UNIKASSEL
VERSITÄT

UNIVERSITÄT KASSEL

BACHELOR THESIS

Neededness Analysis for Model Checking Properties Defined by Order-2 Fixpoints

Author:
Marco Sälzer

Supervisor:
Florian Bruse

Matriculation Number:
33313214

First Examiner:
Prof. Dr. Martin Lange

Date of Submission:
09.08.2018

Second Examiner:
Prof. Dr. Claudia Fohry

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Theoretical Computer Science / Formal Methods
Department of Electrical Engineering and Computer Science

June 7, 2018 - August 9, 2018

Declaration of Authorship

I, Marco Sälzer, declare that this thesis titled, "Neededness Analysis for Model Checking Properties Defined by Order-2 Fixpoints" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Kassel.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at the University of Kassel or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Acknowledgements

I would like to express my thanks to Prof. Martin Lange and Florian Bruse for their comprehensive support and guidance that enabled me to work on this topic at all. I would particularly like to pay tribute to the unwavering patience of Florian Bruse, who has spent hours discussing problems with me. The mentoring i received during this time was nothing but perfect.

On the non-academic side of my life, I am grateful for all the support I have received from my family and friends up to this moment, but above all, I would like to honor one person and dedicate my first academic work to him: my older brother Maik. I am most sure that without his encouragement and advice I would never be the person I am today and that without him I would never have the chance to become the person I want to be.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Whether the design of a system is correct regarding an intended specification is a reoccurring question in almost all fields of computer science. With increasing complexity of the system, the developer loses the ability to make a resilient statement about its correctness. Hence, automatic or at least semi-automatic methods for addressing this question are needed. One common way is simulating and testing the actual implementation of a system. However, this method allows finding errors, but can not make any comprehensive statement about the absence of errors or the correctness of the implementation.

For many cases *formal verification* allows answering this question with mathematical reasoning. A valid verification by a formal verification method implies that all possible behaviors of the intended system have been explored, thus it guarantees that the design is in fact correct with respect to a given specification [CGP99]. *Model checking* is one of the main techniques in this context, which enables checking a given model against a given specification. The model is a formal description of a system and the specification is a property that this system should meet or not meet, encoded in form of a formula. Typically, these formulas are specified formulas of a temporal logic.

The *modal μ-calculus* is such a temporal logic, which possesses a high expressive power. Many other temporal logics like CTL* can be embedded into the modal μ-calculus [DGL16]. But properties that are expressible in the modal μ-calculus are at most regular, a result from its equi-expressiveness to the Monadic Second Order Logic [JW96]. However, there are many interesting properties that are non-regular such as uniform inevitability, infinite counting properties or certain assume-guarantee properties [Eme87][Lan07][VV04].

Higher-order fixpoint logic (HFL) [VV04] achieves expressability of such properties by incorporating a *simply typed λ-calculus* into the modal μ-calculus. This makes it possible that formulas not only describe state sets, but also functions between them, functions between sets of functions and so on. At the same time, the model checking problem for finite transition systems remains decidable in HFL as shown in its introductory work. Unfortunately, this gain in expressive power leads to the fact that the model checking problem in HFL for a fixed model and formula is k-EXPTIME complete, where k is a restriction on the order of usable functions [ALS07]. There exist fragments that can be model checked in $(k-1)$-EXPSPACE [BLL17], but this relatively high computational complexity still creates a need for efficient model checking algorithms.

Axelsson and Lange introduced a symbolic model checker for the first-order fragment HFL1 of HFL, which restricts formulas to denote state sets or functions between them only. The idea is to localize fixpoint computations and thus avoid best-case exponential behavior [AL07]. To achieve this, they transfered the idea of *neededness analysis* from traditional data-flow analysis to model checking HFL1. Originally, neededness analysis states that it is sufficient to only include values that appear non-trivially in the computation of a fixpoint of interest [Jør94]. In the context of model checking HFL, this can be achieved by utilizing the fact that the value of a fixpoint is usually not needed globally, i.e. for all possible arguments. Therefore, it can be localized to needed ones only. It is shown in its introductory work that for NFAs of finite size the universality problem encoded in HFL1 can be model checked with this algorithm with far less computations then a naive model checking algorithm would need.

Therefore, it is a problem of interest to lift this approach to higher-order fragments of HFL. The goal of this thesis is developing a model checking algorithm with neededness analysis for the second-order fragment HFL2 of HFL, where formulas also can denote functions between functions of state sets. Furthermore, it will be discussed, which challenges and difficulties come with lifting the original approach of Axelsson and Lange to a higher-order fragment like HFL2:

In Chap. 2 all needed preliminaries, including syntax, semantic and type-system of HFL are defined as well as needed definitions for the fragments HFL1 and HFL2 are given.

In Chap. 3 the model checking algorithm for HFL1 as it was originally introduced by Axelsson and Lange in [AL07] is described. Furthermore, an example computation, underlining the computational benefits of this approach and the expressive power already gained in HFL1 compared to the modal μ-calculus, is given.

In Chap. 4 an algorithm with neededness analysis for model checking HFL2 and its proof of correctness is presented. This is followed by a discussion of different possible optimizations in a practical context and an example, illustrating the computational benefits compared to a naive approach and the increased expressive power compared to HFL1.

In Chap. 5 challenges coming with a neededness analysis approach for model checking HFL2 are discussed and it shows that the problem of optimizing this approach is significantly more complex in HFL2 than in HFL1.

In Chap. 6 this thesis is summarized and directions for possible future work are given, based on the results of this thesis. Especially, the results are considered in the light of an actual implementation and lifting this approach to other, even higher order fragments and ultimately generalizing it to complete HFL.

In App. A a short introduction to fundamentals of the higher-order fixpoint logic is given, including the modal μ-calculus and the simply typed λ-calculus.

Chapter 2

Preliminaries

2.1 Labeled Transition Systems

Definition 1. A finite *labeled transition system (LTS)* is a structure $\mathcal{T} = \{\mathcal{S}, \{R_a \mid R_a \in \mathcal{S} \times \mathcal{S}, a \in \mathcal{A}\}, \mathcal{L}\}$, where

- $\mathcal{S} = \{s_0, ..., s_k\}$ is a finite set of states
- $\mathcal{A}$ is a finite set of symbols, representing transition names
- $\mathcal{P}$ is a finite set of symbols, representing propositions
- each R_a describes a binary relation on $\mathcal{S}$
- and $\mathcal{L}$ is a labeling function $\mathcal{S} \to 2^{\mathcal{P}}$, which assigns to each s its set of valid propositions.

Typically, transition names are denoted by $a, b, \ldots$ and $p, q, \ldots$ are used to range over proposition names.

Intuitively, an LTS is a directed graph with labeled edges and vertices, where the labels of edges are elements of $\mathcal{A}$ and the valid propositions for each vertex are elements of $\mathcal{P}$. With this in mind, an infix notation $s \xrightarrow{a} s'$ for edges can be used, which means $(s, s') \in R_a$ and identify R_a with $\xrightarrow{a}$.

Example 1. The LTS shown in Fig. 2.1 is formally described by $\{\{s_0, s_1, s_2, s_3\}, \{\xrightarrow{a}, \xrightarrow{b}, \xrightarrow{c}\}, \mathcal{L}_{ex}\}$, where $\mathcal{L}_{ex}$ maps the states s_2, s_3 to the proposition $\{p\}$ and every other state to $\emptyset$. The transitions $\xrightarrow{a}, \xrightarrow{b}, \xrightarrow{c}$ are defined as:

$$\xrightarrow{a} = \{(s_0, s_1), (s_1, s_2), (s_2, s_3)\}$$
$$\xrightarrow{b} = \{(s_1, s_3), (s_3, s_1)\}$$
$$\xrightarrow{c} = \{(s_2, s_2), (s_3, s_3)\}$$

2.2 The HFL Type System

HFL uses a type system, to prevent polymorphic effects and in general ensure that its semantics are well-defined. This type system can be seen as an extension of the

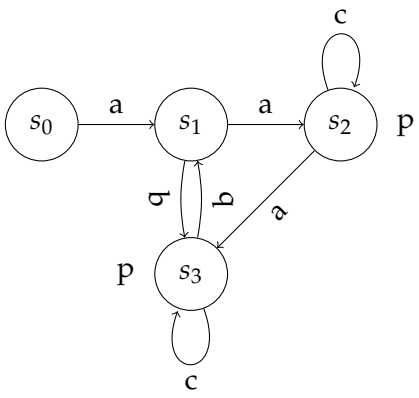

FIGURE 2.1: Simple labeled transition system.

type system of the simply typed λ-calculus. τ is called a *type* if it is producible by the following grammar:

$$\tau ::= \bullet \mid \tau^v \to \tau \quad v \in \{+, -, 0\}$$

$\bullet$ denotes the base-type *predicates*, which are subsets of the underlying state space $\mathcal{S}$. v is called *variance* and it corresponds to the behavior of a function in its arguments. The variance v of a type $\tau^v \to \tau'$ describes how the ordering of τ is respected. If $v = +$ then a function with type $\tau^v \to \tau'$ is monotonic, for $v = -$ it is antimonotonic and for $v = 0$ it is arbitrary in the corresponding arguments.

Over a given LTS $\mathcal{T}$ such a type τ is interpreted as a partially ordered set. This is formally defined in the following.

1. For a relation $R \subseteq U \times U$ on a set U, the relations R^+, R^- and R^0 are defined as follows: $R^+ = R, R^- = \{(b, a) \mid (a, b) \in R\}$ and $R^0 = R^+ \cap R^-$. For any partial order $\mathcal{U} = (U, \sqsubseteq_U)$, the partial order $\mathcal{U}^v$ is defined as $(U, \sqsubseteq_U^v)$ with $v = \{+, -, 0\}$.

2. The semantics $\mathcal{T}[\![\tau]\!]$ of any type τ are inductively defined as

$$\mathcal{T}[\![\bullet]\!] = (2^{\mathcal{S}}, \subseteq)$$
$$\mathcal{T}[\![\tau^v \to \tau']\!] = (\mathcal{T}[\![\tau]\!])^v \to \mathcal{T}[\![\tau']\!]$$

For the base-type $\bullet$ this is the power set of $\mathcal{S}$ with $\subseteq$ as ordering. For a type of the form $\tau^v \to \tau'$ the elements are functions mapping from elements of type τ to elements of type τ' with respect to v ordered pointwise, means that $\mathcal{T}[\![\tau^v \to \tau']\!]$ denotes the set

$$\{f : \tau^v \to \tau' \mid \text{ f.a } x, y \in \mathcal{T}[\![\tau]\!] : x \sqsubseteq^v_{\mathcal{T}[\![\tau]\!]} y \Rightarrow f(x) \sqsubseteq_{\mathcal{T}[\![\tau']\!]} f(y)\}.$$

The ordering of two functions f, g of type $\tau^v \to \tau'$ is denoted by $\sqsubseteq_{\mathcal{T}[\![\tau \to \tau']\!]}$ and defined as:

$$f \sqsubseteq_{\mathcal{T}[\![\tau \to \tau']\!]} g \Leftrightarrow \text{ f.a. } x \in \mathcal{T}[\![\tau]\!] : f(x) \sqsubseteq_{\mathcal{T}[\![\tau']\!]} g(y),$$

It can be shown that each type τ forms a complete lattice, which results from the fact that the base-type $\bullet$ is a complete lattice with $\cap$ and $\cup$ as meet and join and that each

(var)	$\dfrac{}{\Gamma, X^v : \tau, \Gamma' \vdash X : \tau \text{ if } v \in \{0, +\}}$	(prop)	$\dfrac{}{\Gamma \vdash q : \bullet}$
(or)	$\dfrac{\Gamma \vdash \varphi : \bullet \qquad \Gamma \vdash \psi : \bullet}{\Gamma \vdash \varphi \vee \psi : \bullet}$	(mod)	$\dfrac{\Gamma \vdash \varphi : \bullet}{\Gamma \vdash \langle a \rangle \varphi : \bullet}$
(abs)	$\dfrac{\Gamma, X^v : \tau \vdash \varphi : \tau'}{\Gamma \vdash \lambda(X^v : \tau).\varphi : (\tau^v \to \tau')}$	(fix)	$\dfrac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu(X : \tau).\varphi : \tau}$
(app +)	$\dfrac{\Gamma \vdash \varphi : (\tau^+ \to \tau') \quad \Gamma \vdash \psi : \tau}{\Gamma \vdash (\varphi\, \psi) : \tau'}$	(app -)	$\dfrac{\Gamma \vdash \varphi : (\tau^- \to \tau') \quad \Gamma^- \vdash \psi : \tau}{\Gamma \vdash (\varphi\, \psi) : \tau'}$
(app 0)	$\dfrac{\Gamma \vdash \varphi : (\tau^0 \to \tau') \quad \Gamma \vdash \psi : \tau \quad \Gamma^- \vdash \psi : \tau}{\Gamma \vdash (\varphi\, \psi) : \tau'}$	(neg)	$\dfrac{\Gamma^- \vdash \varphi : \bullet}{\Gamma \vdash \neg\varphi : \bullet}$

TABLE 2.1: The type-system and syntax of HFL.

type is a partially ordered set. Therefore the meet and join for higher order types can be computed pointwise.

2.3 The Syntax of HFL

First a set of transition names $\mathcal{A}$, a set of propositions $\mathcal{P}$ as described in Sec. 2.1 and a set of variable names $\mathcal{V} = \{X, Y, \ldots\}$ are fixed. A formula φ is an element of HFL if it is producible by the following grammar:

$$\varphi ::= q \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid \lambda(X^v : \tau).\varphi \mid (\varphi\, \varphi) \mid \mu(X : \tau).\varphi$$

where $q \in \mathcal{P}$, $a \in \mathcal{A}$, $X \in \mathcal{V}$. τ and v are the type and the variance of a formula as described in the previous section. The set of subformulas of a formula φ is denoted by $sub(\varphi)$ and is defined in the usual way.

In addition to this basic grammar there are some standard abbreviations:

$$\begin{aligned} \text{tt} &:= q \vee \neg q & \qquad \text{ff} &:= \neg\text{tt} \\ \varphi \wedge \psi &:= \neg(\neg\varphi \vee \neg\psi) & \qquad [a]\varphi &:= \neg\langle a \rangle \neg\varphi \\ \nu(X : \tau).\varphi &:= \neg\mu(X : \tau).\neg\varphi[\neg X / X] \end{aligned}$$

In Tab. 2.1 it can be seen how the type of a formula is evaluated. Γ is called a *context* and is a sequence of the form $X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$, where $X_i^{v_i} : \tau_i$ means that X_i has type τ_i and variance v_i in this context. Γ^- means $X_1^{v_1^-} : \tau_1, \ldots, X_n^{v_n^-} : \tau_n$, which is evaluated in the following way: $+^- = -$, $-^- = +$ and $0^- = 0$. Intuitively, it means negating the variance. $\Gamma, X^v : \tau$ denotes $\Gamma \cup X^v : \tau$. We say that a formula φ is *well-typed* if for some Γ and τ the statement $\Gamma \vdash \varphi : \tau$ holds and that φ has type τ if $\Gamma \vdash \varphi : \tau$ can be inferred by using the rules in Tab. 2.1.

