

TDK dolgozat

Zászlós Márton György



EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF PROGRAMMING LANGUAGES
AND COMPILERS

Matching Logic Proof System in Coq

Supervisor:

Dániel Horpácsi

Assistant Professor

Author:

Márton György Zászlós

Computer Science MSc

Budapest, 2023

Abstract

Matching logic is a unifying logic for reasoning about program behavior and correctness. It subsumes numerous logical frameworks, including first-order logic with least fixed points. There exist multiple formalizations of matching logic, one in Coq in particular, which this study is concerned with. Firstly, a user-friendly object-level proof mode is presented which has the capability to give names to hypotheses. Previously the formalization enabled object-level reasoning by indices of hypotheses only. Tactics are provided that operate on these hypotheses by their given names.

Secondly, the formalization is expanded with additional methods of reasoning about fixed points. The proof system of matching logic includes a Knaster-Tarski rule for such cases. A deduction theorem is presented which can be used together with Knaster-Tarski. Then natural numbers are captured in a theory of matching logic and the deduction theorem is used to prove statements about this theory, including Peano induction.

Contents

1	Introduction	2
2	Matching Logic	5
2.1	Locally Nameless Representation	5
2.2	Applicative Matching Logic	8
2.2.1	Syntax	8
2.2.2	Semantics	13
2.2.3	Proof System	14
3	Named Proof Mode	19
3.1	Proof Goal	20
3.2	Proof Tactics	22
4	Deduction Theorem	27
4.1	Theory of Definedness	27
4.2	Deduction Theorem with Knaster-Tarski	29
5	Peano Naturals	37
5.1	Refined Definedness Proof System	37
5.2	Theory of Sorts	41
5.3	Theory of Natural Numbers	42
5.4	Peano Induction	43
6	Summary	48
6.1	Future Work	48
6.2	Conclusion	49
	Acknowledgements	51
	Bibliography	52

Chapter 1

Introduction

As time progresses, programs become large, and the larger a program is, the harder it is to reason about its behavior. While conventional testing methods remain popular for ensuring correct operation of programs, there is interest in formal reasoning about programs for higher reassurance of expected behavior.

A proof involving the dynamics of a program tends to be complicated even for cases where the program is of smaller size. If it is expected for formal methods to be practical then it is necessary to provide automation for steps of the proof process, even if the whole process cannot be fully automated.

$\mathbb{K}$ [1] is a language framework for automatic reasoning about programs written in arbitrary programming languages. Languages can be comprehensively defined in $\mathbb{K}$ and the framework generates the necessary tooling for them autonomously, such as interpreters and verifiers. $\mathbb{K}$ aims to be the ideal language framework, that is, it desires to uniformly produce the widest range of language tooling from a formal definition of a language's semantics [2].

Matching logic [3] is a unifying logical framework designed with the intent of reasoning about behavior of programs. Its power to subsume first-order logic with fixed points, reachability logic, many-sorted logics, and many others [2] makes it a desirable framework for formal reasoning. It was proposed as the logic foundation of $\mathbb{K}$ and it has been shown that features of $\mathbb{K}$ can be defined as logical theories in it [2].

Many real-world programming languages have been fully defined as theories in matching logic, including C [4], Java [5], JavaScript [6], x86 assembly [7], and LLVM [8]. It is apparent that $\mathbb{K}$ and thus matching logic have a wide range of practical

applications in formal verification.

A formalization of matching logic exists in Coq [9]. While $\mathbb{K}$ provides a powerful framework for verification, it has no reliable solution to assist a proof which failed through automatic reasoning. In such a case, lemmas may be provided to $\mathbb{K}$ to assist in the proof, but such lemmas need to be reasoned about externally, often informally [9]. The Coq formalization aims to provide a reliable computer-assisted method to reason externally about such lemmas by hand. The formalization is open source and available at [10].

This study is organized as follows. In chapter 2 matching logic and the specific dialect of it used in the formalization is discussed, including its locally nameless representation, its syntax, and its proof system. Afterwards, the real contributions are presented. They can be separated into three large groups:

- Chapter 3 discusses an object-level proof mode for ease of propositional reasoning in matching logic. Specifically, it shows an alternative representation of a matching logic proof goal where hypotheses can be introduced and given names. Through this alternative representation it shows proof rules that refine the minimalist proof system of matching logic with the intent of creating more concise proofs.
- Chapter 4 presents a matching logic theory known as the theory of definedness. A deduction theorem is known to exist for theories that contain this theory. The formalization already had this deduction theorem, except it was constrained to exclude proofs which used the proof rule of Knaster-Tarski. This constraint is lifted, and so the deduction theorem now may be used with any proof rule.
- Chapter 5 proves a statement equivalent to Peano induction with fixed points. In order to discuss this proof, several preliminaries are described: a refined proof system, the subsuming of many-sortedness in applicative matching logic, the sort of natural numbers, and lemmas that aid in the proof of Peano induction.

Because of the wide range of these topics the chapters are mostly self-contained, that is, backgrounds and related works are included in each as relevant. It is notable that there are minor dependencies between them: the propositional proof mode of

chapter 3 is useful in latter chapters, and the deduction theorem of chapter 4 is used for the proof of Peano induction in chapter 5.

Lemmas are presented only if they are necessary for proofs shown. A lemma's proof is omitted if it is identical to how it appeared in prior work, mechanized or otherwise. Likewise, proofs are presented completely only if the proof differs from prior work. If only a segment is different, then only the different segment is presented. The proofs are presented as they were written in Coq and so they are by backwards reasoning.

Finally, the study concludes in chapter 6 after discussing areas of future work.

Chapter 2

Matching Logic

Multiple variants of matching logic exist, for example many-sorted and applicative. This study concerns the applicative variant [2]. In this chapter, applicative matching logic is described in locally nameless representation with its proof system. Finally, an informal understanding of its semantics is given.

2.1 Locally Nameless Representation

In this section an overview of locally nameless representation is given with discussion of why one might pick it over totally named or totally nameless. To minimize new information in this section the overview is with respect to first-order logic. The same principles will apply to matching logic. Locally nameless representation with respect to matching logic itself is given in the next section.

Let the scope of binding quantifiers go as far to the right as it can. A quantifier binds a variable such as in this first-order formula: $\exists x.P(x)$. Variables can also be free such as in this first-order formula: $P(y)$. In the same formula both bound and free variables may be found such as in $P(y) \rightarrow \exists x.P(x)$, where y is free while x is bound. This representation is known as totally named. While this is easy to read for humans, when formalizing a logic for mechanical assistance this can cause inconveniences.

Consider the first-order formula $P(x) \rightarrow \exists x.P(x)$. The variable x is both bound and free, also known as mixed. Mixed variables are generally avoided, as they might be confusing for humans. The earlier formula commonly appears when a naive substitution happens. Consider $\forall x.P(x) \wedge P(y)$. One might substitute x into y giv-

ing $\forall x.P(x) \wedge P(x)$. It is said that x was captured. The meaning of the formula was changed. This is generally not the intent. Capture-avoiding substitution might be used to avoid such a situation: variables might be renamed. A safe capture-avoiding substitution might change the earlier formula to retain meaning like so: $\forall z.P(z) \wedge P(x)$. The bound variable was renamed. Capture-avoiding substitution is unfortunately not simple to mechanize. However, one might observe that the name of a bound variable is irrelevant, as long as it still belongs to the same quantifier.

Instead, it is possible to represent variables with their distance from their quantifier. Variables are thus denoted as $\underline{n}$, where n is the distance to the binding quantifier, 0 indicating the closest. This is called a de Bruijn index.

$$\exists x.P(x) \equiv \exists.P(\underline{0})$$

x is bound by the closest quantifier it occurs under and thus it becomes the de Bruijn index $\underline{0}$.

If a variable is free they are considered as if they were bound by a universal quantifier outside of the whole formula. This is known as de Bruijn or totally nameless representation. Consider the following formula:

$$\exists x.P(x) \wedge P(y) \equiv \exists.P(\underline{0}) \wedge P(\underline{1})$$

$\underline{0}$ is bound by the quantifier. $\underline{1}$ "jumps" over the quantifier, making it a free variable.

By introducing more de Bruijn indices further free variables can be represented, such as in the following formula:

$$\exists x.P(x) \wedge P(y) \wedge P(w) \equiv \exists.P(\underline{0}) \wedge P(\underline{1}) \wedge P(\underline{2})$$

$\underline{0}$ is still bound, but both $\underline{1}$ and $\underline{2}$ "jump" over the quantifier, making them distinct free variables.

It can be observed that free variables can remain named. When free variables are named while bound variables are not, it is referred to as a locally nameless representation. Consider the earlier formulae and their locally nameless variants.

$$P(y) \rightarrow \exists x.P(x) \equiv P(y) \rightarrow \exists.P(\underline{0})$$

y is free and thus it is not affected by conversion to locally nameless representa-

tion. x becomes $\underline{0}$ for the same reason as in the first example. Notice that this way it is impossible to produce mixed variables.

Consider multiple quantifiers. Remember that the de Bruijn index is calculated by the distance to the quantifier.

$$\exists x. \exists y. P(x) \wedge P(y) \equiv \exists. \exists. P(\underline{1}) \wedge P(\underline{0})$$

The bound variable y is bound by the inner quantifier. The bound variable x is bound by the outer quantifier and thus it needs to "pass by" the inner quantifier hence it becoming $\underline{1}$.

The same de Bruijn index can refer to different quantifiers depending on where it occurs. Remember that the scope of a quantifier goes as far to the right as it can.

$$(\exists x. P(x)) \wedge (\exists y. P(y)) \equiv (\exists. P(\underline{0})) \wedge (\exists. P(\underline{0}))$$

$$\exists x. P(x) \wedge \exists y. P(y) \equiv \exists. P(\underline{0}) \wedge \exists. P(\underline{0})$$

Consider the following totally named first-order formulae: $\exists x. x \equiv \exists y. y$. These two formulae are α -equivalent, that is, they can be transformed into one another by renaming bound variables. It's difficult to reason mechanically about this, because of the differing names of the bound variables. Consider the locally nameless version $\exists. \underline{0} \equiv \exists. \underline{0}$. The formulae are identical, and thus reasoning about α -equivalence is unnecessary.

The advantages of de Bruijn indices were described: mixed variables cannot be expressed, and similarly, neither can substitutions capture. These are things that both totally and locally nameless representations provide. However, the formalization represents matching logic in a locally nameless representation.

Consider the totally nameless formula $P(\underline{0}) \wedge \exists. P(\underline{1})$. The index $\underline{0}$ and the index $\underline{1}$ under the quantifier both refer to the same variable. If one wanted to substitute $\underline{1}$ into $\underline{0}$ that would give $P(\underline{1}) \wedge \exists. P(\underline{2})$. Similarly if a quantifier needs to be dropped, a shifting need to occur in the other direction: dropping $\exists$ from $\exists. P(\underline{1})$ gives $P(\underline{0})$. Thus locally nameless provides advantages over totally nameless. Free variable substitution sidesteps the problem of shifting de Bruijn indices. It also retains more of human readability.

2.2 Applicative Matching Logic

In this section, a formal definition of the syntax of applicative matching logic is given in a locally nameless representation. Then an intuitive description of the semantics is provided. Finally, a proof system for applicative matching logic is shown.

2.2.1 Syntax

Definition 1 (Signature). A matching logic signature is a tuple $(EVar, SVar, \Sigma)$ where

- $EVar$ is a countably infinite set of element variables (denoted $x, y, \dots$)
- $SVar$ is a countably infinite set of set variables (denoted $X, Y, \dots$)
- Σ is a countable set of (constant) symbols (denoted $\sigma, f, g, \dots$)

When the variable sets are understood Σ is used to denote the whole signature.

Note. The formalization allows any type to be used as $EVar$ and $SVar$ as long as the type satisfies the requirement of countable infiniteness and a decidable equality exists between instances of the type. However, the formalization provides a signature where $EVar$ and $SVar$ are both strings as it is a convenient default which satisfies all constraints.

