
TDK dolgozat

Benamor Adil



EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF PROGRAMMING LANGUAGES
AND COMPILERS

Formalization of Matching Logic Theories in Coq

Supervisor:

Dr. Dániel Horpácsi

Assistant Professor

Author:

Adil Benamor

Computer Science MSc

Budapest, 2024

Contents

1	Introduction	1
2	Matching Logic Background	3
2.1	Syntax	3
2.2	Semantics	5
2.3	Specification of Basic Theories	7
2.4	Proof System	9
2.5	Proof Mode	12
3	Formalization of ML Theories	15
3.1	Supporting Theorems for Theories	15
3.2	Simple Theories	18
3.2.1	Booleans Theory	18
3.2.2	Natural Numbers Theory	26
3.3	Parametric Theories	29
3.3.1	Sum Sort Theory	29
3.3.2	Product Sort Theory	32
4	Discussion	34
5	Related Work	36
6	Summary	38
6.1	Conclusion	38
6.2	Future Work	39
	Acknowledgements	40
	Bibliography	40

Abstract

The present work is part of a wider research project that intends to explore and aim to establish a solid foundation for machine-checked, and interactive reasoning about systems and languages based on matching logic (ML). The key concept of ML is patterns that are constructed from variables, constant symbols, a binary application, and standard first-order logic formulas such as bottom/false, implication, and existential quantification, with the addition of the least fixpoint.

The central focus of this thesis is the formalization of matching logic theories within a Coq implementation of ML, these theories represent mathematical domains and datatypes like booleans, natural numbers, product, and sum types. We derive definitions, and notations for each theory, and we formulate various theorems that we subsequently prove in the Coq framework. For the construction of these proofs, we employ and reuse the available concepts, a rich library of notations, and the ML proof system offered by the formalization of ML in Coq which we build upon.

Chapter 1

Introduction

In recent years, there has been a surge of interest in reasoning about the static structure and dynamic behavior of programs. Matching logic [1] stands out as a unique logic that finds applications in various domains, including programming languages and theorem proving [2]. This versatility allows ML to function as a unifying logic, facilitating the specification and reasoning about both static and dynamic aspects of programs. However, to effectively utilize formal verification techniques for reasoning about the correctness of machine-checking the paper-based proofs, a crucial first step is formalizing ML theories within a suitable proof assistant framework, which fosters a higher degree of confidence and trust in them.

Adding to that, ML serves as the foundation for the $\mathbb{K}$ semantic framework. The $\mathbb{K}$ framework's effectiveness has been demonstrated by its use in developing practical verifiers for a wide range of real-world programming languages. This list includes well-established languages like C [3], Java [4], JavaScript [5], Python [6], and Rust [7, 8]. ML's foundational role empowers $\mathbb{K}$ to formally define programming languages and subsequently reason about the inherent properties of both these languages and the programs written within them. The successful use of $\mathbb{K}$ in various fields demonstrates the practical significance and power of ML.

At the core of matching logic lies the concept of patterns. These patterns are meticulously constructed from fundamental building blocks: variables, constant symbols, a binary application construct, and established (FOL) constructs such as implication, existential quantification, bottom (false), and the least fixpoint construct [1]. The semantics of patterns, which we will explore in detail in section 2, involve interpreting them as sets of elements that satisfy the pattern's constraints.

Furthermore, our main contributions through this thesis are as follows:

- The formalization of ML theories; Booleans, natural numbers, sum sort, and product sort, that enable the representation of diverse mathematical domains and datatypes within Coq.
- The definition of theorems for each theory and their subsequent proofs utilizing an ML proof system formalized in Coq.
- The extension of the formal ML library in Coq by specifying and proving new auxiliary theorems that capture a variety of properties and use functional patterns to generalize existing theorems and express them at the meta-level.

Finally, The remainder of this thesis is structured as follows:

- In section 2, we provide background information on ML syntax, semantics, the proof system, the proof mode, and the specification of basic theories.
- In section 3, we present the details of our formalization approach, explaining how we represent ML theories within Coq, and theorem proofs for each formalized specification.
- In section 5, we review the related work in the field.
- In section 6, we conclude the thesis and outline potential avenues for future research.