By the Kleene Fixpoint Theorem any formula $\mu X.\psi$ of type $\tau_1 \to \ldots \to \tau_k \to \bullet$ is equivalent to a finite approximation X_ψ^n. This is recursively defined as:

$$X_\psi^0 := \lambda X_1 \ldots \lambda X_k.\text{ff} \qquad X_\psi^{i+1} := \psi[X_\psi^i / X]$$

It is assumed that all formulas are *well-named*. This means that in every formula

$$
\begin{array}{rcl}
\mathcal{T}[\![\Gamma \vdash q : \bullet]\!]_\eta & = & \{s \in \mathcal{S} \mid q \in \mathcal{L}(s)\} \\
\mathcal{T}[\![\Gamma \vdash X : \tau]\!]_\eta & = & \eta(X) \\
\mathcal{T}[\![\Gamma \vdash \neg\varphi : \bullet]\!]_\eta & = & S \setminus \mathcal{T}[\![\Gamma^- \vdash \varphi : \bullet]\!]_\eta \\
\mathcal{T}[\![\Gamma \vdash \varphi \vee \psi : \bullet]\!]_\eta & = & \mathcal{T}[\![\Gamma \vdash \varphi : \bullet]\!]_\eta \cup \mathcal{T}[\![\Gamma \vdash \psi : \bullet]\!]_\eta \\
\mathcal{T}[\![\Gamma \vdash \langle a\rangle\varphi : \bullet]\!]_\eta & = & \left\{s \in \mathcal{S} \mid \text{ex. } t \in \mathcal{T}[\![\Gamma \vdash \varphi : \bullet]\!] : s \xrightarrow{a} t\right\} \\
\mathcal{T}[\![\Gamma \vdash \lambda(X^v : \tau).\varphi : \tau^v \to \tau']\!]_\eta & = & f \in \mathcal{T}[\![\tau^v \to \tau']\!] \; s.t. \\
 & & \text{f. a. } d \in \mathcal{T}[\![\tau]\!] : f(d) = \mathcal{T}[\![\Gamma, X^v : \tau \vdash \varphi : \tau']\!]_{\eta\{X \to d\}} \\
\mathcal{T}[\![\Gamma \vdash \varphi\,\psi : \tau']\!]_\eta & = & \mathcal{T}[\![\Gamma \vdash \varphi : \tau^v \to \tau']\!]_\eta \left(\mathcal{T}[\![\Gamma' \vdash \psi : \tau]\!]_\eta\right) \\
\mathcal{T}[\![\Gamma \vdash \mu(X : \tau).\varphi : \tau]\!]_\eta & = & \bigsqcap \left\{d \in \mathcal{T}[\![\tau]\!] \mid \mathcal{T}[\![\Gamma, X^+ : \tau \vdash \varphi : \tau]\!]_{\eta[X \to d]} \sqsubseteq_{\mathcal{T}[\![\tau]\!]} d\right\}
\end{array}
$$

TABLE 2.2: The semantics of HFL.

$\varphi \in$ HFL every variable X occurs only once as a bound variable. Furthermore, there exist no free-variable X in φ if X already occurs as a bound variable.

Some important restrictions due to the type-system explaining its benefits can be noticed at this point: Firstly, the negation $\neg\varphi$ is only defined for formulas φ of type $\bullet$. Secondly, it is ensured that in the application part $\varphi\,\psi$ the input ψ has a correct type regarding the type of the input expected by φ. Lastly, it forces the X in $\mu X.\varphi$ to be monotonic in its arguments. With this and the fact that each type forms a complete lattice the Tarski-Knaster Theorem [Tar55] guarantees the existence of least (and greatest) fixpoints.

2.4 The Semantics of HFL

Let $\mathcal{T}$ be a fixed LTS. η is called an *environment*, which is a possibly partial map on the variable set. An environment η is called Γ-*respecting* with $\Gamma = X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$ if $\eta \models \Gamma$ which means $\eta(X_i) \in \mathcal{T}[\![\tau_i]\!]$, $i \in \{1, \ldots, n\}$. An *update* of η is denoted by $\eta\{X \to d\}$, which maps X to d and everything else is mapped as given by η. It follows that if $\eta \models \Gamma$ and $d \in \mathcal{T}[\![\tau]\!]$ for some type τ then $\eta\{X \to d\} \models \Gamma, X : \tau$, where $X : \tau \notin \Gamma$. With such an environment η with $\eta \models \Gamma$ the semantics of HFL are defined inductively in Tab. 2.2 for any well-typed formula $\Gamma \vdash \varphi : \tau$ to be an element of $\mathcal{T}[\![\tau]\!]$. In the case of application $\varphi\,\psi$ the context Γ' is equal to Γ if $v \in \{0, +\}$ and Γ^- if $v = -$.

For the sake of completeness, the semantics of the standard abbreviations from Sec. 2.3 are:

$$
\begin{array}{rcl}
\mathcal{T}[\![\Gamma \vdash \mathrm{tt} : \bullet]\!]_\eta & = & \mathcal{S} \\
\mathcal{T}[\![\Gamma \vdash \mathrm{ff} : \bullet]\!]_\eta & = & \emptyset \\
\mathcal{T}[\![\Gamma \vdash \varphi \wedge \psi : \bullet]\!]_\eta & = & \mathcal{T}[\![\Gamma \vdash \varphi : \bullet]\!]_\eta \cap \mathcal{T}[\![\Gamma \vdash \psi : \bullet]\!]_\eta \\
\mathcal{T}[\![\Gamma \vdash [a]\varphi : \bullet]\!]_\eta & = & \left\{s \in \mathcal{S} \mid \text{f.a. } t \in \mathcal{S} : s \xrightarrow{a} t \Rightarrow t \in \mathcal{T}[\![\Gamma \vdash \varphi : \bullet]\!]\right\} \\
\mathcal{T}[\![\Gamma \vdash \nu(X : \tau).\varphi : \tau]\!]_\eta & = & \bigsqcup \left\{d \in \mathcal{T}[\![\tau]\!] \mid d \sqsubseteq_{\mathcal{T}[\![\tau]\!]} \mathcal{T}[\![\Gamma, X^+ : \tau \vdash \varphi : \tau]\!]_{\eta\{X \to d\}}\right\}
\end{array}
$$

With this, the approximation given by the Kleene Fixpoint Theorem can be defined on a semantic level as:

$$F^0_\psi := [\![X^0_\psi]\!]^{\mathcal{T}}_\eta \qquad F^{i+1}_\psi := [\![\psi]\!]^{\mathcal{T}}_{\eta\{X \to F^i_\psi\}}$$

Example 2. Consider the following formula:

$$\varphi_{AG} = \lambda(F^- : \bullet^+ \to \bullet).\lambda(G^+ : \bullet^+ \to \bullet).(\nu(Z : \bullet^- \to \bullet^+ \to \bullet).\\ \lambda(X^- : \bullet).\lambda(Y^+ : \bullet).(\neg X \vee Y) \wedge Z\,(F\,X)\,(G\,Y))\,\mathrm{tt}\,(G\,\mathrm{tt})$$

φ_{AG} is of type $(\bullet^+ \to \bullet)^- \to (\bullet^+ \to \bullet)^+ \to \bullet$. This is an encoding of *assume guarantee properties* [VV04], which means that "*Assuming that X holds, then it is guaranteed that Y holds.*" in a model that satisfies φ_{AG}. For instance, if $\lambda X.\langle a\rangle X$ is applied for F and $\lambda Y.\langle b\rangle Y$ is applied for G then $\varphi_{AG}\,F\,G$ means "*Assuming that there is an a-path of length n, then there is a b-path of length n with* $n \geq 0$*.*"

The following conventions are used in this thesis: All formulas are considered well-typed from now on and types and variances of a formula are omitted if they are not relevant in the respective situation. Furthermore variables with a functional type will range over $F, G, \ldots$, variables of type $\bullet$ over $X, Y, \ldots$ and the shorthand $\lambda X_1, X_2, \ldots X_n$ is used instead of $\lambda X_1.\lambda X_2. \ldots \lambda X_n$. With these conventions the formula of Example 2 looks like

$$\lambda F, G.(\nu Z.\lambda X, Y.(\neg X \vee Y) \wedge Z\,(F\,X)(G\,Y))\,\mathrm{tt}\,(G\,\mathrm{tt}).$$

2.5 The First- and Second-Order Fragment of HFL

Different fragments of HFL are defined by the set of types that are needed to type their respective formulas. First, an order of types is defined in the manner given in [ALS07].

$$ord(\bullet) := 0 \qquad ord(\tau \to \tau') := \max\{ord(\tau) + 1, ord(\tau')\}$$

That a type τ has an order of smaller or equal $i \in \mathbb{N}$ is denoted by $\tau^{\leq i}$.

Definition 2. The first-order fragment of the higher-order fixpoint logic *HFL1* is defined as the set of HFL formulas:

$$\mathrm{HFL1} := \{\varphi \in \mathrm{HFL} \mid \text{f.a. } \psi \in Sub(\varphi) : \psi \text{ is of type } \tau^{\leq 1}\}$$

HFL1 is informally defined as the fragment of all formulas, where all occurring types are of the form $\bullet \to \ldots \to \bullet$. This means there are no applications $\psi_1\,\psi_2 \in Sub(\varphi)$ where ψ_2 has a type other than $\bullet$.

Definition 3. The second-order fragment of the higher-order fixpoint logic *HFL2* is defined as the set of HFL formulas:

$$\mathrm{HFL2} := \{\varphi \in \mathrm{HFL} \mid \text{f.a. } \psi \in Sub(\varphi) : \psi \text{ is of type } \tau^{\leq 2}\}$$

The fragment HFL2 can be understood as the fragment of HFL, where in an included formula φ for all formulas $\psi_1\,\psi_2 \in Sub(\varphi)$ holds that ψ_2 has a type of at most $\bullet \to$

$\ldots \to \bullet$, i.e., there are no applications $\psi_1\,\psi_2$ where ψ_2 expects any input of a type other than $\bullet$. With the above definition this means the type τ of ψ_2 has an order of at most 1.

Chapter 3

Model Checking in HFL1 with Neededness Analysis

3.1 Neededness Analysis in HFL1

In traditional data-flow analysis a neededness analysis approach only includes values that appear non-trivial in the computation of a fixpoint of interest [Jør94]. Axelsson and Lange adopted this approach for model checking HFL1 in [AL07]. They exploited the fact that the value of a fixpoint is usually not needed on all possible arguments, hence, the computation can be localized to the necessary ones only. The result is a model checking algorithm that achieves major computational benefits compared to a naive model checking algorithm, which computes every fixpoint of interest entirely. In this introductory work an extended grammar for HFL1 is used. It is defined that a formula φ is an element of HFL1 if it is producible by the following grammar.

$$\varphi ::= q \mid \neg q \mid X \mid \neg X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \varphi\, \varphi \mid \lambda X.\varphi \mid \mu X^{\tau}.\varphi \mid \nu X^{\tau}.\varphi$$

where $q \in \mathcal{P}$, $X \in \mathcal{V}$ and $a \in \mathcal{A}$ of some given LTS $\mathcal{T}$. The semantics are the same as given in Chap. 2 with the restriction that only τ with $ord(\tau) \leq 1$ are possible. For this chapter it is assumed that if a formula $\varphi \in$ HFL1 then it is built with the grammar above. This allows presenting the algorithm in its original form.

3.2 Model Checker *MC* for HFL1

In Algorithm 1 (on p. 12) the model checker `MC` is given, as it was introduced in [AL07]. This algorithm is a symbolic model checker with neededness analysis, which recursively evaluates a formula $\varphi \in$ *HFL1* along its syntax-tree .

A call of `MC` is denoted by `MC`$(\varphi, [T_1, \ldots, T_k])$, where φ is a HFL1-formula of type $\tau = \bullet \to \ldots \to \bullet$ with k function arrows and $[T_1, \ldots, T_k] \in (2^{\mathcal{S}})^k$ is a list of arguments for φ. If $k = 0$, the formula φ is of type $\bullet$, which means it expects no further arguments. The initial call of the algorithm is then denoted by `MC`$(\varphi, [\,])$, where $[\,]$ represents the empty list of arguments. Note that the algorithm also needs the corresponding LTS $\mathcal{T}$ and environment η, but both are needed as global values only, which makes it unnecessary to include them in the signature of `MC`.

The global variable env represents a map from the variable set $\mathcal{V}$ to the set of possibly partial functions from $\mathcal{T}[\![\tau^{\leq 1}]\!]$. It is needed to map variables to their corresponding arguments in functional parts of a formula. In particular, it is needed to build up approximations of fixpoints. env must be initialized according to the environment η if MC should be able to model check a formula φ including free variables. An update of env, where the value of X for $[T_1, \ldots, T_k]$ is set to d is denoted by $\texttt{env}(X)\{[T_1, \ldots, T_k] \to d\}$.

Due to its recursive nature, the algorithm MC is best described for each possible case separately. Assuming that $\texttt{MC}(\varphi, [T_1, \ldots, T_k])$ is called. Then one of the following cases occurs:

Case (q, $\neg q$, $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$,$\langle a \rangle \psi$, $[a]\psi$)*:* Note that for all of these cases φ is of type $\bullet$, which means $k = 0$. With respect to the given LTS $\mathcal{T}$ and environment η the algorithm computes the semantics described in Chap. 2. For the cases $\psi_1 \vee \psi_2$ and $\psi_1 \wedge \psi_2$ it makes recursive calls to evaluate ψ_1, ψ_2 separately, before evaluating the boolean operator.

Case ($\psi_1\ \psi_2$)*:* Note that ψ_2 has to be of type $\bullet$ because $\varphi \in$ HFL1. The value of interest is $[\![\psi_1]\!]^{\mathcal{T}}_{\eta}([\![\psi_2]\!]^{\mathcal{T}}_{\eta}, T_1, \ldots, T_k])$. To compute this, the algorithm MC makes a recursive call and computes $\texttt{MC}(\psi_2, [\,])$ first and then uses this result as the first argument in the call $\texttt{MC}(\psi_1, [\texttt{MC}(\psi_2, [\,]), T_1, \ldots, T_k])$.

Case ($\lambda X.\psi$)*:* When $\texttt{MC}(\lambda X.\varphi, [T_1, \ldots, T_k])$ is called, $\texttt{env}(X)$ is updated with $\{[\,] \to T_1\}$. Again, from the fact that $\varphi \in$ HFL1 it follows that X is of type $\bullet$. This means the argument T_1 expects no input itself and therefore can be represented as a function from the singleton domain $\{[\,]\}$ to T_1.

Case (X)*:* If φ is a free variable a case distinction is needed: If $\texttt{env}(X)$ is undefined for $[T_1, \ldots, T_k]$, then X has to be a fixpoint variable and it has to be the first application of the arguments $[T_1, \ldots, T_k]$. It can be verified that in all other cases X would have been defined at this point. If this is the case then X is mapped to $\emptyset$ or $\mathcal{S}$, depending on whether X is a least of greatest fixpoint variable, in order to set an initial value for the fixpoint computation. After that and in all other cases, the algorithm returns the value $\texttt{env}(X)([T_1, \ldots, T_k])$. If X is of type $\bullet$ the value is $\texttt{env}(X)([\,])$. This makes clear, why an abstraction of base-type arguments to a singleton function is needed.

Case ($\neg X$)*:* Note that X can only be of type $\bullet$. Therefore the algorithm simply returns $\mathcal{S} \setminus \texttt{env}(X)([\,])$ which is the complement of the value of X in $\mathcal{S}$.

