\mathbf{false}] | \mathsf{keep}_a \in \mathcal{F} \text{ and } a.\alpha[\mathbf{false}] \in \mathcal{F}\} \\
\theta_{11}(\mathcal{F}) &= \{b\langle\mathbf{true}\rangle | \neg\mathsf{keep}_b, b[\mathbf{false}] \in \mathcal{F}\} \\
\theta_{12}(\mathcal{F}) &= \{a.b.\alpha\langle\mathbf{true}\rangle | a[\mathsf{keep}_b], a\langle\mathbf{true}\rangle, a.b.\alpha\langle\mathbf{true}\rangle \in \mathcal{F}\} \\
\theta_{13}(\mathcal{F}) &= \{a.b.\alpha[\mathbf{false}] | a[\mathsf{keep}_b], a\langle\mathbf{true}\rangle, a.b.\alpha[\mathbf{false}] \in \mathcal{F}\} \\
\theta_{14}(\mathcal{F}) &= \{a.b\langle\mathbf{true}\rangle | \neg a[\mathsf{keep}_b], a.b[\mathbf{false}] \in \mathcal{F}\} \\
\theta_{15}(\mathcal{F}) &= \{a.b[\mathbf{false}] | a.b\langle\mathbf{true}\rangle, \mathsf{keep}_b, \neg a[\mathsf{keep}_b] \in \mathcal{F}\}
\end{aligned}
$$

Figure 4.7: Definition of $\theta_9, \ldots, \theta_{15}$

In the next proposition we show that prop. 4.19 holds also for the extended θ.

Proposition 4.24 *Let $\mathcal{F}$ be a set of formulas. Then the following hold:*

1. $\vdash \mathcal{F} \Rightarrow \bigcirc\theta(\mathcal{F})$.
2. *If $\mathcal{F}$ is consistent, then so is $\theta(\mathcal{F})$.*

Proof. The proof is the same as for proposition 4.19, we only need to consider the new cases in the definition of θ.

- $F \equiv a.\alpha\langle\mathbf{true}\rangle \in \theta_9(\mathcal{F})$: by definition we have $\vdash \mathcal{F} \Rightarrow \mathsf{keep}_a \wedge a.\alpha\langle\mathbf{true}\rangle$, hence by (ax12) it follows that $\vdash \mathcal{F} \Rightarrow \bigcirc a.\alpha\langle\mathbf{true}\rangle$.
- $F \equiv a.\alpha[\mathbf{false}] \in \theta_{10}(\mathcal{F})$: using axiom (ax12),(T5) and the definition of θ_{10} we conclude $\vdash \mathcal{F} \Rightarrow \neg\bigcirc a.\alpha\langle\mathbf{true}\rangle$. The assertion follows with the aid of (ax6) and (T5).
- $F \equiv b\langle\mathbf{true}\rangle \in \theta_{11}(\mathcal{F})$: by the definition of θ_{11} holds $\neg\mathsf{keep}_b, b[\mathbf{false}] \in \mathcal{F}$. By (ax14), (prop) and (ax6) it follows $\vdash \mathcal{F} \Rightarrow \bigcirc\neg b[\mathbf{false}]$, hence $\vdash \mathcal{F} \Rightarrow \bigcirc b\langle\mathbf{true}\rangle$.

- $F \equiv a.b.\alpha\langle\mathbf{true}\rangle \in \theta_{12}(\mathcal{F})$: Observe that $a.b.\alpha\langle\mathbf{true}\rangle \in \mathcal{F}$ and $a[\mathsf{keep}_b] \in \mathcal{F}$ by the definition of θ_{12}. We give a derivation of $\mathcal{F} \Rightarrow \bigcirc a.b.\alpha$.

(1)	$\mathcal{F} \Rightarrow a.b.\alpha\langle\mathbf{true}\rangle$	def. of θ_{12}
(2)	$\mathcal{F} \Rightarrow a.b\langle\mathbf{true}\rangle$	(1),(T7),(prop)
(3)	$\mathcal{F} \Rightarrow b.\alpha\langle\mathbf{true}\rangle$	(1),(T7),(prop)
(4)	$\mathcal{F} \Rightarrow a[\mathsf{keep}_b]$	def. of θ_{12}
(5)	$\mathcal{F} \Rightarrow \mathsf{keep}_b$	(ax15),(4),(2),(prop)
(6)	$\mathcal{F} \Rightarrow \bigcirc a.b\langle\mathbf{true}\rangle$	(ax15),(4),(2),(prop)
(7)	$\mathcal{F} \Rightarrow \bigcirc b.\alpha\langle\mathbf{true}\rangle$	(ax12),(3),(5),(prop)
(8)	$\mathcal{F} \Rightarrow \bigcirc(a.b \wedge b.\alpha)$	(6),(7),(T10),(prop)
(9)	$a.b \wedge b.\alpha \Rightarrow a.b.\alpha$	(T7),(prop)
(10)	$\bigcirc(a.b \wedge b.\alpha) \Rightarrow \bigcirc a.b.\alpha$	(nex),(9),(ax7),(prop)
(11)	$\mathcal{F} \Rightarrow \bigcirc a.b.\alpha\langle\mathbf{true}\rangle$	(8),(10),(prop)

- $F \equiv a.b.\alpha[\mathbf{false}] \in \theta_{13}(\mathcal{F})$: Note that $a[\mathsf{keep}_b], a\langle\mathbf{true}\rangle, a.b.\alpha[\mathbf{false}] \in \mathcal{F}$ by the definition of θ_{13}. We give a derivation of $\mathcal{F} \Rightarrow \bigcirc a.b.\alpha[\mathbf{false}]$:

(1)	$\mathcal{F} \Rightarrow a\langle\mathsf{keep}_b\rangle \wedge a.b.\alpha[\mathbf{false}]$	def. of θ_{12},(T6),(prop)
(2)	$a\langle\mathsf{keep}_b\rangle \wedge a.b[\mathbf{false}] \Rightarrow \bigcirc a.b[\mathbf{false}]$	(ax18)
(3)	$a.b[\mathbf{false}] \Rightarrow a.b.\alpha[\mathbf{false}]$	(T7),(T5),(prop)
(4)	$\bigcirc a.b[\mathbf{false}] \Rightarrow \bigcirc a.b.\alpha[\mathbf{false}]$	(3),(nex),(ax7),(prop)
(5)	$\mathcal{F} \wedge a.b[\mathbf{false}] \Rightarrow \bigcirc a.b.\alpha[\mathbf{false}]$	(1),(2),(4),(prop)
(6)	$a\langle\mathsf{keep}_b\rangle \wedge a.b\langle\mathbf{true}\rangle \Rightarrow \mathsf{keep}_b$	(ax15),(prop)
(7)	$a.b\langle\mathbf{true}\rangle \wedge a.b.\alpha[\mathbf{false}] \Rightarrow b.\alpha[\mathbf{false}]$	(T5),(T7),(prop)
(8)	$\mathsf{keep}_b \wedge b.\alpha[\mathbf{false}] \Rightarrow \bigcirc b.\alpha[\mathbf{false}]$	(ax12),(prop)
(9)	$\bigcirc b.\alpha[\mathbf{false}] \Rightarrow \bigcirc a.b.\alpha[\mathbf{false}]$	(ax2),(nex),(ax7),(prop)
(10)	$\mathcal{F} \wedge a.b\langle\mathbf{true}\rangle \Rightarrow \bigcirc a.b.\alpha[\mathbf{false}]$	(1),(6),(7),(8),(9),(prop)
(11)	$\mathcal{F} \Rightarrow \bigcirc a.b.\alpha[\mathbf{false}]$	(5),(10),(prop)

- $F \equiv a.b\langle\mathbf{true}\rangle \in \theta_{14}(\mathcal{F})$: we have $\neg a[\mathsf{keep}_b], a.b[\mathbf{false}] \in \mathcal{F}$ by the definition of θ_{14}. By (ax17) it follows that $\vdash \mathcal{F} \Rightarrow \neg\bigcirc a.b[\mathbf{false}]$, hence $\vdash \mathcal{F} \Rightarrow \bigcirc a.b\langle\mathbf{true}\rangle$ by (ax6), (T5), (nex) and (ax7).

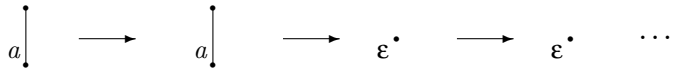
- $F \equiv a.b[\mathbf{false}] \in \theta_{15}(\mathcal{F})$: then it holds $a.b\langle\mathbf{true}\rangle, \mathsf{keep}_b, \neg a[\mathsf{keep}_b] \in \mathcal{F}$ by the definition of θ_{15}. By (ax16) and by propositional reasoning it follows that $\vdash \mathcal{F} \Rightarrow \neg\bigcirc a.b\langle\mathbf{true}\rangle$, hence $\vdash \bigcirc a.b[\mathbf{false}]$ by (ax6), (T5), (nex) and (ax7). ■

The graph $\mathcal{T}(\mathcal{F})$ is redefined: the successors of a node are defined with respect to the new θ. The notion of a complete path is unchanged.

As our next step, we are going to construct a run based on a complete path $\mathcal{F}_0, \mathcal{F}_1 \ldots$ in the graph $\mathcal{T}(\mathcal{F})$. Let σ' denote the sequence $(t'_0, \lambda'_0)(t'_1, \lambda'_1)\ldots$ where (t'_i, λ'_i) is defined as in sec. 4.1, starting with the set $\mathcal{F}_i$. Note that this run is in general no model for the set $\mathcal{F}$, as formulas of the form $\neg\mathsf{keep}_a$ are not necessarily satisfied. In order to see this, consider for example the following set: $\mathcal{F} = \{\neg\mathsf{keep}_a, a\langle\mathbf{true}\rangle, \bigcirc a\langle\mathbf{true}\rangle\}$. Let $\mathcal{G}$ denote an arbitrary completion of $\mathcal{F}$. A possible complete path in $\mathcal{T}(\mathcal{F})$ is the following:

$$\mathcal{G}, \{a\langle\mathbf{true}\rangle\}, \emptyset, \emptyset \ldots$$

The construction yields the following run σ:



Obviously, for σ does not hold the formula $\neg\mathsf{keep}_a$. The problem is that a formula of the form $\neg\mathsf{keep}_a$ possibly requires the existence of some new name that does not occur in $\mathcal{F}$ whereas in the run σ only names occur that occur in some formula in $\mathcal{F}$.

To solve this problem, we have to modify the trees of the run based on a complete path slightly. For every name a for which $\neg\mathsf{keep}_a \in \bigcup_{i\geq 0} \mathcal{F}_i$, let $n_a \in \mathsf{N}$ be a name that does not occur in any of the formulas in $\bigcup_{i\geq 0} \mathcal{F}_i$, with $n_a \neq n_b$ for $a \neq b$. Note that it is possible to choose such names, as $\bigcup_{i\geq 0} \mathcal{F}_i$ is a finite set. Intuitively, the name n_a will have to ensure that the formula $\neg\mathsf{keep}_a$ holds: if $\neg\mathsf{keep}_a$ has to hold for a sub-run $\sigma|_i$ and the name n_a does not occur in t_i, then n_a is put below the name a in the tree t_{i+1}. Conversely, if n_a is already in t_i, then it must not appear in t_{i+1}. This idea is formalised in the following definition.

We define the run $\sigma = (t_0,\lambda_0)(t_1,\lambda_1)\ldots$ by induction on $i \in \omega$. We let again $\sigma' = (t'_0,\lambda'_0)(t'_1,\lambda'_1)\ldots$, with $t'_i = (\mathsf{N}'_i, <'_i)$, be the run based on a complete path $\mathcal{F}_0, \mathcal{F}_1, \ldots$ as described above and let $(t_0,\lambda_0) := (t'_0,\lambda'_0)$. Let $i \in \omega$ and assume that $(t_0,\lambda_0),\ldots,(t_i,\lambda_i)$ are already constructed.

$$\begin{aligned}\mathsf{N}_{i+1} := \mathsf{N}'_{i+1} &\cup \{n_a | \neg\mathsf{keep}_a \in \mathcal{F}_i, a \in \mathsf{N}'_i, a \in \mathsf{N}'_{i+1} \text{ and } n_a \notin \mathsf{N}_i\} \\ &\cup \{n_a | n_a \in \mathsf{N}_i \text{ and } \mathsf{keep}_b \in \mathcal{F}_i \text{ for some } b \text{ with } n_a <_i b\}\end{aligned}$$

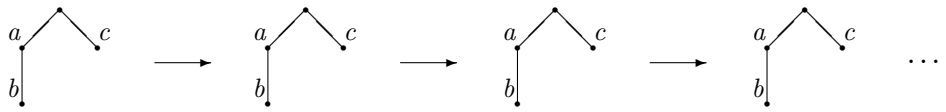
and the binary relation $<_{i+1}$ on N_{i+1} is defined as follows:

$$a <_{i+1} b :\Leftrightarrow \begin{cases} a <'_{i+1} b & \text{if } a,b \in \mathsf{N}'_{i+1} \\ \overline{a} \leq'_{i+1} b & \text{if } b \in \mathsf{N}'_{i+1} \text{ and } a = n_{\overline{a}} \text{ for some } \overline{a} \in \mathsf{N} \\ a <'_{i+1} \overline{b} & \text{if } a \in \mathsf{N}'_{i+1} \text{ and } b = n_{\overline{b}} \text{ for some } \overline{b} \in \mathsf{N} \\ \overline{a} <'_{i+1} \overline{b} & \text{if } a = n_{\overline{a}}, b = n_{\overline{b}} \text{ and } a <'_{i+1} b \end{cases}$$

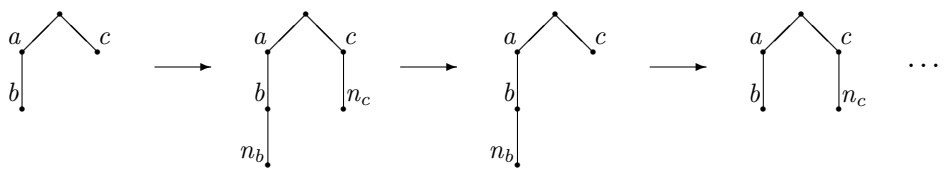
The assignments λ_i are defined by

$$\lambda_i(a) := \begin{cases} \lambda'_i(a) & \text{if } a \in \mathsf{N}'_i \\ \emptyset & \text{if } a = n_{\overline{a}} \text{ for some } \overline{a} \in \mathsf{N}'_i \end{cases}$$

Informally, this definition indicates that n_a is put immediately below a. To illustrate the construction, we consider a sequence $\mathcal{F}^*_0, \mathcal{F}^*_1, \ldots$ where $\mathcal{F}^*_0$ is some completion of $\mathcal{F}_0 = \{\Box ab\langle\textbf{true}\rangle, \Box c\langle\textbf{true}\rangle, \neg\mathsf{keep}_b, \bigcirc\mathsf{keep}_a, \Box\neg\mathsf{keep}_c\}$ and $\mathcal{F}^*_{i+1}$ is some completion of $\theta(\mathcal{F}^*_i)$. The prefix of one possible resulting run – first only applying the original definition from sec. 4.1, that is, without the described modification – is the following:



This is obviously not a model of $\mathcal{F}_0$, as the formulas $\neg\mathsf{keep}_b$ and $\Box\neg\mathsf{keep}_c$ are not satisfied. The modification according to our definition yields the following run:



The first trees are identical. Then the name n_b is put below the name b as well as the name n_c below c in order to satisfy formulas $\neg\mathsf{keep}_b$ and $\Box\neg\mathsf{keep}_c$. In the next step n_c disappears as required by formula $\Box\neg\mathsf{keep}_c$, but node n_b is still kept as required by $\mathsf{keep}_a \in \mathcal{F}_1^*$ (this follows from $\circ\mathsf{keep}_a \in \mathcal{F}_0$) and $a.b\langle\mathbf{true}\rangle \in \mathcal{F}_1^*$. In the last tree n_b does not appear (the auxiliary nodes are not kept if not explicitly required by some keep operator) whereas n_c reappears due to $\neg\mathsf{keep}_c \in \mathcal{F}_2^*$.

We show that t_i is a tree.

Lemma 4.25 *Let $t_i = (\mathsf{N}_i, <_i)$ be defined as described above. Then t_i is a tree.*

Proof. In order to prove the claim we have to show that

1. the relations $<_i$ are irreflexive.
2. the relations $<_i$ are transitive.
3. if $a, b, c \in \mathsf{N}_i$, $a \neq b$, $c <_i a$ and $c <_i b$ then either $a <_i b$ or $b <_i a$.

To 1.: Assume that $a <_i a$. We have to distinguish two cases.

Case: $a \in \mathsf{N}_i'$. Then $a <_i' a$ by definition in contradiction to the irreflexivity of the relation $<_i'$.

Case: $a = n_{\overline{a}}$ for some $\overline{a} \in \mathsf{N}_i'$. Then $\overline{a} <_i' \overline{a}$ by definition, hence we obtain the same contradiction as above.

To 2.: Assume that $a <_i b$ and $b <_i c$. We have to show that $a <_i c$. Again, we have to distinguish different cases.

Case: $a, b, c \in \mathsf{N}_i'$. Then $a <_i' b$ and $b <_i' c$ by definition, hence $a <_i' c$ by the transitivity of $<_i'$. By the definition of $<_i$ this means $a <_i c$.

Case: $a, b \in \mathsf{N}_i'$ and $c = n_{\overline{c}}$ for some $\overline{c} \in \mathsf{N}_i'$. Then $a <_i' b$ and $b <_i' \overline{c}$, hence $a <_i' \overline{c}$. Latter implies $a <_i c$.

Case: $a, c \in \mathsf{N}_i'$ and $b = n_{\overline{b}}$ for some $\overline{b} \in \mathsf{N}_i'$. Then $a <_i' b$ and $b \leq_i' c$ by definition, hence $a <_i' c$ and so $a <_i c$ by the definition of $<_i$.

Case: $b, c \in \mathsf{N}'_i$ and $a = n_{\overline{a}}$ for some $\overline{a} \in \mathsf{N}'_i$. Then $\overline{a} \leq'_i b$ and $b <'_i c$. It follows $\overline{a} <'_i c$, hence $n_{\overline{a}} <_i c$ by definition.

Case: $a \in \mathsf{N}'_i$, $b = n_{\overline{b}}$ and $c = n_{\overline{c}}$ for some $\overline{b}, \overline{c} \in \mathsf{N}'_i$. Then $a <'_i \overline{b}$ and $\overline{b} <'_i \overline{c}$ by definition. It follows $a <'_i \overline{c}$, hence $a <_i c$.