Definition 2 (Pattern). Let Σ be a signature. The set Pattern is inductively defined as

$$\varphi ::= x \mid X \mid \underline{n} \mid \underline{\underline{N}} \mid \sigma \mid \varphi_1 \varphi_2 \mid \perp \mid \varphi_1 \rightarrow \varphi_2 \mid \exists.\varphi \mid \mu.\varphi$$

In the above grammar:

- x is an element variable from $EVar$.
- X is a set variable from $SVar$.
- $\underline{n}$ is a de Bruijn index for an element variable ($n \in \mathbb{N}$).
- $\underline{\underline{N}}$ is a de Bruijn index for a set variable ($N \in \mathbb{N}$).
- σ is a first-order symbol.

- $\varphi_1\varphi_2$ is the application of φ_1 to φ_2 .
- $\perp$ is the pattern known as bottom.
- $\varphi_1 \rightarrow \varphi_2$ is the propositional operation called implication.
- $\exists.\varphi$ is existential quantification from first-order logic.
- $\mu.\varphi$ is a least fixed point.

An intuitive explanation of these and their commonly derived constructs is given later in this section.

Note. Two different kind of de Bruijn indices are present. $\underline{n}$ is always an element variable bound by an $\exists$ quantifier, while $\underline{\underline{N}}$ is always a set variable bound by a μ quantifier. $\underline{n}$ ignores μ quantifiers and $\underline{\underline{N}}$ ignores $\exists$ quantifiers. Consider the following pattern: $\exists.\mu.\underline{0} \wedge \underline{\underline{0}}$.

Not all patterns generated by the above grammar are valid on their own.

Consider the pattern $\mu.\underline{\underline{0}} \rightarrow \perp$. This pattern does not describe a valid least fixed point. It describes the least fixed point under the subset operation of the equation $X = (X \rightarrow \perp)$ with respect to X . One may observe that no such least fixed point exists [11].

Consider the patterns $\exists.\underline{0} \rightarrow \underline{1}$ and $\mu.\underline{\underline{1}}$. The bound variables $\underline{1}$ and $\underline{\underline{1}}$ have no corresponding binding quantifiers.

The following two definitions are mutually recursive.

Definition 3 (Positive Occurrence). Let $\varphi \in \text{Pattern}$, w be an element or set variable (bound or free).

$pos(\varphi, w)$ is defined recursively on φ :

$$\begin{aligned}
 pos(x, w) &\equiv \text{true if } x = w \\
 pos(x, w) &\equiv \text{false if } x \neq w \\
 pos(X, w) &\equiv \text{true if } X = w \\
 pos(X, w) &\equiv \text{false if } X \neq w \\
 pos(\underline{n}, w) &\equiv \text{true if } \underline{n} = w \\
 pos(\underline{n}, w) &\equiv \text{false if } \underline{n} \neq w \\
 pos(\underline{\underline{N}}, w) &\equiv \text{true if } \underline{\underline{N}} = w \\
 pos(\underline{\underline{N}}, w) &\equiv \text{false if } \underline{\underline{N}} \neq w
 \end{aligned}$$

$$\begin{aligned}
 pos(\sigma, w) &\equiv pos(\perp, w) \equiv false \\
 pos(\varphi_1 \varphi_2, w) &\equiv pos(\varphi_1, w) \vee pos(\varphi_2, w) \\
 pos(\varphi_1 \rightarrow \varphi_2, w) &\equiv neg(\varphi_1, w) \vee pos(\varphi_2, w) \\
 pos(Q.\varphi, w) &\equiv pos(\varphi, w) \text{ if } w \text{ is free or if } w \text{ is a bound set variable and } Q \text{ is } \exists \text{ or if } \\
 &\quad w \text{ is a bound element variable and } Q \text{ is } \mu \\
 pos(Q.\varphi, w) &\equiv pos(\varphi, w + 1) \text{ if } w \text{ is a bound set variable and } Q \text{ is } \mu \text{ or if } w \text{ is a} \\
 &\quad \text{bound element variable and } Q \text{ is } \exists
 \end{aligned}$$

Definition 4 (Negative Occurrence). Let $\varphi \in \text{Pattern}$, w be an element or set variable (bound or free).

$neg(\varphi, w)$ is defined recursively on φ :

$$\begin{aligned}
 neg(x, w) &\equiv false \\
 neg(X, w) &\equiv false \\
 neg(\underline{n}, w) &\equiv false \\
 neg(\underline{\underline{n}}, w) &\equiv false \\
 neg(\sigma, w) &\equiv neg(\perp, w) \equiv false \\
 neg(\varphi_1 \varphi_2, w) &\equiv neg(\varphi_1, w) \vee neg(\varphi_2, w) \\
 neg(\varphi_1 \rightarrow \varphi_2, w) &\equiv pos(\varphi_1, w) \vee neg(\varphi_2, w) \\
 neg(Q.\varphi, w) &\equiv neg(\varphi, w) \text{ if } w \text{ is free or } w \text{ is a bound set variable and } Q \text{ is } \exists \text{ or if } w \\
 &\quad \text{is a bound element variable and } Q \text{ is } \mu \\
 neg(Q.\varphi, w) &\equiv neg(\varphi, w + 1) \text{ if } w \text{ is a bound set variable and } Q \text{ is } \mu \text{ or if } w \text{ is a} \\
 &\quad \text{bound element variable and } Q \text{ is } \exists
 \end{aligned}$$

Note. Notice that when w is a bound variable passing a quantifier that may bind it, it must be incremented. This is because a bound variable refers to the structural distance of the quantifier binding it. If it goes under another quantifier then it must remember to "jump" over the quantifier.

Intuitively, the hole occurs negatively if it is nested under the left side for an odd number of implications. Conversely, the hole occurs positively if its occurrence is not nested under an odd number of implications.

Definition 5 (Well-formed-positive). Let $\varphi \in \text{Pattern}$. $well\text{-}formed\text{-}positive(\varphi)$ is defined recursively on φ :

$$well\text{-}formed\text{-}positive(x) = well\text{-}formed\text{-}positive(X) =$$

$$\text{well-formed-positive}(\underline{n}) = \text{well-formed-positive}(\underline{N}) =$$

$$\text{well-formed-positive}(\sigma) = \text{well-formed-positive}(\perp) = \text{true}$$

$$\text{well-formed-positive}(\varphi_1 \varphi_2) = \text{well-formed-positive}(\varphi_1 \rightarrow \varphi_2) =$$

$$\text{well-formed-positive}(\varphi_1) \wedge \text{well-formed-positive}(\varphi_2)$$

$$\text{well-formed-positive}(\exists.\varphi) = \text{well-formed-positive}(\varphi)$$

$$\text{well-formed-positive}(\mu.\varphi) = \neg \text{neg}(\underline{0}) \wedge \text{well-formed-positive}(\varphi)$$

Intuitively, definition 5 says that a pattern is well-formed-positive if in all sub-patterns of the form $\mu.\varphi$ the set variable bound by μ is not nested in an odd number of times on the left side of an $\rightarrow$ in φ . This guarantees the existence of a least fixed point.

Definition 6 (Well-formed-closed). Let $\varphi \in \text{Pattern}$, $\underline{n}$ is an element and $\underline{N}$ is a set variable. $\text{well-formed-closed}(\varphi, \underline{n}, \underline{N})$ is defined recursively on φ :

$$\text{well-formed-closed}(x, \underline{n}, \underline{N}) = \text{well-formed-closed}(X, \underline{n}, \underline{N}) = \text{true}$$

$$\text{well-formed-closed}(\underline{m}, \underline{n}, \underline{N}) = \underline{m} < \underline{n}$$

$$\text{well-formed-closed}(\underline{M}, \underline{n}, \underline{N}) = \underline{M} < \underline{N}$$

$$\text{well-formed-closed}(\sigma, \underline{n}, \underline{N}) = \text{well-formed-closed}(\perp, \underline{n}, \underline{N}) = \text{true}$$

$$\text{well-formed-closed}(\varphi_1 \varphi_2, \underline{n}, \underline{N}) = \text{well-formed-closed}(\varphi_1 \rightarrow \varphi_2, \underline{n}, \underline{N}) =$$

$$\text{well-formed-closed}(\varphi_1, \underline{n}, \underline{N}) \wedge \text{well-formed-closed}(\varphi_2, \underline{n}, \underline{N})$$

$$\text{well-formed-closed}(\exists.\varphi, \underline{n}, \underline{N}) = \text{well-formed-closed}(\varphi, \underline{n} + 1, \underline{N})$$

$$\text{well-formed-closed}(\mu.\varphi, \underline{n}, \underline{N}) = \text{well-formed-closed}(\varphi, \underline{n}, \underline{N} + 1)$$

The de Bruijn indices may be omitted to determine the well-formed-closed property of a pattern:

$$\text{well-formed-closed}(\varphi) \equiv \text{well-formed-closed}(\varphi, \underline{0}, \underline{0})$$

Note. It follows from the locally nameless representation and the above definition that if a variable is free then it is free in all patterns, and if it is bound then it is bound in all well-formed patterns.

Intuitively, definition 6 says that a pattern is well-formed-closed if in all subpatterns

- any element variable $\underline{n}$ is under at least $n + 1$ $\exists$ -quantifiers and
- any set variable $\underline{N}$ is under at least $N + 1$ μ -quantifiers.

Note that this is not identical to the classical closed property, that is, it is not required for all variables to be bound.

Definition 7 (Well-formed). Let $\varphi \in \text{Pattern}$.

$$\text{well-formed}(\varphi) \equiv \text{well-formed-positive}(\varphi) \wedge \text{well-formed-closed}(\varphi)$$

Note. In this study well-formedness of patterns in every statement is implicitly assumed to be as forgiving as possible. For example, if $\exists.\varphi$ appears on its own then it is assumed to be well-formed. However, $\exists.\varphi$ does not necessarily mean that φ is well-formed, unless φ also appears on its own, for instance in $\varphi \wedge \exists.\varphi$. This is justified by the capability of the Coq formalization to reason about well-formedness automatically.

Intuitively, a pattern is well-formed if it is well-formed-positive and well-formed-closed.

Definition 2 aims to be as succinct as possible. The intent is to not have rules in the grammar that are representable by composition of other rules. Next, further useful syntactic constructs are defined as sugar. Classical operator precedence can be assumed, where application is the strongest. Implication is right associative. The scope of quantifiers go as far to the right as possible.

$$\begin{aligned} \neg\varphi &\equiv \varphi \rightarrow \perp & \varphi_1 \vee \varphi_2 &\equiv \neg\varphi_1 \rightarrow \varphi_2 & \varphi_1 \wedge \varphi_2 &\equiv \neg(\neg\varphi_1 \vee \neg\varphi_2) \\ \top &\equiv \neg\perp & \forall.\varphi &\equiv \neg\exists.\neg\varphi & \varphi_1 \leftrightarrow \varphi_2 &\equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) \end{aligned}$$

2.2.2 Semantics

The topics of this study is about the syntax of matching logic and contains no semantic results. However, for better understanding of the results presented, knowledge of semantics might be beneficial. As such, in this subsection, an informal description is provided.

Matching logic has pattern matching semantics. Patterns are interpreted as the sets of elements that match them. Consider the disjunctive pattern $0 \vee 1$ where 0 and 1 are symbols. It means that 0 matches $0 \vee 1$ and 1 also matches $0 \vee 1$, but nothing else. From this relatively simple method rather complicated patterns can be created to match elements of particular structure, dynamic behavior, or other logical constraints.

When discussing the semantics of matching logic it is crucial to consider matching logic models. A model is made up of three parts:

- A model contains a non-empty set, which may be known as the domain or the universe. The elements of this set is what patterns may evaluate to.
- A binary function interpreting application.
- Subsets of the universe for all symbols in the signature, that is, what the symbols evaluate to.

First, the semantics of the basic constructs in definition 2 are discussed.

Element variables are interpreted as singleton sets, that is, they always represent a set of exactly one element. They match this one exact element they are interpreted as.

Set variables are interpreted as arbitrary sets. They match any of the elements they contain. They can be any subset of the interpretation's universe. They can be empty, the total set, or anything between.

Symbols are similar to set variables, because they also represent an arbitrary set of elements, but unlike set variables, the interpretation of a symbol is fixed.

Application is used to construct structures, commonly first-order functions and predicates, and also constructors. For example the first-order predicate $P(x)$ in matching logic can be represented as the applications Px where P is a symbol. The first-order function $F(x, y)$ can be represented as the application Fxy where F is a symbol.

$\perp$ is the pattern matched by exactly nothing.

The pattern $\varphi \rightarrow \psi$ is known as implication and is matched exactly by all elements where if they match φ then they match ψ .