Case ($\varphi = \sigma X.\psi$, $\sigma \in \{\mu, \nu\}$)*:* If a fixpoint case occurs then the algorithm approximates the fixpoint $[\![\sigma X.\psi]\!]^{\mathcal{T}}_{\eta}$ sufficiently to compute its value for the given argument list $[T_1, \ldots, T_k]$. The mapping of env (X) is used to build up this approximation step-by-step. The algorithm starts with mapping $\texttt{env}(\mathrm{X})\,([T_1, \ldots, T_k])$ to the value $\emptyset$ or $\mathcal{S}$ according to the binding fixpoint-operator of X. After this initial operation the actual fixpoint computation begins. In every iteration of the repeat-loop the algorithm computes the value of $[\![\psi]\!]^{\mathcal{T}}_{\eta\{X \to \texttt{env}(X)\}}([T'_1, \ldots, T'_k])$ for all $[T'_1, \ldots, T'_k] \in \mathit{dom}(\texttt{env}(X))$, where $\mathit{dom}(\texttt{env}(X))$ is the domain of the partial function that $\texttt{env}(X)$ currently represents. Note that in the first iteration $\mathit{dom}(\texttt{env}(X)) = \{[T_1, \ldots, T_k]\}$. If an application $\texttt{env}(X)\,([T'_1, \ldots, T'_k])$ occurs during such a computation, where the argument list $[T'_1, \ldots, T'_k]$ is undefined for $\texttt{env}(X)$, the value for this argument list is initialized as described in the variable case above. This means that the argument list is added

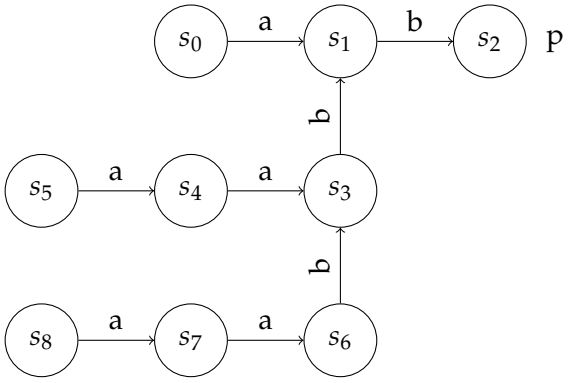

FIGURE 3.1: LTS with paths of the form $\langle a^n b^n \rangle$.

to $dom(\texttt{env}(X))$ and with that included in further iterations of the repeat-loop. This is in fact how the algorithm avoids including unnecessary arguments in a fixpoint computation: Only those argument lists are added to the domain of $\texttt{env}(X)$ for which the values of already included argument lists depend on and ultimately the value $[\![\sigma X.\psi]\!]^{\mathcal{T}}_{\eta}([T_1, \ldots, T_k])$ depends on. The repeat-loop terminates if no changes happened in the last iteration. This means that no new argument list was added to $dom(\texttt{env}(X))$ and no value of $\texttt{env}(X)$ for any argument list in $dom(\texttt{env}(X))$ changed. After this the value of $\texttt{env}(X)$ $([T_1, \ldots, T_k])$ is returned. This is sufficient to ensure that $\texttt{MC}(\sigma X.\psi, [T_1, \ldots, T_k]) = [\![\sigma X.\psi]\!]^{\mathcal{T}}_{\eta}([T_1, \ldots, T_k])$ holds [AL07].

3.3 Application: Existence of Paths Definable in HFL1

To obtain a better understanding of how neededness analysis for model checking HFL1 works as well as showing the expressive power of HFL1 compared to the modal μ-calculus, consider the following HFL1 formula:

$$\varphi := (\mu F.\lambda X.X \vee \langle a \rangle (F \langle b \rangle X))\ p$$

By unfolding the fixpoint part of φ to an infinite disjunction

$$p \vee \langle a \rangle \langle b \rangle p \vee \langle a \rangle \langle a \rangle \langle b \rangle \langle b \rangle p \vee \ldots$$

it can be seen that the formula describes the property "*There exists a word w of the context-free language $\{a^n b^n \mid n \in \mathbb{N}\}$ such that there is a state with proposition p that is reachable under w*". This language is not regular, which means that this property is not expressible in the modal μ-calculus as stated in Chap. 1. Without explicit model checking it can be verified that in Fig. 3.1 this property holds for $\{s_0, s_2, s_5\}$.

In the following this property is model checked for the given LTS with the algorithm MC. From the fact that φ is of type $\bullet$, it follows that MC needs no further arguments and is called with $\texttt{MC}(\varphi, [\,])$. Considering the syntaxtree of φ, the highest-level operation is an application which leads to the case $\psi_1\ \psi_2$ and the call $\texttt{MC}(\psi, [\texttt{MC}(p, [\,])])$, where $\psi := (\mu F.\lambda X.X \vee \langle a \rangle (F \langle b \rangle X))$. From the propositional case of MC and the given LTS it follows directly that $\texttt{MC}(p, [\,]) = \{s_2\}$. This is the first argument for the following fixpoint computation.

Input: $\varphi \in \text{HFL1}$, $[T_1, \ldots, T_k] \in ((2^{\mathcal{S}}))^k$, $\mathcal{T}$ and η
Output: $[\![\varphi]\!]^{\mathcal{T}}_{\eta}\ ([T_1, \ldots, T_k])$

$\texttt{env} : \mathcal{V} \to \mathcal{T}[\![\tau^{\leq 1}]\!]$
Function $\texttt{MC}(\varphi, [T_1, \ldots, T_k])$
  switch φ **do**
    case q **do return** $\mathcal{L}(q)$;
    case $\neg q$ **do return** $\mathcal{S} \setminus \mathcal{L}(q)$;
    case $\psi_1 \vee \psi_2$ **do return** $\texttt{MC}(\psi_1, [\,]) \cup \texttt{MC}(\psi_2, [\,])$;
    case $\psi_1 \wedge \psi_2$ **do return** $\texttt{MC}(\psi_1, [\,]) \cap \texttt{MC}(\psi_2, [\,])$;
    case $\langle a \rangle \psi$ **do return** $\left\{ s \in \mathcal{S} \mid \textit{ex. } t \in \texttt{MC}(\psi, [\,])\ \textit{s.t. } s \xrightarrow{a} t \right\}$;
    case $[a]\psi$ **do return** $\left\{ s \in \mathcal{S} \mid \textit{f.a. } t \in \mathcal{S} : s \xrightarrow{a} t \Rightarrow t \in \texttt{MC}(\psi, [\,]) \right\}$;
    case X **do**
      if $\texttt{env}(X)([T_1, \ldots, T_k]) = \text{UNDEF}$ **then**
        if $\texttt{fixpoint}(X) = \mu$ **then** $T := \emptyset$;
        else $T := \mathcal{S}$;
        $\texttt{env}(X) = \texttt{env}(X)\ \{[T_1, \ldots, T_k] \to T\}$
      return $\texttt{env}(X)([T_1, \ldots, T_k])$
    case $\neg X$ **do return** $\mathcal{S} \setminus \texttt{env}(X)([\,])$;
    case $\lambda X.\psi$ **do**
      $\texttt{env}(X) := \{[\,] \to T_1\}$;
      return $\texttt{MC}(\psi, [T_2, \ldots, T_k])$
    case $\psi_1\ \psi_2$ **do return** $\texttt{MC}(\psi_1, [\texttt{MC}(\psi_2, [\,]), T_1, \ldots, T_k])$;
    case $\sigma X.\psi$ **do**
      if $\sigma = \mu$ **then** $T := \emptyset$;
      else $T := \mathcal{S}$;
      $\texttt{env}(X) := \{[T_1, \ldots, T_k] \to T\}$;
      repeat
        $f := \texttt{env}(X)$;
        forall $[T'_1, \ldots, T'_k] \in \mathit{dom}(\texttt{env}(X))$ **do**
          $\texttt{env}(X) := \texttt{env}(X)\ \{[T'_1, \ldots, T'_k] \to \texttt{MC}(\psi, [T'_1, \ldots, T'_k])\}$
        end
      until $f = \texttt{env}(X)$;
      return $\texttt{env}(X)([T_1, \ldots, T_k])$
  end
end

Algorithm 1: Symbolic model checking algorithm for HFL1.

Iter. \ Arg.	$\{s_2\}$	$\{s_1\}$	$\{s_3\}$	$\{s_6\}$	$\emptyset$
0	$\emptyset$				
1	$\{s_2\}$	$\emptyset$			
2	$\{s_2\}$	$\{s_1\}$	$\emptyset$		
3	$\{s_0, s_2\}$	$\{s_1\}$	$\{s_3\}$	$\emptyset$	
4	$\{s_0, s_2\}$	$\{s_1, s_4\}$	$\{s_3\}$	$\{s_6\}$	$\emptyset$
5	$\{s_0, s_2, s_5\}$	$\{s_1, s_4\}$	$\{s_3, s_7\}$	$\{s_6\}$	$\emptyset$
6	$\{s_0, s_2, s_5\}$	$\{s_1, s_4, s_8\}$	$\{s_3, s_7\}$	$\{s_6\}$	$\emptyset$
7	$\{s_0, s_2, s_5\}$	$\{s_1, s_4, s_8\}$	$\{s_3, s_7\}$	$\{s_6\}$	$\emptyset$

TABLE 3.1: Fixpoint computation of example in Sec. 3.3.

This fixpoint computation is listed in Tab. 3.1. Row i denotes $\texttt{env}(F)$ after the i-th execution of the repeat-loop. Empty cells can be interpreted as still undefined arguments. The initial argument, as stated above, is $\{s_2\}$. Therefore, it is added to $dom(\texttt{env}(F))$ and its value is set to $\emptyset$ before the first iteration of the repeat-loop. This can be seen in Row 0. In the first iteration of the repeat-loop the only update happening is $\texttt{env}(F)\{\{s_2\} \to \texttt{MC}(\psi, [\{s_2\}])\}$. The application $\texttt{env}(F)(\{s_1\})$ appears in the call of $\texttt{MC}(\psi, [\{s_2\}])$. Therefore, the argument $\{s_1\}$ is added to $dom(\texttt{env}(F))$ and its value is set to $\emptyset$. With that the value of $\texttt{env}(F)$ for $\{s_1\}$ changes and is updated. The result can be seen in Row 1. This procedure is repeated until the iteration becomes stable, which means that no further undefined arguments for F occurred and no value of any argument in $dom(\texttt{env}(F))$ changed. This can be seen in comparison of Row 6 and 7. With this the repeat-loop terminates after the 7-th iteration and the current and correct value for $\texttt{env}(F)(\{s_2\})$ is returned, which is $\{s_0, s_2, s_5\}$.

It can be seen that the algorithm localizes the computation to the needed arguments only. In quantitative terms, this means that only the values of 5 arguments were computed instead of the values of all $|2^{\mathcal{S}}| = 2^9 = 512$ possible arguments. This underlines that this neededness analysis approach achieves a major reduction of computational effort compared to a naive approach.

Chapter 4

Model Checking in HFL2 with Neededness Analysis

4.1 Neededness Analysis in HFL2

Model checking HFL2 with neededness analysis can be done in a similar way as model checking HFL1, described in Chap 3. The computation of the value of a fixpoint for a certain argument can be localized by starting the computation with this argument and including only those arguments that the value depends on.

The only difference is the possible type of applicable arguments. In HFL1 the type of ψ_2 in a formula $\psi_1\ \psi_2$ is restricted to the base-type $\bullet$, whereas in HFL2 the formula ψ_2 can be of type $\bullet \to \ldots \to \bullet$. For a given LTS $\mathcal{T}$ this means that a corresponding algorithm for model checking HFL2 with neededness analysis needs to handle arguments from $\mathcal{T}[\![\tau^{\leq 1}]\!]$, respectively their syntactical representation.

It can be considered that the values of such functional arguments are only needed for certain arguments in their specific applications in the fixpoint formula. However, a partial evaluation of these arguments comes with unsolved problems, which are further discussed in Chap. 5.

4.2 Model Checker MC_2 for HFL2

In Alg. 2 (on p. 17) a symbolic model checker with neededness analysis for HFL2 is given, which is denoted by $\texttt{MC}_2$. This model checker should be understood as a development of the approach adopted by the algorithm `MC` described in Chap. 3, hence there can be seen a far-reaching similarity between these two algorithms. Note that the algorithm $\texttt{MC}_2$ uses the grammar defined in Sec. 2.3.

Consider the signature of $\texttt{MC}_2$ and an example call $\texttt{MC}_2(\varphi, [T_1, \ldots, T_k])$, where φ is of type $\tau_1 \to \ldots \tau_k \to \bullet$. Applicable formulas φ are from HFL2. This means that the argument list parameter $[T_1, \ldots, T_k]$ is an element of $(\mathcal{T}[\![\tau^{\leq 1}]\!])^k$. Furthermore, the algorithm needs an LTS $\mathcal{T}$ and an environment η. These inputs are needed as global values only, therefore it is not necessary to include them in the signature of $\texttt{MC}_2$ directly.

The global variable `env` is a map, mapping from the set of variables V to the set of possibly partial functions from $\mathcal{T}[\![\tau^{\leq 2}]\!]$. This includes state sets from $\mathcal{S}$, functions between those state sets and functions between those functions. The variable `env`

has the same functionality in the different functional parts of $\texttt{MC}_2$ as in the algorithm MC: In the λ and variable case it is used to map a variable to the correct argument and in the μ case it is used to compute an approximation of the fixpoint considered. Furthermore, the global variable env needs to be initialized according to η in order to enable model checking formulas including free variables.

The structure of the algorithm $\texttt{MC}_2$ is similar to that of the algorithm MC as well. It is a symbolic model checker that recursively checks a given formula along its syntax tree. Therefore its functionality is best discussed for the different cases separately.

Case (q, $\neg\psi$, $\psi_1 \vee \psi_2$, $\langle a \rangle \psi$)*:* In all of these cases the algorithm $\texttt{MC}_2$ computes the return value according to the semantics defined in Sec. 2.4. For the cases q, $\psi_1 \vee \psi_2$ and $\langle a \rangle \psi$ this is done in the same way as in the algorithm MC. This is possible, because in these cases the formulas are still restricted to the base-type $\bullet$. In the case of $\neg\psi$ the subformula ψ has to be of type $\bullet$ too, hence, the algorithm computes $\texttt{MC}_2(\psi, [\,])$ and returns its complement in $\mathcal{S}$.

Case ($\psi_1\, \psi_2$)*:* In comparison to the algorithm MC this case is computed differently, because ψ_2 is not necessarily of type $\bullet$ anymore. By assumption ψ_2 is of type $\tau_1 \to \ldots \to \tau_{k'} \to \bullet$. The algorithm computes the value of $[\![\psi_2]\!]^{\mathcal{T}}_{\eta}$ first. This can be seen in Alg. 2 in line 9-13. First the algorithm identifies how many arguments the formula ψ_2 expects, which is done in the call of $\texttt{\#args}(\psi_2)$. For instance, in an actual implementation this could be derived from the type of ψ_2. In the next step it computes $[\![\psi_2]\!]^{\mathcal{T}}_{\eta}$ by initializing a map f_{ψ_2} and computing $[\![\psi_2]\!]^{\mathcal{T}}_{\eta}\,([T'_1, \ldots, T'_{k'}])$ for all possible $[T'_1, \ldots, T'_{k'}] \in (2^{\mathcal{S}})^{k'}$. This is done in the for-all loop. Note that in the case that $k' = 0$ this creates a function that maps the empty list $[\,]$ to $[\![\psi_2]\!]^{\mathcal{T}}_{\eta}$. Once the for-all loop terminates, the algorithm uses f_{ψ_2} as the first argument in the call $\texttt{MC}_2$ $(\psi_1, [f_{\psi_2}, T_1, \ldots, T_k])$, which computes the value of interest $[\![\psi_1]\!]^{\mathcal{T}}_{\eta}\,([[\![\psi_2]\!]^{\mathcal{T}}_{\eta}, T_1, \ldots, T_k])$.