Case: $b \in \mathsf{N}'_i$, $a = n_{\overline{a}}$ and $c = n_{\overline{c}}$ for some $\overline{a}, \overline{c} \in \mathsf{N}'_i$. Then $\overline{a} \leq'_i b$ and $b <'_i \overline{c}$ by definition. This implies $\overline{a} <'_i \overline{c}$ as $<'_i$ is transitive, hence $a <_i c$ by definition.

Case: $c \in \mathsf{N}'_i$, $a = n_{\overline{a}}$ and $b = n_{\overline{b}}$ for some $\overline{a}, \overline{b} \in \mathsf{N}'_i$. Then $\overline{a} <'_i \overline{b}$ and $\overline{b} \leq'_i c$ by definition. Hence $\overline{a} <'_i c$ which implies $a <_i c$.

Case: $a = n_{\overline{a}}$, $b = n_{\overline{b}}$ and $c = n_{\overline{c}}$ for some $\overline{a}, \overline{b}, \overline{c} \in \mathsf{N}'_i$. Then $\overline{a} <'_i \overline{b}$ and $\overline{b} <'_i \overline{c}$ by definition, hence $\overline{a} <'_i \overline{c}$. From this follows $a <_i c$ by the definition of $<_i$.

To 3.: Again, different cases have to be considered.

Case: $a, b, c \in \mathsf{N}'_i$. Then $c <'_i a$ and $c <'_i b$ by definition and as $(t'_i, <'_i)$ is a tree it follows $a <'_i b$ or $b <'_i a$. Hence, $a <_i b$ or $b <_i a$.

Case: $a, b \in \mathsf{N}'_i$ and $c = n_{\overline{c}}$ for some $\overline{c} \in \mathsf{N}'_i$. Then $\overline{c} \leq'_i a$ and $\overline{c} \leq'_i b$, which implies $a <'_i b$ or $b <'_i a$. Hence, $a <_i b$ or $b <_i a$ by the definition of $<_i$.

Case: $c \in \mathsf{N}'_i$, $a = n_{\overline{a}}$ and $b = n_{\overline{b}}$ for some $\overline{a}, \overline{b} \in \mathsf{N}'_i$ with $\overline{a} \neq \overline{b}$ (as $n_{\overline{a}} \neq n_{\overline{b}}$ by assumption). Then $c <'_i \overline{a}$ and $c <'_i \overline{b}$ by the definition of $<_i$, hence $\overline{a} <'_i \overline{b}$ or $\overline{b} <'_i \overline{a}$. In the first case it follows $a <_i b$, in the second case $b <_i a$ by the definition of $<_i$.

Case: $a, c \in \mathsf{N}'_i$ and $b = n_{\overline{b}}$ for some $\overline{b} \in \mathsf{N}'_i$. Then $c <'_i a$ and $c <'_i \overline{b}$ by definition. It follows $a <'_i \overline{b}$ or $\overline{b} <'_i a$, hence $a \leq_i b$ or $b <_i a$. Since $a \neq b$ by assumption, the assertion follows.

Case: $b, c \in \mathsf{N}'_i$ and $a = n_{\overline{a}}$ for some $\overline{a} \in \mathsf{N}'_i$. This case is symmetrical to the previous case.

Case: $a \in \mathsf{N}'_i$, $b = n_{\overline{b}}$ and $c = n_{\overline{c}}$ for some $\overline{b}, \overline{c} \in \mathsf{N}'_i$. Then $\overline{c} \leq'_i a$ and $\overline{c} <'_i \overline{b}$. If $\overline{c} = a$ then we have $a <'_i b$ by the latter. Otherwise we obtain $a <'_i \overline{b}$ or $\overline{b} <'_i a$, hence $a <_i b$ or $b <_i a$ by definition.

Case: $b \in \mathsf{N}'_i$, $a = n_{\overline{a}}$ and $c = n_{\overline{c}}$. for some $\overline{a}, \overline{c} \in \mathsf{N}'_i$. This is symmetrical to the previous case.

Case: $a = n_{\overline{a}}$, $b = n_{\overline{b}}$ and $c = n_{\overline{c}}$ for some $\overline{a}, \overline{b}, \overline{c} \in \mathsf{N}'_i$. Then we have $\overline{c} <'_i \overline{a}$ and $\overline{c} <'_i \overline{b}$ by definition, hence $\overline{a} <'_i \overline{b}$ or $\overline{b} <'_i \overline{a}$ as t'_i is a tree. It follows $a <_i b$ or $b <_i a$ by the definition of $<_i$. ■

We prove the following lemma about the sequence σ.

Lemma 4.26 *Let $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ be defined as above and let $i \in \omega$ be arbitrary. Then the following holds:*

1. *If $\mathsf{keep}_b \in \tau\kappa(\mathcal{F}_i)$, then $\mathsf{keep}_b \in \mathcal{F}_i$ iff $t_i{\downarrow}b = t_{i+1}{\downarrow}b$.*
2. *If $a[\mathsf{keep}_b] \in \tau\kappa(\mathcal{F}_i)$, then $a[\mathsf{keep}_b] \in \mathcal{F}_i$ iff $a \notin \mathsf{N}_i$ or $t_i{\downarrow}a.b = t_{i+1}{\downarrow}a.b$.*

Proof.

1. *Only if:* Let $\mathsf{keep}_b \in \mathcal{F}_i$. First we show that $t'_i{\downarrow}b = t'_{i+1}{\downarrow}b$. To this end, it is enough to prove that for all $a, c \in \mathsf{N}$ holds the following:

   (a) $a <'_i b$ iff $a <'_{i+1} b$

   (b) If $a <'_i b$, then $c <'_i a$ iff $c <'_{i+1} a$

   To a): By the definition of the relations $<'_j$ we have to show $b.a\langle\mathbf{true}\rangle \in \mathcal{F}_i$ iff $b.a\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$. Since $\mathsf{keep}_b \in \mathcal{F}_i$ and since $\mathcal{F}_{i+1}$ is a completion of $\theta(\mathcal{F}_i)$, this follows from the definition of θ.

   To b): Let $a <'_i b$, i.e. $b.a\langle\mathbf{true}\rangle \in \mathcal{F}_i$. We have to show that $a.c\langle\mathbf{true}\rangle \in \mathcal{F}_i$ iff $a.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$.

   Only if: Assume $a.c\langle\mathbf{true}\rangle \in \mathcal{F}_i$. Observe that a, b, c are pairwise distinct and hence $b.a.c\langle\mathbf{true}\rangle \in \tau\kappa(\mathcal{F}_i)$. By (T7) it follows $\vdash \mathcal{F}_i \Rightarrow b.a.c\langle\mathbf{true}\rangle$, hence $b.a.c\langle\mathbf{true}\rangle \in \mathcal{F}_i$ by proposition 4.11,1. By the definition of θ it follows that $b.a.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$. Since $\vdash b.a.c\langle\mathbf{true}\rangle \Rightarrow a.c\langle\mathbf{true}\rangle$ by (T0), it follows that $a.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$.

If: Assume $a.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$. Since a, b, c are pairwise distinct and as $b.a\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$ by a), it follows as above that $b.a.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$. As $a, b, c \in \mathsf{nm}(\mathcal{F}_i)$, we have $b.a.c\langle\mathbf{true}\rangle \in \tau\kappa(\mathcal{F}_i)$. By the definition of θ it follows that $b.a.c[\mathbf{false}] \notin \mathcal{F}_i$, hence $b.a.c\langle\mathbf{true}\rangle \in \mathcal{F}_i$. The assertion follows now as in the previous case.

Now we consider the "auxiliary" names n_c. From the definition of the trees t_i it follows directly that if a node n_c occurs in t_i, then its position is immediately below the node c. Hence, it suffices to show for every n_c that $n_c <_i b$ iff $n_c <_{i+1} b$.

Only if: Assume first that $n_c <_i b$. Since $\mathsf{keep}_b \in \mathcal{F}_i$ by assumption, the definition of t_{i+1} implies that $n_c \in \mathsf{N}_{i+1}$. Furthermore, by the definition of $<_{i+1}$ it follows that $n_c <_{i+1} b$, that is, $n_c <_{i+1} b$.

If: For the converse, assume that $n_c <_{i+1} b$. Then $c \leq_{i+1} b$ by definition, that is, either $c = b$ or $b.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$.

Case: $c = b$. Since $\mathsf{keep}_b \in \mathcal{F}_i$ and $n_c \in \mathsf{N}_{i+1}$, the definition of t_{i+1} implies $n_b \in \mathsf{N}_i$. By the definition of $<_i$ it follows $n_b <_i b$.

Case: $c <_{i+1} b$. By definition, this means $b.c\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$. Since $\mathsf{keep}_b \in \mathcal{F}_i$, by the definition of θ it follows $b.c\langle\mathbf{true}\rangle \in \mathcal{F}_i$. Hence, $\vdash \mathcal{F}_i \Rightarrow \mathsf{keep}_c$ by (ax13), in particular $\neg\mathsf{keep}_c \notin \mathcal{F}_i$. Again, it follows by the definition of t_{i+1} that $n_c \in \mathsf{N}_i$. As $c <_i b$ (since $b.c\langle\mathbf{true}\rangle \in \mathcal{F}_i$), we conclude $n_c <_i b$ by the definition of $<_i$.

If: Assume that $\mathsf{keep}_b \notin \mathcal{F}_i$, that is, as $\mathcal{F}_i$ is complete, $\neg\mathsf{keep}_b \in \mathcal{F}_i$.

Case: $n_b \notin \mathsf{N}_i$. There are two possibilities. Either $b \notin \mathsf{N}_i$ or $b \in \mathsf{N}_i$.
In the first case we have $b[\mathbf{false}] \in \mathcal{F}_i$, hence $b\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$ by the definition of θ. Hence, $b \notin \mathsf{N}_i$ but $b \in \mathsf{N}_{i+1}$, in particular, $t_i{\downarrow}b \neq t_{i+1}{\downarrow}b$.
If $b \in \mathsf{N}_i$ and $b \notin \mathsf{N}_{i+1}$, then we already have $t_i{\downarrow}b \neq t_{i+1}{\downarrow}b$ since latter is the empty tree whereas the first is not. If $b \in \mathsf{N}_{i+1}$, then $n_b \in \mathsf{N}_{i+1}$ by the definition of t_{i+1}, hence $n_b <_{i+1} b$. It follows $t_i{\downarrow}b \neq t_{i+1}{\downarrow}b$ since $n_b <_{i+1} b$ but $n_b \not<_i b$.

Case: $n_b \in \mathsf{N}_i$. Assume $n_b \in \mathsf{N}_{i+1}$. By the definition of t_{i+1}, this is only possible if there is a name c with $\mathsf{keep}_c \in \mathcal{F}_i$ and $n_b <_i c$. By the definition

of $<_i$ it follows $b \leq_i c$, that is, $b = c$ or $c.b\langle\mathbf{true}\rangle \in \mathcal{F}_i$. It follows in both cases - in the latter by (ax13) and by lemma 4.13 - that $\vdash \mathcal{F}_i \Rightarrow \mathsf{keep}_b$, in contradiction to the consistency of $\mathcal{F}_i$.

2. *Only if:* Assume that $a[\mathsf{keep}_b] \in \mathcal{F}_i$ and $a\langle\mathbf{true}\rangle \in \mathcal{F}_i$.

   Case: $b <_i a$, that is, $a.b\langle\mathbf{true}\rangle \in \mathcal{F}_i$. By (ax15) and (prop) it follows that $\vdash \mathcal{F}_i \Rightarrow \mathsf{keep}_b$ holds. As $\mathsf{keep}_b \in \tau\kappa(\mathcal{F}_i)$, it holds $\mathsf{keep}_b \in \mathcal{F}_i$ by proposition 4.11,1. Further, as $a[\mathsf{keep}_b], a.b\langle\mathbf{true}\rangle, a\langle\mathbf{true}\rangle \in \mathcal{F}_i$, we have $a.b\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$ by the definition of θ, hence $b <_{i+1} a$ by the definition of t_{i+1}. In particular, $t_i{\downarrow}b = t_i{\downarrow}a.b$ and $t_{i+1}{\downarrow}a.b = t_{i+1}{\downarrow}b$. Since $\mathsf{keep}_b \in \mathcal{F}_i$, this implies by 1. that $t_i{\downarrow}a.b = t_{i+1}{\downarrow}a.b$.

   Case: $t_i{\downarrow}a.b = \mathsf{empty}$, that is, $b \not<_i a$, that is, $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}_i$. If $a = b$, then $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}_{i+1}$ because of the consistency of $\mathcal{F}_{i+1}$, hence $b \not<_{i+1} a$. If $a \neq b$, then $a.b[\mathbf{false}] \in \mathcal{F}_i$ since $\mathcal{F}_i$ is complete, hence $a.b[\mathbf{false}] \in \mathcal{F}_{i+1}$ by the definition of θ, hence $b \not<_{i+1} a$, that is, $t_{i+1}{\downarrow}a.b = \mathsf{empty}$.

If: Let $t_i{\downarrow}a.b = t_{i+1}{\downarrow}a.b$.

   Case: $t_i{\downarrow}a.b = \mathsf{empty}$ and $t_{i+1}{\downarrow}a.b = \mathsf{empty}$, that is $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}_i$ and $a.b\langle\mathbf{true}\rangle \notin \mathcal{F}_{i+1}$.

      Case: If $a = b$, then $\vdash a.b[\mathbf{false}] \wedge \circ a.b[\mathbf{false}]$ by (ax4), (nex) and (prop), hence $\vdash a[\mathsf{keep}_b]$ by (ax17), i.e. $a[\mathsf{keep}_b] \in \mathcal{F}_i$ by proposition 4.11,1.
      Case: If $a \neq b$, then $a.b[\mathbf{false}] \in \mathcal{F}_i$ because of the completeness of $\mathcal{F}_i$. Assuming $\neg a[\mathsf{keep}_b] \in \mathcal{F}_i$ would imply $a.b\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$ by the definition of θ_{14}, in contradiction to the assumption. Hence, it holds that $a[\mathsf{keep}_b] \in \mathcal{F}_i$.

   Case: $t_i{\downarrow}a.b = t_{i+1}{\downarrow}a.b \neq \mathsf{empty}$, that is, $a.b\langle\mathbf{true}\rangle, a.b\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$ and also $t_i{\downarrow}b = t_i{\downarrow}a.b$ as well as $t_{i+1}{\downarrow}b = t_{i+1}{\downarrow}a.b$, hence in particular $t_i{\downarrow}b = t_{i+1}{\downarrow}b$. Since $\mathsf{keep}_b \in \tau\kappa(\mathcal{F}_i)$, it follows by 1. that $\mathsf{keep}_b \in \mathcal{F}_i$. By the definition of θ_{15} and since $a.b\langle\mathbf{true}\rangle \in \mathcal{F}_{i+1}$ we have $\neg a[\mathsf{keep}_b] \notin \mathcal{F}_i$, hence $a[\mathsf{keep}_b] \in \mathcal{F}_i$. ■

Theorem 4.27 *Let $\mathcal{F}$ be a finite and consistent set of pMLTL-formulas and let $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ be the run defined as described above. Let $i \in \omega$. Then the following holds:*

1. *If $F \in \tau\kappa(\mathcal{F}_i)$, then $F \in \mathcal{F}_i$ iff $\sigma|_i, \varepsilon \models F$.*
2. *If $a[F] \in \tau\kappa(\mathcal{F}_i)$, then $a[F] \in \mathcal{F}_i$ iff $a \notin \mathsf{N}_i$ or $\sigma|_i, a \models F$.*

Proof. The proof extends the proof of theorem 4.23. Note that also for the modified trees and all names $a, b \in \mathsf{nm}(\mathcal{F}_i)$ holds the following:

$$a \in \mathsf{N}_i \text{ iff } a\langle\mathbf{true}\rangle \in \mathcal{F}_i \quad \text{and} \quad a <_i b \text{ iff } b.a\langle\mathbf{true}\rangle \in \mathcal{F}_i \quad .$$

Using this observation, it is easy to see that the proof of theorem 4.23 works also for the modified run σ. The additional case of the move operators follows immediately from lemma 4.26. ■

With the aid of this theorem – more precisely, using the proof of the theorem – we can show that every satisfiable pMLTL-formula has a "finite" model in the sense as stated in the following corollary.

Corollary 4.28 *Let F be a pMLTL-formula. Let $\mathsf{nm}(F)$ denote the set of names occurring in F and $\mathsf{at}(F)$ the set of propositional variables occurring in F. Furthermore, for every $a \in \mathsf{nm}(F)$ let n_a be a name with $n_a \notin \mathsf{nm}(F)$ and $n_a \neq n_b$ for $a \neq b$. If F is satisfiable, then there exists a run $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ with $\mathsf{N}_i \subseteq \mathsf{nm}(F) \cup \{n_a | a \in \mathsf{nm}(F)\}$ and $\lambda_i : \mathsf{N}_i^{\varepsilon} \to 2^{\mathsf{at}(F)}$ for which $\sigma, \varepsilon \models F$ holds.*

Proof. Let F be satisfiable. Because of the soundness of Σ_{MLTL} holds $\not\vdash \neg F$, that is, the set $\{F\}$ is consistent. By (the proof of) theorem 4.27, there is a model of F as described above. ■

Chapter 5

Model Checking & Decidability

5.1 Background

In this chapter we explore the model checking problem (for finite state mobile systems) and the decidability problem for propositional MLTL.

The model checking problem is to decide for a given system $\mathcal{M}$ and a formula F whether for all runs σ of $\mathcal{M}$ holds $\sigma \models F$. One well established method to solve this problem makes use of automata-theory. Runs of a finite state system can be regarded as infinite words over a finite alphabet that consists of the system's possible states. Assume now that the system we have to check is given as an automaton $\mathcal{A}_{\mathcal{M}}$. Further, assume that $\mathcal{A}_F$ is an automaton that accepts exactly the models of the formula F, that is, $\sigma \in \mathcal{L}(\mathcal{A}_F)$ iff $\sigma \models F$. Then solving the model checking problem is equivalent with deciding whether the language $\mathcal{L}(\mathcal{A}_{\mathcal{M}}) \cap \mathcal{L}(\mathcal{A}_{\neg F})$, that is, the language $\mathcal{L}(\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg F})$ is empty (where $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\neg F}$ denotes a product automaton that accepts exactly the words that are accepted by both automata). Hence, the model checking problem is reduced to the non-emptiness problem for appropriate automata. Comprehensive introductions to the field of model checking can be found for example in [12, 13].

The decidability problem means to decide for a given formula whether it is satisfiable, that is, whether it has a model. As we have shown in corollary 4.28,

propositional MLTL has the finite model property, that is, if a formula has some model, then it has also a model over a special finite alphabet. Consequently, if we are able to translate formulas into automata, the decidability problem for propositional MLTL can also be reduced to the non-emptiness problem: the formula F is satisfiable if and only if $\mathcal{L}(\mathcal{A}_F) \neq \emptyset$, where $\mathcal{A}_F$ is defined over an appropriate alphabet.