Chapter 2

Matching Logic Background

This section provides essential background information for understanding the formalization of Matching Logic (ML) within the Coq proof assistant framework. For all technical details, we refer to [1, 2].

2.1 Syntax

Definition 1. An ML signature $(EVar, SVar, \Sigma)$ has a set $EVar$ of *element variables* $x, y, \dots$, a set $SVar$ of *set variables* $X, Y, \dots$, and a set Σ of (constant) *symbols* $\sigma, f, g, \dots$. We often omit $EVar$ and $SVar$ and use Σ to denote the signature.

Definition 2. Given a signature Σ , the set of Σ -patterns (or simply patterns) is inductively defined by the following grammar:

$$\varphi ::= x \mid X \mid \sigma \mid \varphi_1 \varphi_2 \mid \perp \mid \varphi_1 \rightarrow \varphi_2 \mid \exists x. \varphi \mid \mu X. \varphi$$

where in $(\mu X. \varphi)$ we require that (φ) is positive in (X) , i.e., (X) is not nested in an odd number of times on the left of an implication $(\varphi_1 \rightarrow \varphi_2)$. Pattern $(\varphi_1 \varphi_2)$ is called application and assumed associative to the left. The scope of binders $(\exists)$ and (μ) goes farthest to the right. We employ the notation $(\varphi[\psi/x])$ for capture-avoiding substitution and denote the set of free (element or set) variables of (φ) as $FV(\varphi)$. In addition, a pattern context C is defined as a pattern containing a distinguished variable represented by $\square$. The notation $(C[\varphi])$ indicates the substitution of $\square$ with (φ) within the context C .

Grigore et al. argue that the above syntax is minimal while still being useful and expressive to formalize language semantics and program properties [9]. Element vari-

ables, akin to their counterparts in first-order logic, range over individual elements which can be quantified using the existential quantifier ($\exists$) to express propositions about the existence of specific elements. Set variables, on the other hand, function similarly to propositional variables in modal logic [9]. They range over sets of elements, which can also be interpreted as predicates. Notably, set variables can be quantified using the (μ) operator to create least fixpoint.

A cornerstone of ML's elegance is the concept of symbols. These symbols provide a unified approach to representing various functionalities within the logic. Functions, predicates, constructors, and even modal operators can all be represented as symbols. This uniformity simplifies the syntax and eliminates the need for separate constructs for each type of functionality. The power of symbols is further amplified by the application construct. This construct allows us to combine symbols with patterns (represented by variables, other symbols, or even combinations of these) to create more complex patterns.

Finally, we have the usual ($\perp$) and ($\varphi_1 \rightarrow \varphi_2$) to build arbitrary logical constraints. We define the following constructs as syntactic sugar in the usual way, and we assume the standard precedence among logical connectives, where application binds the tightest.

$$\begin{array}{ll}
 \neg\varphi \equiv \varphi \rightarrow \perp & \varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \rightarrow \varphi_2 \\
 \varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2) & \top \equiv \neg\perp \\
 \varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) & \forall x.\varphi \equiv \neg\exists x.\neg\varphi \\
 \nu X.\varphi \equiv \neg\mu X.\neg\varphi[\neg X/X] &
 \end{array}$$

We draw attention to the fact that the previous constructs are syntactic sugar, i.e., notation, and as such, they may be completely removed without extending the logic. Furthermore, it's critical to remember that these constructs are not symbols. As an illustration, let us take the following pattern ($\mu X.0 \vee s X$) which defines all natural numbers using the zero (0) and the successor symbol (s) along with least fixpoint (μ) as we will define later on 3.2.2.