Case ($\lambda X.\psi$)*:* In this case the algorithm sets $\texttt{env}(X) := T_1$. Considering the variable case, this can be understood as mapping X to T_1 in all following occurrences of X. After this, the algorithm returns the value of the call $\texttt{MC}_2(\psi, [T_2, \ldots, T_k])$. Note that from the application case it is guaranteed that T_1 is a function, assuming that $\texttt{MC}_2$ is called correctly initially.

Case (X)*:* In this case the algorithm $\texttt{MC}_2$ checks if the value of $\texttt{env}(X)$ is undefined for $[T_1, \ldots, T_k]$. If so, X has to be bound by a μ-operator and it has to be the first occurrence of this application, hence, the value of $\texttt{env}(X)\,([T_1, \ldots, T_k])$ is set to $\emptyset$. After this and in every other case the algorithm returns the value $\texttt{env}(X)\,([T_1, \ldots, T_k])$. Note that for an initially unbound variable env is initialized corresponding to η.

Case ($\mu X.\psi$)*:* In the fixpoint part the algorithm works similar to the algorithm MC. It starts with setting the value of $\texttt{env}(X)\,([T_1, \ldots, T_k])$ to $\emptyset$, followed by the actual fixpoint computation. In each iteration of the repeat-loop the algorithm updates the values for the arguments included in $dom(\texttt{env}(X))$. The value of newly occurring arguments is initialized with $\emptyset$, as can be seen in the variable case, hence, it is included in further iterations. This is done until stability is reached, which means that no new arguments occurred and no values for included arguments changed. After this the algorithm returns the value of $\texttt{env}(\mathrm{X})\,([T_1, \ldots, T_k])$.

Input: $\varphi \in$ HFL2, $[T_1, \ldots, T_k] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^k$, $\mathcal{T}$ and η
Output: $[\![\varphi]\!]^{\mathcal{T}}_{\eta}\ ([T_1, \ldots, T_k])$

$\texttt{env} : \mathcal{V} \to \mathcal{T}[\![\tau^{\leq 2}]\!]$
Function $\texttt{MC}_2(\varphi, [T_1, \ldots, T_k])$
  switch φ **do**
    case q **do return** $\mathcal{L}(q)$;
    case $\neg\psi$ **do return** $\mathcal{S} \setminus \texttt{MC}_2(\psi, [T_1, \ldots, T_k])$;
    case $\psi_1 \vee \psi_2$ **do return** $\texttt{MC}_2(\psi_1, []) \cup \texttt{MC}_2(\psi_2, [])$;
    case $\langle a \rangle \psi$ **do return** $\left\{ s \in \mathcal{S} \mid ex.\ t \in \texttt{MC}_2(\psi, [])\ s.t.\ s \xrightarrow{a} t \right\}$;
    case $\psi_1\ \psi_2$ **do**
      $k' = \texttt{\#args}(\psi_2)$;
      $f_{\psi_2} := \{ \text{UNDEF} \}$;
      forall $[T'_1, \ldots, T'_{k'}] \in (2^{\mathcal{S}})^{k'}$ **do**
        $f_{\psi_2} := f_{\psi_2} \{ T \to \texttt{MC}_2(\psi_2, [T'_1, \ldots, T'_{k'}]) \}$
      end
      return $\texttt{MC}_2(\psi_1, [f_{\psi_2}, T_1, \ldots, T_k])$
    case X **do**
      if $\texttt{env}(X)\ \{[T_1, \ldots, T_k]\} = \text{UNDEF}$ **then**
        $T := \emptyset$;
        $\texttt{env}(X) := \texttt{env}(X)\ \{[T_1, \ldots, T_k] \to T\}$
      return $\texttt{env}(X)\ ([T_1, \ldots, T_k])$
    case $\lambda X.\psi$ **do**
      $\texttt{env}(X) := T_1$;
      return $\texttt{MC}_2(\psi, [T_2, \ldots, T_k])$
    case $\mu X.\psi$ **do**
      $T := \emptyset$;
      $\texttt{env}(X) := \{[T_1, \ldots T_k] \to T\}$;
      repeat
        $f := \texttt{env}(X)$;
        forall $[T'_1, \ldots, T'_k] \in dom(\texttt{env}(X))$ **do**
          $\texttt{env}(X) := \texttt{env}(X) \left\{ [T'_1, \ldots, T'_k] \to \texttt{MC}_2(\psi, [T'_1, \ldots, T'_k]) \right\}$
        end
      until $f = \texttt{env}(X)$;
      return $\texttt{env}(X)\ ([T_1, \ldots, T_k])$
  end
end

Algorithm 2: Symbolic model checking algorithm for HFL2.

4.3 Correctness of MC_2

Due to the far-reaching similarity between the algorithms MC and $\mathtt{MC}_2$, proving correctness of $\mathtt{MC}_2$ can be done similar to the correctness proof of MC in [AL07]. For the μ case a fundamentally different proof is given.

In general this proof is done inductively on the structure of a formula $\varphi \in$ HFL2. For that the following assumption is made: The call $\mathtt{MC}_2(\varphi, [T_1, \ldots, T_k])_\eta$ denotes the result of $\mathtt{MC}_2$ with formula φ and argument list $[T_1, \ldots, T_k]$ if the global variable env has the following form before the call: For all $X \in \mathcal{V}$ of type $\tau_1 \to \ldots \to \tau_k \to \bullet$ and all $[T_1, \ldots, T_k] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^k$

- if X is λ-bound then $\mathtt{env}(X)[T_1, \ldots, T_k] = \eta(X)[T_1, \ldots, T_k]$
- if X is μ-bound then $\eta(X)([T_1, \ldots, T_k]) \neq \emptyset$ implies $\mathtt{env}(X)([T_1, \ldots, T_k]) = \eta(X)([T_1, \ldots, T_k])$
- if X is ν-bound then $\eta(X)([T_1, \ldots, T_k]) \neq \mathcal{S}$ implies $\mathtt{env}(X)([T_1, \ldots, T_k]) = \eta(X)([T_1, \ldots, T_k])$

In order to prove correctness of $\mathtt{MC}_2$, the following lemma is needed. Let ϵ be an element of $\{\in, \notin\}$, $\epsilon^+ := \epsilon$ and $\epsilon^- := \neg\, \epsilon$.

Lemma 1. *For all LTS $\mathcal{T}$, all environments η, all $\epsilon \in \{\in, \notin\}$, all $\varphi \in$ HFL2 of type $\tau_1 \to \ldots \to \tau_k \to \bullet$, where the variance v of the variable X is $+$ or $-$, all $[T_1, \ldots, T_k]$, where $T_i \in \mathcal{T}[\![\tau_i]\!]$ and all partial functions F_1, F_2 of type $\tau'_1 \to \ldots \to \tau'_{k'} \to \bullet$, such that $[\![\varphi]\!]^{\mathcal{T}}_{\eta\{X \to F_i\}}([T_1, \ldots, T_k])$ is defined and f.a. $\psi_1\, \psi_2 \in sub(\varphi)$ the value $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_i\}}$ is defined:*

If there is $s \in S$ s.t. $s\, \epsilon([\![\varphi]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}([T_1, \ldots, T_k]))$ and $s\, \epsilon^-([\![\varphi]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}([T_1, \ldots, T_k]))$, then there is $s' \in S$ s.t. $s'\, \epsilon^v(F_1([T'_1, \ldots, T'_{k'}]))$ and $s'\, \epsilon^{-^v}(F_2([T'_1, \ldots, T'_{k'}]))$.

Informally, this lemma states: If the value of a formula φ is different under the environment $\eta\{X \to F_1\}$ than it is under the environment $\eta\{X \to F_2\}$, at least one of the occurring applications of F_1 must differ from the corresponding application of F_2.

Proof. This lemma can be proven inductively on the structure of the formula φ. In the case that φ is of type $\bullet$ the application $[\![\varphi]\!]^{\mathcal{T}}_\eta([\,])$ is denoted by $[\![\varphi]\!]^{\mathcal{T}}_\eta$.

Note that from the fact that $[\![\varphi]\!]^{\mathcal{T}}_{\eta\{X \to F_i\}}([T_1, \ldots, T_k])$ is defined for F_1 and F_2 follows that all occurring applications of F_1 and F_2 in $[\![\varphi]\!]^{\mathcal{T}}_{\eta\{X \to F_i\}}([T_1, \ldots, T_k])$ are defined. In Sec. 2.3 it is described that any fixpoint formula $\mu X.\psi$ is equivalent to a finite approximation X^n_ψ. Hence, all $\mu X.\psi \in sub(\varphi)$ can be replaced by the corresponding X^n_ψ, which means w.l.o.g. it can be assumed that a fixpoint formula does not occur. For the cases $\varphi = q$ and $\varphi = X'$, where $X' \neq X$, the premise is wrong in any case. The statement is proven for the case that the variance v of X is $+$. The case that $v = -$ is proven analogously.

Case ($\varphi = X$)*:* It can be inferred directly, that if there is a state $s \in S$ such that $s\, \epsilon([\![X]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}([T_1, \ldots, T_k]))$ and $s\, \epsilon^-([\![X]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}([T_1, \ldots, T_k]))$, then it follows that $s\, \epsilon(F_1([T_1, \ldots, T_k]))$ and $s\, \epsilon^-(F_2([T_1, \ldots, T_k]))$, because $\eta\{X \to F_i\}(X) = F_i$ for $i \in \{1, 2\}$.

Case ($\varphi = \neg\psi$)*:* Note that φ is of type $\bullet$ in this case. By definition, $[\![\neg\psi]\!]^{\mathcal{T}}_{\eta}$ is the same as $\mathcal{S} \setminus [\![\psi]\!]^{\mathcal{T}}_{\eta}$. This means that if there is an $s \in \mathcal{S}$ such that $s\,\epsilon\,[\![\neg\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}$ and $s\,\epsilon^{-}\,[\![\neg\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$, then $s\,\epsilon^{-}\,[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}$ and $s\,\epsilon\,[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$. With use of the induction hypothesis, it follows that there is a state $s' \in S$ such that $s'\,\epsilon^{-^{v'}}(F_1([T'_1,\ldots,T'_{k'}]))$ and $s'\,\epsilon^{v'}(F_2([T'_1,\ldots,T'_{k'}]))$. But this v' is the variance of X regarding ψ, which has to be $-$, because the variance of X in $\neg\psi$ is $+$. Then it follows directly that $s'\,\epsilon(F_1([T'_1,\ldots,T'_{k'}]))$ and $s'\,\epsilon^{-}(F_2([T'_1,\ldots,T'_{k'}]))$.

Case ($\varphi = \psi_1 \vee \psi_2$)*:* Note that φ is of type $\bullet$ in this case. By definition $[\![\psi_1 \vee \psi_2]\!]^{\mathcal{T}}_{\eta} = [\![\psi_1]\!]^{\mathcal{T}}_{\eta} \cup [\![\psi_2]\!]^{\mathcal{T}}_{\eta}$. Then there exists at least one $i \in \{1,2\}$ such that if there is a state $s \in \mathcal{S}$ such that $s\,\epsilon[\![\psi_1 \vee \psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}$ and $s\,\epsilon^{-}[\![\psi_1 \vee \psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$, then $s\,\epsilon[\![\psi_i]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}$ and $s\,\epsilon^{-}[\![\psi_i]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$. Using the induction hypothesis, it immediately follows that the statement is true.

Case ($\varphi = \langle a\rangle\psi$)*:* Note that φ is of type $\bullet$ in this case. By definition $[\![\langle a\rangle\psi]\!]^{\mathcal{T}}_{\eta} = \{\tilde{s} \in \mathcal{S} \mid \text{ex. } \tilde{s}' \in [\![\psi]\!]^{\mathcal{T}}_{\eta} : \tilde{s} \xrightarrow{a} \tilde{s}'\}$. With this it follows that if there is a state $s \in \mathcal{S}$ such that $s\,\epsilon[\![\langle a\rangle\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}$ and $s\,\epsilon^{-}[\![\langle a\rangle\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$, then there is a state $s' \in \mathcal{S}$ such that $s'\,\epsilon[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}$ and $s'\,\epsilon^{-}[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$. Applying the induction hypothesis, it immediately follows that the statement is true.

Case ($\varphi = \psi_1\,\psi_2$)*:* By definition $[\![\psi_1\,\psi_2]\!]^{\mathcal{T}}_{\eta} = [\![\psi_1]\!]^{\mathcal{T}}_{\eta}\,[\![\psi_2]\!]^{\mathcal{T}}_{\eta}$. With this it can be inferred that if there is a state $s \in \mathcal{S}$ such that $s\,\epsilon([\![\psi_1\,\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}([T_1,\ldots,T_k]))$ and $s\,\epsilon^{-}([\![\psi_1\,\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}([T_1,\ldots,T_k]))$, then $s\,\epsilon([\![\psi_1]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}([[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}},T_1,\ldots,T_k]))$ and $s\,\epsilon^{-}([\![\psi_1]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}([[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}},T_1,\ldots,T_k]))$. At this point a case distinction is needed:

If $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}} = [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$ then the premise of the lemma is true. This means that the lemma can be applied to the statement $s\,\epsilon([\![\psi_1]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}([[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}},T_1,\ldots,T_k]))$ and $s\,\epsilon^{-}([\![\psi_1]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}([[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}},T_1,\ldots,T_k])$. Therefore the conclusion follows immediately in this case.

Otherwise, if $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}} \neq [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$ then a case distinction between $\epsilon = \in$ and $\epsilon = \notin$ and for each between the three possibilities of inequality $\sqsupset$, $\sqsubset$ and incomparability is needed. But it turns out that the proofs of the cases $\in, \sqsupset$ and $\notin, \sqsubset$ are identical and the proofs of the cases $\in, \sqsubset$ and $\notin, \sqsupset$ are identical too. The incomparability case is proven equally for both possibilities of ϵ. Therefore, it suffices to show the three inequality cases for $\epsilon = \in$.

From the inequality follows that there has to be at least one occurrence of X in ψ_2. Note that this means that ψ_1 has to be monotonic or anti-monotonic regarding ψ_2, depending on the variance of X in ψ_2. Furthermore, note that $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_i\}}$ is defined for F_1 and F_2. Assuming that ψ_2 is of type $\tau_1 \to \ldots \to \tau_{\tilde{k}} \to \bullet$:

1. If $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}} \sqsupset [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}$, then there exists a $[\tilde{T}_1,\ldots,\tilde{T}_{\tilde{k}}] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^{\tilde{k}}$ such that $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}([\tilde{T}_1,\ldots,\tilde{T}_{\tilde{k}}]) \supsetneq [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}([\tilde{T}_1,\ldots\tilde{T}_{\tilde{k}}])$. This means there exists at least one state $\tilde{s} \in \mathcal{S}$ such that $\tilde{s}\,\epsilon([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_1\}}([\tilde{T}_1,\ldots,\tilde{T}_{\tilde{k}}]))$ and $\tilde{s}\,\epsilon^{-}([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X\to F_2\}}([\tilde{T}_1,\ldots,\tilde{T}_{\tilde{k}}]))$. Using the induction hypothesis, it follows that there is $s' \in \mathcal{S}$ such that $s'\,\epsilon^{v'}(F_1([T'_1,\ldots,T'_{k'}]))$ and $s'\,\epsilon^{-^{v'}}(F_2([T'_1,\ldots,T'_{k'}]))$.

This v' is the variance of X regarding ψ_2. From the fact that the premise of the lemma holds for $\psi_1\ \psi_2$ it follows that ψ_1 has to be monotonic regarding ψ_2 in this case. Therefore v' must be $+$ to ensure that $v = +$. This means that $s' \epsilon (F_1\,([T'_1, \ldots, T'_{k'}]))$ and $s' \epsilon^- (F_2\,([T'_1, \ldots, T'_{k'}]))$, which is in fact the conclusion of the lemma for the original premise regarding $\psi_1\ \psi_2$.