$\exists.\varphi$ is matched by some valuation of $\underline{0}$, meaning it abstracts away the irrelevant part of the pattern φ . Note that the quantifier $\exists$ binds only element variables and never set variables.

Least fixed point $\mu.\varphi$ represents recursion by matching the smallest set $\underline{0}$ of elements that satisfy $\underline{0} = \varphi$. Note that the quantifier μ binds set variables only and never element variables.

Now, the semantics of some more refined constructs are discussed. As they are defined by other already known patterns, all of their semantics should follow from what has been described in the above paragraphs.

The pattern $\neg\varphi$ is known as negation or colloquially "not", and is matched exactly by patterns that do not match φ .

The pattern $\varphi \vee \psi$ is known as disjunction or colloquially "or", and is matched by patterns that match either φ or ψ .

The pattern $\varphi \wedge \psi$ is known as conjunction or colloquially "and", and is matched by patterns that match both φ and ψ .

$\top$ is the pattern matched by everything.

2.2.3 Proof System

In this subsection, the necessary definitions are given to understand matching logic's Hilbert-style proof system. Afterwards the proof system itself is described.

Definition 8 (Theory). Let $\varphi_1, \dots, \varphi_n \in \text{Pattern}$, $n \in \mathbb{N}$.

$\Gamma \equiv \{\varphi_1, \dots, \varphi_n\}$ is a set of patterns known as a matching logic theory.

$\varphi_1, \dots, \varphi_n$ are known as axioms.

Definition 9 (Derives). Let Γ be a matching logic theory and $\varphi \in \text{Pattern}$.

If φ can be proven from the axioms in Γ through the proof system then $\Gamma \vdash \varphi$. It is said that Γ derives φ or φ is provable from Γ .

Definition 10 (Substitution). Let $\varphi, \psi \in \text{Pattern}$, w be an element or set variable (bound or free).

$\varphi[\psi/w]$ is defined recursively on φ :

$$\begin{aligned}
 x[\psi/w] &\equiv x \text{ if } x \neq w \\
 x[\psi/w] &\equiv \psi \text{ if } x = w \\
 X[\psi/w] &\equiv X \text{ if } X \neq w \\
 X[\psi/w] &\equiv \psi \text{ if } X = w \\
 \underline{n}[\psi/w] &\equiv \underline{n} \text{ if } \underline{n} \neq w \\
 \underline{n}[\psi/w] &\equiv \psi \text{ if } \underline{n} = w \\
 \underline{\underline{n}}[\psi/w] &\equiv \underline{\underline{n}} \text{ if } \underline{\underline{n}} \neq w \\
 \underline{\underline{n}}[\psi/w] &\equiv \psi \text{ if } \underline{\underline{n}} = w \\
 \sigma[\psi/w] &\equiv \sigma \\
 (\varphi_1 \varphi_2)[\psi/w] &\equiv \varphi_1[\psi/w] \varphi_2[\psi/w] \\
 \perp[\psi/w] &\equiv \perp \\
 (\varphi_1 \rightarrow \varphi_2)[\psi/w] &\equiv \varphi_1[\psi/w] \rightarrow \varphi_2[\psi/w] \\
 (Q.\varphi)[\psi/w] &\equiv Q.\varphi[\psi/w] \text{ if } w \text{ is a free variable and } Q \in \{\exists, \mu\} \\
 (Q.\varphi)[\psi/w] &\equiv Q.\varphi[\psi/w] \text{ if } w \text{ is a bound set variable and } Q \text{ is } \exists \text{ or if } w \text{ is a bound} \\
 &\quad \text{element variable and } Q \text{ is } \mu \\
 (Q.\varphi)[\psi/w] &\equiv Q.\varphi[\psi/w + 1] \text{ if } w \text{ is a bound set variable and } Q \text{ is } \mu \text{ or if } w \text{ is a} \\
 &\quad \text{bound element variable and } Q \text{ is } \exists
 \end{aligned}$$

Note. Intuitively, $\varphi[\psi/w]$ replaces all occurrences of w in φ to ψ . Due to the locally nameless representation substitution is always capture avoiding as is.

Some rules allow elimination of quantifiers. However, quantified patterns might stop being well-formed-closed as seen in definition 6 after dropping the quantifier. For example if an $\exists$ quantifier is dropped from $\exists.\underline{0}$ then the new pattern is $\underline{0}$ which is not well-formed-closed as the element variable has no corresponding binder. This can be avoided by substituting a fresh variable into the bound variable.

Definition 11 (Fresh Variable). Let w be a free element or set variable and $\varphi \in \text{Pattern}$.

w is fresh in φ if w is not a subpattern of φ .

Definition 12 (Opening). Let $\varphi \in \text{Pattern}$, x a fresh element variable and X a fresh set variable in φ .

$\varphi[x/\underline{0}]$ is called the opening of $\exists.\varphi$.

$\varphi[X/\underline{\underline{0}}]$ is called the opening of $\mu.\varphi$.

Intuitively, opening drops the quantifier and substitutes a fresh variable into the pattern. This way if the initial pattern is well-formed then the opened pattern is also well-formed. Opening may be defined as a separate function as sugar for substituting into the zeroth de Bruijn index. The proofs in this study would always immediately unfold opening into substitution and as such no separate notation is introduced.

Definition 13 (Size). Let $\varphi \in \text{Pattern}$. $\text{size}(\varphi)$ is defined recursively on φ :

$$\text{size}(x) = \text{size}(X) = \text{size}(\underline{n}) = \text{size}(\underline{N}) = \text{size}(\sigma) = \text{size}(\perp) = 0$$

$$\text{size}(\varphi_1 \varphi_2) = \text{size}(\varphi_1 \rightarrow \varphi_2) = 1 + \text{size}(\varphi_1) + \text{size}(\varphi_2)$$

$$\text{size}(Q.\varphi) = 1 + \text{size}(\varphi) \text{ where } Q \in \{\exists, \mu\}$$

Note. Intuitively, the above definition counts the number of matching logic constructs in a pattern.

This is useful, as this study contains inductive proofs or more specifically proofs that use strong induction, that is, the induction asserts that the induction hypothesis is true for all patterns of smaller size than the pattern the induction is on.

Definition 14 (Context). Let $C \in \text{Pattern}$, $\square$ be a distinguished element variable called the hole.

C pattern is a context if it may contain the hole.

Definition 15 (Application Context). C context is an application context if from the root of C to the hole there are only applications, that is, if it's defined by the following grammar, where $\varphi \in \text{Pattern}$:

$$C ::= \square \mid \varphi C \mid C \varphi$$

Note. Do not confuse definitions 14 and 15, contexts are not constrained to application contexts unless explicitly said otherwise.

Definition 16 (Context Substitution). Let $\varphi \in \text{Pattern}$, C be a context.

$$C[\varphi] \equiv C[\varphi/\square]$$

Note. If it is understood that a substitution is against a hole then the hole is omitted. For example: if $C \equiv \varphi$ then $C[\psi] \equiv \varphi[\psi]$.

Definition 17 (Positive Context). Let C be a context.

If $\neg neg(C, \square)$ then it is said that C is a positive context.

Definition 18 (Negative Occurrence). Let C be a context.

If $\neg pos(C, \square)$ then it is said that C is a negative context.

Intuitively, definition 17 says that a context is positive if the hole does not occur negatively. Conversely, definition 18 says that a context is negative if the hole does not occur positively.

The rules of the proof system can be seen in table 2.1. Matching logic was originally defined with only three rule categories, fixed points were added later in [3], giving the current four categories. The proof system is known to be sound, however it is known to be complete only without fixed points [3].

- The category of FOL reasoning provides the well-known rules for comprehensive first-order and propositional reasoning. It is equivalent to the one seen in [2], but the rules $\exists$ -Quantifier and $\exists$ -Generalization had to be adjusted to the locally nameless representation.
- The category of frame reasoning is as seen in [9]. Due to the difficulties of mechanization it is easier to define application rules separated into left and right. It can be shown that the composite rules as seen in [2] are equivalent to the separated rules.
- The category of fixpoint reasoning contains rules for dealing with set variables and the quantifier μ as seen in [3]. Note that rules Pre-Fixpoint and Knaster-Tarski were adjusted to the locally nameless representation.
- The category of technical rules provides miscellaneous rules.

Category	Name	Rule
FOL Reasoning	Proposition 1	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
	Proposition 2	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
	Proposition 3	$((\varphi \rightarrow \perp) \rightarrow \perp) \rightarrow \varphi$
	Modus Ponens	$\frac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
	$\exists$ -Quantifier	$\varphi[x/\underline{0}] \rightarrow \exists.\varphi$
	$\exists$ -Generalization	$\frac{\varphi_1[x/\underline{0}] \rightarrow \varphi_2}{(\exists.\varphi_1) \rightarrow \varphi_2} \quad x \text{ is fresh in } \varphi_2$
Frame Reasoning	Propagation Left $_{\perp}$	$\perp\varphi$
	Propagation Right $_{\perp}$	$\varphi\perp$
	Propagation Left $_{\vee}$	$(\varphi_1 \vee \varphi_2)\psi \rightarrow (\varphi_1\psi) \vee (\varphi_2\psi)$
	Propagation Right $_{\vee}$	$\psi(\varphi_1 \vee \varphi_2) \rightarrow (\psi\varphi_1) \vee (\psi\varphi_2)$
	Propagation Left $_{\exists}$	$(\exists.\varphi)\psi \rightarrow \exists.\varphi\psi$
	Propagation Right $_{\exists}$	$\psi(\exists.\varphi) \rightarrow \exists.\psi\varphi$
	Framing Left	$\frac{\varphi_1 \rightarrow \varphi_2}{\varphi_1\psi \rightarrow \varphi_2\psi}$
	Framing Right	$\frac{\varphi_1 \rightarrow \varphi_2}{\psi\varphi_1 \rightarrow \psi\varphi_2}$
Fixpoint Reasoning	Set Variable Substitution	$\frac{\varphi}{\varphi[\psi/X]}$
	Pre-Fixpoint	$\varphi[\mu.\varphi/\underline{0}] \rightarrow \mu.\varphi$
	Knaster-Tarski	$\frac{\varphi[\psi/\underline{0}] \rightarrow \psi}{\mu.\varphi \rightarrow \psi}$
Technical Rules	Existence	$\exists.\underline{0}$
	Singleton	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

 Table 2.1: The Hilbert-style proof system of matching logic (C_1 and C_2 are application contexts as per definition 15.)

Chapter 3

Named Proof Mode

Writing proofs using the proof system of matching logic directly can be verbose and thus inconvenient. An object-level proof mode was made before for the matching logic formalization where proofs could be created with tactics that mirror the meta-level Coq tactics. The intention was to make it easy for a user familiar with Coq to reason about the embedded matching logic formalization by providing tools that look and behave similarly to that provided by Coq natively.

This proof mode could manipulate the proof context primarily by handling object-level hypotheses. The left side of implications in the goal could be pulled into a sequence of hypotheses and manipulated in there.

Previously, all of these hypotheses were addressable by their indices. This was in a sharp contrast with the proof mode of Coq where all hypotheses were given names either by the user or by Coq automatically. It is a significant quality of life feature to be able to name hypotheses.

In this chapter, such a proof mode is presented where all hypotheses are nameable by the user similarly to Coq’s meta solution. This proof mode was built upon the old index based proof mode.

For convenience if no name is given then the proof mode generates fresh names on its own, that is, it makes sure that every hypothesis has a unique name, even if the user shown no explicit intent of ensuring that. Consequently, it also ensures that names provided by the user do not collide with already existing names. While the primary goal is to mirror the meta tactics of Coq, many of these concepts were based on or inspired by the Iris Proof Mode [12].

3.1 Proof Goal

In this section, proof states are discussed. It is shown how the formalization represents a matching logic statement, and an alternative representation is given for the proof mode.

When a matching logic statement is made it simply appears as the provability relation seen in definition 9: $\Gamma \vdash \varphi$ where Γ is theory and φ is a pattern. Coq's tactics can be used to progress the proof, but that frequently makes a lot of boilerplate, and thus is unwieldy.

Coq's built-in logic is referred to as meta logic. Matching logic is formalized and reasoned about in this meta logic, and so matching logic is referred to as object logic. Object-level reasoning effectively refers to operating directly on a pattern φ in $\Gamma \vdash \varphi$.