2.2 Semantics

Within ML, pattern-matching semantics govern the interpretation of patterns over a specified carrier set. Each pattern denotes a set of elements that fulfill the constraints imposed by the pattern. The bottom pattern ($\perp$) intuitively matches no elements, while the top pattern ($\top$) encompasses all elements within the carrier set. Pattern $(\varphi_1 \wedge \varphi_2)$ matched by the elements that match both φ_1 and φ_2 and pattern $(\varphi_1 \vee \varphi_2)$ captures elements that satisfy at least one of the patterns φ_1 or φ_2 . Pattern $(\neg \varphi)$ is matched by the elements that do not match φ .

Element variables, denoted by symbols like (x) , play a crucial role in pattern matching. Each element variable is associated with a specific element within the carrier set through a valuation function (to be formally defined later in definition 4). This valuation essentially assigns a meaning or interpretation to the variable, dictating which element it refers to in a particular context. Set variables, represented by symbols like (X) , also participate in pattern matching. Unlike element variables that match individual elements, set variables match sets of elements. Symbols in ML also correspond to sets, but their interpretation mechanism differs from that of variables. Valuations determine the meaning of variables (both element and set variables) by associating them with specific elements or sets within a given carrier set.

On one hand, the existential quantifier, denoted by $(\exists)$, plays a crucial role in performing abstractions within ML patterns. Consider a pattern of the form $(\exists x. \varphi)$. This pattern successfully matches elements that satisfy the sub-pattern (φ) under some valuation for the element variable (x) .

Next, we recall the standard semantics of ML patterns [1].

Definition 3. An ML structure $(M, _ \bullet _, \{M_\sigma\}_{\sigma \in \Sigma})$ has:

- a nonempty set M , called *domain*,
- a binary *application* function $_ \bullet _ : M \times M \rightarrow \mathcal{P}(M)$,
- a subset $M_\sigma \subseteq M$ called *interpretation* of $\sigma \in \Sigma$.

We use M to denote the above structure and also call it a model.

Definition 4. In order to define the semantics of matching logic patterns, we need to introduce a variable valuation given a signature (Σ) and a model M , which is a

function such that:

$\rho : (EVar \cup SVar) \rightarrow (M \cup \mathcal{P}(M))$ with $\rho(x) \in M$ for all $x \in EVar$, and $\rho(X) \subseteq M$ for all $X \in SVar$.

Definition 5. From this, we can introduce the semantics as follows using a pattern interpretation ($|_|_\rho : \text{PATTERN} \rightarrow \mathcal{P}(M)$):

$$\begin{array}{ll}
 |x|_\rho = \{\rho(x)\} & |X|_\rho = \rho(X) \\
 |\sigma|_\rho = M_\rho & |\perp|_\rho = \emptyset \\
 |\varphi_1 \varphi_2|_\rho = |\varphi_1|_\rho \bullet |\varphi_2|_\rho & |\varphi_1 \rightarrow \varphi_2|_\rho = M \setminus (|\varphi_1|_\rho \setminus |\varphi_2|_\rho) \\
 |\exists x. \varphi|_\rho = \bigcup_{a \in M} |\varphi|_{\rho[a/x]} & |\mu X. \varphi|_\rho = \mu \mathcal{F}_{X, \varphi}^\rho
 \end{array}$$

where $\rho[a/x]$ is the valuation ρ' such that $(\rho'(x) = a)$, $(\rho'(y) = \rho(y))$ for any $(y \in EVar)$ distinct from (x) , and $(\rho'(X) = \rho(X))$ for any $(X \in SVar)$.

Here, $\mathcal{F}_{X, \varphi}^\rho : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$ is the function defined as $(\mathcal{F}_{X, \varphi}^\rho(A) = |\varphi|_{\rho[A/X]})$ for every $(A \subseteq M)$, where $\rho[A/X]$ is the valuation ρ' such that $(\rho'(X) = A)$, $(\rho'(Y) = \rho(Y))$ for any $(Y \in SVar)$ distinct from (X) , and $(\rho'(x) = \rho(x))$ for any $(x \in EVar)$.