2. If $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}} \sqsubset [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}$, then there exists a $[\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^{\tilde{k}}$ such that $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}\,([\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}]) \subsetneq [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}\,([\tilde{T}_1, \ldots \tilde{T}_{\tilde{k}}])$. This means there exists at least one state $\tilde{s} \in \mathcal{S}$ such that $\tilde{s}\,\epsilon^-([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}\,([\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}]))$ and $\tilde{s}\,\epsilon([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}\,([\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}]))$. Using the induction hypothesis, it follows that there is $s' \in \mathcal{S}$ such that $s'\,\epsilon^{-^{v'}}(F_1\,([T'_1, \ldots, T'_{k'}]))$ and $s'\,\epsilon^{v'}(F_2\,([T'_1, \ldots, T'_{k'}]))$. This v' is the variance of X regarding ψ_2. From the fact that the premise of the lemma holds for $\psi_1\ \psi_2$ follows that ψ_1 has to be anti-monotonic regarding ψ_2 in this case. Therefore v' must be $-$ to ensure that $v = +$. This means that $s' \epsilon (F_1\,([T'_1, \ldots, T'_{k'}]))$ and $s' \epsilon^- (F_2\,([T'_1, \ldots, T'_{k'}]))$, which is in fact the conclusion of the lemma for the original premise regarding $\psi_1\ \psi_2$.

3. If $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}} \not\sqsupseteq [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}$ and $[\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}} \not\sqsubseteq [\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}$ is the case, then there has to exist a $[\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^{\tilde{k}}$ such that there exists a state $s_1 \in \mathcal{S}$ such that $s_1\,\epsilon([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}\,([\tilde{T}_1, \ldots \tilde{T}_{\tilde{k}}]))$ and $s_1\,\epsilon^-([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}\,([\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}]))$ and there exists another state $s_2 \in \mathcal{S}$ such that $s_2\,\epsilon^-([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}\,([\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}]))$ and that $s_2\,\epsilon([\![\psi_2]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}\,([\tilde{T}_1, \ldots, \tilde{T}_{\tilde{k}}]))$. Depending on whether ψ_1 is monotonic or anti-monotonic regarding ψ_2, the state s_1 respectively s_2 can be chosen. Then the conclusion of the statement follows with the same argumentation as in the $\sqsupset$, respectively the $\sqsubset$ case.

Case ($\varphi = \lambda X'.\psi$)*:* Using β-reduction[1], it follows that $[\![\lambda X'.\psi]\!]^{\mathcal{T}}_{\eta}\,([T_1, \ldots, T_k]) = [\![\psi]\!]^{\mathcal{T}}_{\eta\{X' \to T_1\}}\,([T_2, \ldots, T_k])$. With this, it can be inferred that if there is $s \in \mathcal{S}$ such that $s\,\epsilon([\![\lambda X'.\psi]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}}\,([T_1, \ldots, T_k]))$ and $s\,\epsilon^-([\![\lambda X'.\psi]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}}\,([T_1, \ldots, T_k]))$, it follows that $s\,\epsilon([\![\psi]\!]^{\mathcal{T}}_{\eta\{X \to F_1\}\{X' \to T_1\}}\,([T_2, \ldots, T_k]))$ and $s\,\epsilon^-([\![\psi]\!]^{\mathcal{T}}_{\eta\{X \to F_2\}\{X' \to T_1\}}\,([T_2, \ldots, T_k]))$. Let $\eta' := \eta\{X' \to T_1\}$. Then, it follows from the fact that X' is not an element of $sub(X)$ that $s\,\epsilon([\![\psi]\!]^{\mathcal{T}}_{\eta'\{X \to F_1\}}\,([T_2, \ldots, T_k]))$ and $s\,\epsilon^-([\![\psi]\!]^{\mathcal{T}}_{\eta'\{X \to F_2\}}\,([T_2, \ldots, T_k]))$. Applying the induction hypothesis, it immediately follows that the statement is true.

□

With this the following theorem can be proven, which shows correctness of the algorithm $\mathtt{MC}_2$.

Theorem 1. *For all LTS $\mathcal{T}$, all environments η, all formulas $\varphi \in$ HFL2 of type $\tau_1 \to \ldots \to \tau_k \to \bullet$ and all $[T_1, \ldots, T_k] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^k$: $\mathtt{MC}_2(\varphi, [T_1, \ldots, T_k])_\eta = [\![\varphi]\!]^{\mathcal{T}}_{\eta}\,([T_1, \ldots, T_k])$*

Proof. This theorem can be proven by induction on the structure of the formula φ. Consider the call $\mathtt{MC}_2(\varphi, [T_1, \ldots, T_k])_\eta$, where $[T_1, \ldots, T_k] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^k$. Then one of the following cases occurs:

Case ($\varphi = q$)*:* Comparing the defined semantics in Sec. 2.4 for a proposition and this case of $\mathtt{MC}_2$, the statement is true immediately.

[1] The semantic of HFL inherits invariance under β-reduction from the simply typed λ-calculus.

Case ($\varphi = X$)*:* It can be verified in Alg. 2 that $\texttt{MC}_2(X, [T_1, \ldots, T_k])_\eta$ is the same as $\texttt{env}(X)([T_1, \ldots, T_k])$. With the above assumption about the form of env before a call follows that $\texttt{env}(X)([T_1, \ldots, T_k]) = \eta(X)([T_1, \ldots, T_k])$. By definition this is $[\![X]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$.

Case ($\varphi = \neg\psi$)*:* Note that $\neg\psi$ and ψ are of type $\bullet$. It can be verified in Alg. 2 that $\texttt{MC}_2(\neg\psi, [\,])_\eta = \mathcal{S} \setminus \texttt{MC}_2(\psi, [\,])_\eta$. By induction hypothesis follows that $\texttt{MC}_2(\psi, [\,])_\eta = [\![\psi]\!]^{\mathcal{T}}_\eta$. Therefore $\texttt{MC}_2(\neg\psi, [\,])_\eta = \mathcal{S} \setminus [\![\psi]\!]^{\mathcal{T}}_\eta$, which by definition is the same as $[\![\neg\psi]\!]^{\mathcal{T}}_\eta$.

Case ($\varphi = \psi_1 \vee \psi_2$)*:* Note that $\psi_1 \vee \psi_2, \psi_1$ and ψ_2 are of type $\bullet$. It can be verified in Alg. 2 that $\texttt{MC}_2(\psi_1 \vee \psi_2, [\,])_\eta = \texttt{MC}_2(\psi_1, [\,])_\eta \cup \texttt{MC}_2(\psi_2, [\,])_\eta$. Using the induction hypothesis, it can be inferred that this is equal to $[\![\psi_1]\!]^{\mathcal{T}}_\eta \cup [\![\psi_2]\!]^{\mathcal{T}}_\eta$, which in turn is the same as $[\![\psi_1 \vee \psi_2]\!]^{\mathcal{T}}_\eta$.

Case ($\varphi = \psi_1\,\psi_2$)*:* By assumption φ_2 is of type $\tau_1 \to \ldots \to \tau_{k'} \to \bullet$. It can be verified in Alg. 2 that $\texttt{MC}_2(\varphi_1\,\varphi_2, [T_1, \ldots, T_k])_\eta = \texttt{MC}_2(\varphi_1, [f_{\psi_2}, T_1, \ldots, T_k])_\eta$ and that $f_{\psi_2}([T'_1, \ldots, T'_{k'}]) = \texttt{MC}_2(\varphi_2, [T'_1, \ldots, T'_{k'}])_\eta$ for all arguments $[T'_1, \ldots T'_{k'}] \in (\mathcal{T}[\![\tau^{\leq 1}]\!])^{k'}$. Using induction hypothesis, it follows that $f_{\psi_2} = [\![\psi_2]\!]^{\mathcal{T}}_\eta$ after termination of the for-all loop. This means $\texttt{MC}_2(\varphi_1, [f_{\psi_2}, T_1, \ldots, T_k])_\eta = \texttt{MC}_2(\varphi_1, [[\![\psi_2]\!]^{\mathcal{T}}_\eta, T_1, \ldots, T_k])_\eta$. Using the induction hypothesis again, it follows that this equals $[\![\psi_1]\!]^{\mathcal{T}}_\eta([[\![\psi_2]\!]^{\mathcal{T}}_\eta, T_1, \ldots, T_k])$, which by definition is the same as $[\![\psi_1\,\psi_2]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$.

Case ($\varphi = \lambda X.\psi$)*:* It can be verified in Alg. 2 that $\texttt{env}(X) = T_1$ after the call of $\texttt{MC}_2(\lambda X.\psi, [T_1, \ldots, T_k])_\eta$. This means that $\texttt{MC}_2(\lambda X.\psi, [T_1, \ldots, T_k])_\eta$ is the same as $\texttt{MC}_2(\psi, [T_2, \ldots, T_k])_{\eta\{X \to T_1\}}$. With use of the induction hypothesis, it can be inferred that $\texttt{MC}_2(\psi, [T_2, \ldots, T_k])_{\eta\{X \to T_1\}} = [\![\psi]\!]^{\mathcal{T}}_{\eta\{X \to T_1\}}([T_2, \ldots, T_k])$ and using β-reduction this is the same as $[\![\lambda X.\psi]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$.

Case ($\varphi = \langle a \rangle \psi$)*:* Note that $\langle a \rangle \psi$ and ψ are of type $\bullet$. It can be verified in Alg. 2 that $\texttt{MC}_2(\langle a \rangle \psi, [\,])_\eta = \left\{ s \in \mathcal{S} \mid \text{ex. } t \in \texttt{MC}_2(\psi, [])_\eta \text{ s.t. } s \xrightarrow{a} t \right\}$. Using the induction hypothesis, this is the same as $\left\{ s \in \mathcal{S} \mid \text{ex. } t \in [\![\psi]\!]^{\mathcal{T}}_\eta \text{ s.t. } s \xrightarrow{a} t \right\}$, which by definition is the same as $[\![\langle a \rangle \psi]\!]^{\mathcal{T}}_\eta$.

Case ($\mu X.\psi$)*:* To prove correctness of the μ case, the case $\texttt{MC}_2(\mu X.\psi, [T_1, \ldots, T_k])_\eta \subseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$ and the case $\texttt{MC}_2(\mu X.\psi, [T_1, \ldots, T_k])_\eta \supseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$ are proven separately.

Case ($\subseteq$ of μ case)*:* In order to show this case, the following statement is proven to be an invariant of the repeat-loop of the μ case in Alg. 2.

$$\text{f.a. } [T_1, \ldots, T_k] \in \mathit{dom}(\texttt{env}(X)) : \texttt{env}(X)([T_1, \ldots, T_k]) \subseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$$

The invariant holds before the loop, because $\mathit{dom}(\texttt{env}(X)) = \{[T_1, \ldots, T_k]\}$ and env maps this argument to $\emptyset$, which is a subset of $[\![\mu X.\psi]\!]^{\mathcal{T}}_\eta([T_1, \ldots, T_k])$. For the next step it must be shown, that if the invariant holds at the beginning of an iteration of the repeat-loop, it follows that it holds at its end. Let $\texttt{env}(X)$ be the global variable at the beginning of an arbitrary repeat-loop iteration and $[T'_1, \ldots, T'_k] \in \mathit{dom}(\texttt{env}(X))$ at this point. From the fact that this is a fixpoint formula follows that ψ is monotonic in X. Therefore, with the assumption that the invariant holds at the beginning of a

repeat-loop iteration and the fact that $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}$ is a fixpoint of ψ follows that:

$$[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to \mathtt{env}(X)\}}([T'_1,\ldots,T'_k]) \subseteq [\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to [\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}\}}([T'_1,\ldots,T'_k])$$

But $[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to \mathtt{env}(X)\}}([T'_1,\ldots,T'_k])$ is, using the induction hypothesis, equivalent to $\mathtt{MC}_2(\psi,[T'_1,\ldots,T'_k])_{\eta\{X\to \mathtt{env}(X)\}}$, which is the value of $\mathtt{env}(X)$ for $[T'_1,\ldots,T'_k]$ at the end of this repeat-loop iteration and by definition $[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to [\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}\}}([T'_1,\ldots,T'_k])$ is the same as $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}([T'_1,\ldots,T'_k])$. This shows that the invariant holds after this iteration. For finite transition systems the loop terminates eventually and since the domain of env grows or stays the same in each iteration it is valid that $[T_1,\ldots,T_k] \in dom(\mathtt{env}(X))$ at this point. This means the invariant is applicable to $[T_1,\ldots,T_k]$ after the repeat-loop. With the fact that $\mathtt{MC}_2(\mu X.\psi,[T_1,\ldots,T_k])_{\eta} = \mathtt{env}(X)\,([T_1,\ldots,T_k])$, where env is the global variable upon termination of the repeat-loop, this proves the $\subseteq$ direction of the μ case.

Case ($\supseteq$ of μ case)*:* In order to show this direction of the μ case, it must be shown that the following statement is true.

$$\mathtt{MC}_2(\varphi,[T_1,\ldots,T_k])_{\eta} \supseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}([T_1,\ldots,T_k]) \tag{4.1}$$

As it is described in Sec. 2.3, the fixpoint $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}$ is the same as some finite approximation F^n_{ψ}, which in turn is the same as $[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to F^{n-1}_{\psi}\}}$. It can be seen in Alg. 2 that $\mathtt{MC}_2(\mu X.\psi,[T_1,\ldots,T_k]) = \mathtt{env}(X)([T_1,\ldots,T_k])$, where $\mathtt{env}(X)$ denotes the value of the global variable for X upon termination of the repeat-loop. From the termination criterion of the loop follows that $\mathtt{env}(X)$ equals $\mathtt{env}(X)$ after the second to last iteration. This means that $\mathtt{env}(X)([T_1,\ldots,T_k]) = \mathtt{MC}_2(\psi,[T_1,\ldots,T_k])_{\eta\{X\to \mathtt{env}(X)\}}$ and by induction hypothesis it follows that this is the same as $[\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to \mathtt{env}(X)\}}([T_1,\ldots,T_k])$. To prove statement 4.1 it is assumed that the opposite is true and shown that this assumption leads to a contradiction. If the statement is wrong, then the following must be true:

$$\text{ex. } n\in\mathbb{N}\ \text{ex. } s\in\mathcal{S} : s\in F^n_{\psi}([T_1,\ldots,T_k]) \text{ and } s\notin \mathtt{env}(X)([T_1,\ldots,T_k]) \tag{4.2}$$

With the above argumentation, this means there is an $n\in\mathbb{N}$ such that there is an $s\in\mathcal{S}$ such that $s\in [\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to F^{n-1}_{\psi}\}}([T_1,\ldots,T_k]))$ and $s\notin ([\![\psi]\!]^{\mathcal{T}}_{\eta\{X\to \mathtt{env}(X)\}}([T_1,\ldots,T_k]))$. Let n be fixed as such an approximation. From the fact that X is a fixpoint variable follows that the variance of X in ψ is $+$. This means, that Lem. 1 can be applied with $\epsilon = \in$ and it follows that:

$$\text{ex. } s'\in S : s'\in (F^{n-1}_{\psi}([T'_1,\ldots,T'_{k'}])) \text{ and } s'\notin (\mathtt{env}(X)\,([T'_1,\ldots,T'_{k'}]))$$

If $n-1=0$ this would be a contradiction because F^0_{ψ} maps to $\emptyset$ for all arguments. If $n-1>0$ the argumentation starting from statement 4.2 can be repeated until this case occurs. Therefore, the assumption must be wrong and it must in fact be true that statement 4.1 holds, which proves the $\supseteq$ part of the μ case.