It is important to stress that the proof mode is merely a different representation of a matching logic proof statement. The proof mode can be shown to possess the same expressive power as the meta-level reasoning: it simply composes the original proof system's rules for convenience. Proofs made with and without the proof mode are equally valid.

The proof mode can be entered by the tactic `toMLGoal`. After said tactic the matching logic statement is transferred into a goal that initially looks similar to before of the application, but now the proof environment can be divided into four parts instead of the initial three.

- Meta-level hypotheses. Lemmas that were proven earlier, or things that are not directly related to the matching logic proof itself.
- Object-level theory. Γ carries over into the proof mode unchanged.
- Object-level hypotheses. This is the new part that appears upon entering proof mode and it is initially empty. There are multiple ways to fill it up, such as parts of the goal can be moved here, and meta-level matching logic hypotheses can be added into it.
- Object-level goal.

The latter three are all actually part of the meta-level goal, and as such equivalent to where the proof was before entering the proof mode. It is important that

what is known is actually not altered, simply new tools become available as the representation of the proof goal is changed.

It is also important to mention that the object-level hypothesis sequence is different from the matching logic theory Γ . While the proof mode's behavior is similar to operating on a theory, it must be stressed that it merely alters the representation of that goal pattern, and is in no relation to the actual theory.

Object-level hypotheses are represented as a sequence of string-pattern tuples. The strings are for convenience of the user and thus they carry no actual meaning to the proof.

The proof mode can be left with `fromMLGoal`. The proof mode tactics can no longer be used. To not lose any information the hypotheses are flushed as implications into the goal.

Next these object-level hypotheses are defined.

Definition 19 (Named Hypotheses). Let $\mathbb{S}$ represent the set of strings, $\times$ be the Cartesian product operator, and $_*$ represent the set of (potentially empty) finite sequences.

$$\mathbb{H} \equiv (\mathbb{S} \times \text{Pattern})^*$$

$\mathbb{H}$ is the set of all possible named hypotheses.

Definition 20. $|hyps| : \mathbb{H} \rightarrow \mathbb{N}$ is used to denote the length of $hyps$.

Definition 21. Given a $hyps \in \mathbb{H}$ and $n \in 0..|hyps|$, the operation $hyps_n$ returns the n th element of the sequence $hyps$.

Definition 22. The function $names : \mathbb{H} \rightarrow \mathbb{S}$ projects the sequence $hyps \in \mathbb{H}$ to the sequence of names maintaining their order. $names(hyps)_n$ is used to denote the n th element of the sequence $names(hyps)$ where $n \in |hyps|$.

Definition 23. The function $patterns : \mathbb{H} \rightarrow \text{Pattern}$ projects the sequence $hyps \in \mathbb{H}$ to the sequence of patterns maintaining their order. $patterns(hyps)_n$ is used to denote the n th element of the sequence $patterns(hyps)$ where $n \in |hyps|$.

Note. $patterns(hyps)$ transforms $hyps$ into the form that also happens to be the representation prior to the addition of names. This is useful to minimize the changes necessary from indices only. It is generally true that where $hyps$ appears a sequence of patterns appeared before.

Definition 24 (Proof Mode Goal). Let Γ be a theory, $hyps \in \mathbb{H}$, $\varphi \in \text{Pattern}$, and $n = |hyps|$.

$$hyps \Rightarrow_{\Gamma} \varphi \equiv \Gamma \vdash \varphi_1 \rightarrow \dots \rightarrow \varphi_n \rightarrow \varphi$$

where $patterns(hyps)_m = \varphi_m$ ($m \in 1..n$).

Definition 25 (Concatenation). Let $hyps_1, hyps_2 \in \mathbb{H}$.

$hyps_1 \oplus hyps_2$ is used to denote the concatenation of the sequences.

Note. When adding a single element to a sequence consider it as a concatenation against a sequence of length 1.

3.2 Proof Tactics

In this section, lemmas are presented for backing tactics. In practice, every lemma presented is identical to one tactic. The tactics do more than the lemmas themselves, but these things are merely convenience features, such as turning the proof state to the right shape for the lemma to be directly applied, like dividing a hypothesis list into concatenations of multiple lists.

Focus is on how the lemmas changed with the transition from indices to names; therefore, not all tactics provided by the formalization are listed (for example, splitting on a conjunctive pattern in the goal).

It should be emphasized that the tactics merely compose valid proof steps, and thus they are equivalent to a proof done by hand through meta-level reasoning. All proofs can be deconstructed to the rules of the proof system.

The basic proof idea behind the following lemmas did not change from how they looked like prior to the introduction of names. They can be easily shown by propositional reasoning through definition 24.

Remember that $patterns(hyps)$ projects $hyps$ to its shape before the introduction of names, that is, hypotheses can be referred to by their indices in the sequence. By this observation all prior proofs can be reused by substituting $patterns(hyps)$ to where $hyps$ was used before [10].

In the next lemmas let $hyps, hyps_1, hyps_2 \in \mathbb{H}$ fixed.

Lemma 1 (To Goal). Let Γ be a matching logic theory, $|hyps| = 0$, $\varphi \in \text{Pattern}$.

$$\frac{hyps \Rightarrow_{\Gamma} \varphi}{\Gamma \vdash \varphi}$$

Lemma 2 (From Goal). *Let Γ be a matching logic theory, $|hyps| = n$, $\varphi \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi_1 \rightarrow \dots \rightarrow \varphi_n \rightarrow \varphi}{hyps \Rightarrow_{\Gamma} \varphi}$$

where $patterns(hyps)_m = \varphi_m$ ($m \in 1..n$).

Intuitively, lemma 1 enters the proof mode and lemma 2 leaves it. They provide a convenient dividing point between object-level and meta-level reasoning.

It is notable that the reasoning is backwards, as the tactics operate on the meta-level Coq proof goal.

It is also worth mentioning that one matching logic goal may be identical to many proof mode goals, that is, the relation which leaves the proof mode is not injective. Consider the following:

$$\begin{aligned} (s_1, \varphi_1), (s_2, \varphi_2) &\Rightarrow_{\Gamma} \varphi \equiv \\ (s_1, \varphi_1) &\Rightarrow_{\Gamma} \varphi_2 \rightarrow \varphi \equiv \\ \Gamma \vdash \varphi_1 &\rightarrow \varphi_2 \rightarrow \varphi \end{aligned}$$

Lemma 3 (Introduction). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $names(hyps)$.*

$$\frac{hyps \oplus (s, \phi) \Rightarrow_{\Gamma} \varphi}{hyps \Rightarrow_{\Gamma} \phi \rightarrow \varphi}$$

Note. Lemma 3 is similar to a deduction theorem, but it should be noted that Γ is unchanged.

Lemma 4 (Revert). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $names(hyps)$.*

$$\frac{hyps \Rightarrow_{\Gamma} \phi \rightarrow \varphi}{hyps \oplus (s, \phi) \Rightarrow_{\Gamma} \varphi}$$

The tactics so far did not actually alter the matching logic statement. They are the primary means of manipulating the shape of the statement, but they do not actually progress the proof.

Lemma 5 (Exact). *Let Γ be a matching logic theory, $\varphi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $names(hyps_1)$ and $names(hyps_2)$.*

$$hyps_1 \oplus (s, \varphi) \oplus hyps_2 \Rightarrow_{\Gamma} \varphi$$

Lemma 5 is the only tactic which is capable of finishing a proof in the proof mode.

Lemma 6 (Apply). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\text{hyps}_1 \oplus (s, \phi \rightarrow \varphi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \phi}{\text{hyps}_1 \oplus (s, \phi \rightarrow \varphi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}$$

Lemma 7 (Assert). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\text{hyps}_1 \Rightarrow_{\Gamma} \phi \quad \text{hyps}_1 \oplus (s, \phi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \phi}{\text{hyps}_1 \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}$$

Lemma 8 (Add). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\Gamma \vdash \phi \quad (s, \phi) \oplus \text{hyps} \Rightarrow_{\Gamma} \phi}{\text{hyps} \Rightarrow_{\Gamma} \varphi}$$

Lemma 9 (Clear). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\text{hyps}_1 \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}{\text{hyps}_1 \oplus (s, \phi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}$$

Lemma 10 (Destruct Or). *Let Γ be a matching logic theory, $\varphi, \phi, \psi \in \text{Pattern}$, $s, z, w \in \mathbb{S}$ where s, z, w do not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\text{hyps}_1 \oplus (z, \phi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi \quad \text{hyps}_1 \oplus (w, \psi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}{\text{hyps}_1 \oplus (s, \phi \vee \psi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}$$

Lemma 11 (Apply Meta). *Let Γ be a matching logic theory, $\varphi, \phi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $\text{names}(\text{hyps})$.*

$$\frac{\Gamma \vdash \phi \rightarrow \varphi \quad \text{hyps} \Rightarrow_{\Gamma} \phi}{\text{hyps} \Rightarrow_{\Gamma} \varphi}$$

Lemma 12 (Apply Meta In). *Let Γ be a matching logic theory, $\varphi, \phi, \psi \in \text{Pattern}$, $s \in \mathbb{S}$ where s does not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\Gamma \vdash \phi \rightarrow \psi \quad \text{hyps}_1 \oplus (s, \psi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}{\text{hyps}_1 \oplus (s, \phi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}$$

Note. The proof mode works with backwards reasoning. However, note that when the whole proof is backwards then the reasoning in the hypothesis sequence is actually going forward.

Lemma 13 (Destruct And). *Let Γ be a matching logic theory, $\varphi, \phi, \psi \in \text{Pattern}$, $s, z, w \in \mathbb{S}$ where s, z, w do not occur in $\text{names}(\text{hyps}_1)$ and $\text{names}(\text{hyps}_2)$.*

$$\frac{\text{hyps}_1 \oplus (z, \phi) \oplus (w, \psi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}{\text{hyps}_1 \oplus (s, \phi \wedge \psi) \oplus \text{hyps}_2 \Rightarrow_{\Gamma} \varphi}$$

When hypotheses were using indices their access was always unique. Thus, if the user gave an index it either identified a hypothesis uniquely or it over-indexed. By transitioning to strings now the *hyps* sequence can contain the same string multiple times, assuming no precautions are taken. This would mean that a string would identify multiple hypotheses.

In practice, this would make it impossible to operate on some of those hypotheses, as one would shadow all the others. The new tactics are designed in such a way that attempting to add a string which is already present is dealt with in a similar way to over-indexing, and as such is not doable.

The user may omit names when the name does not appear in the conclusion. In such a scenario `Std++[13]` is used to generate fresh names, that is, a string is automatically created which is not contained in $\text{names}(\text{hyps})$.

In later chapters, to solve some steps the above proof mode will be used. When the steps are understood to be simple and self-contained propositional steps then the proof mode notation with the actual names may be omitted. Remember that the proof mode is just an alternative representation of the same matching logic statement, as such entering and leaving the proof mode is merely cosmetic and may be omitted as well.

Next some lemmas that describe commonly known propositional truths are shown which will be useful later when it is more cumbersome to enter the proof mode. These lemmas can be trivially proven by the proof mode rules, and thus their proofs are omitted for brevity.

Lemma 14. *Let Γ be a matching logic theory, $\varphi, \psi \in \text{Pattern}$.*

$$\Gamma \vdash \varphi \wedge \psi \leftrightarrow \psi \wedge \varphi$$

Lemma 15. *Let Γ be a matching logic theory, $\varphi, \psi, \phi \in \text{Pattern}$.*

$$\Gamma \vdash (\varphi \wedge \psi \rightarrow \phi) \leftrightarrow (\varphi \rightarrow \psi \rightarrow \phi)$$

Lemma 16. *Let Γ be a matching logic theory, $\varphi, \psi, \phi \in \text{Pattern}$.*

$$\Gamma \vdash (\psi \rightarrow \varphi) \rightarrow (\phi \wedge \psi \rightarrow \phi \wedge \varphi)$$

Lemma 17. *Let Γ be a matching logic theory, $\varphi \in \text{Pattern}$.*

$$\Gamma \vdash \varphi \leftrightarrow \varphi$$

Chapter 4

Deduction Theorem

In this chapter, a matching logic theory is described called definedness and a deduction theorem is shown for all theories containing definedness axioms [3]. It is discussed why a deduction theorem is useful, and what peculiarities of matching logic cause the need for the theory of definedness for stating a deduction theorem. While in [3] new cases are introduced in the deduction theorem for all fixpoint reasoning rules, the formalization already has the cases of Set Variable Substitution and Pre-Fixpoint mechanized. As the proof system of the formalization already contained the rule Knaster-Tarski, the deduction theorem was stated by saying that the proof may not use Knaster-Tarski. As such only this constraint is what needs to be lifted.