In the following we present a translation of propositional pMLTL-formulas into weak alternating automata. Our construction is based on the translation of LTL into alternating automata as given in [51].

5.2 Büchi automata

Finite automata running on finite words or finite trees are well known in connection with the theory of formal languages. If verification of nonterminating systems is concerned, it is useful to consider finite automata that run on infinite objects like infinite words or infinite trees. As the models of pMLTL can be regarded as infinite words over a set S of configurations, we will only consider automata on infinite words here.

In this section, we briefly describe non-deterministic *Büchi automata* and cite some well known results that we will need later to determine upper bounds on the complexity of different decision problems.

Definition 5.1 *A non-deterministic* Büchi automaton *over the finite alphabet* Σ *is given by a tuple* $\mathcal{A} = (\Sigma, Q, q_I, \delta, F)$ *with*

- Q *is a finite set of states*
- $q_I \in Q$ *is the initial state*
- $\delta : Q \times \Sigma \to 2^Q$ *is the transition function*
- $F \subseteq Q$ *is the accepting condition*

A run *of the automaton $\mathcal{A}$ on an infinite word $\sigma = s_0 s_1 \ldots \in \Sigma^\omega$ is an infinite sequence $q_0 q_1 \ldots \in Q^\omega$ of states such that $q_0 = q_I$ and that for all $i \in \omega$ it holds $q_{i+1} \in \delta(q_i, s_i)$.*

A run $\rho = q_0 q_1 \ldots \in Q^\omega$ is accepting *iff $F \cap \mathsf{Inf}(\rho) \neq \emptyset$, where $\mathsf{Inf}(\rho)$ denotes the set of states that occur in ρ infinitely often, that is,*

$$\mathsf{Inf}(\rho) = \{q \in Q | \forall i \in \omega \exists j \geq i : q_j = q\} \quad .$$

A word $\sigma \in \Sigma^\omega$ is accepted by $\mathcal{A}$ iff there is an accepting run of the automaton on σ.

The definition says that an accepting run has to pass one of the accepting states infinitely often. The language accepted by the automaton $\mathcal{A}$ will be denoted by $\mathcal{L}(\mathcal{A})$.

In many applications, it is an important question whether the language accepted by an automaton is empty. The following proposition cites the well known result that the non-emptiness problem for Büchi automata is decidable in time linear in the size of the automaton.

Proposition 5.2 *Let $\mathcal{A}$ be a Büchi automaton with n states. The question whether $\mathcal{L}(\mathcal{A})$ is empty can be decided in time $O(n)$, that is, it is linear in the size of the automaton.*

Another classical result is the construction for two Büchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$ a Büchi automaton that defines exactly the intersection of the two languages $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$. We do not present the construction, but only record its complexity.

Proposition 5.3 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be non-deterministic Büchi automata over the alphabet Σ with n_1 resp. n_2 states. There is a non-deterministic Büchi automaton $\mathcal{A}$ with $O(n_1 n_2)$ states for which it holds*

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \quad .$$

5.3 Alternating Automata on Infinite Words

For the translation of propositional MLTL into automata we will use *weak alternating automata*, first introduced by Muller et al. in [41]. In our presentation of weak alternating automata we follow [32], where, in contrast to most other definitions, runs of alternating automata are described in terms of dags with bounded width, instead of finitely branching trees.

Alternating automata combine existential (nondeterministic) and universal branching mode. As in universal automata, several states of the automaton may be active at the same time. Additionally, as in nondeterministic automata, when reading an input letter, the automaton can choose from different sets of states as successors. The different branching modes are given by the transition function that assigns to every state/letter pair a positive boolean expression over the set Q of the automaton's states. For example, $q_0 \vee (q_1 \wedge q_2)$ means that the automaton can choose between activating q_0 or both, q_1 and q_2 simultaneously.

Alternating automata whose transition function is given in disjunctive normal form, can be illustrated by hypergraphs. A hypergraph corresponding to a (weak) alternating automaton with state set $Q = \{q_I, q_1, q_2\}$, initial state q_I and the following transition function:

$$\begin{aligned}
&\delta(q_I, a) = \delta(q_I, b) = q_I \vee (q_1 \wedge q_2) && \delta(q_I, c) = \delta(q_I, d) = q_I \\
&\delta(q_1, a) = \delta(q_1, b) = q_1 \wedge q_2 && \delta(q_1, c) = \delta(q_1, d) = \textbf{false} \\
&\delta(q_2, a) = \delta(q_1, c) = q_2 && \delta(q_2, b) = \delta(q_2, d) = \textbf{true}
\end{aligned}$$

appears in figure 5.1. The numbers that appear in brackets next to the states indicate the *ranks* of the states. In weak alternating automata, every state has a rank, and transitions are not allowed to lead to a state of higher rank.

A hyper-edge labelled by a letter $a \in \Sigma$ indicates that on input a the automaton can simultaneously activate the states the hyper-edge leads to. For example, the above automaton in its initial state can activate on input a either q_I, or q_1 and q_2 simultaneously.

A *run* of a weak alternating automaton on an (infinite) input word $s_0 s_1 \ldots$ is, roughly speaking, an acyclic graph that arises if we follow the (hyper-)edges along

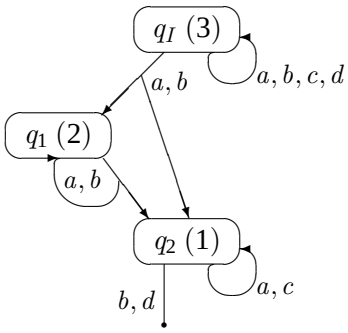


Figure 5.1: Illustration of a weak alternating automaton

the input word. Figure 5.2 shows (the first segment of) a possible run of the automaton given in fig. 5.1 on the infinite word $dcabba^{\omega}$.

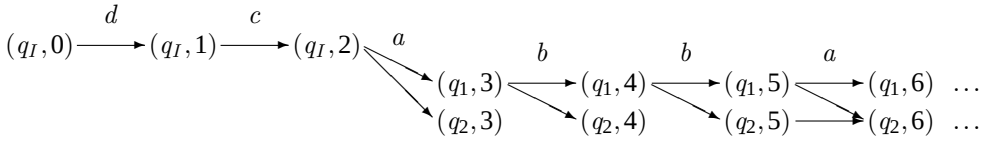


Figure 5.2: Run of an alternating automaton

Such a run graph is *accepting*, if all infinite paths in the graph satisfy the accepting condition of the automaton. In the case of weak alternating automata, this condition requires that the minimal rank occurring on the path be even.

For example, the run given in fig. 5.2 is not accepting, because the infinite path $\cdots(q_2,5)(q_2,6)\cdots$ gets trapped in state q_2 which has rank one. Figure 5.3 presents an accepting run of the same automaton (the one in fig. 5.1) on the word $aaab^{\omega}$. The only infinite path finally contains only state q_1 which has rank 2.

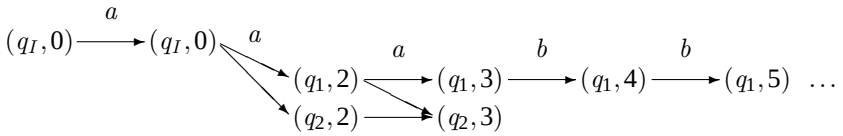


Figure 5.3: An accepting run

After these informal explanations let us introduce all the notions in a formal way.

For a finite set X, the set of *positive boolean expressions* over X is the set of all expressions built from the elements of the set X using $\wedge$ and $\vee$, plus the formulas **true** and **false**. This set is denoted by $\mathcal{B}^+(X)$. We say that a subset Y *satisfies* $\rho \in \mathcal{B}^+(X)$ iff the truth assignment that assigns **true** to the elements in Y and **false** to the elements in $X \setminus Y$ satisfies ρ. The set of subsets of X which satisfy a positive boolean expression $\rho \in \mathcal{B}^+(X)$ is denoted by $\mathrm{Mod}(\rho)$, the set of elements $x \in X$ occurring in a positive boolean expression $\rho \in \mathcal{B}^+(X)$ by $\mathsf{at}(\rho)$.

In the next proposition we list properties of such models. All of them are easy to prove.

Proposition 5.4 *Let X be a set and $\rho \in \mathcal{B}^+(X)$. The following holds:*

1. *If $S \in \mathrm{Mod}(\rho)$ and $S \subseteq S' \subseteq X$, then $S' \in \mathrm{Mod}(\rho)$.*
2. *If $S \in \mathrm{Mod}(\rho)$ and $\mathsf{at}(\rho) \subseteq S'$, then $S \cap S' \in \mathrm{Mod}(\rho)$.*
3. *If $S \in \mathrm{Mod}(\rho_1) \cap \mathrm{Mod}(\rho_2)$, then $S \in \mathrm{Mod}(\rho_1 \wedge \rho_2)$.*

Now we define weak alternating automata operating on infinite words over a finite alphabet.

Definition 5.5 *A* weak alternating automaton – WAA *for short – on infinite words is a tuple $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ where*

- *Σ is a finite alphabet*
- *Q is a finite set of states*
- *$q_I \in Q$ is the initial state*
- *$\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is the transition function*
- *$r : Q \to \omega$ is a function with the following property:*

$$\forall q, q' \in Q \forall s \in \Sigma : \; q' \in \mathsf{at}(\delta(q, s)) \Rightarrow r(q') \leq r(q)$$

The number $r(q)$ is called the rank *of q.*

We already gave an informal explanation of a run of a WAA. Here we present the precise definition.

Definition 5.6 *A* run *of an alternating automaton $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ on a word $\sigma = s_0 s_1 \ldots \in \Sigma^\omega$ is a directed acyclic graph (dag) $\mathcal{G} = (V, E, v_I)$ that satisfies the following conditions:*

- $V \subseteq Q \times \omega$
- $((q,i),(q',j)) \in E \Rightarrow j = i+1$
- $\{q' | ((q,i),(q',i+1)) \in E\} \in \text{Mod}(\delta(q, s_i))$
- $\{q' | ((q,i),(q',i+1)) \in E\} \subseteq \mathsf{at}(\delta(q, s_i))$
- $v_I = (q_I, 0)$
- $(q,0) \in V \Leftrightarrow q = q_I$

An infinite *path* of a dag $\mathcal{G}$ is a map $\pi : \omega \to Q \times \omega$ such that

- $\pi(0) = v_I$
- $\forall i \in \omega : (\pi(i), \pi(i+1)) \in E$

A run dag $\mathcal{G}$ is called *accepting* iff for all infinite paths $\pi : \omega \to Q \times \omega$ in $\mathcal{G}$ the minimal rank occurring in π, that is, $\min\{r(q) | \exists i < \omega : (q,i) = \pi(i)\}$, is even. A WAA $\mathcal{A}$ *accepts* the word $\sigma \in \Sigma^\omega$ iff there exists an accepting run of $\mathcal{A}$ on σ. Again, the language accepted by an automaton $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

Weak alternating automata can be translated into non-deterministic Büchi automata as stated in the next proposition.

Proposition 5.7 *Let $\mathcal{A}$ be a* WAA *with n states. There is a non-deterministic Büchi automaton $\mathcal{A}_B$ with $O(2^n)$ states and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_B)$.*

The translation is based on a subset construction by Miyano and Hayashi [40] and is exponential in the size of the automaton. For WAAs in our format the translation is described for example in [49].

The following proposition will often be used.

Proposition 5.8 *Let* $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ *be a* WAA *and* $\sigma = s_0 s_1 \ldots \in \Sigma^\omega$. *The following holds:*

1. *If* $\delta(q_I, s_0) =$ **true**, *then* $\sigma \in \mathcal{L}(\mathcal{A})$.
2. *If* $\delta(q_I, s_0) =$ **false**, *then* $\sigma \notin \mathcal{L}(\mathcal{A})$.

Proof. *To 1.:* As the empty set satisfies **true**, the dag consisting only of the root $(q_I, 0)$ is a run of $\mathcal{A}$ on any word beginning with s_0. This dag does not contain any infinite path and so it is accepting.

To 2.: As **false** does not have any model, there is no run of $\mathcal{A}$ on σ. ■

Now we prove several lemmas about the languages accepted by weak alternating automata. These lemmas will be helpful to prove the correctness of our translation.

The first lemma shows how to construct for two WAAs $\mathcal{A}_1$ and $\mathcal{A}_2$, an automaton that accepts exactly the words that are accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$. The construction is simple: the state space consists of the union of the state spaces and a "new" initial state q_I. For all "old" states, the ranks and the transition function are unchanged. As the automaton has to imitate both automata simultaneously, from the initial state it can perform the transitions that are allowed for both automata in their initial states. Technically, the transition function on q_I is the conjunction of the transition functions of the two automata on the initial states.

Lemma 5.9 *Let* $\mathcal{A}_i = (\Sigma, Q^i, q_I^i, \delta^i, r^i)$, *for* $i = 1, 2$, *be* WAA*s such that*

$$\forall q \in Q^1 \cap Q^2 \forall s \in \Sigma : \; (\delta^1(q,s) = \delta^2(q,s)) \wedge (r^1(q) = r^2(q)) \quad .$$

Further, let $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ *be defined as:*

- $Q = Q^1 \cup Q^2 \cup \{q_I\}$ *with* $q_I \notin Q^1 \cup Q^2$
- $\delta(q,s) = \begin{cases} \delta^i(q,s) & \text{if } q \in Q^i \\ \delta^1(q_I^1,s) \wedge \delta^2(q_I^2,s) & \text{if } q = q_I \end{cases}$
- $r(q) = \begin{cases} r^i(q) & \text{if } q \in Q^i \\ \max\{r^1(q_I^1), r^2(q_I^2)\} & \text{if } q = q_I \end{cases}$

Then $\mathcal{A}$ *is a* WAA *and the following holds:*

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \quad .$$

Proof. First note that δ and r are well defined because of the condition on the two automata that their rank- and transition functions agree on all common states. As the rank function is obviously "decreasing" by the definition of δ and r, it follows that $\mathcal{A}$ is a WAA. Now we have to prove two inclusions.
"$\subseteq$": Let $\mathcal{G} = (V, E, v_I)$ be an accepting run of $\mathcal{A}$ on $\sigma \in \Sigma^\omega$. Using this dag we construct an accepting run for $\mathcal{A}_i$ as follows: we replace the root $(q_I, 0)$ by $(q_I^i, 0)$ and take the sub-graph of $\mathcal{G}$ that contains only states in Q^i. (Additionally, we could remove all nodes that are not reachable from $(q_I^i, 0)$.)
More formally, let the dags $\mathcal{G}_i = (V^i, E^i, v_I^i)$, for $i = 1, 2$, be defined as follows:

- $v_I^i = (q_I^i, 0)$
- $V^i = \{v_I^i\} \cup \{(q,j) | (q,j) \in V, j \geq 1, q \in Q^i\}$
- $E^i = \{(v_I^i, (q,1)) | q \in Q^i, (v_I, (q,1)) \in E\} \cup$
  $\{((q,j),(q',j+1)) | j \geq 1, q, q' \in Q^i, ((q,j),(q',j+1)) \in E\}$

Figure 5.4 illustrates the definition with $q_I^2, q_2^2 \in Q^2$ and $q_1^1 \notin Q^2$.

We want to show that $\mathcal{G}_i$ is an accepting run of $\mathcal{A}_i$ on σ for $i = 1, 2$. In order to see that $\mathcal{G}_i$ is a run of $\mathcal{A}_i$ on σ, it suffices to prove the following:

$$\forall j \in \omega \forall (q,j) \in V^i : \{q' | ((q,j),(q',j+1)) \in E^i\} \in \text{Mod}(\delta^i(q, s_j)) \quad .$$

We prove the claim for $j = 0$ and $j \geq 1$ separately.

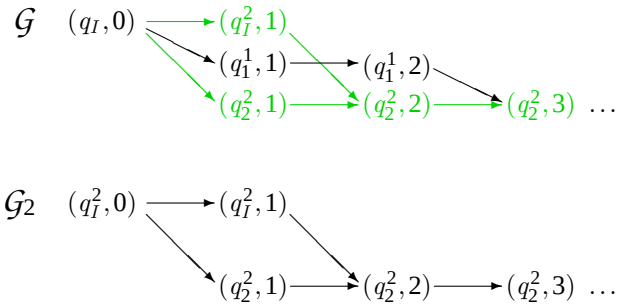


Figure 5.4: Construction of $\mathcal{G}_2$ from $\mathcal{G}$

Case: $j = 0$: Observe that $\{q|(v_I^i,(q,1)) \in E^i\} = \{q|(v_I,(q,1)) \in E\} \cap Q^i$. Since $\mathcal{G}$ is a run of $\mathcal{A}$ on σ, we know that $\{q|(v_I,(q,1)) \in E\} \in \mathrm{Mod}(\delta(q_I,s_0))$. On the other hand we know that $\mathrm{Mod}(\delta(q_I,s_0)) \subseteq \mathrm{Mod}(\delta^i(q_I^i,s_0))$ (cf. the definition of δ). Hence, $\{q|(v_I,(q,1)) \in E\} \in \mathrm{Mod}(\delta^i(q_I^i,s_0))$. Because of $\mathsf{at}(\delta^i(q_I^i,s_0)) \subseteq Q^i$ it follows that $\{q|(v_I,(q,1)) \in E\} \cap Q^i \in \mathrm{Mod}(\delta^i(q_I^i,s_0))$ by using prop. 5.4.

Case: $j \geq 1$: Note first that for every $q \in Q^i$ it holds $\delta^i(q,s_j) = \delta(q,s_j)$ and $\{q'|((q,j),(q',j+1)) \in E^i\} = \{q'|((q,j),(q',j+1)) \in E\} \cap Q^i$. As for all states $q \in Q^i$ we have that $\mathsf{at}(\delta^i(q,s_j)) \subseteq Q^i$, it follows by prop. 5.4,2 that $\{q'|((q,j),(q',j+1)) \in E^i\} \in \mathrm{Mod}(\delta^i(q,s_j))$.

Hence, $\mathcal{G}_i$ is a run of $\mathcal{A}_i$ on σ. It satisfies the weak acceptance condition, because every infinite path corresponds to a path in $\mathcal{G}$, and as $\mathcal{G}$ is accepting, it satisfies the accepting condition.