This means Th. 1 holds for all possible cases of φ, which finishes the proof and shows that the algorithm $\mathtt{MC}_2$ is correct. □

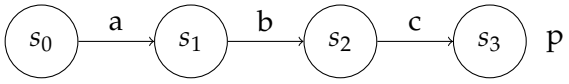

FIGURE 4.1: LTS with path of the form $\langle abc \rangle$.

4.4 Application: Existence of Paths Definable in HFL2

Consider the following formula.

$$(\mu F.\lambda G_1, G_2, G_3.\, ((G_1 \circ G_2 \circ G_3)\, p) \vee F\, (G_1 \circ \langle\langle a \rangle\rangle)\, (G_2 \circ \langle\langle b \rangle\rangle)\, (G_3 \circ \langle\langle c \rangle\rangle))\ \mathit{id}\ \mathit{id}\ \mathit{id}$$

where $\circ := \lambda F_1, F_2, X.\, F_1\, (F_2\, X)$, $f \circ g := \circ\, f\, g$, $\langle\langle x \rangle\rangle := \lambda X.\, \langle x \rangle X$ and $\mathit{id} := \lambda X.\, X$. Furthermore let $\langle\langle x^0 \rangle\rangle := \mathit{id}$ and $\langle\langle x^{n+1} \rangle\rangle := \langle x^n \rangle \circ \langle\langle x \rangle\rangle$. The formula beneath the μ operator is denoted by ψ. Informally, this formula describes the property "*There exists a word w of the not context-free language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ s.t. there is a state with proposition p that is reachable under w*". This is a not-regular property, which means it is not definable in the modal μ-calculus. Furthermore, it is believed that this property is not expressible in HFL1.

Consider the LTS in Fig. 4.1. In this LTS the property described above is true for the set $\{s_0, s_3\}$. In Tab. 4.1 the computation of $\mathtt{MC}_2(\mu F.\psi, [\mathit{id}, \mathit{id}, \mathit{id}])$ is given. Each column corresponds to an included argument and Row i denotes the form of $\mathtt{env}(X)$ after the i-th iteration of the repeat-loop. Empty cells are understood as still undefined. It can be seen in Row 0, that the algorithm starts with setting the value of the initial argument list $[\mathit{id}, \mathit{id}, \mathit{id}]$ to $\emptyset$. In Alg. 2 this is done before the repeat-loop of the μ case. After this the actual fixpoint computation starts: In each iteration of the repeat-loop the values for all arguments of $\mathit{dom}(\mathtt{env}(X))$ are updated until stability is reached. Row 1 shows that $\mathit{dom}(\mathtt{env}(X)) = \{[\mathit{id}, \mathit{id}, \mathit{id}], [\langle\langle a \rangle\rangle, \langle\langle b \rangle\rangle, \langle\langle c \rangle\rangle]\}$ after the first iteration of the repeat-loop. The argument list $[\langle\langle a \rangle\rangle, \langle\langle b \rangle\rangle, \langle\langle c \rangle\rangle]$ is added to the domain during the computation of $\mathtt{MC}_2(\psi, [\mathit{id}, \mathit{id}, \mathit{id}])$ in the corresponding variable case of the algorithm $\mathtt{MC}_2$. In Row 2 it can be seen, that the argument list $[\langle\langle a^2 \rangle\rangle, \langle\langle b^2 \rangle\rangle, \langle\langle c^2 \rangle\rangle]$ is added to $\mathit{dom}(\mathtt{env}(X))$ in the second iteration of the repeat-loop. This happens during the computation of $\mathtt{MC}_2(\psi, [\langle\langle a \rangle\rangle, \langle\langle b \rangle\rangle, \langle\langle c \rangle\rangle])$. After the second iteration all necessary arguments are included. The argument list $[\langle\langle a^3 \rangle\rangle, \langle\langle b^3 \rangle\rangle, \langle\langle c^3 \rangle\rangle]$ occurs during the computation of $\mathtt{MC}(\psi, [\langle\langle a^2 \rangle\rangle, \langle\langle b^2 \rangle\rangle, \langle\langle c^2 \rangle\rangle])$, but it can be verified that $[\![\langle\langle a^2 \rangle\rangle]\!]^{\mathcal{T}}_{\eta} = [\![\langle\langle a^3 \rangle\rangle]\!]^{\mathcal{T}}_{\eta}$, $[\![\langle\langle b^2 \rangle\rangle]\!]^{\mathcal{T}}_{\eta} = [\![\langle\langle b^3 \rangle\rangle]\!]^{\mathcal{T}}_{\eta}$ and $[\![\langle\langle c^2 \rangle\rangle]\!]^{\mathcal{T}}_{\eta} = [\![\langle\langle c^3 \rangle\rangle]\!]^{\mathcal{T}}_{\eta}$. This means that the argument list $[\langle\langle a^3 \rangle\rangle, \langle\langle b^3 \rangle\rangle, \langle\langle c^3 \rangle\rangle]$ is the same as $[\langle\langle a^2 \rangle\rangle, \langle\langle b^2 \rangle\rangle, \langle\langle c^2 \rangle\rangle]$ semantically and therefore not needed. In comparison of Row 3 and 4 it can be seen that no changes happened, which means that the termination criterion of the repeat loop is fulfilled and stability is reached. Therefore, the algorithm $\mathtt{MC}_2$ returns the value of $[\mathit{id}, \mathit{id}, \mathit{id}]$ of Row 4, which is $\{s_3, s_0\}$ at this point.

Tab. 4.1 shows that the algorithm needed 3 argument lists to compute the value of interest. For the LTS in Fig. 4.1 there exists a total of $(2^4)^{2^4}$ functions of the form $2^{\mathcal{S}} \to 2^{\mathcal{S}}$. This means that in fact there is a total of $((2^4)^{2^4})^3 \approx 6 \times 10^{57}$ possible argument lists, which underlines the efficiency gained by adopting neededness analysis for model checking HFL2.

Iter. \ Arg.	$[id, id, id]$	$[\langle\langle a \rangle\rangle, \langle\langle b \rangle\rangle, \langle\langle c \rangle\rangle]$	$[\langle\langle a^2 \rangle\rangle, \langle\langle b^2 \rangle\rangle, \langle\langle c^2 \rangle\rangle]$
0	$\emptyset$		
1	$\{s_3\}$	$\emptyset$	
2	$\{s_3\}$	$\{s_0\}$	$\emptyset$
3	$\{s_3, s_0\}$	$\{s_0\}$	$\emptyset$
4	$\{s_3, s_0\}$	$\{s_0\}$	$\emptyset$

TABLE 4.1: Fixpoint computation of example in Sec. 4.4.

4.5 Practical Optimizations of MC_2 for HFL2

First of all it can be noticed, that the operators $\wedge$, $[\,]$ and ν do not occur in $\texttt{MC}_2$. In the way that the grammar of HFL is defined in Sec. 2.3 this is not necessary, but would require all formulas that are applied to $\texttt{MC}_2$ to use the correspondences given in Sec. 2.3 instead of the above mentioned operators. To avoid this the algorithm $\texttt{MC}_2$ should be supplemented by the corresponding cases in an actual implementation.

It was already mentioned that the values of functional arguments may only be needed for certain arguments. However, the algorithm $\texttt{MC}_2$ computes these arguments completely, which means that it performs unnecessary computations. But there are some practical adjustments that can attenuate this efficiency gap:

One can distinguish between arguments of fixpoint formulas and λ formulas. In fact the full information about a functional argument is only needed if it is applied to a fixpoint variable or fixpoint formula to check for equality. Therefore a short case-distinction can lead to computational improvements, especially if there is a high number of different λ applications in a given formula. But this requires information about the structure of the formula in application cases.

Arguments for a fixpoint formula are typically applied again in later iterations of a fixpoint computation. In a practical context arguments, which are already computed, should be identified and reused instead of recomputed.

Example 3. Consider the following formula.

$$\varphi = \mu F.\, \lambda G_1, G_2.\, (G_1\, q) \vee (G_2\, p\, q)(X\, (G_2\, p)\, G_2)$$

An argument for G_2 has to be of type $\bullet \to \bullet \to \bullet$. Hence, in the first application it would be evaluated for all $[T_1, T_2] \in (2^{\mathcal{S}})^2$. But in the next iteration of the fixpoint it appears in $G_2\, p$ as a new argument. Here it would be redundant to recompute it for all possibilities of its second argument. It makes much more sense to use some kind of interface for the original G_2 argument, which simply fixes its first argument to q.

Another optimization is based on the underlying data-structure of $dom(\texttt{env}(X))$, where X is a fixpoint variable. The main idea of the localization of a fixpoint computation is to include only needed arguments, which are identified through their dependencies regarding the value of interest. This means that the computed value of one argument depends on the computed value of another argument. To illustrate this, consider the example from Sec. 3.3 again. In Tab. 4.2 the computation of Tab. 3.1 is given. Additionally, dependencies between the values are marked through arrows. Some of these arrows show that redundant computations happen

Iter. \ Arg.	$\{s_2\}$	$\{s_1\}$	$\{s_3\}$	$\{s_6\}$	$\emptyset$
0	$\emptyset$				
1	$\{s_2\}$	$\emptyset$			
2	$\{s_2\}$	$\{s_1\}$	$\emptyset$		
3	$\{s_0, s_2\}$	$\{s_1\}$	$\{s_3\}$	$\emptyset$	
4	$\{s_0, s_2\}$	$\{s_1, s_4\}$	$\{s_3\}$	$\{s_6\}$	$\emptyset$
5	$\{s_0, s_2, s_5\}$	$\{s_1, s_4\}$	$\{s_3, s_7\}$	$\{s_6\}$	$\emptyset$
6	$\{s_0, s_2, s_5\}$	$\{s_1, s_4, s_8\}$	$\{s_3, s_7\}$	$\{s_6\}$	$\emptyset$
7	$\{s_0, s_2, s_5\}$	$\{s_1, s_4, s_8\}$	$\{s_3, s_7\}$	$\{s_6\}$	$\emptyset$

TABLE 4.2: Fixpoint computation with dependencies between arguments.

in this computation. For instance, it can be seen that the value for $\{s_2\}$ in iteration 1 and 2 is computed with the same value of $\{s_1\}$. This is a result from the fact that in this example computation $dom(\texttt{env}(X))$ is considered as some sort of ordered structure e.g. a list, where new found arguments are appended at the end. The reason for this becomes clearer if these dependencies between values are reduced to dependencies between the corresponding arguments. This is given in form of a dependency graph for the included arguments in Fig. 4.2. This dependency graph shows that it would be more efficient to add new found arguments at the beginning of $dom(\texttt{env}(X))$, if it continues to be considered as a list. This is because it has a chain-like structure starting from the node $\emptyset$, which is dependent on itself, and ending at the node $\{s_2\}$, which is ultimately dependent on the values for s_1, s_3, s_6 and $\emptyset$. If this approach is adopted, the same computation can be performed with fewer computations. This can be seen in the computation given in Fig. 4.3. There, a row $i \,||\quad \ldots \quad s \to s'; \quad \ldots \quad$, $i \in \{0, \ldots, 5\}$, $s, s' \in dom(\texttt{env}(X))$ denotes that the argument s is evaluated to s' in the i-th iteration. Obviously, this computations needs two iterations less than the original one. This shows that an ordering of included arguments based on their dependency graph promises computational advantages against a naive succession.

Another optimization related to dependencies between included arguments is updating only the values of those arguments where the dependencies were updated. In fact, this is part of the traditional neededness analysis approach, where in advanced iterations of a fixpoint computation only those values are updated for which the values they depend on change [Jør94]. In Alg. 2 in the μ case it can be seen that in each iteration of the repeat-loop all arguments of the domain of the global variable `env` are updated. But considering Tab. 4.2 and Fig. 4.2 the value of the argument $\emptyset$ is dependent on itself, which means that it is sufficient to compute it once, because the value will not change in further iterations. And this would mean that the argument $\{s_6\}$ has to be computed just once, too.

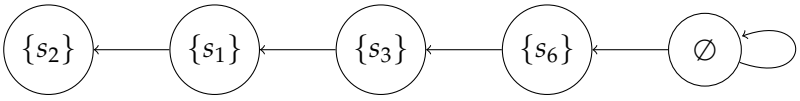

FIGURE 4.2: Dependency graph of arguments.

0	$\{s_2\} \to \emptyset$				
1	$\{s_2\} \to \emptyset$;	$\{s_1\} \to \emptyset$			
2	$\{s_1\} \to \{s_1\}$;	$\{s_3\} \to \emptyset$;	$\{s_2\} \to \{s_0, s_2\}$		
3	$\{s_3\} \to \{s_3\}$;	$\{s_6\} \to \emptyset$;	$\{s_1\} \to \{s_1, s_4\}$;	$\{s_2\} \to \{s_0, s_2, s_5\}$	
4	$\{s_6\} \to \{s_6\}$;	$\emptyset \to \emptyset$;	$\{s_3\} \to \{s_3, s_7\}$;	$\{s_1\} \to \{s_1, s_4\}$;	$\{s_2\} \to \{s_0, s_2, s_5\}$
5	$\emptyset \to \emptyset$;	$\{s_6\} \to \{s_6\}$;	$\{s_3\} \to \{s_3, s_7\}$;	$\{s_1\} \to \{s_1, s_4\}$;	$\{s_2\} \to \{s_0, s_2, s_5\}$

FIGURE 4.3: Optimized fixpoint computation.

In general such a dependency graph would be more complex than in the given example and will not include binary dependencies only. In this case a broader approach could be helpful, where the computation of a fixpoint starts with collecting all needed arguments first. With this an optimal ordering of the arguments could be found, which allows avoiding redundancies in the following computation of the fixpoint approximation.

It should be taken into considerations that the efficiency gain of most of the optimizations discussed depends heavily on the actual applied formula. This means, it should be decided situationally if and which adjustments of $\texttt{MC}_2$ promise relevant computational advantages.

Chapter 5

Efficient Model Checking in Higher-Order Fragments of HFL

5.1 Model Checking HFL with Neededness Analysis

As mentioned before the approach of the algorithms MC and $\mathtt{MC}_2$ for avoiding unnecessary computations in a fixpoint computation is an adoption of the neededness analysis approach of traditional data-flow analysis. The goal here is to only include values that appear non-trivial in the computation of a fixpoint of interest [Jør94]. In the context of a fixpoint computation, this means including needed arguments only.

Alg. 3 is an informal description of the functionality of the algorithms MC and $\mathtt{MC}_2$. The input for this algorithm is a fixpoint formula $\mu X.\psi$, an initial argument list $\overline{T}_0$, an LTS $\mathcal{T}$ over which the formula is evaluated and the corresponding environment η. The output is the value of interest $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}(\overline{T}_0)$. It can be seen that such an algorithm needs to initialize some sort of map M_X first, which holds the approximation of $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}$ created in the following. After this the algorithm starts with computing this approximation, beginning with the computation of the value of $[\![\psi]\!]^{\mathcal{T}}_{\eta}(\overline{T}_0)$. If a new argument occurs during this it is added to M_X and included in following iterations. The algorithm terminates and returns the value of interest if nothing changes anymore. This is checked by comparing the map M_X with its snapshot S_X, which is created before the last iteration.

5.2 Efficient Model Checking HFL2 with Neededness Analysis

In the case of model checking the first-order fragment HFL1, the avoidance of unnecessary computations, in other words, neededness analysis, is fully met by the functionality of MC. In the case of HFL2, the algorithm $\mathtt{MC}_2$ only partially meets this requirement. Indeed, unnecessary arguments are omitted, but necessary ones are computed completely. Considering Alg. 3 this is needed in Line 7 to check if a new argument for the fixpoint variable is already defined in the corresponding map and therefore already included in the computation. However, the values of functional arguments are only required for certain arguments.