4.1 Theory of Definedness

In this section, theory of definedness is described formally and then an intuitive understanding of it is given.

Definition 26 (Definedness). Let $\llbracket _ \rrbracket$ be a symbol known as the definedness symbol. $\llbracket \varphi \rrbracket$ is used as sugar to denote the application $\llbracket _ \rrbracket \varphi$.

Let $\Gamma^{def} \equiv \{\forall.\llbracket 0 \rrbracket\}$, known as the theory of definedness.

$\forall.\llbracket 0 \rrbracket$ is known as the definedness axiom.

It is said that if $\llbracket \varphi \rrbracket$ then φ is defined ($\varphi \in \text{Pattern}$).

Note. When considering a theory Γ which contains Γ^{def} , that is, $\Gamma^{def} \subseteq \Gamma$, the signature is assumed to contain the symbol $\llbracket _ \rrbracket$.

Using the definedness symbol further matching logic constructs are defined as notations.

Definition 27.

$$\lfloor \varphi \rfloor \equiv \neg \lceil \neg \varphi \rceil \quad \varphi_1 = \varphi_2 \equiv \lfloor \varphi_1 \leftrightarrow \varphi_2 \rfloor$$

$$x \in \varphi \equiv \lceil x \wedge \varphi \rceil \quad \varphi_1 \subseteq \varphi_2 \equiv \lfloor \varphi_1 \rightarrow \varphi_2 \rfloor$$

$$\varphi_1 \neq \varphi_2 \equiv \neg(\varphi_1 = \varphi_2) \quad x \notin \varphi \equiv \neg(x \in \varphi)$$

Intuitively, $\lceil \varphi \rceil$ means that something matches pattern φ . In practice, this means that if some element of the domain matches φ then every element of the domain matches $\lceil \varphi \rceil$.

$\lfloor \varphi \rfloor$ is known as totality. If $\lfloor \varphi \rfloor$ then it is said that φ is total. If φ is matched by all elements of the domain then every element of the domain matches $\lfloor \varphi \rfloor$.

$\varphi_1 = \varphi_2$ is equality. $x \in \varphi$ is membership. $\varphi_1 \subseteq \varphi_2$ is inclusion. Equality, membership, and inclusion have the expected semantics.

One might notice that definedness and totality behave as $\top$ and $\perp$, that is, they either match everything or nothing. Since matching logic has no concepts of true and false, this can be used to subsume Boolean logic.

Remember that the universe is never empty and so there is no confusion: there is no model in which $\top = \perp$, and thus the subsuming of Boolean logic is correct for all models.

Now that an equality construct has been made, predicates can be defined.

Definition 28 (Predicate). Let $\Gamma^{def} \subseteq \Gamma$ and $\psi \in \text{Pattern}$.

If $\Gamma \vdash \psi = \top \vee \psi = \perp$ then it is said that ψ is a predicate pattern.

Note. Operations preserve the predicate property, that is if φ and ψ are predicates then $\varphi \rightarrow \psi$, $\varphi \vee \psi$, $\mu.\varphi$ and so on are also predicates.

Lemma 18. Let $\Gamma^{def} \subseteq \Gamma$ and $\varphi \in \text{Pattern}$.

$\lceil \varphi \rceil$ is a predicate pattern.

Lemma 19. Let $\Gamma^{def} \subseteq \Gamma$ and $\varphi \in \text{Pattern}$.

$\lfloor \varphi \rfloor$ is a predicate pattern.

4.2 Deduction Theorem with Knaster-Tarski

In this section, the deduction theorem is discussed, then lemmas are shown that are necessary for proving the case of Knaster-Tarski in the deduction theorem. Afterwards the deduction theorem itself is proven.

A deduction theorem is capable of moving a part of the conclusion into the theory. This allows the proof to then progress on the remaining part in the goal, without losing the information which is still available in the theory. Consider a deduction theorem of first-order logic which may look like so (ignoring constraints on the structure of the proof):

If $\Gamma \cup \{\psi\} \vdash \varphi$ then $\Gamma \vdash \psi \rightarrow \varphi$.

If Γ is a first-order theory and ψ and φ are first-order formulae then the above deduction theorem may be proven. Remember that formulae in first-order logic are two-valued, that is, they either evaluate to true or false. This is not true in matching logic.

For instance, the set variable X may evaluate to any subset of the domain, including the empty set, the total set, and anything between. It would be necessary for φ to always contain the valuation of ψ , even in interpretations where ψ does not hold. As such, the first-order deduction theorem may not be directly adapted into matching logic.

Remember definition 28, that is, predicates can be represented in matching logic by requiring a pattern to always evaluate to either the empty set or the whole universe. This enables us to represent two-valued patterns, since the universe may not be empty.

One may observe that it would be possible to constrain all patterns participating in a deduction theorem to be predicates. However, a more allowing constraint may be used:

If $\Gamma \cup \{\psi\} \vdash \varphi$ then $\Gamma \vdash [\psi] \rightarrow \varphi$.

This is by the fact that as per lemma 19, totality is always a predicate pattern.

The proofs rely on opening of quantifiers (definition 12) and manipulation of contexts (definition 14), since they are substitution-heavy both for free and bound variables. As such, next lemmas are presented about composing substitutions of element and set variables (free and bound).

Lemma 20. *Let $\varphi, \psi \in \text{Pattern}$. If φ is well-formed-closed then*

$$\varphi[\psi/\underline{0}] = \varphi$$

Lemma 21. *Let $\varphi, \psi, \phi \in \text{Pattern}$, x be a free element variable. If x does not occur in ϕ then*

$$\varphi[\psi/x][\phi/\underline{0}] = \varphi[\phi/\underline{0}][\psi/x]$$

Proof. Do induction on the structure of φ .

- $\varphi \equiv x_0$.

Suppose $x = x_0$.

$\psi[\phi/\underline{0}] = \psi$ is the goal.

Which is proven by lemma 20.

Suppose $x \neq x_0$.

$x_0[\phi/\underline{0}] = x_0$ is the goal.

Which is proven by definition 10.

- $\varphi \equiv \underline{0}$.

$\phi = \phi[\psi/x]$ is the goal.

Which is then proven from the fact that x does not occur in ϕ .

- Every other case follows immediately from the induction hypothesis.

That is all cases. □

Lemma 22. *Let $\varphi, \psi \in \text{Pattern}$, x be a free element variable. If x does not occur in φ then*

$$\varphi[x/\underline{0}][\psi/x] = \varphi[\psi/\underline{0}]$$

Proof. Do structural induction on φ .

The cases where φ is a set variable, bound element variable, a symbol, or $\perp$ are trivial.

- $\varphi \equiv x_0$.

$x_0[x/\underline{0}][\psi/x] = x_0[\psi/\underline{0}]$ is the goal.

$x_0 = x_0$ because x does not occur in x_0 .

Which is trivial.

- $\varphi \equiv \underline{0}$.

$\underline{0}[x/\underline{0}][\psi/x] = \underline{0}[\psi/\underline{0}]$ is the goal.

Which is proven by definition 10.

- Every other case follows immediately from the induction hypothesis.

That is all cases. □

Next, lemmas leading to the deduction theorem are presented. Most notably, they are about monotonicity of quantifiers and contexts, and the propagation of predicates through quantifiers and contexts.

Lemma 23. *Let $\varphi, \psi \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi \rightarrow \psi}{\Gamma \vdash (\exists. \varphi[\underline{0}/x]) \rightarrow \exists. \psi[\underline{0}/x]}$$

Lemma 24. *Let $\varphi, \psi \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi \rightarrow \psi}{\Gamma \vdash (\mu. \varphi[\underline{0}/X]) \rightarrow \mu. \psi[\underline{0}/X]}$$

Proof. See [3, Lemma 85]. □

Intuitively, lemma 23 and 24 say that quantifiers in matching logic are monotonic regarding implication. Note the consequences of the locally nameless representation, that is, the variables becoming bound need to be substituted into a different syntactic construct.

Lemma 25. *Let $\varphi, \psi \in \text{Pattern}$.*

If C is a positive context (definition 17) then

$$\frac{\Gamma \vdash \varphi \rightarrow \psi}{\Gamma \vdash C[\varphi] \rightarrow C[\psi]}$$

If C is a negative context (definition 18) then

$$\frac{\Gamma \vdash \varphi \rightarrow \psi}{\Gamma \vdash C[\psi] \rightarrow C[\varphi]}$$

Proof. Based on [3, Lemma 87]. Only the positive variant is shown. The negative variant is constructed identically to the positive direction.

Do induction on the size of C then on the structure of C . The base cases are trivial.

The proof is split by the structure of C . The cases where C is a bound element or set variable are proven by contradiction with the well-formedness of C .

- $C \equiv x$.

Suppose x is the hole.

$x \rightarrow x$ is the goal.

Which is proven by lemma 17.

Suppose x is not the hole.

$\varphi \rightarrow \psi$ is the goal.

Which is the induction hypothesis.

- $C \equiv X$.

$X \rightarrow X$ is the goal.

Which is proven by lemma 17.

- $C \equiv \sigma$.

$\sigma \rightarrow \sigma$ is the goal.

Which is proven by lemma 17.

- $C \equiv \xi_1 \xi_2$.

$\xi_1[\varphi]\xi_2[\varphi] \rightarrow \xi_1[\psi]\xi_2[\psi]$ is the goal by definition 10.

Note that $\xi_1[\varphi]\xi_2[\varphi] \rightarrow \xi_1[\psi]\xi_2[\varphi]$ by Framing Left and the induction hypothesis $\xi_1[\varphi] \rightarrow \xi_1[\psi]$.

$\xi_1[\psi]\xi_2[\varphi] \rightarrow \xi_1[\psi]\xi_2[\psi]$ by the above observation.

$\xi_2[\varphi] \rightarrow \xi_2[\psi]$ by Framing Right.

Which is the other induction hypothesis.

- $C \equiv \perp$.

$\perp \rightarrow \perp$ is the goal.

Which is proven by lemma 17.

- $C \equiv \xi_1 \rightarrow \xi_2$.

$(\xi_1[\varphi] \rightarrow \xi_2[\varphi]) \rightarrow (\xi_1[\psi] \rightarrow \xi_2[\psi])$ is the goal by definition 10.

Note that the induction hypotheses are $\xi_1[\psi] \rightarrow \xi_1[\varphi]$ and $\xi_2[\varphi] \rightarrow \xi_2[\psi]$.

ξ_1 must be a negative context and ξ_2 must be a positive context. This follows from the initial assumption that C is positive and the definition of a positive context. Convert the goal into the proof mode by lemma 1 and then add the induction hypotheses to the named hypotheses by lemma 8. The left side of the goal may be added to the named hypotheses by lemma 3:

$$(IH_1, \xi_1[\psi] \rightarrow \xi_1[\varphi]),$$

$$(IH_2, \xi_2[\varphi] \rightarrow \xi_2[\psi]),$$

$$(H_0, \xi_1[\varphi] \rightarrow \xi_2[\varphi]),$$

$$(H, \xi_1[\psi]) \Rightarrow_{\Gamma} \xi_2[\psi].$$

In the following steps the named hypotheses are omitted as they are identical to the above.

$\xi_2[\psi]$ is the goal of the proof mode.

$\xi_2[\varphi]$ by lemma 6 with $s = IH_2$.

$\xi_1[\varphi]$ by lemma 6 with $s = H_0$.

$\xi_1[\psi]$ by lemma 6 with $s = IH_1$.

Which is then proven by lemma 5 with $s = H$.

- $C \equiv \exists.\xi$.

$(\exists.\xi[\varphi]) \rightarrow \exists.\xi[\psi]$ is the goal by definition 10.

$\xi[\varphi][x/\underline{0}] \rightarrow \xi[\psi][x/\underline{0}]$ by lemma 23. x is a fresh element variable in ξ .

$\xi[x/\underline{0}][\varphi] \rightarrow \xi[x/\underline{0}][\psi]$ by lemma 21.

Which is the induction hypothesis.