"$\supseteq$": Let $\mathcal{G}_i = (V^i,E^i,v_I^i)$ be an accepting run of $\mathcal{A}_i$ on σ, for $i = 1,2$. We construct an accepting run of $\mathcal{A}$ by "putting together" the two graphs. Formally, let the dag $\mathcal{G}$ be defined as follows:

- $v_I = (q_I,0)$
- $V = (V^1 \setminus \{v_I^1\}) \cup (V^2 \setminus \{v_I^2\}) \cup \{v_I\}$
- $E = \{(v_I,v')|(v_I^i,v') \in E^i \text{ for an } i \in \{1,2\}\} \cup$
  $\{(v,v')|(v,v') \in E^i \text{ for an } i \in \{1,2\}, v,v' \in V\}$

Figure 5.5 illustrates the construction of the run $\mathcal{G}$ based on the runs $\mathcal{G}_1$ and $\mathcal{G}_2$ of the original automata. We want to show that $\mathcal{G}$ is an accepting run of $\mathcal{A}$ on σ.

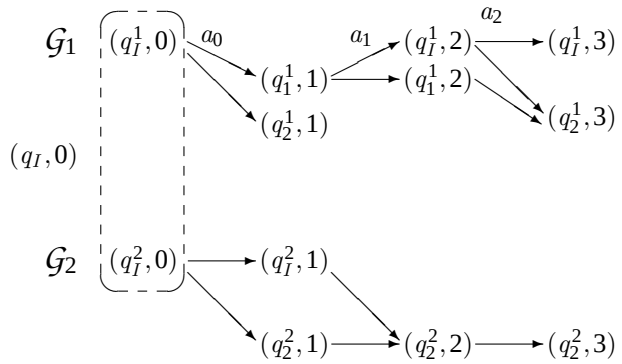


Figure 5.5: Run dag of the intersection automaton

First we have to prove that $\mathcal{G}$ is a run of $\mathcal{A}$ on σ, that is, that the set of successors of any vertex (q,j) is a model of $\delta(q,s_j)$. Formally, this means

$$\forall j \in \omega \forall (q,j) \in V : \{q' | ((q,j),(q',j+1)) \in E\} \in \text{Mod}(\delta(q,s_j)) \quad .$$

Again, we examine the cases $j = 0$ and $j \geq 1$ separately.

Case: $j = 0$: Let $S := \{q' | (v_I,(q',1)) \in E\}$ be the set of successors of the root of $\mathcal{G}$. Note that $S = \{q' | (v_I^1,(q',1)) \in E^1\} \cup \{q' | ((v_I^2,(q',1)) \in E^2\}$ and that $\delta(q_I,s_0) = \delta^1(q_I^1,s_0) \wedge \delta^2(q_I^2,s_0)$. Since $\mathcal{G}_i$ is a run of $\mathcal{A}_i$ on σ, it follows that $\{q' | (v_I^i,(q',1)) \in E^i\} \in \text{Mod}(\delta^i(q_I^i,s_0))$. By prop. 5.4,1 it follows that $S \in \text{Mod}(\delta^1(q_I^1,s_0)) \cap \text{Mod}(\delta^2(q_I^2,s_0))$. Hence, $S \in \text{Mod}(\delta(q_I,s_0))$ (cf. prop. 5.4,3).

Case: $j \geq 1$: Let $(q,j) \in V$ and let $S := \{q' | ((q,j),(q',j+1)) \in E\}$. Observe that $S = \{q' | ((q,j),(q',j+1)) \in E^1\} \cup \{q' | ((q,j),(q',j+1)) \in E^2\}$. Because of $j \geq 1$ it follows that $q \in Q^1$ or $q \in Q^2$. We assume w.l.o.g. that $q \in Q^1$. Thus, $\delta(q,s_j) = \delta^1(q,s_j)$. Since $\mathcal{G}_1$ is a run of $\mathcal{A}_1$ on σ, it holds

$$\{q' | ((q,j),(q',j+1)) \in E^1\} \in \text{Mod}(\delta^1(q,s_j)) \quad .$$

Therefore, $S \in \text{Mod}(\delta(q,s_j))$.

Altogether we obtain that $\mathcal{G}$ is a run of $\mathcal{A}$ on σ. Since every infinite path of $\mathcal{G}$ corresponds either to a path in $\mathcal{G}_1$ or in $\mathcal{G}_2$, this run is an accepting run. ■

The next lemma shows how to construct the union automaton for two weak alternating automata. It is similar to the previous one, the only difference is in the transition function. As the automaton should be able choose between the two automata, we allow a non-deterministic choice in the initial state q_I, that is, the transition function on q_I is the disjunction of the original transition functions on the respective initial states.

Lemma 5.10 *Let* $\mathcal{A}_i = (\Sigma, Q^i, q_I^i, \delta^i, r^i)$*, for* $i = 1, 2$*, be* WAA*s such that*

$$\forall q \in Q^1 \cap Q^2 \forall s \in \Sigma : \ (\delta^1(q,s) = \delta^2(q,s)) \wedge (r^1(q) = r^2(q)) \quad .$$

Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ *be defined as:*

- $Q = Q^1 \cup Q^2 \cup \{q_I\}$ *where* $q_I \notin Q^1 \cup Q^2$
- $\delta(q,s) = \begin{cases} \delta^i(q,s) & \text{if } q \in Q^i \\ \delta^1(q_I^1, s) \vee \delta^2(q_I^2, s) & \text{if } q = q_I \end{cases}$
- $r(q) = \begin{cases} r^i(q) & \text{if } q_I \neq q \in Q^i \\ \max\{r^1(q_I^1), r^2(q_I^2)\} & \text{if } q = q_I \end{cases}$

Then $\mathcal{A}$ *is a* WAA *and the following holds:*

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) \quad .$$

Proof. The proof is similar to the previous one. ■

Now we present a construction that will help us to translate formulas of the form $\Box F$. By the semantics of $\Box F$, formula F will be checked only as long as the current location of evaluation exists. Hence, the construction depends also on a set C. For the translation, this set will contain the configurations in which the current name does not appear.

Let us ignore the set C for a moment, and assume that we are given a WAA $\mathcal{A}$. Our goal is to construct an automaton $\mathcal{A}'$ that accepts a word σ iff $\mathcal{A}$ accepts all suffixes of σ. The construction works as follows: we take all the states of $\mathcal{A}$ plus a "fresh" initial state q_I'. The transition function on the old states is unchanged. The initial state can perform all transitions of the original initial state, but additionally each transition from q_I' has to activate q_I' again (in order to check the suffix). As an infinite repetition of the initial state is desired, we assign to q_I' an even rank. The ranks of the old states do not change.

The following lemma gives the formal definition of this construction.

Lemma 5.11 *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ be a* WAA *and let $C \subseteq \Sigma$. Let the automaton $\mathcal{A}' = (\Sigma, Q', q_I', \delta', r')$ be defined as follows:*

- $Q' = Q \dot\cup \{q_I'\}$
- $\delta'(q,s) = \begin{cases} \textbf{true} & \text{if } q = q_I' \text{ and } s \in C \\ \delta(q_I, s) \wedge q_I' & \text{if } q = q_I' \text{ and } s \notin C \\ \delta(q,s) & \text{if } q \neq q_I' \end{cases}$
- $r'(q) = \begin{cases} r(q) & \text{if } q \neq q_I' \\ 2\lceil r(q_I)/2 \rceil & \text{if } q = q_I' \end{cases}$

Then $\mathcal{A}'$ is a WAA *and the following holds:*

$$\mathcal{L}(\mathcal{A}') = \{\sigma = s_0 s_1 \ldots \in \Sigma^\omega | \forall i \in \omega : (\sigma|_i \in \mathcal{L}(\mathcal{A}) \vee \exists j \leq i : s_j \in C)\} \quad .$$

Proof. Since $r'(q_I) \leq r'(q_I')$, $\mathcal{A}'$ is a WAA (cf. also the definition of δ').
"$\subseteq$": Let $\mathcal{G}' = (V', E', v_I')$ be an accepting run of $\mathcal{A}'$ on σ and let $i \in \omega$ be an arbitrary natural number. We have to prove the following:

$$\sigma|_i \in \mathcal{L}(\mathcal{A}) \vee \exists j \leq i : s_j \in C \quad .$$

Assume that $\forall j \leq i : s_j \notin C$. Now we have to show that $\sigma|_i \in \mathcal{L}(\mathcal{A})$.
Let $V \subseteq Q \times \omega$ and $E \subseteq V \times V$ be the smallest sets, such that the following holds:

- $(q_I, 0) \in V$
- $\forall q \in Q : ((q'_I, i), (q, i+1)) \in E' \Rightarrow (q, 1) \in V \wedge ((q_I, 0), (q, 1)) \in E$
- $\forall j \geq 1 \forall q' \in Q : \big((q, j) \in V \wedge ((q, i+j), (q', i+j+1)) \in E'\big)$
  $\Rightarrow \big((q', j+1) \in V \wedge ((q, j), (q', j+1)) \in E\big)$

Then $\mathcal{G} = (V, E, (q_I, 0))$ is a dag. Observe that the assumption $\forall j \leq i : s_j \notin C$ implies that $(q'_I, i) \in V'$. Figure 5.6 illustrates the construction of $\mathcal{G}$ for given $\mathcal{G}'$.

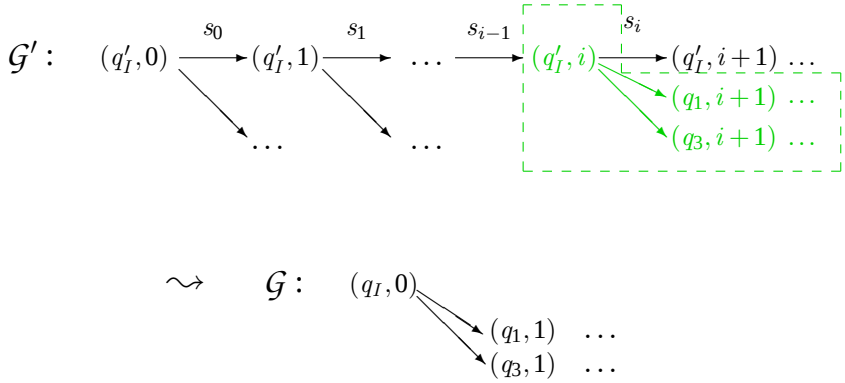


Figure 5.6: Construction of $\mathcal{G}$ from $\mathcal{G}'$

In a similar way as in the proof of lemma 5.9, we can show that $\mathcal{G}$ is a run of $\mathcal{A}$ on $\sigma|_i$. Since $\mathcal{G}$ is essentially a subgraph of $\mathcal{G}'$, and since $\mathcal{G}'$ satisfies the weak acceptance condition, $\mathcal{G}$ is an accepting run.

"$\supseteq$": Let L denote the set $\{s_0 s_1 \ldots \in \Sigma^\omega | \forall i \in \omega : (\sigma|_i \in \mathcal{L}(\mathcal{A}) \vee \exists j \leq i : s_j \in C)\}$. Let $\sigma = s_0 s_1 \ldots \in L$, that is, for all $i \in \omega$ either $\sigma|_i \in \mathcal{L}(\mathcal{A})$ or there is a $j \leq i$ such that $s_j \in C$ is true. We have to construct an accepting run of $\mathcal{A}'$ on σ. Let

$$m := \begin{cases} \min\{j | s_j \in C\} & \text{if } \exists j \in \omega : s_j \in C \\ \omega & \text{otherwise} \quad . \end{cases}$$

Since $\sigma \in L$, it holds $\sigma|_i \in \mathcal{L}(\mathcal{A})$ for all $i < m$. For $i < m$, let $\mathcal{G}_i = (V^i, E^i, v_I^i)$ be an accepting run of $\mathcal{A}$ on $\sigma|_i$. Let $V' \subseteq Q' \times \omega$ and $E' \subseteq V' \times V'$ be the smallest sets with the following properties:

- $(q'_I, 0) \in V'$

- $\forall j < m : (q'_I, j+1) \in V' \wedge ((q'_I, j), (q'_I, j+1)) \in E'$
- $\forall j < m \forall k \geq 1 : (q,k) \in V^j \Rightarrow (q, k+j) \in V'$
- $\forall j < m : (v^j_I, (q,1)) \in E^j \Rightarrow ((q'_I, j), (q, j+1)) \in E'$
- $\forall j < m \forall k \geq 1 : ((q,k),(q',k+1)) \in E^j \Rightarrow ((q, k+j), (q', k+j+1)) \in E'$

We give an illustration of this construction in fig. 5.7.

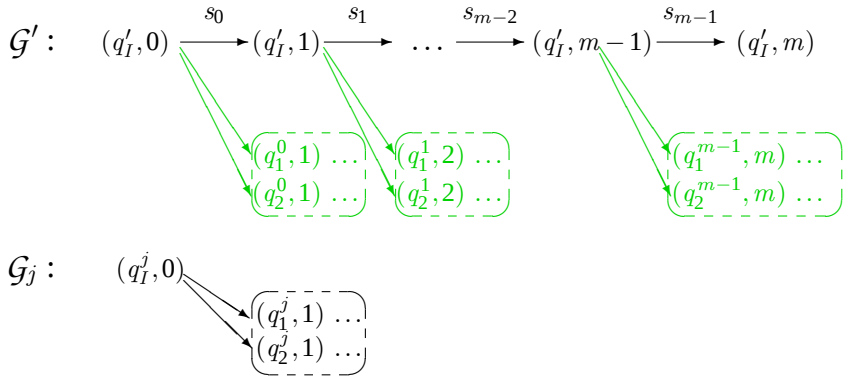


Figure 5.7: Construction of the dag $\mathcal{G}'$ from $\mathcal{G}_0, \ldots \mathcal{G}_m$

It is not hard to prove that $\mathcal{G}' = (V', E', (q'_I, 0))$ is a run of $\mathcal{A}$ on σ. In order to prove that it satisfies the weak acceptance condition we distinguish the cases $m < \omega$ and $m = \omega$.

Case: $m < \omega$: Every infinite path of $\mathcal{G}'$ corresponds to a path of one of the $\mathcal{G}_i$'s, thus it satisfies the acceptance condition. Hence, $\mathcal{G}'$ is an accepting run of $\mathcal{A}$ on σ.

Case: $m = \omega$: Let π be an infinite path in $\mathcal{G}'$. There are two possibilities. Either there is a $j < \omega$ such that $\pi(j) = (q, j)$ with $q \neq q'_I$ and j is minimal, or for all $j < \omega$ holds $\pi(j) = (q'_I, j)$. In the first case, (a suffix of) π corresponds to an infinite path in $\mathcal{G}_j$ and as such it satisfies the acceptance condition. In the second case, every node occurring in the path π has rank $2\lceil r(q_I)/2 \rceil$, which is even. Hence, π satisfies the acceptance condition also in this case, and it follows that $\mathcal{G}'$ is an accepting run of $\mathcal{A}'$ on σ. ■

The next lemma gives the construction for the dual case of the sometime operator.

Lemma 5.12 *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ be a* WAA *and let $C \subseteq \Sigma$. Let the automaton $\mathcal{A}' = (\Sigma, Q', q_I', \delta', r')$ be defined as follows:*

- $Q' = Q \dot\cup \{q_I'\}$
- $\delta'(q,s) = \begin{cases} \textbf{false} & \text{if } q = q_I' \text{ and } s \in C \\ \delta(q_I, s) \vee q_I' & \text{if } q = q_I' \text{ and } s \notin C \\ \delta(q,s) & \text{if } q \neq q_I' \end{cases}$
- $r'(q) = \begin{cases} r(q) & \text{if } q \neq q_I' \\ 2\lfloor r(q_I)/2 \rfloor + 1 & \text{if } q = q_I' \end{cases}$

Then $\mathcal{A}'$ is a WAA *and the following holds:*

$$\mathcal{L}(\mathcal{A}') = \{s_0 s_1 \ldots \in \Sigma^\omega | \exists i \in \omega : (\sigma|_i \in \mathcal{L}(\mathcal{A}) \wedge \forall j \leq i : s_j \notin C)\} \quad .$$

Proof. The claim follows in a similar way as in the previous case. ■

Finally we present the constructions for the weak and strong next-time operators, respectively. Also in these cases, the automata depend on a set C. In our translation, this set will control whether or not the current location of evaluation exists in the next configuration.

Lemma 5.13 *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ be a* WAA *and $C \subseteq \Sigma$. Further, let the automaton $\mathcal{A}' = (\Sigma, Q', q_I', \delta', r')$ be defined as follows:*

- $Q' = Q \dot\cup \{q_I'\}$
- $\delta'(q,s) = \begin{cases} \textbf{true} & \text{if } q = q_I' \text{ and } s \in C \\ q_I & \text{if } q = q_I' \text{ and } s \notin C \\ \delta(q,s) & \text{if } q \in Q \end{cases}$

- $r'(q) = \begin{cases} r(q) & \text{if } q \in Q \\ r(q_I) & \text{if } q = q'_I \end{cases}$

Then $\mathcal{A}'$ is a WAA *and the following holds:*

$$\mathcal{L}(\mathcal{A}') = \{s_0 s_1 \ldots \in \Sigma^\omega | s_0 \in C \vee \sigma|_1 \in \mathcal{L}(\mathcal{A})\} \quad .$$

Proof. It is easy to see that $\mathcal{A}'$ is a WAA, so we only prove the equality of the considered languages.
"$\subseteq$": Let $\mathcal{G}' = (V', E', v'_I)$ be an accepting run of $\mathcal{A}'$ on $\sigma = s_0 s_1 \ldots$. Assume that $s_0 \notin C$ holds. We have to show that $\sigma|_1$ is accepted by the automaton $\mathcal{A}$. Let the dag $\mathcal{G} = (V, E, v_I)$ be defined by

- $v_I = (q_I, 0)$
- $V = \{(q,j) | (q, j+1) \in V'\}$
- $E = \{((q,j),(q',j+1)) | ((q,j+1),(q',j+2)) \in E'\}$.

Note that it holds that $(q_I, 1) \in V'$ because of the assumption $s_0 \notin C$ and by the definition of δ'. Therefore, it also holds $v_I \in V$.

The graph $\mathcal{G}$ basically arises form $\mathcal{G}'$ by "cutting off" the root. A picture to illustrate the definition appears in fig. 5.8.

As every infinite path in $\mathcal{G}$ is a suffix of an infinite path in $\mathcal{G}'$, it follows that $\mathcal{G}$ is an accepting run.