Example 4. Consider following formula:

$$(\mu F.\lambda G_1, G_2.\,((G_1 \circ G_2)\,p) \vee F\,(G_1 \circ \langle\langle a \rangle\rangle)\,(G_2 \circ \langle\langle b \rangle\rangle))\;\; id\;\; id$$

Input: A fixpoint formula $\mu X.\psi$, a list of arguments $\overline{T}_0$, a LTS $\mathcal{T}$ and an environment η.
Output: The value of $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}(\overline{T}_0)$.

initialize a map M_X and set $M_X(\overline{T}_0)$ to $\emptyset$;
repeat
  save snapshot S_X of M_X;
  forall $\overline{T}$ *defined in* M_X **do**
    calculate value $[\![\psi]\!]^{\mathcal{T}}_{\eta}([\overline{T}])$ with M_X replacing X **begin**
      ...;
      if *undefined* $M_X(\overline{T}')$ *occurs* **then** set $M_X(\overline{T}')$ to $\emptyset$;
      ...;
    end
    update $M_X(\overline{T})$ with calculated value;
  end
until M_X *equals* S_X;
return $M_X(\overline{T}_0)$

Algorithm 3: Fixpoint computation in HFL with neededness analysis.

where $\circ := \lambda F_1, F_2, X.\, F_1\,(F_2\,X)$, $f \circ g := \circ\, f\, g$, $\langle\langle x \rangle\rangle := \lambda X.\, \langle x \rangle X$ and $id := \lambda X.\, X$. Furthermore, let $\langle\langle x^0 \rangle\rangle := id$ and $\langle\langle x^{n+1} \rangle\rangle := \langle\langle x^n \rangle\rangle \circ \langle\langle x \rangle\rangle$ and let ψ denote the formula beneath the μ operator. Informally, this formula describes the property "*There exists a path of the form $a^n b^n$ for some $n \in \mathbb{N}$ to a state where p is true.*". It can be seen that in a fixpoint computation following the approach of Alg. 3 new arguments occur according to the pattern $[id, id], [\langle\langle a \rangle\rangle, \langle\langle b \rangle\rangle], [\langle\langle a^2 \rangle\rangle, \langle\langle b^2 \rangle\rangle], \ldots$ and so on. Nevertheless, regardless of the arguments currently applied to ψ, the value of G_2 is only needed for p and the value of G_1 is only needed for the value of $G_2\, p$. Hence, a full computation of the arguments for G_1 and G_2 is not necessary.

As seen in Ex. 4, to achieve a complete neededness analysis in HFL2 a weaker concept of equality is needed, which avoids unnecessary computations of functional arguments but still ensures correctness of a fixpoint computation.

Let $\approx_\psi$ denote a relation between two possible arguments in a computation of $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}$ with $\mu X.\psi \in$ HFL2 for some argument $\overline{T}_0 \in HFL1^k$. If $\overline{T} \approx_\psi \overline{T}'$ holds for some $\overline{T}, \overline{T}' \in \text{HFL1}^k$, then $\overline{T}'$ can be omitted from the computation if $\overline{T}$ is included.

A first hypothesis for the definition of the relation $\approx_\psi$ is comparing two arguments based on their actual occurrences within the fixpoint formula only. This would avoid computing values of functional arguments for unnecessary arguments.

Hypothesis 1. Assuming that $\overline{T}, \overline{T}' \in \text{HFL1}^k$ with $\overline{T} = [\alpha_1, \ldots, \alpha_k]$ and $\overline{T}' = [\alpha'_1, \ldots, \alpha'_k]$ are possible arguments in a computation of the value of $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}$ for some initial argument list. Then $\overline{T} \approx_\psi \overline{T}'$ holds if and only if for all α_i and α'_i holds that their occurring applications in ψ are equal.

Practically, this can be checked by applying $\overline{T}$ and $\overline{T}'$ to ψ, saving the occurring applications as well as their values for each single argument in $\overline{T}$ and $\overline{T}'$ and comparing them afterward.

Example 5. Consider the formula

$$(\mu F.\lambda G_1, G_2.\,((G_1 \circ G_2)\,p) \vee F\,(G_1 \circ \langle\langle a \rangle\rangle)\,(G_2 \circ \langle\langle b \rangle\rangle))\;\; id\;\; id$$

again and the LTS in Fig. 5.1. In Tab. 5.1 the approach of Hyp. 1 is performed for this example. Each row represents one of the variables G_1, G_2. Column 1 represents the initial argument list respectively arguments and the small tables in the cells represent their applications. For example, the initial argument corresponding to G_2 is id, whose value is needed for $\{s_2\}$ only. Considering the formula this is correct, because the value of the current argument for G_2 is only needed for p, which in this LTS evaluates to $\{s_2\}$. The arguments of Column 2 occur as arguments for the variable F if $[id, id]$ is applied to ψ. The arguments of Column 3 occur if the arguments of Column 2 are applied to ψ and so on. In fact, this is the scheme described in Alg. 3. It can be verified that the applications listed in Column 4 and 5 are identical. Following Hyp. 1 it is true that:

$$[\langle\langle a^3 \rangle\rangle, \langle\langle b^3 \rangle\rangle] \approx_\psi [\langle\langle a^4 \rangle\rangle, \langle\langle b^4 \rangle\rangle]$$

which means that the argument list $[\langle\langle a^4 \rangle\rangle, \langle\langle b^4 \rangle\rangle]$ does not need to be included in the computation if $[\langle\langle a^3 \rangle\rangle, \langle\langle b^3 \rangle\rangle]$ is already included. Taking a look at Fig. 5.1 this is correct: The longest path of the form $\langle a^n b^n \rangle$ is for $n = 2$, hence, all checks for the existence of paths $\langle a^{n'} b^{n'} \rangle$ with $n' > 2$ will return false, i.e., the empty set. This means that including $[\langle\langle a^3 \rangle\rangle, \langle\langle b^3 \rangle\rangle]$ is necessary, but all argument lists up from $[\langle\langle a^4 \rangle\rangle, \langle\langle b^4 \rangle\rangle]$ are redundant because they evaluate to the same values as $[\langle\langle a^3 \rangle\rangle, \langle\langle b^3 \rangle\rangle]$.

However, for a different example it turns out that Hyp. 1 is wrong.

Example 6. Consider the following formula:

$$(\mu F.\lambda G_1, G_2.\,(G_1\,p) \vee F\,G_2\,(G_1 \circ \langle\langle b \rangle\rangle))\,\langle\langle a \rangle\rangle\,\langle\langle a \rangle\rangle$$

Informally, this formula denotes the property "*There is a path of the form* $\langle ab^n \rangle$ *for some* $n \in \mathbb{N}$ *that leads to a state where* p *is true.*" Due to the way the arguments are applied to the variable F every possible path is checked twice. This is best understood if the formula is unfolded to an infinite disjunction:

$$\langle\langle a \rangle\rangle\,p \vee \langle\langle a \rangle\rangle\,p \vee \langle\langle a \rangle\rangle\langle\langle b \rangle\rangle\,p \vee \langle\langle a \rangle\rangle\langle\langle b \rangle\rangle\,p \vee \langle\langle a \rangle\rangle\langle\langle b^2 \rangle\rangle\,p \vee \langle\langle a \rangle\rangle\langle\langle b^2 \rangle\rangle\,p \vee \ldots$$

This leads to the computation in Tab. 5.2 for the LTS in Fig. 5.1. It can be verified that for this LTS the property is true for the set $\{s_0, s_4\}$. But the applications in Column 1 and Column 2 are equal, which means that an algorithm adopting Hyp. 1 would not include the argument list of Column 2 in the computation. But then, according to the scheme in Alg. 3, Column 3, 4 and 5 would not occur, which means that state s_4 is not considered. This would be incorrect and therefore the hypothesis must be wrong.

The failure of Hyp. 1 shows that equality in the occurring applications of different arguments is an insufficient relation. Alg. 3, Ex. 4 and Ex. 6 show that new occurring arguments are in fact modifications, combinations or permutations of the initially applied arguments. Therefore, a next idea is comparing two argument lists $\overline{T}$ and $\overline{T}'$ based on the occurring applications of the initial arguments if $\overline{T}$ and $\overline{T}'$ are applied to the fixpoint formula.

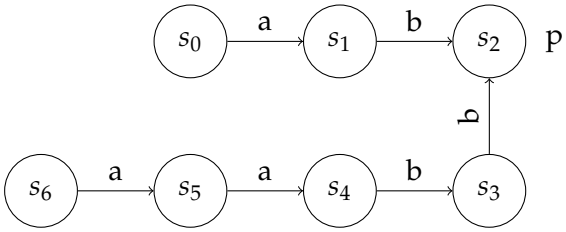

FIGURE 5.1: LTS with paths of the form $\langle a^n b^n \rangle$.

Hypothesis 2. Assuming that $\overline{T}, \overline{T}' \in \text{HFL1}^k$ are possible argument lists in the computation of the value of $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\eta}$ for some argument list $\overline{T}_0 = [\alpha_1, \ldots, \alpha_k]$. Then $\overline{T} \approx_{\psi} \overline{T}'$ holds if and only if for all α_i holds that the occurring applications of α_i in the application of $\overline{T}$ to ψ are the same as the occurring applications of α_i in the application of $\overline{T}'$ to ψ.

Practically, the approach of Hyp. 2 is checked in almost the same way as for Hyp. 1: Applying $\overline{T}$ and $\overline{T}'$ to ψ, saving the occurring applications of the initial arguments and comparing them afterward.

Example 7. Consider the formula

$$(\mu F.\lambda G_1, G_2.\, (G_1\, p) \vee F\, G_2\, (G_1 \circ \langle\langle b \rangle\rangle))\, \langle\langle a \rangle\rangle\, \langle\langle a \rangle\rangle$$

and the LTS in Fig. 5.1 again. Let ψ denote the formula beneath the μ operator. In Tab. 5.3 the approach of Hyp. 2 is performed for this example. Each column represents a possible argument list and each of the two rows stands for one of the initial arguments. $\langle\langle a \rangle\rangle_1$ denotes the initial argument $\langle\langle a \rangle\rangle$ for the variable G_1 and $\langle\langle a \rangle\rangle_2$ the initial argument $\langle\langle a \rangle\rangle$ for the variable G_2. This index is also used to denote where which initial argument occurs in the argument list currently considered e.g. $\langle\langle ab^2 \rangle\rangle_1$ includes $\langle\langle a \rangle\rangle_1$. The small table in a cell shows the applications of the initial argument of this row, if the argument-list of this column is applied to the fixpoint formula. It can be seen that with the column corresponding to $[\langle\langle ab^3 \rangle\rangle_1, \langle\langle ab^3 \rangle\rangle_2]$ the applications of $\langle\langle a \rangle\rangle_1$ and $\langle\langle a \rangle\rangle_2$ start to repeat. Therefore, by Hyp. 2 it follows that

$$[\langle\langle ab^3 \rangle\rangle_1, \langle\langle ab^3 \rangle\rangle_2] \approx_{\psi} [\langle\langle ab^4 \rangle\rangle_1, \langle\langle ab^4 \rangle\rangle_2] \text{ and } [\langle\langle ab^3 \rangle\rangle_2, \langle\langle ab^4 \rangle\rangle_1] \approx_{\psi} [\langle\langle ab^4 \rangle\rangle_2, \langle\langle ab^5 \rangle\rangle_1]$$

Considering the given LTS this is correct, because there exists no path $\langle b^n \rangle$ with $n > 2$, which means there exists no path $\langle ab^n \rangle$ with $n > 2$. Therefore, all arguments of G_1 checking for such a path to $\{s_2\}$ must evaluate to the empty set and the argument lists $[\langle\langle ab^4 \rangle\rangle_1, \langle\langle ab^4 \rangle\rangle_2]$ as well as $[\langle\langle ab^4 \rangle\rangle_2, \langle\langle ab^5 \rangle\rangle_1]$ can be omitted in the computation.

However, for a similar example it turns out that the approach of this hypothesis is insufficient, too.

Example 8. Consider the following formula:

$$(\mu F.\lambda G_1, G_2.\, (G_1\, p) \vee F\, G_2\, (\langle\langle b \rangle\rangle \circ G_1))\, \langle\langle a \rangle\rangle\, \langle\langle a \rangle\rangle$$

and the LTS in Fig. 5.2. Let ψ denote the formula beneath the μ operator. In comparison to the formula of Ex. 6 and Ex. 7 the only change in the formula above is that the order of the composition of $\langle\langle b \rangle\rangle$ and G_1 changed. Hence, this formula describes

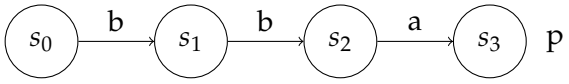

FIGURE 5.2: LTS with path of the form $\langle b^2 a \rangle$.

the property "*There is a path of the form* $\langle b^n a \rangle$ *for some* $n \in \mathbb{N}$ *that leads to a state where* p *is true.*" Tab. 5.4 shows the occurring applications of the initial arguments for the different possible argument lists. Following the approach of Hyp. 2, it can be seen that:

$$[\langle\langle a \rangle\rangle_1, \langle\langle a \rangle\rangle_2] \approx_\psi [\langle\langle ba \rangle\rangle_1, \langle\langle ba \rangle\rangle_2] \text{ and } [\langle\langle a \rangle\rangle_2, \langle\langle ba \rangle\rangle_1] \approx_\psi [\langle\langle ba \rangle\rangle_2, \langle\langle b^2 a \rangle\rangle_1]$$

This means an algorithm adopting the approach of this hypothesis omits the argument lists $[\langle\langle ba \rangle\rangle_1, \langle\langle ba \rangle\rangle_2]$ and $[\langle\langle ba \rangle\rangle_2, \langle\langle b^2 a \rangle\rangle_1]$. But then the path $\langle b^2 a \rangle$ is not considered, which in fact is incorrect for the given LTS. Therefore, Hyp. 2 must be wrong.

In the failure of Hyp. 1 and Hyp. 2 it can be seen that partial computation is most likely not sufficient for the definition of $\approx_\psi$. Different variations of these two hypothesis can be considered, but it should be taken into account that Hyp. 2 already uses the atomic components of possible arguments in a fixpoint computation in form of the initial arguments. This makes it unlikely that other approaches of this form, which necessarily consider combinations or variations of the initial arguments, promise success.

	$[id, id]$	$[\langle\langle a\rangle\rangle, \langle\langle b\rangle\rangle]$	$[\langle\langle a^2\rangle\rangle, \langle\langle b^2\rangle\rangle]$	$[\langle\langle a^3\rangle\rangle, \langle\langle b^3\rangle\rangle]$	$[\langle\langle a^4\rangle\rangle, \langle\langle b^4\rangle\rangle]$
G_1	$\frac{\{s_2\}}{\{s_2\}}$	$\frac{\{s_1, s_3\}}{\{s_0\}}$	$\frac{\{s_4\}}{\{s_6\}}$	$\frac{\emptyset}{\emptyset}$	$\frac{\emptyset}{\emptyset}$
G_2	$\frac{\{s_2\}}{\{s_2\}}$	$\frac{\{s_2\}}{\{s_1, s_3\}}$	$\frac{\{s_2\}}{\{s_4\}}$	$\frac{\{s_2\}}{\emptyset}$	$\frac{\{s_2\}}{\emptyset}$

TABLE 5.1: Table corresponding to Example 5.