An ML pattern can evaluate to the empty set ($\emptyset$), signifying that no elements within the domain satisfy the constraints imposed by the pattern. Conversely, an ML pattern can encompass the entire domain (M), indicating that every element within the domain aligns with the pattern's requirements. ML's expressive power extends even further, allowing patterns to match any subset of the domain that falls between these two extremes. Since the domain (M) is guaranteed to be non-empty by definition, we can establish a clear mapping between elements in the domain and truth values. By convention, we can designate the empty set ($\emptyset$) to represent “logical false” and the entire domain (M) to represent “logical true”.

Formally, a pattern φ in ML is classified as a predicate if, its evaluation denoted by $|\varphi|_\rho$ always results in either the empty set ($\emptyset$) or the entire domain M for all ρ . In essence, predicates in ML represent patterns that adhere strictly to a two-valued interpretation, functioning as logical expressions that can be definitively classified as either true (matching all elements in the domain) or false (matching none).

Proposition 1. The derived constructs defined in section 2.1 have the expected semantics:

$$\begin{aligned}
 |\neg\varphi|_\rho &= M \setminus |\varphi|_\rho & |\varphi_1 \leftrightarrow \varphi_2|_\rho &= M \setminus (|\varphi_1|_\rho \Delta |\varphi_2|_\rho) \\
 |\varphi_1 \wedge \varphi_2|_\rho &= |\varphi_1|_\rho \cap |\varphi_2|_\rho & |\varphi_1 \vee \varphi_2|_\rho &= |\varphi_1|_\rho \cup |\varphi_2|_\rho \\
 |\forall x.\varphi|_\rho &= \bigcap_{a \in M} |\varphi|_{\rho[a/x]} & |\top|_\rho &= M \\
 |\nu X.\varphi|_\rho &= \nu \mathcal{F}_{X,\varphi}^\rho
 \end{aligned}$$

where set symmetric difference $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

Definition 6. The concept of satisfaction forms the cornerstone of the connection between ML structures (models) and patterns (formulas). We say a model M satisfies a pattern φ , denoted by $M \models \varphi$, if, regardless of the specific valuation ρ chosen, $|\varphi|_\rho = M$. Let Γ be a set of patterns. Extending this concept to sets of patterns, we define satisfaction for a set of patterns Γ (often referred to as a theory) as follows: $M \models \Gamma$ holds if and only if $M \models \psi$ is true for every individual pattern $\psi \in \Gamma$. Conversely, we can define the satisfaction relation for a theory Γ concerning a pattern φ as $\Gamma \models \varphi$ holds if and only if, whenever $M \models \Gamma$ is true for a model M , then $M \models \varphi$ must also be true. Following the established terminology, we refer to a set of patterns Γ as a theory, and the individual patterns within Γ are called axioms. Additionally, a model M is considered a model of a theory Γ if it satisfies the entire theory, meaning $M \models \Gamma$.

2.3 Specification of Basic Theories

Here we explore how various essential mathematical tools, including equality, membership, functions, and tuples, can be elegantly expressed within the ML framework. We achieve this by introducing notations and constructing theories that capture the essence of these concepts, but, before we dive into the supporting mathematical instruments, we introduce an important concept in ML that allows us to define any kind of theory, more specifically, the theories that we are formalizing (booleans, natural numbers, sum sort, and product sort). This concept is known as the ML specification.

Definition 7. Formally, a matching logic specification (**SPEC**), consists of a signature (set of symbols), a set of notations, and a set of axioms for the theory. We use the keyword (**spec**) to introduce the specification. Moreover, specifications can import other specifications as we will witness in later examples.

Definition 8. A crucial element in our approach is the introduction of a symbol denoted by $\llbracket _ \rrbracket$. This symbol serves as the definedness symbol. It signifies that the expression within it is well-defined and is matched by at least one element. For convenience, we adopt the shorthand notation $\llbracket \varphi \rrbracket$ instead of $\llbracket _ \rrbracket \varphi$. Also, let (**DEFINEDNESS**) be the axiom $\llbracket x \rrbracket$. We introduce the following important notations:

spec DEFINEDNESS

Symbol: $\llbracket _ \rrbracket$

Notation:

(TOTALITY): $\llbracket \varphi \rrbracket \equiv \neg \llbracket \neg \varphi \rrbracket$

(EQUALITY): $\varphi_1 = \varphi_2 \equiv \llbracket \varphi_1 \leftrightarrow \varphi_2 \rrbracket$ $\varphi_1 \neq \varphi_2 \equiv \neg(\varphi_1 = \varphi_2)$

(MEMBERSHIP): $x \in \varphi \equiv \llbracket x \wedge \varphi \rrbracket$ $x \notin \varphi \equiv \neg(x \in \varphi)$

(INCLUSION): $\varphi_1 \subseteq \varphi_2 \equiv \llbracket \varphi_1 \rightarrow \varphi_2 \rrbracket$ $\varphi_1 \not\subseteq \varphi_2 \equiv \neg(\varphi_1 \subseteq \varphi_2)$

Axiom:

(DEFINEDNESS): $\llbracket x \rrbracket$

endspec

For convenience, we tacitly assume the definedness symbol and the (**DEFINEDNESS**) axiom, as well as all the above notations. It's crucial to emphasize that this approach does not constitute an extension of the core ML framework. Instead, we are introducing an additional symbol and axiom that will be consistently included within all subsequent theories. Definedness and related notations are important in writing axioms and defining theories.

In addition, the concept of definedness ensures that expressions possess significance by ensuring the existence of at least one element in the model that corresponds to a given pattern. Building upon this notion, totality extends the requirement, stipulating that every element within the model must match the specified pattern. Equality, on the other hand, facilitates comparison between two elements within the model, ensuring their match. Membership, defined in terms of an element vari-

able (denoted as x) and a pattern (φ), signifies that any element matching (x) also matches with (φ). Lastly, inclusion is established if all model elements matching (φ_1) also match with (φ_2).

Definition 9. ML is known to be an unsorted logic, which is why, we use a special symbol, called inhabitant, which we denote by $\llbracket _ \rrbracket$ to define any sort s as a distinct ML symbol, and we write $\llbracket s \rrbracket$, which means that this pattern is matched by all elements that have sort s . Therefore, we can introduce a specification for sorts as follows:

spec SORTS

Import: DEFINEDNESS

Symbol: $\llbracket _ \rrbracket$, *Sorts*

Notation:

$$\llbracket s \rrbracket \equiv \llbracket _ \rrbracket s$$

$$\neg_s \varphi \equiv (\neg \varphi) \wedge \llbracket s \rrbracket$$

$$\forall x : s. \varphi \equiv \forall x. (x \in s) \rightarrow \varphi$$

$$\exists x : s. \varphi \equiv \exists x. (x \in s) \wedge \varphi$$

endspec

We present the sort set symbol *Sorts* as well as the inhabitant symbol, which will be used to include all symbols that denote sorts. The notation $(\neg_s \varphi)$ is also known as *sorted negation*, in which all items with sort s that do not match (φ) match the pattern. *sorted quantification* refers to the two remaining notations, in which (x) only varies across the members of sort s .

2.4 Proof System

This section introduces the ML proof system [1]. The system provides a formal framework for reasoning about patterns and properties within ML structures.

The core of the ML proof system lies in the set of proof rules, which are presented in table 2.1. These rules can be broadly categorized into four main groups:

1. First-order logic (FOL) Reasoning: This category provides a complete set of rules for reasoning within first-order logic [10].

2. Frame reasoning: consisting of three propagation rules and one framing rule, which allow us to propagate formal reasoning that is carried out within an application context throughout the context [2].
3. Fixpoint reasoning: It contains three rules for fixpoint reasoning [11], where (Knaster-Tarski), also known as Park induction [12], is a logical incarnation of the Knaster-Tarski fixpoint theorem into ML and is the key to reasoning about inductively-defined data structures and dynamic properties such as program reachability.
4. Technical rules: The final category comprises two technical rules that are essential for establishing specific completeness results within the ML proof system, as detailed in [13].