"$\supseteq$": Let $\sigma = s_0 s_1 \ldots \in L$, that is, either $s_0 \in C$ or $\sigma|_1 \in \mathcal{L}(\mathcal{A})$. We have to show that σ is accepted by $\mathcal{A}$'. In the case that $s_0 \in C$ holds, the dag consisting only of a root $(q'_I, 0)$ is an accepting run of $\mathcal{A}'$ on σ. So we assume that $\sigma|_1 \in \mathcal{L}(\mathcal{A})$. Let $\mathcal{G} = (V, E, v_I)$ be an accepting run of $\mathcal{A}$ on the suffix $\sigma|_1$. We define the dag $\mathcal{G}' = (V', E', v'_I)$ as follows:

- $v'_I = (q'_I, 0)$
- $V' = \{v'_I\} \cup \{(q, j+1) | (q,j) \in V\}$

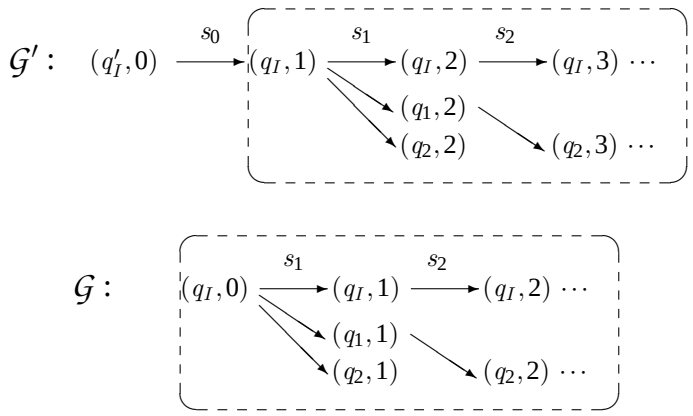


Figure 5.8: Connection between $\mathcal{G}$ and $\mathcal{G}'$

- $E' = \{(v_I', (q_I, 1))\} \cup \{((q, j+1), (q', j+2)) | ((q, j), (q', j+1)) \in E\}$

The graph $\mathcal{G}'$ is essentially $\mathcal{G}$ prefixed by the initial node $(q_I', 0)$. Again, fig. 5.8 shows the connection between $\mathcal{G}$ and $\mathcal{G}'$. It follows immediately from the definition of $\mathcal{A}'$ that $\mathcal{G}'$ is a run of $\mathcal{A}'$ on σ. As $\mathcal{G}$ satisfies the accepting condition, so does $\mathcal{G}'$. ■

The last lemma presents the dual case.

Lemma 5.14 *Let $\mathcal{A} = (\Sigma, Q, q_I, \delta, r)$ be a* WAA *and $C \subseteq \Sigma$. Let the automaton $\mathcal{A}' = (\Sigma, Q', q_I', \delta', r')$ be defined as follows:*

- $Q' = Q \dot\cup \{q_I'\}$
- $\delta'(q, s) = \begin{cases} \textbf{false} & \text{if } q = q_I' \text{ and } s \in C \\ q_I & \text{if } q = q_I' \text{ and } s \notin C \\ \delta(q, s) & \text{if } q \in Q \end{cases}$
- $r'(q) = \begin{cases} r(q) & \text{if } q \in Q \\ r(q_I) & \text{if } q = q_I' \end{cases}$

Then $\mathcal{A}'$ is a WAA *and the following holds:*

$$\mathcal{L}(\mathcal{A}') = \{\sigma = s_0 s_1 \ldots \in \Sigma^\omega | s_0 \notin C \wedge \sigma|_1 \in \mathcal{L}(\mathcal{A})\} \quad .$$

Proof. The proof is similar to the previous one. ■

5.4 Alternating Automaton for propositional MLTL

The aim of this section is to construct for a given pMLTL-formula F and a name $n \in \mathsf{N}^{\varepsilon}$ a weak alternating automaton $\mathcal{A}_{F,n}$ that accepts exactly the runs for which the formula F at node n holds. The states of the automaton will be pairs of the form (n, G) where G is a sub-formula of F and n is a name occurring in F or ε. Intuitively, an accepting run on σ should exist from a state (n, G) iff $\sigma, n \models G$ holds. For technical reasons, the alphabet for the automaton will consist of pairs of states instead of states as otherwise the transition function for the keep operator would be rather complicated. The transition function of the automaton is defined by induction on the structure of the formula.

In the following let $\mathcal{S}$ denote a finite set of states of the form (t, λ). For a formula F we write again $\mathsf{nm}(F)$ to denote the set of names occurring in F. Clearly, this set is finite for every formula.

For formulas G and F we write $G \leq F$ iff G is a sub-formula of F. The length of a formula A, that is, the number of occurrences of symbols in the formula, is denoted by $|A|$.

Now we present the translation of pMLTL-formulas in *positive normal form* into weak alternating automata. The set of pMLTL-formulas in positive normal form is given by:

$$v \mid \neg v \mid \mathsf{keep}_m \mid \neg\mathsf{keep}_m \mid F \wedge G \mid F \vee G \mid m[F] \mid m\langle F\rangle \mid \Box F \mid \Diamond F \mid \circ F \mid \odot F \quad .$$

The main feature of the positive normal form is that negations can only occur in front of propositional variables and the keep_m-operators.

Note that to every pMLTL-formula F one can construct an equivalent pMLTL-formula $\widetilde{F}$ in positive normal form with $|\widetilde{F}| \leq 2|F|$.

Definition 5.15 *For a pMLTL-formula F in positive normal form and for $a \in \mathsf{N}^{\varepsilon}$ the weak alternating automaton $\mathcal{A}_{F,a} = (\Sigma, Q, q_I, \delta, r)$ is defined as follows:*

- $\Sigma = \mathcal{S} \times \mathcal{S}$
- $Q = \{(n, G) | n \in \mathsf{nm}(F) \cup \{\varepsilon, a\}, G \leq F\}$
- $q_I = (a, F)$
- *Let* $s = (t, \lambda)$ *and* $s' = (t', \lambda')$. *The transition function* δ *is defined inductively.*
  - $\delta((n, v), (s, s')) = \begin{cases} \textbf{true} & \textit{if } v \in \lambda(n) \textit{ and } n \in \mathsf{N}_t^\varepsilon \\ \textbf{false} & \textit{otherwise} \end{cases}$
  - $\delta((n, \neg v), (s, s')) = \begin{cases} \textbf{false} & \textit{if } v \in \lambda(n) \textit{ and } n \in \mathsf{N}_t^\varepsilon \\ \textbf{true} & \textit{otherwise} \end{cases}$
  - $\delta((n, \mathsf{keep}_m), (s, s')) = \begin{cases} \textbf{true} & \textit{if } t{\downarrow}n.m = t'{\downarrow}n.m \\ \textbf{false} & \textit{otherwise} \end{cases}$
  - $\delta((n, \neg\mathsf{keep}_m), (s, s')) = \begin{cases} \textbf{false} & \textit{if } t{\downarrow}n.m = t'{\downarrow}n.m \\ \textbf{true} & \textit{otherwise} \end{cases}$
  - $\delta((n, G_1 \vee G_2), (s, s')) = \delta((n, G_1), (s, s')) \vee \delta((n, G_2), (s, s'))$
  - $\delta((n, G_1 \wedge G_2), (s, s')) = \delta((n, G_1), (s, s')) \wedge \delta((n, G_2), (s, s'))$
  - $\delta((n, m\langle G \rangle), (s, s')) = \begin{cases} \textbf{false} & \textit{if } m \not<_t n \\ \delta((m, G), (s, s')) & \textit{otherwise} \end{cases}$
  - $\delta((n, m[G]), (s, s')) = \begin{cases} \textbf{true} & \textit{if } m \not<_t n \\ \delta((m, G), (s, s')) & \textit{otherwise} \end{cases}$
  - $\delta((n, \Box G), (s, s')) = \begin{cases} \textbf{true} & \textit{if } n \notin \mathsf{N}_t^\varepsilon \\ \delta((n, G), (s, s')) \wedge (n, \Box G) & \textit{otherwise} \end{cases}$
  - $\delta((n, \Diamond G), (s, s')) = \begin{cases} \textbf{false} & \textit{if } n \notin \mathsf{N}_t^\varepsilon \\ \delta((n, G), (s, s')) \vee (n, \Diamond G) & \textit{otherwise} \end{cases}$
  - $\delta((n, \circ G), (s, s')) = \begin{cases} \textbf{true} & \textit{if } n \notin \mathsf{N}_{t'} \\ (n, G) & \textit{otherwise} \end{cases}$

– $\delta((n, \odot G), (s, s')) = \begin{cases} \textbf{false} & \textit{if } n \notin \mathsf{N}_{t'} \\ (n, G) & \textit{otherwise} \end{cases}$

- *The rank $r((n, G))$ of a state equals $\alpha(G)$ where $\alpha(G)$ is defined inductively as follows:*
  - $\alpha(v) = \alpha(\neg v) = \alpha(\mathsf{keep}_m) = \alpha(\neg \mathsf{keep}_m) = 0$
  - $\alpha(G_1 \wedge G_2) = \alpha(G_1 \vee G_2) = \max(\alpha(G_1), \alpha(G_2))$
  - $\alpha(m[G]) = \alpha(m\langle G \rangle) = \alpha(\circ G) = \alpha(\odot G) = \alpha(G)$
  - $\alpha(\Box G) = 2\lceil \alpha(G)/2 \rceil$
  - $\alpha(\Diamond G) = 2\lfloor \alpha(G)/2 \rfloor + 1$

For a sequence $\sigma = s_0 s_1 \ldots \in \mathcal{S}^\omega$ of states let $\tilde{\sigma} \in \Sigma^\omega$ denote the corresponding sequence of transitions:

$$\tilde{\sigma} = (s_0, s_1)(s_1, s_2)(s_2, s_3)\ldots$$

Now we can show that the presented translation is correct with respect to pMLTL in the sense as stated in the following theorem.

Theorem 5.16 *Let $\sigma = s_0 s_1 \ldots \in \mathcal{S}^\omega$ be a run, $n \in \mathsf{N}^\varepsilon$ and F a pMLTL-formula. Let $\mathcal{A}_{F,n}$ be defined as above. Then the following holds:*

$$\sigma, n \models F \Leftrightarrow \tilde{\sigma} \in \mathcal{L}(\mathcal{A}_{F,n}) \quad .$$

Proof. We prove the claim by induction on the structure of the formula F.

Case: $F = v \in \mathcal{V}$.

Only if: Assume that $\sigma, n \models v$. By the definition of the semantics of pMLTL-formulas and the definition of the transition function of the automaton $\mathcal{A}_{v,n}$ it follows that $\delta((n, v), (s_0, s_1)) = \textbf{true}$. The claim follows by prop. 5.8.

If: Assume $\tilde{\sigma} \in \mathcal{L}(\mathcal{A}_{v,n})$. By the definition of the transition function of $\mathcal{A}_{v,n}$ it follows that $\delta((n, v), (s_0, s_1))$ equals either **true** or **false**. Because

of prop. 5.8 it can not be **false**, so it is **true**. Therefore, $v \in \lambda_0(n)$ and $n \in \mathsf{N}_0^{\varepsilon}$ holds. By the definition of the semantics of pMLTL it follows that $\sigma, n \models v$ is true.

Case: $F = \neg v$, $F = \mathsf{keep}_m$ or $F = \neg\mathsf{keep}_m$. In all these cases the claim follows in a similar way as in the previous case.

Case: $F = G_1 \wedge G_2$. The claim follows by lemma 5.9.

Case: $F = G_1 \vee G_2$. The claim follows by lemma 5.10.

Case: $F = m\langle G\rangle$.

Only if: Assume $\sigma, n \models F$, i. e. $m <_o n$ and $\sigma, m \models G$. It follows by the induction hypothesis that $\tilde{\sigma} \in \mathcal{L}(\mathcal{A}_{G,m})$, i.e. there is an accepting run $\mathcal{G}$ of $\mathcal{A}_{G,m}$ on $\tilde{\sigma}$. We obtain an accepting run of $\mathcal{A}_{F,n}$ on $\tilde{\sigma}$ by replacing the root $((m, G), 0)$ of $\mathcal{G}$ by $((n, F), 0)$.

If: Let $\mathcal{G}$ be an accepting run of $\mathcal{A}_{F,n}$ on $\tilde{\sigma}$. It follows by prop. 5.8 and by the definition of the transition function that $m <_0 n$ and that we can construct an accepting run of $\mathcal{A}_{G,m}$ on $\tilde{\sigma}$ by replacing the root of $\mathcal{G}$ with $((m, G), 0)$. The induction hypothesis implies that $\sigma, m \models G$. Since $m <_0 n$, we obtain $\sigma, n \models F$.

Case: $F = m[G]$. The proof of this case is similar as in the previous case.

Case: $F = \Box G$. Let $C = \{((t,\lambda),(t',\lambda'))|n \notin \mathsf{N}_t\}$. The claim follows by lemma 5.11.

Case: $F = \Diamond G$. With $C = \{((t,\lambda),(t',\lambda'))|n \notin \mathsf{N}_t\}$, the claim follows by lemma 5.12.

Case: $F = \circ G$. Let $C = \{((t,\lambda),(t',\lambda'))|n \notin \mathsf{N}_{t'}\}$. The claim follows by lemma 5.13.

Case: $F = \odot G$. Let $C = \{((t,\lambda),(t',\lambda'))|n \notin \mathsf{N}_{t'}\}$. The claim follows by lemma 5.14. ■

We give some examples to illustrate the translation. The first example, given in fig. 5.9, is the automaton for formula $\Diamond m\langle\Box v\rangle$ at the root. According to the definition, the initial node is $(\varepsilon, \Diamond m\langle\Box v\rangle$ and it has rank 1. An edge labelled by

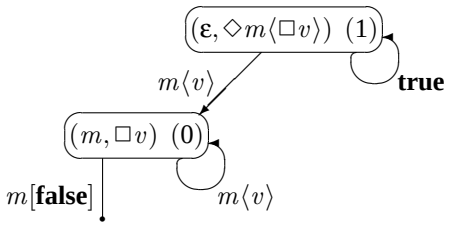


Figure 5.9: Automaton $\mathcal{A}_{\Diamond m\langle\Box v\rangle,\varepsilon}$

a (pure spatial) formula F indicates that the transition is possible exactly for the configuration pairs (s, s') with $s \models F$.

If $m\langle v\rangle$ holds in the current configuration, then we have the choice between activating the initial node again ($m\langle v\rangle$ holds now, but maybe not $m\langle\Box v\rangle$), or activating node $(m, \Box v)$ (to prove that $m[\Box v]$ holds for the suffix). From $(m, \Box v)$, if m[**false**] holds, then no state is activated, if $m\langle v\rangle$ holds, then the same state is activated again (as in this case m occurs in the current tree, according to the definition we have to prove that $\Box v$ holds at m also for the suffix). In the remaining case that $m\langle\neg v\rangle$ holds, the automaton does not accept the run.

If $m\langle v\rangle$ does not hold, then $m\langle\Box v\rangle$ cannot hold for the current input σ, so the initial node is activated again in order to check whether it holds for the suffix $\sigma|_1$.

The rank 1 of the original formula reflects the fact that activating the initial state again and again (i.e. postponing the proof of the eventually formula) should not lead to an accepting run.

The second example, depicted in fig. 5.10, is the automaton that checks whether $\Diamond\Box(v \wedge m\langle\Diamond w\rangle)$ holds at location n.

5.5 Applications to decision problems

As already mentioned, theorem 5.16 enables us to solve the model checking and the satisfiability problem for propositional MLTL.

In order to state a theorem about the model checking problem we introduce the

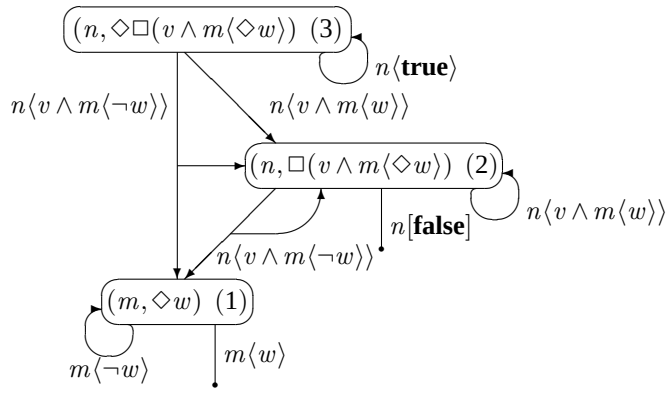


Figure 5.10: Automaton $\mathcal{A}_{\Diamond\Box(v\wedge m\langle\Diamond w\rangle),n}$

notion of a *spatial transition system*.

Definition 5.17 *Let* N *be a denumerable set of names and* $\mathcal{V}$ *a denumerable set of propositional variables. A* spatial transition system *is a tuple* $\mathcal{M} = (S, R, s_I)$ *with*

- $S \subseteq \{(t,\lambda) | t = (\mathsf{N}_t, <_t)$, *finite tree over* $\mathsf{N}, \lambda : \mathsf{N}_t^{\varepsilon} \to 2^{\mathcal{V}}\}$ *is the set of states.*
- $R \subseteq S \times S$ *is the transition relation.*
- $s_I \in S$ *is the initial state of* $\mathcal{M}$.

A *finite state spatial transition system* is a spatial transition system $\mathcal{M} = (S, R, s_I)$ with $|S| < \omega$.

Note that such a finite state transition system can be regarded as a Büchi automaton over the alphabet $S \times S$, with state space S and with trivial acceptance condition. We will denote this corresponding automaton by $\mathcal{A}_{\mathcal{M}}$.

Theorem 5.18 (Model Checking) *Let* F *be a propositional MLTL-formula in positive normal form and* $\mathcal{M} = (S, R, s_I)$ *a finite state spatial transition system. The question, whether for all runs* σ *of* $\mathcal{M}$ *holds* $\sigma, \varepsilon \models F$*, can be solved in time* $O(|S| * 2^{|F|^2})$.

Proof. First note that for a pMLTL-formula in positive normal form it is easy to find a formula G in positive normal form equivalent to $\neg F$ and of (essentially) the same length as F: one only needs to dualise all operators. Let G be such a formula and let n denote the length $|F|$ of F.