- $C \equiv \mu.\xi$.

$(\mu.\xi[\varphi]) \rightarrow \mu.\xi[\psi]$ is the goal by definition 10.

$\xi[\varphi][X/\underline{0}] \rightarrow \xi[\psi][X/\underline{0}]$ by lemma 24 and $\mu.\xi$ is assumed positive. X is a fresh set variable in ξ .

$\xi[X/\underline{0}][\varphi] \rightarrow \xi[X/\underline{0}][\psi]$. by lemma 21.

Which is the induction hypothesis.

That is all cases. □

Intuitively, lemma 25 says that contexts are monotonic depending on their positivity.

Lemma 26. *Let $\Gamma^{def} \subseteq \Gamma$ and $\varphi, \psi \in \text{Pattern}$. If ψ is a predicate pattern and C is a context where the hole may not appear under a μ binder then*

$$\Gamma \vdash \psi \wedge C[\varphi] \leftrightarrow \psi \wedge C[\psi \wedge \varphi]$$

Proof. See [3, Lemma 88]. □

Lemma 27. *Let X be a fresh set variable in φ and ψ . If ψ is a predicate pattern then*

$$\Gamma \vdash (\mu.\psi \wedge \varphi) \leftrightarrow \psi \wedge \mu.\varphi$$

Proof. Based on [3, Lemma 89]. The proof is split into two cases.

- $(\mu.\psi \wedge \varphi) \rightarrow \psi \wedge \mu.\varphi$ is the goal.

The proof is split further into two cases.

- $(\mu.\psi \wedge \varphi) \rightarrow \psi$ is the goal.

$(\psi \wedge \varphi)[\psi/\underline{0}] \rightarrow \psi$ by Knaster-Tarski.

$\psi \wedge \varphi[\psi/\underline{0}] \rightarrow \psi$ by well-formedness of ψ .

$\psi \rightarrow \varphi[\psi/\underline{0}] \rightarrow \psi$ by lemma 13.

$\psi \rightarrow \psi$ by lemma 9.

Which is then proven by lemma 5.

- $(\mu.\psi \wedge \varphi) \rightarrow \mu.\varphi$ is the goal.

$(\psi \wedge \varphi)[X/\underline{0}] \rightarrow \varphi[X/\underline{0}]$ by lemma 24. X is being substituted as it was assumed to be fresh in φ and ψ .

$\psi \wedge \varphi[X/\underline{0}] \rightarrow \varphi[X/\underline{0}]$ by the definition 10.

$\psi \rightarrow \varphi[X/\underline{0}] \rightarrow \varphi[X/\underline{0}]$ by lemma 13.

$\varphi[X/\underline{0}] \rightarrow \varphi[X/\underline{0}]$ by lemma 9.

Which is then proven by lemma 5.

- $\psi \wedge (\mu.\varphi) \rightarrow \mu.\psi \wedge \varphi$ is the goal.

$(\mu.\varphi) \wedge \psi \rightarrow \mu.\psi \wedge \varphi$ by lemma 14.

$(\mu.\varphi) \rightarrow \psi \rightarrow \mu.\psi \wedge \varphi$ by lemma 13.

$\varphi[\psi \rightarrow \mu.\psi \wedge \varphi/\underline{0}] \rightarrow \psi \rightarrow \mu.\psi \wedge \varphi$ by Knaster-Tarski.

$\varphi[\psi \rightarrow \mu.\psi \wedge \varphi/\underline{0}] \wedge \psi \rightarrow \mu.\psi \wedge \varphi$ by lemma 15.

$\psi \wedge \varphi[\psi \rightarrow \mu.\psi \wedge \varphi/\underline{0}] \rightarrow \mu.\psi \wedge \varphi$ by lemma 14.

$\psi \wedge \varphi[\psi \rightarrow \mu.\psi \wedge \varphi/\underline{0}] \rightarrow (\psi \wedge \varphi)[\mu.\psi \wedge \varphi/\underline{0}]$ by Pre-Fixpoint.

$\psi \wedge \varphi[\psi \rightarrow \mu.\psi \wedge \varphi/\underline{0}] \rightarrow \psi \wedge \varphi[\mu.\psi \wedge \varphi/\underline{0}]$ by the definition 10.

$\psi \wedge \varphi[\psi \wedge (\psi \rightarrow \mu.\psi \wedge \varphi)/\underline{0}] \rightarrow \psi \wedge \varphi[\mu.\psi \wedge \varphi/\underline{0}]$ by lemma 26. It should be noted that the substitutions $\varphi[x/\underline{0}][\psi \rightarrow \mu.\psi \wedge \varphi/x]$ and $\varphi[x/\underline{0}][\psi \wedge (\psi \rightarrow \mu.\psi \wedge \varphi)/x]$ were "collapsed" by lemma 22 (x is fresh in φ).

$\varphi[\psi \wedge (\psi \rightarrow \mu.\psi \wedge \varphi)/\underline{0}] \rightarrow \varphi[\mu.\psi \wedge \varphi/\underline{0}]$ by lemma 16

$\psi \wedge (\psi \rightarrow \mu.\psi \wedge \varphi) \rightarrow \mu.\psi \wedge \varphi$ by lemma 25. It should be noted that the substitutions $\varphi[x/\underline{0}][\psi \wedge (\psi \rightarrow \mu.\psi \wedge \varphi)/x]$ and $\varphi[x/\underline{0}][\mu.\psi \wedge \varphi/x]$ were "collapsed" by lemma 22 (x is fresh in φ).

$\psi \rightarrow (\psi \rightarrow \mu.\psi \wedge \varphi) \rightarrow \mu.\psi \wedge \varphi$ by lemma 13.

$\psi \rightarrow (\psi \rightarrow \mu.\psi \wedge \varphi) \rightarrow \psi$ by lemma 6.

$\psi \rightarrow \psi$ by lemma 9.

Which is then proven by lemma 5.

That is all cases. □

Intuitively, lemma 26 and 27 say that predicates in conjunction with contexts or a μ quantifier can be freely moved inside or outside of the context or the quantifier.

The deduction theorem now can be proven.

Theorem 1 (Deduction Theorem). *Let $\Gamma^{def} \subseteq \Gamma$ and $\varphi, \psi \in Pattern$. If*

1. *the proof does not generalize on free element variables of ψ ,*
2. *the proof does not substitute free set variables of ψ , and*
3. *the proof uses Knaster-Tarski then $\underline{0}$ may not appear under a μ binder in φ*

then

$$\frac{\Gamma \cup \{\psi\} \vdash \varphi}{\Gamma \vdash [\psi] \rightarrow \varphi}$$

Proof. Do induction on the proof rule. All cases are proven as if Knaster-Tarski was forbidden except the case of Knaster-Tarski itself. In case of Knaster-Tarski

$\varphi \equiv (\mu.\varphi_1) \rightarrow \varphi_2$ and so

$\lfloor \psi \rfloor \rightarrow (\mu.\varphi_1) \rightarrow \varphi_2$ is the goal.

$\lfloor \psi \rfloor \wedge (\mu.\varphi_1) \rightarrow \varphi_2$ by lemma 15.

$(\mu.\lfloor \psi \rfloor \wedge \varphi_1) \rightarrow \varphi_2$ by lemma 27. $\lfloor \psi \rfloor$ is a predicate by lemma 19.

$(\lfloor \psi \rfloor \wedge \varphi_1)[\varphi_2/\underline{0}] \rightarrow \varphi_2$ by Knaster-Tarski.

$\lfloor \psi \rfloor \wedge \varphi_1[\varphi_2/\underline{0}] \rightarrow \varphi_2$ by definition 10 and well-formedness of $\lfloor \psi \rfloor$.

That is the induction hypothesis. □

Note. Realize that the constraints of the deduction theorem are similar to the constraints of the first-order version: what is said about element variables is simply extended to set variables with respect to the fixpoint reasoning rules.

Chapter 5

Peano Naturals

In this chapter, a theorem based on least fixed points that is equivalent to Peano induction is proven.

First, a proof system is shown which is refined from the proof system presented in chapter 2 and the definedness axiom presented in the last chapter. This proof system is specifically designed to deal with membership. Having seen lemmas about membership, many-sortedness is axiomatized as an applicative matching logic theory. Using such sorting, a theory of natural numbers based on Peano axioms is presented. Afterwards a statement based on fixed points equivalent to Peano induction is proven.

The refined proof system is not shown with all rules as seen in [3]. This chapter's goal is to present a proof equivalent to Peano induction. As such, what is seen here is limited to what is necessary for that proof.

5.1 Refined Definedness Proof System

For the ease of reasoning about theories with definedness symbols and specifically membership, a refined proof system is presented in [3]. Some lemmas of this refined proof system are shown here as they are useful when proving Peano induction. The formalization did not contain the full refined proof system, and as such lemmas that were not formalized are now proven with respect to differences to the mechanization as usual.

First, some lemmas are presented which are not related to definedness, but they will be useful for the refined proof system.

Lemma 28 (Universal Generalization). *Let Γ be a matching logic theory, $\varphi \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall.\varphi[\underline{0}/x]}$$

Lemma 29. *Let Γ be a matching logic theory, $\varphi \in \text{Pattern}$, x be fresh in φ .*

$$\frac{\Gamma \vdash \forall.\varphi}{\Gamma \vdash \varphi[x/\underline{0}]}$$

The above well-known first-order-like lemmas follow from the definition of $\forall$ and the proof rules regarding $\exists$. Remember that in matching logic $\forall$ is defined by $\exists$.

The rest of the lemmas in this section are all in the theory of definedness. Where Γ is used, it is assumed that Γ is a matching logic theory and $\Gamma^{\text{def}} \subseteq \Gamma$.

Lemma 30. *Let $\varphi, \psi, \phi \in \text{Pattern}$. If φ, ψ and ϕ are μ free and ψ is a predicate pattern, then*

$$\Gamma \vdash \psi \wedge \phi\varphi \leftrightarrow \phi(\psi \wedge \varphi)$$

Proof. The proof is split into two cases.

- $\psi = \top$.

The goal is $\top \wedge \phi\varphi \leftrightarrow \phi(\top \wedge \varphi)$.

$\phi\varphi \leftrightarrow \phi\varphi$ by $\top \wedge \xi = \xi$ for any ξ .

Which is then proven by lemma 17.

- $\psi = \perp$.

The goal is $\perp \wedge \phi\varphi \leftrightarrow P(\perp \wedge \varphi)$.

$\perp \leftrightarrow \phi\perp$ by $\perp \wedge \xi = \perp$ for any ξ .

Which is then proven by Propagate Right $_{\perp}$.

That is all cases. □

Intuitively, lemma 30 says that predicate patterns propagate through application.

Lemma 31. *Let $\varphi \in \text{Pattern}$, x be an element variable.*

$$\Gamma \vdash [x \wedge [\varphi]] \rightarrow [\varphi]$$

Proof. This is proven through lemma 34 and the fact that $[[\varphi]] \rightarrow [\varphi]$. For details see [10]. □

Lemma 32. *Let $\varphi \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \lfloor \varphi \rfloor}$$

Proof. See [3, Lemma 46]. □

Note. $\Gamma \vdash \varphi \rightarrow \lfloor \varphi \rfloor$ does not hold. Consider an interpretation where φ holds for some element of the universe, but not all.

Lemma 33. *Let $\varphi \in \text{Pattern}$.*

$$\Gamma \vdash \lfloor \varphi \rfloor \rightarrow \varphi$$

Proof. See [3, Lemma 57]. □

Lemma 34. *Let $\varphi_1, \varphi_2 \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi_1 \rightarrow \varphi_2}{\Gamma \vdash \lceil \varphi_1 \rceil \rightarrow \lceil \varphi_2 \rceil}$$

Intuitively, lemma 34 says that definedness is monotone with regards to implication. Realize that from the rules Framing Right and Framing Left all applications can be shown monotone.