	$[\langle\langle a\rangle\rangle, \langle\langle a\rangle\rangle]$	$[\langle\langle a\rangle\rangle, \langle\langle ab\rangle\rangle]$	$[\langle\langle ab\rangle\rangle, \langle\langle ab\rangle\rangle]$	$[\langle\langle ab\rangle\rangle, \langle\langle ab^2\rangle\rangle]$	$[\langle\langle ab^2\rangle\rangle, \langle\langle ab^2\rangle\rangle]$
G_1	$\frac{\{s_2\}}{\emptyset}$	$\frac{\{s_2\}}{\emptyset}$	$\frac{\{s_2\}}{\{s_0\}}$	$\frac{\{s_2\}}{\{s_0\}}$	$\frac{\{s_4\}}{\{s_5\}}$
G_2					

TABLE 5.2: Table corresponding to Example 6.

	$[\langle\langle a\rangle\rangle_1, \langle\langle a\rangle\rangle_2]$	$[\langle\langle a\rangle\rangle_2, \langle\langle ab\rangle\rangle_1]$	$[\langle\langle ab\rangle\rangle_1, \langle\langle ab\rangle\rangle_2]$	$[\langle\langle ab\rangle\rangle_2, \langle\langle ab^2\rangle\rangle_1]$	$[\langle\langle ab^2\rangle\rangle_1, \langle\langle ab^2\rangle\rangle_2]$
$\langle a\rangle_1$	$\frac{\{s_2\}}{\emptyset}$		$\frac{\{s_1, s_3\}}{\{s_0\}}$		$\frac{\{s_4\}}{\{s_5\}}$
$\langle a\rangle_2$		$\frac{\{s_2\}}{\emptyset}$		$\frac{\{s_1, s_3\}}{\{s_0\}}$	
	$[\langle\langle ab^2\rangle\rangle_2, \langle\langle ab^3\rangle\rangle_1]$	$[\langle\langle ab^3\rangle\rangle_1, \langle\langle ab^3\rangle\rangle_2]$	$[\langle\langle ab^3\rangle\rangle_2, \langle\langle ab^4\rangle\rangle_1]$	$[\langle\langle ab^4\rangle\rangle_1, \langle\langle ab^4\rangle\rangle_2]$	$[\langle\langle ab^4\rangle\rangle_2, \langle\langle ab^5\rangle\rangle_1]$
$\langle a\rangle_1$		$\frac{\emptyset}{\emptyset}$		$\frac{\emptyset}{\emptyset}$	
$\langle a\rangle_2$	$\frac{\{s_4\}}{\{s_5\}}$		$\frac{\emptyset}{\emptyset}$		$\frac{\emptyset}{\emptyset}$

TABLE 5.3: Table corresponding to Example 7.

	$[\langle\langle a\rangle\rangle_1, \langle\langle a\rangle\rangle_2]$	$[\langle\langle a\rangle\rangle_2, \langle\langle ba\rangle\rangle_1]$	$[\langle\langle ba\rangle\rangle_1, \langle\langle ba\rangle\rangle_2]$	$[\langle\langle ba\rangle\rangle_2, \langle\langle b^2a\rangle\rangle_1]$	$[\langle\langle b^2a\rangle\rangle_1, \langle\langle b^2a\rangle\rangle_2]$
$\langle a\rangle_1$	$\frac{\{s_3\}}{\{s_2\}}$		$\frac{\{s_3\}}{\{s_2\}}$		$\frac{\{s_3\}}{\{s_2\}}$
$\langle a\rangle_2$		$\frac{\{s_3\}}{\{s_2\}}$		$\frac{\{s_3\}}{\{s_2\}}$	

TABLE 5.4: Table corresponding to Example 8.

Chapter 6

Conclusion

6.1 Results

This thesis presented a symbolic model checker $\texttt{MC}_2$ for HFL2 using neededness analysis. In the example computation of Sec. 4.4 it was shown that even for a simple LTS and formula the adoption of neededness analysis for model checking HFL2 promises major computational benefits compared to a naive approach. Furthermore, it can be seen that there is not much difference between neededness analysis for model checking HFL1 and HFL2, which suggests that an adaptation for higher order fragments or complete HFL can be achieved in a similar way.

An actual implementation of the algorithm $\texttt{MC}_2$ given in Alg. 2 can achieve further computational advantages. The chosen data-structure for the map `env` and its internal sorting as well as informations about dependencies between included arguments can reduce the number of necessary iterations of a fixpoint computation further. Additionally, the neededness analysis idea can be extended such that not all included arguments in a fixpoint computation are updated in each iteration, but only those, whose dependencies were updated before.

Considering efficiency, this thesis showed that the algorithm $\texttt{MC}_2$ has limitations. Functional arguments need to be computed completely, even if their value is only needed for certain arguments in the formula of interest. This is necessary to identify which arguments need to be included and which do not need to be included in a fixpoint computation. It was shown that it is most likely not sufficient to compute corresponding arguments partially and compare them solely based on computed values. Moreover, a more complex approach seems to be needed to further improve the efficiency of a model checker for HFL2 with neededness analysis.

6.2 Outlook

Currently, there are three possible continuations of this work, which have already been mentioned implicitly. Firstly, an actual implementation of the algorithm $\texttt{MC}_2$ is a topic of interest. This would not only give an efficient model checker for HFL2 at hand, but opens a way for more profound studies of its computational benefits and drawbacks due to the unsolved problem regarding efficiency. Furthermore, the practical optimizations discussed in Sec. 4.5 could be verified and extended.

Secondly, this neededness analysis approach is most likely applicable on even higher order fragments. As it turns out, the structure of $\texttt{MC}_2$ is quite similar to the structure

of the algorithm MC by Axelsson and Lange. This is not surprising, since the algorithm $\mathtt{MC}_2$ can be seen as an advancement of MC, however, it shows that the adopted idea of neededness analysis works in more or less the same way for model checking HFL2 as for model checking HFL1. This implies that the approach can be used for model checking HFL with neededness analysis and that the corresponding algorithm will not differ that much from $\mathtt{MC}_2$ respectively MC. A next step is investigating this notion further.

Lastly, the problem of necessary complete computation of applied arguments unsolved in this thesis should be addressed further. If it is considered to lift this approach to higher order fragments than HFL2, the drawbacks become bigger. This results from the fact that in HFL2 arguments are at most functions between state sets, but for instance in HFL3 they can be functions between these functions, which are more expensive to compute. Therefore, an efficient evaluation of arguments becomes more relevant with the increasing order of fragments. In this context, the insufficient approaches shown in this thesis are to be seen as a foundation for more profound research of this problem.

Appendix A

Fundamentals

In this appendix basic knowledge and short introductions for necessary fundamentals are presented.

A.1 The modal μ-calculus

The *modal μ-calculus* is an extension of the propositional modal logic and todays commonly used definition was given by D. Kozen in [Koz83], and a sufficient introduction can be found in [BS01].

First we fix a set of atomic propositions $\mathcal{P}$ and a set of transitions $\{\xrightarrow{a}, \xrightarrow{b}, \ldots\}$ with transition names from $\mathcal{A}$. Additionally, we need a set $\mathcal{V} = \{X, Y, ...\}$ of *variables*. The syntax of MMC is then defined as:

$$\varphi ::= p \mid X \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mu X.\varphi$$

where $p \in \mathcal{P}$, $a \in \mathcal{A}$ and $X \in \mathcal{V}$. μ denotes the *least fixpoint operator* and $\langle\rangle$ is called *diamond operator*. In addition there are some standard abbreviations. Let φ and ψ be formulas of MMC, then the following are formulas of MMC as well:

$$\begin{aligned} \varphi \wedge \psi &:= \neg(\neg\phi \vee \neg\psi) \\ [a]\varphi &:= \neg\langle a \rangle \neg\varphi \\ \nu X.\phi &:= \neg\mu X.\neg\varphi[\neg X / X] \end{aligned}$$

where $[\neg X / X]$ means substituting every free occurrence of X with $\neg X$, $[\,]$ is the *box operator* and ν denotes the *greatest fixpoint operator*.

The expressiveness of MMC and its added value compared to propositional modal logic becomes clear, taking a look at its actual semantics. To describe these we need an *environment* η, which is a partial function $\mathcal{V} \to 2^{\mathcal{S}}$ mapping each used variable to a subset of the state set $\mathcal{S}$. With such the semantics are described as follows:

$$\begin{aligned} [\![p]\!]^{\mathcal{T}}_{\eta} &:= \{s \in \mathcal{S} \mid p \in \mathcal{L}(s)\} & [\![\neg\varphi]\!]^{\mathcal{T}}_{\eta} &:= S \setminus [\![\varphi]\!]^{\mathcal{T}}_{\eta} \\ [\![\varphi \vee \psi]\!]^{\mathcal{T}}_{\eta} &:= [\![\varphi]\!]^{\mathcal{T}}_{\eta} \cup [\![\psi]\!]^{\mathcal{T}}_{\eta} & [\![\langle a \rangle \varphi]\!]^{\mathcal{T}}_{\eta} &:= \{s \in S \mid \exists t \in [\![\varphi]\!]^{\mathcal{T}}_{\eta} : s \xrightarrow{a} t\} \\ [\![X]\!]^{\mathcal{T}}_{\eta} &:= \eta(X) & [\![\mu X.\varphi]\!]^{\mathcal{T}}_{\eta} &:= \bigcap\{T \subseteq S \mid [\![\varphi]\!]^{\mathcal{T}}_{\eta[T/X]} \subseteq T\} \end{aligned}$$

At this point it can be seen that the defining feature of MMC is the μ operator. This operator provides semantics of *recursive* expressions, which allows a neat way of

expressing usual temporal logic operators [BS01].

Example 9. Given the following MMC formula φ:

$$\varphi = \mu X.p \vee \langle a \rangle X \tag{A.1}$$

with $p \in \mathcal{P}$ and $a \in A$ the name of the transition denoted by $\xrightarrow{a}$. φ then expresses the property *"There is an a-path where p holds"*. This can best be seen by unfolding the fixpoint formula to $p \vee \langle a \rangle p \vee \langle a \rangle \langle a \rangle p \vee \ldots$, showing that each length of a possible a-path is checked.

A.2 Simply typed λ-calculus

The *simply typed λ-calculus* is a typed form of the classical lambda calculus, which is a model for computable functions. The prefix "simply" denotes that it has one type-constructor only. It was first described by A. Church in [Chu40] and a detailed description can be found in [Bar93].

Terms of the classical (untyped) λ-calculus are described by the grammar

$$f ::= x \mid \lambda x.f \mid f\, f$$

where x is a *term-variable*. $\lambda x.f$ and $f\, f$ are the two basic operations of the lambda calculus: abstraction and application.

An *abstraction* is an expression of the form $\lambda x.f$, where f is some kind of statement possibly dependent on x. Intuitively this term represents the mapping $x \to f(x)$.

The *application* is an expression of the form $f_1\, f_2$ which simply means applying f_1 considered as some algorithm or function on the input f_2. Application is left associative, means $f_1\, f_2\, f_3 \ldots f_n$ is to be read as $(\ldots (f_1\, f_2)\, f_3) \ldots f_n)$. If $f_1 = f_2$, means they denote the same kind of object, $f_1\, f_2$ would still be valid, which is a result of the untyped lambda calculus being type-free.

Both operations in combination can intuitively be understood as $(\lambda x.f(x))\, c = f(c)$ which formally means

$$(\lambda x.f[x])c = f[c/x]$$

thus the substitution of x in the expression f with c. This equation is called *β-conversion* and is the only essential axiom of the lambda calculus.

As mentioned above the simply typed λ-calculus adds types with just one kind of type-constructor $\to$ to the untyped lambda calculus, hence we can skip the discussion of the general typed lambda calculus here and start with this simplified type system. To denote different types we range over $\sigma, \tau, \ldots$. Let $\mathcal{A}$ be a set of base-types. Then the set of STLC-types is inductively defined as

$$\tau ::= \alpha \mid \tau \to \tau \qquad \alpha \in \mathcal{A}$$

Furthermore $\to$ is defined to be right associative, which means that $\sigma_1 \to \sigma_2 \to \ldots \to \sigma_n$ is to be read as $(\sigma_1 \to (\sigma_2 \to \ldots \to (\sigma_{n-1} \to \sigma_n))$.

To determine if a term is well-formed, there exist three typing rules

$$(1)\ \frac{(x:\sigma)\in\Gamma}{\Gamma\vdash x:\sigma} \qquad (2)\ \frac{\Gamma\vdash f_1:(\sigma\to\tau)\quad \Gamma\vdash f_2:\sigma}{\Gamma\vdash (f_1\,f_2):\tau} \qquad (3)\ \frac{\Gamma,x:\sigma\vdash f:\tau}{\Gamma\vdash(\lambda x.f):(\sigma\to\tau)}$$

the *axiom* (1), *elimination* (2) and the *introduction* (3) rule. A term of the form $f:\sigma$ is called *statement* and means that f has type σ. For a variable x the term $x:\sigma$ is called *declaration*. Γ is called *context* and represents a set of such declarations and $\Gamma\vdash f:\sigma$ denotes that $f:\sigma$ is derivable from Γ.

A term is called *well-typed* if it is derivable from a context Γ with the given typing-rules.

Bibliography

[AL07] Roland Axelsson and Martin Lange. "Model Checking the First-Order Fragment of Higher-Order Fixpoint Logic". In: *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*. 2007, pp. 62–76.

[ALS07] Roland Axelsson, Martin Lange, and Rafal Somla. "The Complexity of Model Checking Higher-Order Fixpoint Logic". In: *Logical Methods in Computer Science* 3.2 (2007).

[Bar93] H. P. Barendregt. "Lambda Calculi with Types". In: *Handbook of Logic in Computer Science (Vol. 2): Background: Computational Structures*. New York, USA: Oxford University Press, Inc., 1993.

[BLL17] Florian Bruse, Martin Lange, and Étienne Lozes. "Space-Efficient Fragments of Higher-Order Fixpoint Logic". In: *Reachability Problems - 11th International Workshop, RP 2017, London, UK, September 7-9, 2017, Proceedings*. 2017, pp. 26–41.

[BS01] J. Bradfield and C. Stirling. "Modal logics and mu-calculi: an introduction". In: *Handbook of process algebra*. Elsevier, 2001, pp. 293–330.

[CGP99] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[Chu40] Alonzo Church. "A Formulation of the Simple Theory of Types". In: *The Journal of Symbolic Logic* 5.2 (1940), pp. 56–68.

[DGL16] Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal logics in computer science : Finite-state systems*. Cambridge, 2016.

[Eme87] E.Allen Emerson. "Uniform inevitability is tree automation ineffable". In: *Information Processing Letters* 24.2 (1987), pp. 77 –79.

[JW96] David Janin and Igor Walukiewicz. "On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic". In: *CONCUR '96: Concurrency Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 263–277.

[Jør94] Niels Jørgensen. "Finding fixpoints in finite function spaces using neededness analysis and chaotic iteration". In: *Static Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 329–345.

[Koz83] D. Kozen. "Results on the Propositional *mu*-Calculus". In: *Theoretical Computer Science*, 27 (Jan. 1983), pp. 333–354.

[Lan07] Martin Lange. "Temporal logics beyond regularity". In: *BRICS Report Series* 14.13 (2007).

[Tar55] A. Tarski. "A lattice-theoretical fixpoint theorem and its applications". In: *Pacific journal of Mathematics* 5.2 (1955), pp. 285–309.

[VV04] Mahesh Viswanathan and Ramesh Viswanathan. "A Higher Order Modal Fixed Point Logic". In: *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*. 2004, pp. 512–528.