Now we construct the automaton $\mathcal{A}_{G,\varepsilon}$ as defined in theorem 5.16. Its state space is a subset of $(\mathsf{nm}(G) \cup \{\varepsilon\}) \times \mathrm{SF}(G)$, where $\mathrm{SF}(G)$ denotes the set of sub-formulas of G. As the set of names occurring in G as well as the set of its sub-formulas are bounded by the size of F, this set is at most of size $O(n^2)$. By prop. 5.7, $\mathcal{A}_{G,\varepsilon}$ can be translated into a Büchi automaton $\mathcal{A}_B$ of size exponential in the size of $\mathcal{A}_{G,\varepsilon}$. By prop. 5.3, one can construct a Büchi automaton $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_B$ with $O(|S| * 2^{n^2})$ states and

$$\mathcal{L}(\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_B) = \mathcal{L}(\mathcal{A}_{\mathcal{M}}) \cap \mathcal{L}(\mathcal{A}_B) = \big(\mathcal{L}(\mathcal{A}_{\mathcal{M}}) \cap \mathcal{L}(\mathcal{A}_{G,\varepsilon})\big) \quad .$$

This language contains exactly the runs of $\mathcal{A}_{\mathcal{M}}$ that satisfy G, that is, the runs that do not satisfy F. Hence, all runs of $\mathcal{M}$ have the property described by F iff this language is empty. As the non-emptiness of Büchi automata can be checked in time linear in the size of the automaton, the proof is finished. ■

Using the translation, we also can show the decidability of propositional MLTL:

Theorem 5.19 (Satisfiability) *Let F be a pMLTL-formula. The question, whether F is satisfiable, can be solved in time $O(2^{|F|^2})$.*

Proof. By corollary 4.28, F is satisfiable iff there is a run

$$\sigma = ((\mathsf{N}_0, <_0), \lambda_0)((\mathsf{N}_1, <_1), \lambda_1) \ldots$$

such that $\mathsf{N}_i \subseteq \mathsf{nm}(F) \cup \{n_a | a \in \mathsf{nm}(F)\}$, $\lambda_i : \mathsf{N}_i^{\varepsilon} \to 2^{\mathsf{at}(F)}$ and $\sigma, \varepsilon \models F$. Hence, it follows by theorem 5.16 that F is satisfiable iff $\mathcal{L}(\mathcal{A}_{F,\varepsilon}) \neq \emptyset$ where the automaton $\mathcal{A}_{F,\varepsilon}$ is defined as in definition 5.15 over the alphabet $\mathcal{S} \times \mathcal{S}$ with

$$\mathcal{S} = \big\{(t, \lambda) | \mathsf{N}_t = \mathsf{nm}(F) \cup \{n_a | a \in \mathsf{nm}(F)\}, \lambda : \mathsf{N}_t^{\varepsilon} \to 2^{\mathsf{at}(F)} \quad .\big\}$$

As the state space of $\mathcal{A}_{F,\varepsilon}$ is bounded by $|F|^2$, the assertion follows from the fact that the non-emptiness problem for weak alternating automata can be solved in time exponential in the size of the automaton (via a translation to a Büchi automaton with size exponential in the size of the WAA). ■

Chapter 6

Extensions of MTLA

This chapter discusses limitations of the logic MTLA introduced so far in the context of the dynamic creation of agents. We suggest possible extensions to allow to talk about dynamically created agents. The first extension, motivated and presented in sec. 6.2 and sec. 6.3, introduces a "rigid" quantifier for names. The second extension given in sec. 6.4 is more radical – flexible quantification over sets of names is allowed. This enables us to describe the hiding of dynamically created agents. We would like to point out that the properties of the proposed new quantifiers and the benefits they offer are not elaborated with the thoroughness as in the case of the operators presented earlier. They are rather ad hoc suggestions to handle the problem of the dynamic creation of mobile agents.

6.1 Dynamic creation of k agents

In section 2.2 we have presented an MTLA-specification of a simple agent that collects flights on behalf of a user (cf. fig. 2.3, p. 11). As a modification of this example, we could consider a travel agent that collects offers for flights and hotels in a network for a set of potential destinations, but this time an agent that does not work alone. Every time it finds a flight to some destination d, the agent produces a new agent to collect offers for hotels in d. These agents deliver their collected

offers to the home location by changing variable $home.res_h$. The main agent itself collects flights and on return it puts the collected flights into $home.res_f$.

If we know in the beginning how many hotel agents are going to be created, we can specify the system in MTLA without any further extension. Assume that at most k agents will be needed. Assume further that $hag_1, \ldots, hag_k$ are pairwise distinct names with $hag_i \notin Net$ and $hag_i \neq ag$ for all $i \in \{1, \ldots, k\}$. Figure 6.1 shows the overall specification, using the actions defined in well as the definition of the hotel agents in fig. 6.4.

$$
\begin{aligned}
Init1 &\equiv \wedge\, home.ag\langle \textbf{true} \rangle \wedge ag.ctl = \text{“idle”} \\
&\quad \wedge \textstyle\bigwedge_{i=1}^{k} hag_i[\textbf{false}] \\
Network &\equiv \ldots \\
vars &\equiv \langle ag.ctl, ag.dests, ag.time, ag.rest, ag.found, ag.sent \rangle \\
vars_{home} &\equiv \langle home.res_f, home.res_h \rangle \\
IDynAgent1 &\equiv \wedge\, Init1 \\
&\quad \wedge \textstyle\bigwedge_{n \in Net} \Box \big[\bigvee_{m \in Net} DynMove1_{n,m} \big]_{-n.ag} \\
&\quad \wedge \Box \big[DynHomeActs1 \vee \bigvee_{n \in Net} DynActions1_n \big]_{vars, vars_{home}} \\
&\quad \wedge \textstyle\bigwedge_{i=1}^{k} \Box \big[Create1_n(hag_i) \big]_{+hag_i} \\
\\
DynAgent1 &\equiv \exists\, hag_1 \ldots \exists\, hag_k : IDynAgent1
\end{aligned}
$$

Figure 6.1: Dynamically created agents of bounded number

Initially, no one of the hotel agents should exists yet, expressed by the conjunct

$$\bigwedge_{i=1}^{k} hag_i[\textbf{false}]$$

of the formula $Init1$. The possible actions at the home location are similar to those of the original $FlightAgent$: the agent can be sent to search for offers, expressed by $DynPrep1(ds, t)$ where ds is now a list of possible destinations and t a time period. However, we now have the additional variables $dests$, for the possible destinations, $rest$, that contains a destination d iff the agent still has to look for a flight to d, and $sent$ that contains a destination d iff a hotel agent has already

$$
\begin{aligned}
DynPrep1(ds,t) \;\equiv\;& \wedge\ home.ag\langle\textbf{true}\rangle \wedge \circ home.ag\langle\textbf{true}\rangle \\
& \wedge\ ag.ctl = \text{“idle”} \wedge ag.ctl' = \text{“busy”} \\
& \wedge\ ag.dests' = ds \wedge ag.dest' = \mathsf{hd}(ds) \wedge ag.time' = t \\
& \wedge\ ag.rest' = \mathsf{tl}(ds) \wedge ag.found' = \emptyset \wedge ag.sent' = \emptyset \\
& \wedge\ \text{UNCHANGED}\ vars_{home} \\
DynGetFlight1_n \;\equiv\;& \wedge\ n.ag\langle\textbf{true}\rangle \wedge \circ n.ag\langle\textbf{true}\rangle \\
& \wedge\ ag.ctl = \text{“busy”} \wedge \langle ag.dest, ag.time\rangle \in n.flights \\
& \wedge\ ag.found' = ag.found \cup \\
& \qquad \big\{[loc : n.id, \\
& \qquad\ dest : ag.dest, \\
& \qquad\ time : ag.time, \\
& \qquad\ fl : getFlight(\langle ag.dest, ag.time\rangle, n.flights)]\big\} \\
& \wedge\ \text{UNCHANGED}\ ag.ctl, ag.time, ag.dests, ag.dest \\
& \wedge\ \text{UNCHANGED}\ ag.rest, ag.sent, vars_{home} \\
Create1_n(hag) \;\equiv\;& \wedge\ n.ag\langle\textbf{true}\rangle \wedge \circ n.ag\langle\textbf{true}\rangle \wedge ag.ctl = \text{“busy”} \\
& \wedge\ ag.dest \notin ag.sent \wedge \exists f \in ag.found : f.dest = ag.dest \\
& \wedge\ ag.sent' = ag.sent \cup \{ag.dest\} \\
& \wedge\ hag[\textbf{false}] \wedge \circ n.hag\langle\textbf{true}\rangle \\
& \wedge\ \circ HAgent(hag, \langle ag.dest, ag.time\rangle) \\
& \wedge\ \text{UNCHANGED}\ ag.ctl, ag.time, ag.dests, ag.dest \\
& \wedge\ \text{UNCHANGED}\ ag.rest, ag.found, vars_{home} \\
NewItem1_n \;\equiv\;& \wedge\ n.ag\langle\textbf{true}\rangle \wedge \circ n.ag\langle\textbf{true}\rangle \\
& \wedge\ \vee\ ag.dest \in ag.sent \\
& \quad\ \vee\ \neg\exists f \in n.flights : f.dest = ag.dest \\
& \wedge\ ag.rest \neq \langle\rangle \wedge ag.rest' = \mathsf{tl}(ag.rest) \wedge ag.dest' = \mathsf{hd}(ag.rest) \\
& \wedge\ \text{UNCHANGED}\ ag.ctl, ag.dests, ag.found, ag.sent, ag.time \\
& \wedge\ \text{UNCHANGED}\ vars_{home}
\end{aligned}
$$

Figure 6.2: Actions of the first travel agent, part 1

been created to look for a hotel in d. Further, the agent can deliver the offers it has found, described by $DynDeliver1$ that is essentially like $Deliver$ in $FlightAgent$, but it puts the collected offers into variable res_f instead of res. Finally, one of the hotel agents may come home, as expressed by

$$\bigvee_{i=1}^{k} DynRcv1(hag_i) \quad .$$

The actions $DynRcv1(hag)$ are similar to $DynDeliver1$, but they modify res_h instead of res_f to indicate that an offer for a *hotel* has been delivered. Furthermore, agent hag is destroyed afterwards. This has the advantage that in principle the name can be reused to create a new agent.

At a network node n, there are four different kinds of actions. The agent can take an offer by performing $DynGetFlight1_n$, that is essentially the same as the action $GetFlight_n$ of $FlightAgent$, but it specifies more precisely the structure of the collected offers. The second possibility is to change the item the agent is looking for. This is described by formula $NewItem1_n$. It is executed if either ag has sent a hotel agent to look for a hotel in the current destination ($ag.dest \in ag.sent$) or if there is no flight offer at the current location to the current destination, and if the agent still has some destinations to check for flights ($ag.rest \neq \langle\rangle$). The current destination is set to the first one of the remaining destinations ($ag.dest' = \mathsf{hd}(ag.rest)$) and the old destination is removed from $ag.rest$ ($ag.rest' = \mathsf{tl}(ag.rest)$). The third kind of action is the moving of an agent from one site to another ($DynMove1_{n,m}$). It can be executed as soon as the hotel offers for all destinations have been checked at the current location ($ag.rest = \langle\rangle$). Apart from moving the agent it also resets the variables $ag.dest$ to $\mathsf{hd}(ag.dests)$ and $ag.rest$ to $\mathsf{tl}(ag.dests)$. Finally, a hotel agent can be created, as described by $Create1_n(hag)$. It has the preconditions that hag does not exist yet, that no hotel agent is in charge with the current destination, and that a flight to this destination has been found. In this case, an agent that satisfies specification $HAgent(hag, \langle ag.dest, ag.time\rangle)$ is created. Note that we make use of MTLA's feature of allowing arbitrary temporal formulas to describe transitions. Further, $ag.dest$ is added to $ag.sent$. This is done in order to remember that there is already a hotel agent in charge with $ag.dest$.

$$
\begin{aligned}
DynMove1_{n,m} \;\equiv\;& \wedge\, n.ag\langle \textbf{true}\rangle \wedge \circ m.ag\langle \textbf{true}\rangle \\
& \wedge\, ag.ctl = \text{“busy”} \wedge ag.rest = \langle\rangle \\
& \wedge\, ag.rest' = \mathsf{tl}(dests) \wedge \mathsf{keep}_{ag} \\
& \wedge\, ag.dest' = \mathsf{hd}(dests) \\
& \wedge\, \text{UNCHANGED } ag.ctl, ag.dests.ag.found, ag.time \\
& \wedge\, \text{UNCHANGED } ag.sent, vars_{home} \\
DynDeliver1 \;\equiv\;& \wedge\, home.ag\langle \textbf{true}\rangle \wedge \circ home.ag\langle \textbf{true}\rangle \\
& \wedge\, ag.ctl = \text{“busy”} \wedge ag.ctl' = \text{“idle”} \\
& \wedge\, home.res_f' = ag.found \\
& \wedge\, \text{UNCHANGED } home.res_h \\
DynRcv1(hag) \;\equiv\;& \wedge\, home.hag\langle \textbf{true}\rangle \wedge \circ hag[\textbf{false}] \\
& \wedge\, home.res_h' = home.res_h \cup hag.found \\
& \wedge\, \text{UNCHANGED } ag.ctl, ag.item, ag.dests, ag.rest \\
& \wedge\, \text{UNCHANGED } ag.found, ag.sent, home.res_f \\
DynHomeActs1 \;\equiv\;& \vee\, (\exists ds, t : DynPrep1(ds, t)) \\
& \vee\, DynDeliver1 \\
& \vee\, \bigvee_{i=1}^{k} DynRcv1(hag_i) \\
DynActions1_n \;\equiv\;& \vee\, DynGetFlight1_n \vee NewItem1_n \\
& \vee\, \bigvee_{m \in Net} DynMove1_{n,m} \\
& \vee\, \bigvee_{i=1}^{k} Create1_n(hag_i) \\
vars \;\equiv\;& \langle ag.ctl, ag.dests, ag.item, ag.rest, ag.found, \\
& ag.sent, home.res_f, home.res_h\rangle
\end{aligned}
$$

Figure 6.3: Actions of the first travel agent, part 2

Specification $HAgent(hag, \langle d, t\rangle)$ is depicted in fig. 6.4. It is – up to names and names of variables – very similar to the specification of the $FlightAgent$. One important difference is that it does not claim that variable res_h can only be modified by the transitions performed by the particular hotel agent hag, because any of the hotel agents can do this. The control over res_h is taken by specification $DynAgent1$ of the overall system.

The specification $IDynAgent1$ in fig. 6.1 is the inner specification of the system. It puts together the actions described above and additionally requires that the

$$
\begin{aligned}
HInit((d,t),k) \;\equiv\; & \wedge\, hag\langle \textbf{true}\rangle \wedge hag.ctl = \text{“busy”} \\
& \wedge\, hag.found = \emptyset \wedge hag.item = \langle d,t\rangle \\
GetHotel_n \;\equiv\; & \wedge\, n.hag\langle \textbf{true}\rangle \wedge \circ n.hag\langle \textbf{true}\rangle \\
& \wedge\, hag.ctl = \text{“busy”} \wedge hag.item \in n.hotels \\
& \wedge\, hag.found' = hag.found \\
& \qquad\qquad \cup \{(n.id, getHotel(hag.item, n.hotels))\} \\
& \wedge\, \text{UNCHANGED}\ hag.ctl, hag.item \\
HMove_{n,m} \;\equiv\; & \wedge\, n.hag\langle \textbf{true}\rangle \wedge \circ m.hag\langle \textbf{true}\rangle \\
& \wedge\, hag.ctl = \text{“busy”} \wedge \mathsf{keep}_{hag} \\
& \wedge\, \text{UNCHANGED}\ hag.ctl, hag.item, hag.found \\
HDeliver \;\equiv\; & \wedge\, home.hag\langle \textbf{true}\rangle \wedge \circ hag[\textbf{false}] \\
& \wedge\, hag.ctl = \text{“busy”} \wedge hag.ctl' = \text{“idle”} \\
& \wedge\, home.res_h' = home.res_h \cup hag.found \\
vars_h \;\equiv\; & \langle hag.ctl, hag.found, hag.item\rangle \\
HAgent(hag,(d,t)) \;\equiv\; & \wedge\, HInit((d,t)) \\
& \wedge\, \textstyle\bigwedge_{n\in Net} \Box\big[\bigvee_{m\in Net} HMove_{m,n}\big]_{-n.hag} \\
& \wedge\, \Box\big[HDeliver \vee \bigvee_{n\in Net} GetHotel_n\big]_{vars_h}
\end{aligned}
$$

Figure 6.4: MTLA-Specification of the Hotel Agent

agents hag_i can only arise by creation by ag. This requirement is expressed by the conjunct

$$\bigwedge_{i=1}^{k} \Box\big[\bigvee_{n\in Net} Create1_n(hag_i)\big]_{+hag_i} \quad .$$

This inner specification, however, does not describe the dynamic creation of agents as all the names are explicitely given. So as the last step, we hide the names of all these agents, and obtain the overall specification

$$\exists\, hag_1, \ldots, \exists\, hag_k : IDynAgent1 \quad .$$

6.2 Dynamic creation of arbitrarily many agents

The method to create agents described in the previous section works only if we can fix the maximal number of agents to be created. The natural way to modify specification $DynAgent1$ to allow to create arbitrarily many sub-agents would be to replace the disjunctions in $\bigvee_{i=1}^{k} DynRcv1(hag_i)$ and $\bigvee_{i=1}^{k} Create1_n(hag_i)$ by existential quantifiers. This would lead to formulas like $\exists\, hag : DynRcv1(hag)$ and $\exists\, hag : Create1_n(hag)$. Unfortunately, this does not work. The problem with this specification is that it does not guarantee authenticity of the returning hotel agents. Note that in $DynAgent1$ authenticity was (essentially) ensured by the formula

$$\Box\big[\bigvee_{i=1}^{k} DynRcv1(hag_i)\big]_{home.res_h} \quad .$$

If we replace the disjunction by the name quantifier, it will be impossible to distinguish authentic agents (created and sent by ag) from intruders, basically because formula

$$\Box\big[\exists\, hag : DynRcv1(hag)\big]_{home.res_h}$$

does not say anything about the hidden agent hag. This has to do with the fact that the name quantifier in a formula $\exists\, n : F$ implicitly quantifies over all local variables at location n.

Nevertheless, one can try to define an existential quantifier over names that allows to control the local variables at the quantified names. This is what we are going to do now.

Specification $DynAgent$, given in fig. 6.5, carries out this idea by using a new existential quantifier. It uses also the formulas given in fig. 6.6 and fig. 6.7. The omitted parts of the specification – indicated by dots – are the corresponding formulas of $DynAgent1$ in the previous section. The semantics of the quantifier will be introduced in the next section. Informally, a formula $\exists z : F$ asserts the existence of an "anonymous" location, that is, of a location whose name is not known, for which F holds.

The main difference compared with specification $DynAgent1$ is, apart from the replacement of the disjunctions by quantifiers, the way authenticity is ensured.

$$
\begin{array}{rcl}
\mathit{Init} & \equiv & \mathit{home.ag}\langle\textbf{true}\rangle \wedge \mathit{ag.ctl} = \text{“idle”} \\
\mathit{vars} & \equiv & \langle \mathit{ag.ctl}, \mathit{ag.dests}, \mathit{ag.time}, \mathit{ag.rest}, \mathit{ag.found}, \mathit{ag.sent}, \mathit{ag.key}\rangle \\
\mathit{vars}_{\mathit{home}} & \equiv & \langle \mathit{home.res}_f, \mathit{home.res}_h, \mathit{home.key}\rangle \\
\mathit{IDynAgent} & \equiv & \wedge\ \mathit{Init} \\
 & & \wedge\ \bigwedge_{n\in \mathit{Net}} \Box[\bigvee_{m\in \mathit{Net}} \mathit{DynMove}_{n,m}]_{-n.ag} \\
 & & \wedge\ \Box\big[\mathit{DynHomeActs} \vee \bigvee_{n\in \mathit{Net}} \mathit{DynActions}_n\big]_{\mathit{vars},\mathit{vars}_{\mathit{home}}} \\
 & & \\
\mathit{Security} & \equiv & \forall z : \Box[\mathit{IDynAgent}[z/ag] \vee \bigvee_{n\in \mathit{Net}} \mathit{Create}_n(z)]_{+(z.\mathit{key}=\mathit{home.key})} \\
\mathit{DynAgent} & \equiv & \wedge\ \mathit{Network} \\
 & & \wedge\ \mathit{IDynAgent} \\
 & & \wedge\ \mathit{Security}
\end{array}
$$

Figure 6.5: Agent with arbitrarily many created sub-agents

For this purpose we introduce a variable key. The result delivered by an agent is only accepted if the agent has the right key as asserted by the conjunct $hag.key = home.key$ of formula $HDeliver$. This together with formula $Security$ guarantees authenticity: $Security$ requires that an agent that obtains the right key (whenever formula $z.key = home.key$ becomes true during a transition) either behaves like the “main” agent or like one of the hotel agents.