Lemma 35 (Membership Elimination). *Let $\varphi \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \forall \underline{0} \in \varphi}{\Gamma \vdash \varphi}$$

Proof. See [3, Lemma 50]. □

Lemma 36. *Let $\varphi_1, \varphi_2 \in \text{Pattern}$.*

$$\Gamma \vdash (x \in (\varphi_1 \rightarrow \varphi_2)) \leftrightarrow ((x \in \varphi_1) \rightarrow (x \in \varphi_2))$$

Proof. This is proven by the propagation of negation through symbols. Remember that conjunction is defined by disjunction and negation. For details see [10]. □

Lemma 37. *Let $\varphi_1, \varphi_2 \in \text{Pattern}$.*

$$\frac{\Gamma \vdash \varphi_1 \rightarrow \varphi_2}{\Gamma \vdash x \in \varphi_1 \rightarrow x \in \varphi_2}$$

Proof. This is proven through lemma 34. For details see [10]. □

Intuitively, lemma 37 says that membership is monotonic with regards to implication, assuming the the subject of the membership is an element variable.

Lemma 38 (Membership_∧). *Let $\varphi_1, \varphi_2 \in \text{Pattern}$.*

$$\Gamma \vdash (x \in (\varphi_1 \wedge \varphi_2)) \leftrightarrow ((x \in \varphi_1) \wedge (x \in \varphi_2))$$

Proof. See [3, Lemma 54]. □

Lemma 39 (Membership_∃). *Let $\varphi \in \text{Pattern}$.*

$$\Gamma \vdash (x \in (\exists.\varphi)) \leftrightarrow (\exists.x \in \varphi)$$

Proof. See [3, Lemma 55]. □

Intuitively, lemmas 38 and 39 say that propositional and first-order constructs propagate through membership assuming the subject of the membership is an element variable. These two rules are used for proving the final lemma of this subsection, which is imperative for the proof of peano induction.

Lemma 40. *Let $\varphi \in \text{Pattern}$.*

$$\Gamma \vdash \varphi \rightarrow \exists.\underline{0} \in \varphi \wedge \underline{0}$$

Proof. See [3, Lemma 62]. □

Lemma 41 (Membership Symbol). *Let $\varphi, \psi \in \text{Pattern}$. If φ and ψ are μ free, then*

$$\Gamma \vdash x \in \psi\varphi \rightarrow \exists.\underline{0} \in \varphi \wedge x \in \psi\underline{0}$$

Proof. First, it is shown that $x \in \psi\varphi \rightarrow x \in \psi(\exists.\underline{0} \in \varphi \wedge \underline{0})$.

$\psi\varphi \rightarrow \psi(\exists.\underline{0} \in \varphi \wedge \underline{0})$ by lemma 37.

$\varphi \rightarrow (\exists.\underline{0} \in \varphi \wedge \underline{0})$ by Framing Right.

Which is proven by lemma 40.

Then, it is shown that $x \in \psi(\exists.\underline{0} \in \varphi \wedge \underline{0}) \rightarrow x \in (\exists.\psi(\underline{0} \in \varphi \wedge \underline{0}))$.

$\psi(\exists.\underline{0} \in \varphi \wedge \underline{0}) \rightarrow (\exists.\psi(\underline{0} \in \varphi \wedge \underline{0}))$ by lemma 37.

$\psi(\exists.\underline{0} \in \varphi \wedge \underline{0}) \rightarrow \psi(\exists.\underline{0} \in \varphi \wedge \underline{0})$ by Propagation Right_∃.

$(\exists.\underline{0} \in \varphi \wedge \underline{0}) \rightarrow \exists.\underline{0} \in \varphi \wedge \underline{0}$ by Framing Right.

Which is then proven by lemma 17.

The goal so far is $x \in (\exists.\psi(\underline{0} \in \varphi \wedge \underline{0})) \rightarrow \exists.\underline{0} \in \varphi \wedge x \in \psi\underline{0}$.

$(x \in \exists.\psi(\underline{0} \in \varphi \wedge \underline{0})) \rightarrow \exists.\underline{0} \in \varphi \wedge x \in \psi\underline{0}$ by lemma 39.

$x \in \psi(y \in \varphi \wedge y) \rightarrow y \in \varphi \wedge x \in \psi y$ by lemma 23. y is fresh in x , φ , and ψ .

$x \in (y \in \varphi \wedge \psi y) \rightarrow y \in \varphi \wedge x \in \psi y$ by lemma 30. $y \in \psi$ is a predicate by lemma 18.

$x \in (y \in \varphi) \wedge x \in \psi y \rightarrow y \in \varphi \wedge x \in \psi y$ by lemma 38.

$y \in \varphi \wedge x \in \psi y \rightarrow y \in \varphi \wedge x \in \psi y$ by lemma 31.

$x \in \psi y \rightarrow x \in \psi y$ by lemma 16.

Which is then proven by lemma 5. □

Intuitively, lemma 41 says that if an element variable is member of an application then there is a particular element of that application that the variable belongs to. This is best understood by considering the application to be a function.

5.2 Theory of Sorts

Applicative matching logic is unsorted. Fortunately, applicative matching logic can subsume its many-sorted variant [2].

Recall that definedness as seen in section 4.1 provides necessary constructs to recreate sorting, notably membership and inclusion. Only sorted quantification is missing which can be redefined using definedness without introducing more axioms.

Definition 29 (Inhabitant). Let $\llbracket _ \rrbracket$ be known as the inhabitant symbol. The application $\llbracket _ \rrbracket s$ is shortened as $\llbracket s \rrbracket$.

Intuitively, s is the sort name while the inhabitant can be used to define the set captured by sort s .

Definition 30. Let $Sorts$ be a symbol. It represents the name of the sort of sorts, that is, $\llbracket Sorts \rrbracket$ represents the set which evaluates to the names of sorts.

Definition 31 (Sorted Quantification). Let $\varphi \in \text{Pattern}$, s be a symbol.

$$\forall s. \varphi \equiv \forall. (\underline{0} \in \llbracket s \rrbracket) \rightarrow \varphi \quad \exists s. \varphi \equiv \exists. (\underline{0} \in \llbracket s \rrbracket) \wedge \varphi$$

Note. Remember that the representation is locally nameless. $\forall s. \varphi$ does not mean that s is the bound variable, instead it specifies the sort of the variable. Consider $\forall x:s. \varphi \equiv \forall s. \varphi$ where the left is totally named and the right is locally nameless.

While $\forall.\underline{0}$ ranges over the whole universe, $\forall s.\underline{0}$ ranges over only elements that inhabit sort s . Likewise for exists: $\exists.\underline{0}$ shows that an element exists, $\exists s.\underline{0}$ shows that an element exists which is also of sort s . Note that practically this means that sorted existence abstracts away the irrelevant part of a pattern but only if it is of the correct sort.

Through definedness functional constants can be defined, that is, a pattern which always evaluates to a single element. This is equivalent to a nullary function.

Definition 32 (Functional Pattern). Let $\varphi \in \text{Pattern}$. If

$$\Gamma \vdash \exists.\varphi = \underline{0}$$

then it is said that φ is a functional pattern.

Based on the above definition through sorted quantification as seen in definition 31 it is possible to define sorted functions of arbitrary arity similarly to many-sorted first-order logic.

Definition 33 (Function). Given symbol f and some symbols $s_1, \dots, s_n, s$ ($n \in \mathbb{N}$) if

$$\Gamma \vdash \forall s_1. \dots \forall s_n. \exists s. f \ \underline{n} \dots \underline{1} = \underline{0}$$

then it is said that f is a function of arity n where its arguments are of sort $s_1, \dots, s_n$ and its valuation is of sort s .

Then the theory of definedness along with signatures that contain definedness and inhabitant symbols can be used to define arbitrary sorts. An example is given in the next section.

5.3 Theory of Natural Numbers

In this section, a sort of natural numbers is captured in a matching logic theory based on Peano axioms [2].

Definition 34 (Naturals). Let Γ^{nat} be the axiom set which contains Γ^{def} and the following axioms:

- Sort Name. $(Nat \in \llbracket \text{Sorts} \rrbracket) \wedge \exists. Nat = \underline{0}$
- Non-Empty Inhabitant. $\llbracket Nat \rrbracket \neq \perp$

- Function 1. $\exists Nat.zero = \underline{0}$
- Function 2. $\forall Nat.\exists Nat.succ \underline{1} = \underline{0}$
- Function 3. $\forall Nat.\forall Nat.\exists Nat.plus \underline{2} \underline{1} = \underline{0}$
- No Confusion 1. $\forall Nat.zero \neq succ \underline{0}$
- No Confusion 2. $\forall Nat.\forall Nat.succ \underline{1} = succ \underline{0} \rightarrow \underline{1} = \underline{0}$
- Inductive Domain. $\llbracket Nat \rrbracket = \mu.zero \vee succ \underline{0}$

Intuitively, sort name says that *Nat* is a sort name and some element of the universe must represent that sort name.

Non-empty inhabitant says that some natural number must exist in the domain of an interpretation.

The three function axioms specify that symbols *zero*, *succ*, and *plus* represent first-order-like functions as per definition 33.

The two no confusion axioms specify that *zero* is not a successor of any number and if two successors are equal then their predecessors must be equal as well, that is, all naturals are unique.

Inductive domain specifies that naturals are the smallest set composed by *zero* and application of *succ*.

Note. *plus* according to Function 3 is a binary function. Writing *plus**x* describes a partial application, requiring an interpretation to provide what it means when *plus* is partially applied. This is called curried style. As this study is not interested in interpretations, the curried style is considered simpler, as the syntactical description is easier. An alternative would be the uncurried style: to ensure that the application instead uses some sort of tuple construct, but that is syntactically more complicated, as said construct needs to be defined on the matching logic level, while chaining of applications is readily available.

5.4 Peano Induction

In this section, lemmas are defined for proving Peano induction. Afterwards a statement based on fixed points is proven which can be shown equivalent to Peano induction.

Lemma 42. *Let $\varphi, \phi \in \text{Pattern}$. If ϕ is μ free, then*

$$\Gamma \vdash (\exists.\varphi = \underline{0}) \rightarrow (\exists.\phi = \underline{0}) \rightarrow (\varphi \in \phi) \rightarrow (\varphi = \phi)$$

Intuitively, lemma 42 says that if two patterns are functional, that is, they evaluate to singleton sets, then equality between the two follows from membership. Consider any two singleton sets: it should be obvious that they are equal only if they contain the same element which is the same as to say that one contains what the other contains.

Lemma 43. *Let $\varphi, \psi \in \text{Pattern}$. If φ and ψ are μ free, then*

$$\frac{\Gamma \vdash \exists.\varphi = \underline{0} \quad \Gamma \vdash \varphi \in \psi}{\Gamma \vdash \varphi \subseteq \psi}$$

Proof. The goal is $\varphi \rightarrow \psi$ by lemma 32.

$\forall.\underline{0} \in (\varphi \rightarrow \psi)$ by 35.

$x \in (\varphi \rightarrow \psi)$ by lemma 28. x is fresh in $\varphi \rightarrow \psi$.

$x \in \varphi \rightarrow x \in \psi$ by lemma 36.

$x = \varphi \rightarrow x \in \psi$ by lemma 42. The condition $\exists.\varphi = 0$ is assumed.

$\varphi \in \psi$ by equational reasoning.

That is the assumption. □

Lemma 43 says that if two patterns are functional, then membership and inclusion are the same between them. The idea is the same as in lemma 42. Note while lemma 43 is on the meta level, it could be stated on the object level as well, similarly to lemma 42.

Lemma 44. *Let $\varphi \in \text{Pattern}$ and s be a symbol.*

$$\Gamma \vdash (\exists s.\varphi) \rightarrow \exists.\varphi$$

Proof. Suppose x is a fresh element variable in $\exists s.\varphi$.

$(\exists.\underline{0} \in \llbracket s \rrbracket) \wedge \varphi \rightarrow \exists.\varphi$ by definition 31.

$((\underline{0} \in \llbracket s \rrbracket) \wedge \varphi)[x/\underline{0}] \rightarrow \exists.\varphi$ by $\exists$ -Generalization.

$(x \in \llbracket s \rrbracket) \wedge \varphi[x/\underline{0}] \rightarrow \exists.\varphi$ by definition 10.

$(x \in \llbracket s \rrbracket) \wedge \varphi[x/\underline{0}] \rightarrow \varphi[x/\underline{0}]$ by $\exists$ -Quantifier.

$(x \in \llbracket s \rrbracket) \rightarrow \varphi[x/\underline{0}] \rightarrow \varphi[x/\underline{0}]$ by lemma 13.

$\varphi[x/\underline{0}] \rightarrow \varphi[x/\underline{0}]$ by lemma 9.