6.3 Rigid quantification over names

In the previous section we introduced informally a rigid name quantifier. Now we give the precise definition of its semantics.

Technically, we have to extend the definition of MTLA as well as our model definition. We assume a further denumerable set $\mathcal{V}^n$ of *name variables*. We modify the definition of formulas accordingly: for an (im)pure formula A and a name variable $z \in \mathcal{V}^n$ we let

$$z[A] \mid \exists z : A$$

also be (im)pure formulas. The model notion is modified by requiring that the valuation ξ assigns to every name variable a name, i.e. $\xi : \mathcal{V}_r \cup \mathcal{V}^n \to |\mathfrak{I}| \cup \mathsf{N}$ with

$$
\begin{aligned}
DynPrep(ds,t) &\equiv \land \ldots \\
&\quad \land ag.key' = home.key \\
&\quad \land \text{UNCHANGED } vars_{home} \\
DynGetFlight_n &\equiv \land \ldots \\
&\quad \land \text{UNCHANGED } \ldots, ag.key, home.key \\
Create_n(hag) &\equiv \land \ldots \\
&\quad \land HAgent(hag, \langle ag.dest, ag.time \rangle, ag.key) \\
&\quad \land \text{UNCHANGED } \ldots, ag.key, home.key \\
NewItem_n &\equiv \land \ldots \\
&\quad \land \text{UNCHANGED } \ldots, ag.key, home.key \\
DynMove_{n,m} &\equiv \land \ldots \\
&\quad \land \text{UNCHANGED } \ldots, ag.key, home.key \\
DynDeliver &\equiv \land \ldots \\
&\quad \land \text{UNCHANGED } \ldots, home.key \\
DynRcv(hag) &\equiv \land \ldots \\
&\quad \land home.key = hag.key \\
&\quad \land \text{UNCHANGED } \ldots, ag.key, home.key \\
DynHomeActs &\equiv \lor \exists ds, t : DynPrep(ds,t) \\
&\quad \lor DynDeliver \\
&\quad \lor \exists hag : DynRcv(hag) \\
DynActions_n &\equiv \lor DynGetFlight_n \lor NewItem_n \\
&\quad \lor \bigvee_{m \in Net} DynMove_{n,m} \\
&\quad \lor \exists hag : Create_n(hag) \\
vars &\equiv \langle \ldots, ag.key, home.key \rangle
\end{aligned}
$$

Figure 6.6: Actions of $DynAgent$

$\xi(\mathcal{V}_r) \subseteq |\mathfrak{I}|$ and $\xi(\mathcal{V}^n) \subseteq \mathsf{N}$. The definition of the semantics of MTLA-formulas is extended by the following clauses:

- $\sigma, n, \xi \models z[A]$ iff $\sigma, n, \xi \models m[A]$ for $m = \xi(z)$
- $\sigma, n, \xi \models \exists z : A$ iff $\sigma, n, \xi[z := m] \models A$ for some name $m \in \mathsf{N}$

$$
\begin{aligned}
HInit((d,t),k) &\equiv \wedge \ldots \\
&\quad \wedge hag.key = k \\
GetHotel_n &\equiv \wedge \ldots \\
&\quad \wedge \text{UNCHANGED } \ldots, hag.key \\
HMove_{n,m} &\equiv \wedge \ldots \\
&\quad \wedge \text{UNCHANGED } \ldots, hag.key \\
HDeliver &\equiv \wedge \ldots \\
&\quad \wedge hag.key = home.key \\
&\quad \wedge \text{UNCHANGED } home.key \\
vars_h &\equiv \langle hag.ctl, hag.dest, hag.found, hag.item, hag.key \rangle \\
HAgent(hag,(d,t),k) &\equiv \wedge HInit((d,t),k) \\
&\quad \wedge \bigwedge_{n \in Net} \Box\big[\bigvee_{m \in Net} HMove_{m,n}\big]_{-n.hag} \\
&\quad \wedge \Box\big[HDeliver \vee \bigvee_{n \in Net} GetHotel_n\big]_{vars_h}
\end{aligned}
$$

Figure 6.7: MTLA-Specification of the Hotel Agent

where $\xi[z := m]$ is defined as follows:

$$\xi[z := m](\tilde{z}) = \begin{cases} m & \text{if } \tilde{z} = z \\ \xi(\tilde{z}) & \text{otherwise} \end{cases} .$$

We do not aim here at giving a complete proof system for the extended logic. However, let us mention that the standard quantification rules

$$(\exists_{\mathsf{N}}\text{-I}) \quad A[m/z] \Rightarrow \exists z : A \qquad\qquad (\exists_{\mathsf{N}}\text{-E}) \quad \frac{A \Rightarrow B}{(\exists z : A) \Rightarrow B}$$

are sound on the usual condition for ($\exists_{\mathsf{N}}$-E) that z must not have free occurrences in B. Furthermore, as the semantics of MTLA allows only for finite trees, there are always names that do not occur in the current tree. Hence, an axiomatisation would have to contain an axiom like

$$(\text{fin}) \quad \exists z : z[\textbf{false}] \quad .$$

Further, most axioms of Σ_{MLTL} could be modified by replacing every name by a universally quantified name variable. (As usual, we write $\forall z : A \equiv \neg\exists z : \neg A$.)

For example, (ax2) would by replaced by

$$\forall z_0, z_1 : z_0[F] \Rightarrow z_1[z_0[F]] \quad .$$

However, as we did not introduce equality on name variables, we cannot do the same for

$$\text{(ax5)} \quad (a_1.b \wedge a_2.b) \Rightarrow (a_1.a_2 \vee a_2.a_1) \qquad (\text{for } a_1 \neq a_2) \quad .$$

We either could keep the original axiom, or take something like

$$\forall z, z_1, z_2 : \big((\exists z_0 : z_1.z_0 \wedge \neg z_2.z_0) \wedge (z_1.z \wedge z_2.z) \Rightarrow (z_1.z_2 \vee z_2.z_1)\big),$$

where the first conjunct makes sure that the two variables z_1 and z_2 cannot assume the same value.

Observe that using the name quantifier, the operators keep_m can be defined as follows:

$$\begin{aligned} \mathsf{keep}_m \equiv \wedge\ & m\langle \mathbf{true} \rangle \Leftrightarrow \circ m\langle \mathbf{true} \rangle \\ & \wedge \forall z_0 \forall z_1 : \wedge\ m.z_0\langle \mathbf{true} \rangle \Leftrightarrow \circ m.z_0\langle \mathbf{true} \rangle \\ & \qquad \wedge\ m.z_0.z_1\langle \mathbf{true} \rangle \Leftrightarrow \circ m.z_0.z_1\langle \mathbf{true} \rangle \quad . \end{aligned}$$

The first conjunct ensures that m occurs either in both (the current and the next) configurations or in neither of them. The second conjunct says that the names beneath node m are the same as in the next tree. The last conjunct makes sure that also the order between these names is preserved.

We would like to point out that the concept of rigid quantification over names – and already the keep operator - does not really fit in TLA's philosophy, because it destroys MTLA's property of (spatial) stuttering invariance. However, this does not mean that all the refinement principles presented in chapter 3 are useless now, it just means that one cannot apply them "blindly", but has to be careful in the presence of keep_m and the rigid name quantifier. For example, in our situation it is perfectly legitimate to extend the newly created hotel agents by sub-nodes, essentially because the following formula is valid:

$$\exists z : z.n[F] \Rightarrow \exists z : z.m.n[F] \quad .$$

6.4 Hiding of anonymous agents

In the previous section we presented a way to extend MTLA that enables us to describe the dynamic creation of agents. However, the specification given in fig. 6.5 does not allow for implementations where the task of the agents – searching for offers – is realised by other techniques then by mobile agents, because the formulas $\exists hag : Create_n(hag)$ require the presence of sub-agents. If we want to enable such refinements, we have to be able to hide such "anonymous" agents. As their names are not known, we cannot use the name quantifier defined in sec. 3.3 (in connection with the virtualisation of locations) for this purpose. Instead, we allow to hide sets of locations, and require the created locations to belong to the hidden set.

Figure 6.8 contains a part of the MTLA-specification of a travel agent that is, similarly to the one specified by $DynAgent$ in fig. 6.5, able to create arbitrarily many sub-agents to collect offers for hotels, but this time all the sub-agents are hidden from the interface as expressed by the existential quantification over the name set variable HAG.

Intuitively, (the value of) variable HAG contains the names of all possible sub-agents. The inner specification is almost the same as $DynAgent$. The main differences are to observe when a sub-agent is created (formula $TACreate_n(hag)$) and when it returns to the home location ($TARcv(hag)$). In both cases, we require the name variable hag to belong to the set given by the variable HAG. Further, we do not need the key anymore, as the authenticity of the agents is now ensured by the conjunct $hag \in HAG$, the additional conjunct in $Init$ that asserts that initially no agent in HAG exists, and the last conjunct in the inner specification that says that whenever a new agent from HAG appears, then it behaves correctly. So we leave out all formulas containing variable key. In the overall specification we have omitted some of the actions as they are almost the same as the corresponding formulas of $DynAgent$, the main difference being that no variable key is needed.

In addition to the sets $\mathcal{V}_r$, $\mathcal{V}_f$ and $\mathcal{V}^n$ of rigid, flexible and name variables, we assume a set $\mathcal{V}^{ns}$ of *name set variables* so that all the different variable sets are pairwise disjoint. We will denote the variables in this set by capital letters

$$
\begin{aligned}
TAInit &\equiv \wedge \ldots \\
&\quad \wedge \forall z : (z \in HAG \Rightarrow z[\textbf{false}]) \\
TACreate_n(hag) &\equiv \wedge \ldots \\
&\quad \wedge hag \in HAG \wedge \circ n.hag\langle \textbf{true} \rangle \\
&\quad \wedge \ldots \\
TARcv(hag) &\equiv \wedge \ldots \\
&\quad \wedge hag \in HAG \wedge \ldots \\
&\quad \wedge \ldots \\
TAHomeActs &\equiv \vee \exists ds, t : TAPrep(ds, t) \\
&\quad \vee Deliver \\
&\quad \vee \exists hag : TARcv(hag) \\
TAActions_n &\equiv \vee\ TAGetFlight_n \vee TANewItem_n \\
&\quad \vee \bigvee_{m \in Net} TAMove_{n,m} \\
&\quad \vee \exists hag : TACreate_n(hag) \\
ITravelAgent &\equiv \wedge\ Init \\
&\quad \wedge\ Network \\
&\quad \wedge \bigwedge_{n \in Net} \Box[\bigvee_{m \in Net} TAMove_{n,m}]_{-n.ag} \\
&\quad \wedge \Box\big[TAHomeActs \vee \bigvee_{n \in Net} TAActions_n\big]_{vars} \\
&\quad \wedge \forall z \in HAG : \Box[\bigvee_{n \in Net} TACreate_n(z)]_{+z\langle \textbf{true} \rangle} \\
\\
TravelAgent &\equiv \mathbf{\exists}\, HAG : ITravelAgent
\end{aligned}
$$

Figure 6.8: Travel Agent with hidden sub-agents

$X, Y, X_i, Y_i, \ldots$. Further, we assume that the valuation ξ provides values for the name set variables, i.e. ξ assigns to every $X \in \mathcal{V}^{ns}$ a set of names $S \subseteq \mathsf{N}$.

We extend the syntax of MTLA as follows:

$$F \quad ::= \quad \ldots \mid z \in X \mid m \in X \mid \mathbf{\exists}\, X : F \quad .$$

For the definition of the semantics we will need the following definitions. For every set $S \subseteq \mathsf{N}$ of names we define a relation $<_S$ on the set of configurations.

$$
\begin{array}{lll}
(s,\lambda) <_S (t,\mu) \quad \text{iff} \quad & \mathsf{N}_s = \mathsf{N}_t \setminus S & \text{and} \\
& <_s = <_t \cap (\mathsf{N}_s \times \mathsf{N}_s) & \text{and} \\
& \lambda(a) = \mu(a) \text{ for all } a \in \mathsf{N}_s &
\end{array}
$$

For runs $\sigma = (s_0,\lambda_0)(s_1,\lambda_1)\ldots$ and $\tau = (t_0,\mu_0)(t_1,\mu_1)\ldots$ and a set of names S we define $\sigma <_S \tau$ to hold iff $(s_i,\lambda_i) <_S (t_i,\mu_i)$ for all $i \in \omega$.

The definition of the semantics of the new formulas is given in fig. 6.9.

$\sigma^{(n,\xi)}(X) \;=\; \xi(X)$ for $X \in \mathcal{V}^{ns}$

$\sigma, n, \xi \models z \in X$ iff $\xi(z)$ is an element of $\xi(X)$

$\sigma, n, \xi \models m \in X$ iff m is an element of $\xi(X)$

$\sigma, n, \xi \models \exists X : F$ iff there are runs ρ, τ and a set $S \subseteq \mathsf{N}$ with $\rho \simeq \sigma$, $\rho <_S \tau$ and $\tau, n, \xi[X := S] \models F$

Figure 6.9: Semantics of extended MTLA

The intuitive interpretation of this existential quantifier is analogous to the one of the (flexible) name quantifier. A formula $\exists X : F$ means that we can extend the run by names so that F holds for the extended run.

Chapter 7

Conclusion

In the present thesis we have introduced and studied a novel spatio-temporal logic called MTLA intended for the specification and refinement of systems that make use of mobile code. In contrast to most previous logics for similar purposes, the semantics of our logic is based on Kripke structures instead of process algebras. In chapter 3 we have considered different notions of refinement that we believe to make sense in the context of mobile systems. We have shown MTLA to support these kinds of refinement. For the sake of simplicity, we have considered the logic MLTL of which MTLA is a fragment to carry out the theoretical work in chapter 4 and chapter 5. In chapter 4 we have presented a proof system that we have proven to be sound and complete with respect to the (propositional fragment of the) logic MLTL. In chapter 5 we have given a translation of MLTL into weak alternating automata and used this result to prove the satisfiability problem to be decidable. The same result also provided us with a solution of the model checking problem. The last chapter has considered some rather ad hoc extensions of MTLA.

There is still work to be done on every level. We have presented only small toy examples. In the future, we would like to validate the logic on the basis of more realistic applications. Our axiomatisation only has considered the quantifier free part of the logic, possible axioms for the different quantifiers have been studied only informally. This gap has to be filled. We also plan to investigate the adequacy of the extensions presented in chapter 6 more carefully.

Another interesting question is the applicability of MTLA to other problems than the specification of mobile systems. Tree structures that change over time appear in many contexts, for example, XML documents can be regarded as trees and their update corresponds to modification of the tree structure.

Appendix A

Auxiliary derivations

We give derivations of the theorems and rules used in chapter 4.

Boxed version of (ax3), that is, $n[\neg a[F] \Rightarrow a[\neg F]]$:

(1)	$n[\neg a[F]] \Leftrightarrow (n.a[F] \Rightarrow n[\mathbf{false}])$	(T2),(T3),(T4),(prop)
(2)	$(n.a[F] \Rightarrow n[\mathbf{false}]) \Leftrightarrow \neg n.a[F] \vee n[\mathbf{false}]$	(ax0)
(3)	$\neg n.a[F] \Rightarrow n.a[\neg F]$	(T1)
(4)	$n[\mathbf{false}] \Rightarrow n.a[\neg F]$	(T4)
(5)	$\neg n.a[F] \vee n[\mathbf{false}] \Rightarrow n.a[\neg F]$	(3),(4),(prop)
(6)	$n[\neg a[F]] \Rightarrow n.a[\neg F]$	(1),(2),(5),(prop)
(7)	$n[\neg a[F] \Rightarrow a[\neg F]]$	(T3),(6),(prop)

Boxed version of (ax4), that is, $n[a.a[\mathbf{false}]]$:

(1)	$a.a[\mathbf{false}] \Rightarrow n[a.a[\mathbf{false}]]$	(ax2)
(2)	$a.a[\mathbf{false}]$	(ax3)
(3)	$n[a.a[\mathbf{false}]]$	(1),(2),(mp)

Boxed version of (ax5), that is,

$n[a_1.b\langle\textbf{true}\rangle \wedge a_2.b\langle\textbf{true}\rangle \Rightarrow a_1.a_2\langle\textbf{true}\rangle \vee a_2.a_1\langle\textbf{true}\rangle]$:

(1)	$n[a_1.b] \wedge n[a_2.b] \Rightarrow n[\textbf{false}] \vee (n.a_1.b \wedge n.a_2.b)$	(T6),(prop)
(2)	$n.a_1.b \wedge n.a_2.b \Rightarrow a_1.b \wedge a_2.b$	(ax2),(prop)
(3)	$n.a_1.b \wedge n.a_2.b \Rightarrow a_1.a_2 \vee a_2.a_1$	(ax5),(2),(prop)
(4)	$n.a_1.b \wedge a_1.a_2 \Rightarrow n.a_1.a_2$	(ax2),(prop),(T7)
(5)	$n.a_2.b \wedge a_2.a_1 \Rightarrow n.a_2.a_1$	(ax2),(prop),(T7)
(6)	$n.a_1.b \wedge n.a_2.b \Rightarrow n.a_1.a_2 \vee n.a_2.a_1$	(3),(4),(5),(prop)
(7)	$n.a_1.a_2 \Rightarrow n[a_1.a_2]$	(T6),(prop)
(8)	$n.a_2.a_1 \Rightarrow n[a_2.a_1]$	(T6),(prop)
(9)	$n.a_1.b \wedge n.a_2.b \Rightarrow n[a_1.a_2] \vee n[a_2.a_1]$	(6),(7),(8),(prop)
(10)	$n[a_1.a_2] \vee n[a_2.a_1] \Rightarrow n[a_1.a_2 \vee a_2.a_1]$	(T9a),(prop)
(11)	$n.a_1.b \wedge n.a_2.b \Rightarrow n[a_1.a_2 \vee a_2.a_1]$	(9),(10),(prop)
(12)	$n[\textbf{false}] \Rightarrow n[a_1.a_2 \vee a_2.a_1]$	(T4)
(13)	$n[a_1.b] \wedge n[a_2.b] \Rightarrow n[a_1.a_2 \vee a_2.a_1]$	(1),(11),(12)
(14)	$n[a_1.b \wedge a_2.b] \Rightarrow n[a_1.b] \wedge n[a_2.b]$	(T9b),(prop)
(15)	$n[a_1.b \wedge a_2.b] \Rightarrow n[a_1.a_2 \vee a_2.a_1]$	(14),(13),(prop)
(16)	$n[a_1.b \wedge a_2.b \Rightarrow a_1.a_2 \vee a_2.a_1]$	(15),(T3)

(T10): $\circ F \wedge \circ G \Leftrightarrow \circ(F \wedge G)$:

(1)	$\circ(F \Rightarrow \neg G) \Rightarrow (\circ F \Rightarrow \circ\neg G)$	(ax7)
(2)	$\circ(F \Rightarrow \neg G) \Rightarrow (\circ F \Rightarrow \neg\circ G)$	(1),(ax6),(prop)
(3)	$\neg(\circ F \Rightarrow \neg\circ G) \Rightarrow \neg\circ(F \Rightarrow \neg G)$	(2),(prop)
(4)	$\neg(\circ F \Rightarrow \neg\circ G) \Rightarrow \circ\neg(F \Rightarrow \neg G)$	(3),(ax6),(prop)
(5)	$\circ F \wedge \circ G \Rightarrow \circ(F \wedge G)$	(4),(prop)
(6)	$\circ(F \wedge G \Rightarrow F)$	(ax0),(nex)

(7)	$\bigcirc(F \wedge G) \Rightarrow \bigcirc F$	(6),(ax7),(mp)
(8)	$\bigcirc(F \wedge G) \Rightarrow \bigcirc G$	(ax0),(nex),(ax7),(mp)
(9)	$\bigcirc(F \wedge G) \Rightarrow \bigcirc F \wedge \bigcirc G$	(7),(8),(prop)
(10)	$\bigcirc F \wedge \bigcirc G \Leftrightarrow \bigcirc(F \wedge G)$	(5),(9),(prop)

(T11): $F \wedge \bigcirc\Box F \Rightarrow \Box F$:

(1)	$\bigcirc\Box F \Rightarrow \bigcirc(F \wedge \bigcirc\Box F)$	(nex),(ax8),(ax7),(mp)
(2)	$F \wedge \bigcirc\Box F \Rightarrow \bigcirc(F \wedge \bigcirc\Box F)$	(1),(prop)
(3)	$F \wedge \bigcirc\Box F \Rightarrow F$	(ax0)
(4)	$F \wedge \bigcirc\Box F \Rightarrow \Box F$	(ind),(2),(3)

(T12): $n[\Box F] \Leftrightarrow (n[F] \wedge n[\bigcirc\Box F])$:

(1)	$n[\Box F \Leftrightarrow F \wedge \bigcirc\Box F]$	(axn8)
(2)	$n[\Box F] \Leftrightarrow n[F \wedge \bigcirc\Box F]$	(ax1),(T3),(1),(prop)
(3)	$n[\Box F] \Leftrightarrow n[F] \wedge n[\bigcirc\Box F]$	(2),(T9b),(prop)

(alw): $F \vdash \Box F$:

(1)	F	assumption
(2)	$\bigcirc F$	(nex),(1)
(3)	$F \Rightarrow \bigcirc F$	(prop),(2)
(4)	$F \Rightarrow \Box F$	(ind),(3),(prop)
(5)	$\Box F$	(mp),(1),(4)

(T13): $\Box(F \Rightarrow G) \Rightarrow (\Box F \Rightarrow \Box G)$:

By the deduction theorem it suffices to show $F \Rightarrow G, F \vdash \Box G$.

(1)	$F \Rightarrow G$	assumption
(2)	F	assumption

(3)	G	(1),(2),(mp)
(4)	$\Box G$	(alw),(3)

(T14): $\Box(F \wedge G) \Leftrightarrow \Box F \wedge \Box G$:

(1)	$\Box(F \wedge G \Rightarrow F)$	(alw),(ax0)
(2)	$\Box(F \wedge G \Rightarrow G)$	(alw),(ax0)
(3)	$\Box(F \wedge G) \Rightarrow \Box F$	(T14),(1),(mp)
(4)	$\Box(F \wedge G) \Rightarrow \Box G$	(T14),(2),(mp)
(5)	$\Box(F \wedge G) \Rightarrow \Box F \wedge \Box G$	(prop),(4)
(6)	$\Box F \wedge \Box G \Rightarrow \bigcirc(\Box F \wedge \Box G)$	(ax8),(prop),(T10)
(7)	$\Box F \wedge \Box G \Rightarrow F \wedge G$	(ax8),(prop)
(8)	$\Box F \wedge \Box G \Rightarrow \Box(F \wedge G)$	(6),(7),(ind)
(9)	$\Box(F \wedge G) \Leftrightarrow \Box F \wedge \Box G$	(5),(8),(prop)

Boxed version of one direction of (ax6), that is, $n[\neg\bigcirc F \Rightarrow \bigcirc\neg F]$:

(1)	$n\langle\neg\bigcirc F\rangle \Rightarrow \bigcirc\neg n[F]$	(ax9),(T5)
(2)	$\bigcirc\neg n[F] \Rightarrow \bigcirc n[\neg F]$	(ax3),(nex),(ax7),(mp)
(3)	$\bigcirc n[\neg F] \Rightarrow n[\bigcirc\neg F]$	(ax9),(ax6),(prop)
(4)	$n\langle\neg\bigcirc F\rangle \Rightarrow n[\bigcirc\neg F]$	(1),(2),(3),(prop)
(5)	$n[\mathbf{false}] \Rightarrow n[\bigcirc\neg F]$	(T4)
(6)	$n[\neg\bigcirc F] \Rightarrow n[\mathbf{false}] \vee n\langle\neg\bigcirc F\rangle$	(T5),(T6),(prop)
(7)	$n[\neg\bigcirc F] \Rightarrow n[\bigcirc\neg F]$	(4),(5),(6),(prop)
(8)	$n[\neg\bigcirc F \Rightarrow \bigcirc\neg F]$	(7),(T3),(prop)

Boxed version of (ax7), that is, $n[\bigcirc(F \Rightarrow G) \Rightarrow (\bigcirc F \Rightarrow \bigcirc G)]$:

(1)	$n\langle\bigcirc(F \Rightarrow G)\rangle \Rightarrow \bigcirc n[F \Rightarrow G]$	(ax10)
(2)	$n\langle\bigcirc F\rangle \Rightarrow \bigcirc n[F]$	(ax10)

(3)	$\circ(n[F \Rightarrow G] \Rightarrow (n[F] \Rightarrow n[G]))$	(nex),(ax1)
(4)	$\circ n[F \Rightarrow G] \Rightarrow \circ(n[F] \Rightarrow n[G])$	(ax7),(3),(mp)
(5)	$\circ(n[F] \Rightarrow n[G]) \Rightarrow (\circ n[F] \Rightarrow \circ n[G])$	(ax7)
(6)	$n\langle\circ(F \Rightarrow G)\rangle \Rightarrow (\circ n[F] \Rightarrow \circ n[G])$	(1),(4),(5),(prop)
(7)	$n\langle\circ(F \Rightarrow G)\rangle \wedge n\langle\circ F\rangle \Rightarrow \circ n[G]$	(2),(6),(prop)
(8)	$n\langle\circ(F \Rightarrow G)\rangle \wedge n\langle\circ F\rangle \Rightarrow n[\circ G]$	(7),(ax9),(prop)
(9)	$n[\mathbf{false}] \Rightarrow n[\circ G]$	(T4)
(10)	$n[\circ(F \Rightarrow G)] \wedge n[\circ F] \Rightarrow$ $n[\mathbf{false}] \vee (n\langle\circ(F \Rightarrow G)\rangle \wedge n\langle\circ F\rangle)$	(T6),(prop)
(11)	$n[\circ(F \Rightarrow G)] \Rightarrow (n[\circ F] \Rightarrow n[\circ G])$	(8),(9),(10),(prop)
(12)	$(n[\circ F] \Rightarrow n[\circ G]) \Rightarrow n[\circ F \Rightarrow \circ G]$	(T3)
(13)	$n[\circ(F \Rightarrow G)] \Rightarrow n[\circ F \Rightarrow \circ G]$	(11),(12),(prop)
(14)	$n[\circ(F \Rightarrow G) \Rightarrow (\circ F \Rightarrow \circ G)]$	(13),(T3),(prop)

Boxed version of (ax10), that is, $n[\neg m[\neg\circ F] \Rightarrow \circ m[F]]$:

(1)	$n.m\langle\circ F\rangle \Rightarrow m\langle\circ F\rangle$	(ax2), (prop)
(2)	$m\langle\circ F\rangle \Rightarrow \circ m[F]$	(ax10)
(3)	$\circ(m[F] \Rightarrow n.m[F])$	(nex), (ax2)
(4)	$\circ m[F] \Rightarrow \circ n.m[F]$	(ax7),(3),(mp)
(5)	$\circ n.m[F] \Rightarrow n[\circ m[F]]$	(ax9),(prop)
(6)	$n.m\langle\circ F\rangle \Rightarrow n[\circ m[F]]$	(1),(2),(4),(5),(prop)
(7)	$n[m\langle\circ F\rangle] \Rightarrow n[\mathbf{false}] \vee n.m\langle\circ F\rangle$	(T6),(prop)
(8)	$n[\mathbf{false}] \Rightarrow n[\circ m[F]]$	(T4)
(9)	$n[m\langle\circ F\rangle] \Rightarrow n[\circ m[F]]$	(6),(7),(8),(prop)
(10)	$n[\neg m[\neg\circ F] \Rightarrow \circ m[F]]$	(9),(T3),(prop)

(T15): $\bigcirc\Box F \Leftrightarrow \Box\bigcirc F$:

(1)	$\bigcirc\Box F \Rightarrow \bigcirc(F \wedge \bigcirc\Box F)$	(nex),(ax8),(ax7),(mp)
(2)	$\bigcirc\Box F \Rightarrow \bigcirc F \wedge \bigcirc\bigcirc\Box F$	(1),(T10),(prop)
(3)	$\bigcirc\Box F \Rightarrow \bigcirc\bigcirc\Box F$	(2),(prop)
(4)	$\bigcirc\Box F \Rightarrow \bigcirc F$	(2),(prop)
(5)	$\bigcirc\Box F \Rightarrow \Box\bigcirc F$	(ind),(3),(4)
(6)	$\Box\bigcirc F \wedge F \Rightarrow \bigcirc(\Box\bigcirc F \wedge F)$	(ax8),(prop),(T10)
(7)	$\Box\bigcirc F \wedge F \Rightarrow \Box(\Box\bigcirc F \wedge F)$	(ind),(6),(prop)
(8)	$\Box\bigcirc F \Rightarrow (F \Rightarrow \Box F)$	(7),(T14),(prop)
(9)	$\bigcirc\Box\bigcirc F \Rightarrow \bigcirc(F \Rightarrow \Box F)$	(nex),(8),(ax7),(mp)
(10)	$\Box\bigcirc F \Rightarrow \bigcirc\Box\bigcirc F$	(ax8),(prop)
(11)	$\Box\bigcirc F \Rightarrow (\bigcirc F \Rightarrow \bigcirc\Box F)$	(9),(10),(ax7),(prop)
(12)	$\Box\bigcirc F \Rightarrow \bigcirc F$	(ax8),(prop)
(13)	$\Box\bigcirc F \Rightarrow \bigcirc\Box F$	(11),(12),(prop)
(14)	$\bigcirc\Box F \Leftrightarrow \Box\bigcirc F$	(5),(13),(prop)

Bibliography

[1] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proc. Fourth ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[2] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 81(2):253–284, May 1991.

[3] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[4] Lorenzo Bettini, Rocco De Nicola, and Michele Loreti. Formalizing properties of moblie agent systems. In *COORDINATION'02: Proceedings of the 5th International Conference on Coordination Models and Languages*, pages 72–87, London, UK, 2002. Springer-Verlag.

[5] Nathaniel Borenstein. Email with a mind of its own: The safe-tcl language for enabled mail. In *Proceedings of the 1994 IFIP WG6.5 Conference on Upper Layer Protocols, Architecture, and Applications*, May 1994.

[6] J. Richard Büchi. On a decision method in restricted second-order arithmetics. In *International Congress on Logic, Method and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[7] Luís Caires and Luca Cardelli. A spatial logic for concurrency (part I). In *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science, pages 1–37. Springer-Verlag, 2001.

[8] Luca Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995.

[9] Luca Cardelli. Abstractions for mobile computation. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*, pages 51–94. Springer, 1999.

[10] Luca Cardelli and Andrew D. Gordon. Anytime, anywhere. modal logics for mobile ambients. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 365–377. ACM Press, 2000.

[11] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.

[12] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking. In Manfred Broy, editor, *Deductive Program Design*, volume F-152 of *NATO ASI series*, pages 305–350. Springer-Verlag, Berlin, 1996.

[13] Edmund M. Clarke and Holger Schlingloff. Model checking. In A. Voronkov, editor, *Handbook of Automated Deduction*. Elsevier, 2000. To appear.

[14] William D. Clinger. Foundations of actor semantics. Technical report, Cambridge, MA, USA, 1981.

[15] Rocco de Nicola, Gian Luigi Ferrari, and Rosario Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Trans. on Software Engineering*, 24(5):315–330, 1998.

[16] Rocco de Nicola and Michele Loreti. A modal logic for Klaim. In T. Rus, editor, *Proc. Algebraic Methodology and Software Technology (AMAST 2000)*, volume 1816 of *Lecture Notes in Computer Science*, pages 339–354. Springer-Verlag, 2000.

[17] Uffe Engberg and Mogens Nielsen. A calculus of communicating systems with label-passing. Technical Report DAIMI PB-208, Computer Science Departement, University of Aarhus, 1986.

[18] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *POPL*, pages 372–385. ACM Press, 1996.

[19] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421. Springer-Verlag, 1996.

[20] Alfonso Fugetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Trans. on Software Engineering*, 24(5):342–361, 1998.

[21] Rob Gerth, Doron Peled, Moshe Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[22] Matthew Hennessy and Robin Milner. Algebraic laws for non-determinism and concurrency. *ACM*, 32:137–161, 1985.

[23] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, Mass., 1979.

[24] Frederick C. Knabe. *Language Support for Mobile Agents*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, December 1995.

[25] Frederick C. Knabe. An overwiev of mobile agent programming. In *Proceedings of the fifth LOMAPS workshop on Analysis and Verification of Multiple - Agent Languages*, volume 1192 of *Lecture Notes in Computer Science*, Stockholm, Sweden, 1996. Springer-Verlag.

[26] Alexander Knapp, Stephan Merz, and Martin Wirsing. Refining mobile UML state machines. In Savi Maharaj, Charles Rattray, and Carron Shankland, editors, *10th Intl. Conf. Algebraic Methodology and Software Technology (AMAST 2004)*, Lecture Notes in Computer Science, Stirling, Scotland, July 2004. Springer-Verlag.

[27] Alexander Knapp, Stephan Merz, Martin Wirsing, and Júlia Zappe. Specification and refinement of mobile systems in MTLA and mobile UML. *Theoretical Computer Science*, 2005. Special Issue AMAST 2004, to appear.

[28] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[29] Fred Kröger. *Temporal Logic of Programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987.

[30] Leslie Lamport. How to write a long formula. Research Report 119, Digital Equipment Corporation, Systems Research Center, December 1993.

[31] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[32] Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words. volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer-Verlag, 2000.

[33] Stephan Merz. A user's guide to TLA. In F. Cassez, C. Jard, O. Roux, and B. Rozoy, editors, *Modélisation et vérification des processus parallèles: Actes de l'école d'été*, pages 29–44, Nantes, July 1998. Ecole centrale de Nantes.

[34] Stephan Merz. A more complete TLA. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 — Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1226–1244, Toulouse, September 1999. Springer-Verlag.

[35] Stephan Merz. Model checking: A tutorial overview. In F. Cassez et al., editor, *Modeling and Verification of Parallel Processes*, volume 2067 of *Lecture Notes in Computer Science*, pages 3–38. Springer-Verlag, Berlin, 2001.

[36] Stephan Merz, Martin Wirsing, and Júlia Zappe. A spatio-temporal logic for the specification and refinement of mobile systems. In *FASE'03, Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science. "Springer-Verlag", 2003.

[37] Robin Milner. Communication and concurrency. Prentice Hall, 1989.

[38] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.

[39] Robin Milner, Joachim Parrow, and David Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.

[40] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[41] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *13th ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 275–283. Springer-Verlag, 1986.

[42] Gary Ostertag, editor. *Definite Descriptions: A Reader*. MIT Press, Cambridge, Mass., 1998.

[43] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, November 1977.

[44] Amir Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–80, 1981.

[45] Amir Pnueli. System specification and refinement in temporal logic. In R.K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 1992.

[46] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.

[47] Davide Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. of the 28th Intl. Conf. on Principles of Programming Languages (POPL'01)*, pages 4–17. ACM Press, 2001.

[48] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[49] Wolfgang Thomas. Complementation of Büchi automata revisited. In J. Karhumäki, editor, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer-Verlag, 2000.

[50] Tommy Thorn. Programming languages for mobile code. *ACM Comput. Surv.*, 29(3):213–239, 1997.

[51] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. volume 1043 of *LNCS*, pages 238–266. Springer-Verlag New York, Inc., 1996.

[52] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In *ICCL Workshop: Internet Programming Languages*, pages 47–77, 1998.

[53] Jim White. Telescript technology: The foundation for the electronic marketplace. White paper, General Magic, Inc., Mountain View, CA, 1994.

[54] Jim White. Mobile agents. In J. Bradshaw, editor, *Software Agents*. AAAI Press and MIT Press, 1996.

[55] Silvano Dal Zilio. Mobile processes: A commented bibliography. pages 206–222, 2001.

LEBENSLAUF

JÚLIA ZAPPE

PERSÖNLICHE DATEN

Geburtstag: 16. Juni 1974
Geburtsort: Budapest, Ungarn
Staatsangehörigkeit: ungarisch

SCHULAUSBILDUNG UND STUDIUM

1992	Abitur am József Attila Gimnázium in Budapest
1994–2000	Studium der Mathematik an der LMU in München
2000	Diplom im Fach Mathematik an der LMU in München
2000–2005	Promotionsstudium im Fach Informatik Betreuer: Prof. Dr. F. Kröger