Which is then proven by lemma 5. □

Intuitively, lemma 44 means that if something exists in a sort, then it exists in the whole universe with no respect to the sort as well. This is crucial, as the formalization only subsumes many-sortedness, that is, not everything is sorted: functional patterns are considered unsorted. This needs attention, as the axioms of Peano naturals require sorting. This lemma bridges this gap.

Now that lemmas that need only definedness were shown, lemmas in the theory of naturals are stated.

Lemma 45.

$$\Gamma^{nat} \vdash \llbracket Nat \rrbracket \subseteq X \rightarrow \forall Nat.\underline{0} \in X$$

Proof. The goal is

$\llbracket Nat \rrbracket \subseteq X \rightarrow \forall \underline{0} \in \llbracket Nat \rrbracket \rightarrow \underline{0} \in X$ by definition 31.

$\llbracket \llbracket Nat \rrbracket \rightarrow X \rrbracket \rightarrow \forall \underline{0} \in \llbracket Nat \rrbracket \rightarrow \llbracket \underline{0} \wedge X \rrbracket$ by definition 27.

$\forall \underline{0} \in \llbracket Nat \rrbracket \rightarrow \llbracket \underline{0} \wedge X \rrbracket$ by theorem 1. $\Gamma = \Gamma^{nat} \cup \{\llbracket Nat \rrbracket \rightarrow X\}$

$\llbracket x \wedge \llbracket Nat \rrbracket \rrbracket \rightarrow \llbracket x \wedge X \rrbracket$ by lemma 28.

$x \wedge \llbracket Nat \rrbracket \rightarrow x \wedge X$ by lemma 34.

$\llbracket Nat \rrbracket \rightarrow X$ by lemma 16.

That is in the theory. □

Peano induction now can be proven. It is possible to show that the following theorem is equivalent to $\varphi(zero) \wedge (\forall \underline{0} \rightarrow \varphi(succ \underline{0})) \rightarrow \forall \underline{0} \rightarrow \varphi(\underline{0})$, which is classically understood as Peano induction [3]. The statement is based on [14] but there the proof is semantic and so what can be seen here is more closely resembled by the proof seen in [2].

For the sake of simplicity of the proof the theorem is stated with meta-level conditions. They are lifted into the proof goal as the proof progresses. It should be possible to state the theorem with a single object-level statement along with a slightly more cumbersome proof.

Theorem 2 (Peano Induction). *If*

$$1. \Gamma^{nat} \vdash X \subseteq \llbracket Nat \rrbracket$$

$$2. \Gamma^{nat} \vdash zero \in X$$

$$3. \Gamma^{nat} \vdash \forall Nat.\underline{0} \in X \rightarrow succ \underline{0} \in X$$

then

$$\Gamma^{nat} \vdash \forall Nat. \underline{0} \in X$$

Proof. The goal is

$\llbracket Nat \rrbracket \subseteq X$ by lemma 45.

$(\mu.zero \vee succ \underline{0}) \subseteq X$ by axiom Inductive Domain.

$\lfloor (\mu.zero \vee succ \underline{0}) \rightarrow X \rfloor$ by definition 27.

$(\mu.zero \vee succ \underline{0}) \rightarrow X$ by lemma 32.

$zero \vee succ X \rightarrow X$ by Knaster-Tarski.

The proof is split into two cases.

- The goal is $zero \rightarrow X$.

$\lfloor zero \rightarrow X \rfloor$ by lemma 33.

By application of lemma 43 assumption 2 is reached. Now it is only necessary to show that $\exists.zero = \underline{0}$.

$\exists Nat.zero = \underline{0}$ by lemma 44.

That is axiom Function 1.

- The goal is $succ X \rightarrow X$.

$\forall.\underline{0} \in (succ X \rightarrow X)$ by lemma 35.

$x \in (succ X \rightarrow X)$ by lemma 28.

$x \in succ X \rightarrow x \in X$ by lemma 36.

$(\exists.\underline{0} \in X \wedge x \in succ \underline{0}) \rightarrow x \in X$ by lemma 41.

$y \in X \wedge x \in succ y \rightarrow x \in X$ by $\exists$ -Generalization.

$y \in X \wedge x = succ y \rightarrow x \in X$ by lemma 42. Note that the conditions are proven by the definedness axiom and axiom Functional 2 respectively.

$y \in X \rightarrow succ y \in X$ by equational reasoning.

$(y \in \llbracket Nat \rrbracket \rightarrow y \in X \rightarrow succ y \in X) \rightarrow y \in X \rightarrow succ y \in X$ by assumption 3. y was substituted into assumption 3 by lemma 29.

$(y \in \llbracket Nat \rrbracket \wedge y \in X \rightarrow succ y \in X) \rightarrow y \in X \rightarrow succ y \in X$ by lemma 15.

$(y \in \llbracket Nat \rrbracket \wedge y \in X \rightarrow succ y \in X) \rightarrow y \in X \rightarrow y \in \llbracket Nat \rrbracket \wedge y \in X$ by lemma 6.

$y \in X \rightarrow y \in \llbracket Nat \rrbracket \wedge y \in X$ by lemma 9.

The proof is split further into two cases.

– The goal is $y \in X \rightarrow y \in \llbracket Nat \rrbracket$.

$y \wedge X \rightarrow y \wedge \llbracket Nat \rrbracket$ by 34.

$X \subseteq \llbracket Nat \rrbracket \rightarrow y \wedge X \rightarrow y \wedge \llbracket Nat \rrbracket$ by assumption 1.

$(X \rightarrow \llbracket Nat \rrbracket) \rightarrow y \wedge X \rightarrow y \wedge \llbracket Nat \rrbracket$ by lemma 33.

$(X \rightarrow \llbracket Nat \rrbracket) \rightarrow X \rightarrow \llbracket Nat \rrbracket$ by lemma 16.

Which is then proven by lemma 5.

– The goal is $y \in X \rightarrow y \in X$

Which is then proven by lemma 17.

That is all cases.

□

Chapter 6

Summary

6.1 Future Work

- The tactics presented in chapter 3 support propositional logic only. While they are quite powerful on their own, the proof mode should be made more complete. For convenience, if something can be proven outside of the proof mode then it should also be possible to prove it without leaving the proof mode. Specifically, tactics should be created for first-order reasoning.
- While the Coq formalization provides means of reasoning automatically about the well-formedness of patterns, this automatization fails to properly account for fixed points. This causes inconveniences in the mechanization of proofs in chapter 4. It should be possible to completely automatize this.

In addition, as the deduction theorem has been mechanized two enhancements could be made:

Firstly, the formalization currently has two deduction theorems, one with Knaster-Tarski and one without. They are separately proven. It should be possible to unify them better.

Secondly, if the deduction theorem is applied in a proof then it may not be applied again. This is a consequence of a simplification of the deduction theorem's restrictions in the formalization. This should be corrected by following the statement of the deduction theorem in [3] more properly.

- It is of particular interest to mechanize the refined proof rules of fixed point reasoning as provided by [15]. A tactic may be created which could apply the proof rules algorithmically.

It is notable that these proof rules were already seen working in the $\mathbb{K}$ framework, but the consequences of their formalization in an interactive theorem prover is unknown.

An alternative proof of Peano induction as seen in chapter 5 is a good candidate for experimenting with the refined fixed point rules. Theoretically, it should be possible to automate the whole proof.

The matching logic formalization provides an embedding of first-order logic into Coq and shows that matching logic subsumes it. This embedding is unsorted. It is known that applicative matching logic subsumes not only its own many-sorted variant, but many-sorted first-order logic itself [2]. Likewise, a many-sorted embedding may also be created.

6.2 Conclusion

In this study, three different, but related results were discussed.

The thesis started with the description of applicative matching logic in a locally nameless representation. A formal description of its syntax, informal description of its semantics, and formal description of its proof system was given.

Then a named proof mode was shown for ease of propositional reasoning in matching logic, with emphasis on how the introduction of names affects the system. Through the proof mode more concise proofs could be written than using the proof rules manually, even though it is equivalent to the proof system rules.

Afterwards, a deduction theorem to be used with Knaster-Tarski and the lemmas leading to it were presented along with a theory that allows said deduction theorem to exist. The proof mode of the last chapter was shown working through this chapter.

Finally, a refined proof system of definedness was shown, which through a subsuming of many-sorted matching logic was presented. They were used to support a theory of natural numbers from Peano axioms adapted to applicative matching logic. A proof of a statement equivalent to Peano induction based on fixed points was

given. The deduction theorem was shown working and used alongside Knaster-Tarski in this proof.

While all of these had a solid foundation before, here they were presented with respect to the difficulty of their mechanization with matching logic in Coq. This highlighted some important differences against their unmechanized versions.

It is notable that this formalization is the first mechanization of matching logic in a proof assistant, specifically Coq. The above presented results are thus the first time that they were mechanized in an interactive framework of matching logic.

Acknowledgements

I would like to express my gratitude to Runtime Verification Inc. for the financial support of the project. In addition, I would like to thank Dániel Horpácsi for supervising, Péter Berezky and Jan Tusil for their assistance, and Simon John Thompson for his guidance.

Bibliography

- [1] G. Rosu. “K: A semantic framework for programming languages and formal analysis tools”. In: Jan. 2017, pp. 186–206. DOI: 10.3233/978-1-61499-810-5-186.
- [2] Xiaohong Chen and Grigore Roşu. *Applicative Matching Logic: Semantics of K*. Tech. rep. University of Illinois at Urbana-Champaign, 2019. URL: <http://hdl.handle.net/2142/104616> (visited on 11/19/2022).
- [3] Xiaohong Chen and Grigore Roşu. “Matching μ -Logic”. In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2019. DOI: 10.1109/LICS.2019.8785675.
- [4] Chris Hathhorn, Chucky Ellison, and Grigore Roşu. “Defining the Undefinedness of C”. In: *SIGPLAN Not.* 50 (2015). DOI: 10.1145/2813885.2737979.
- [5] D. Bogdanas and G. Roşu. “K-Java: A complete semantics of Java”. In: *Conference Record of the Annual ACM Symposium on Principles of Programming Languages* 2015 (2015). DOI: 10.1145/2676726.2676982.
- [6] Daejun Park, Andrei Stefănescu, and Grigore Roşu. “KJS: A Complete Formal Semantics of JavaScript”. In: *SIGPLAN Not.* 50 (2015). DOI: 10.1145/2813885.2737991.
- [7] Sandeep Dasgupta et al. “A Complete Formal Semantics of X86-64 User-Level Instruction Set Architecture”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2019. DOI: 10.1145/3314221.3314601.
- [8] Theodoros Kasampalis et al. “Language-Parametric Compiler Validation with Application to LLVM”. In: *Proceedings of the 26th ACM International*

- Conference on Architectural Support for Programming Languages and Operating Systems*. 2021. DOI: 10.1145/3445814.3446751.
- [9] Péter Berczky et al. “Mechanizing Matching Logic in Coq”. In: *Proceedings of the Sixth Working Formal Methods Symposium, FROM 2022, "Al. I. Cuza University", Iasi, Romania, 19-20 September, 2022*. EPTCS 369 (2022). DOI: 10.4204/EPTCS.369.2.
- [10] *AML Formalization*. URL: <https://github.com/harp-project/AML-Formalization/releases/tag/v1.0.10> (visited on 11/19/2022).
- [11] Yuri Gurevich and Saharon Shelah. “Fixed-point extensions of first-order logic”. In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. 1985, pp. 346–353. DOI: 10.1109/SFCS.1985.27.
- [12] Robbert Krebbers, Amin Timany, and Lars Birkedal. “Interactive Proofs in Higher-Order Concurrent Separation Logic”. In: *SIGPLAN Not.* 52 (2017). DOI: 10.1145/3093333.3009855.
- [13] Max Planck Institute for Software Systems. *Coq-std++: An extended "standard library" for Coq*. URL: <https://gitlab.mpi-sws.org/iris/stdpp> (visited on 11/24/2022).
- [14] Xiaohong Chen, Dorel Lucanu, and Grigore Roşu. “Matching logic explained”. In: *Journal of Logical and Algebraic Methods in Programming* 120 (2021). DOI: 10.1016/j.jlamp.2021.100638.
- [15] Xiaohong Chen et al. “Towards a Unified Proof Framework for Automated Fixpoint Reasoning Using Matching Logic”. In: *OOPSLA* 4 (2020). DOI: 10.1145/3428229.