This proof system defines the provability relation in the ML, denoted by $(\Gamma \vdash \varphi)$. It indicates that the pattern (φ) can be proved with the Hilbert-style proof system of table 2.1 the system, utilizing the patterns in the set (Γ) as additional axioms alongside the existing proof rules.

Category	Proof rule name	Proof rule
FOL Reasoning	Tautology	φ if φ is a propositional tautology over patterns
	Modus Ponens	$\frac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
	$\exists$ -Quantifier	$\varphi[y/x] \rightarrow \exists x. \varphi$
	$\exists$ -Generalization	$\frac{\varphi_1 \rightarrow \varphi_2}{(\exists x. \varphi_1) \rightarrow \varphi_2} \text{ if } x \notin FV(\varphi_2)$
Frame Reasoning	Propagation $_{\perp}$	$C[\perp] \rightarrow \perp$
	Propagation $_{\vee}$	$C[\varphi_1 \vee \varphi_2] \rightarrow C[\varphi_1] \vee C[\varphi_2]$
	Propagation $_{\exists}$	$C[\exists x. \varphi] \rightarrow \exists x. C[\varphi] \text{ if } x \notin FV(C)$
	Framing	$\frac{\varphi_1 \rightarrow \varphi_2}{C[\varphi_1] \rightarrow C[\varphi_2]}$
Fixpoint Reasoning	Substitution	$\frac{\varphi}{\varphi[\psi/X]}$
	Pre-Fixpoint	$\varphi[\mu X. \varphi/X] \rightarrow \mu X. \varphi$
	Knaster-Traski	$\frac{\varphi[\psi/X] \rightarrow \psi}{(\mu X. \varphi) \rightarrow \psi}$
Technical Rules	Existence	$\exists x. x$
	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

Next, we show the basic properties of equality in theories containing definedness.

Proposition 2. For theories Γ with definedness, we have

$\Gamma \vdash \varphi = \varphi$	(REFLEXIVITY)
$\Gamma \vdash \varphi_1 = \varphi_2 \text{ and } \Gamma \vdash \varphi_2 = \varphi_3 \text{ implies } \Gamma \vdash \varphi_1 = \varphi_3$	(TRANSITIVITY)
$\Gamma \vdash \varphi_1 = \varphi_2 \text{ implies } \Gamma \vdash \varphi_2 = \varphi_1$	(SYMMETRY)
$\Gamma \vdash \varphi_1 = \varphi_2 \text{ implies } \Gamma \vdash \psi[\varphi_1/x] = \psi[\varphi_2/x]$	(CONGRUENCE)

2.5 Proof Mode

This section presents the ML proof mode, a significant contribution of the formalization of ML within the Coq proof assistant [2]. This formalization employs a locally nameless encoding for ML [14], and implements the syntax, models, and semantics using a deep embedding approach. The ML proof mode plays a central role in our theorem proofs.

The core design philosophy behind the ML proof mode prioritizes user-friendliness for interactive ML reasoning within Coq. This is achieved by establishing a clear separation between the ML-proof state and the underlying Coq-proof state. This separation empowers users to focus on the essential structure of their ML proofs, fostering a more intuitive and manageable proof development process. The current proof context is designed to enhance clarity and simplify navigation during proof development. It presents a collection of named hypotheses alongside a single goal pattern. It allows users to easily track the assumptions made (local context) and the statement they are trying to prove (goal pattern). The structure is further divided into the following key components:

1. A meta-level context
2. A global matching logic context
3. A local matching logic context
4. A matching logic goal

We can point to the previous components in the example 2.1.

```

1 1 goal
2  $\Sigma$ : Signature
3  $\Gamma$ : Theory
4  $\varphi$  : Pattern
5 i : ProofInfo
6 H : well_formed  $\varphi$ 
7 H0 :  $\Gamma \vdash i \varphi$  using i
8 ----- (1/1)
9  $\Gamma \vdash$ 
10 "H0":  $\varphi$ ,
11 -----
12 Top

```

Code 2.1: Example of a theorem proof in Coq

We have the meta-level context from line 2 to line 7 where you can have for example hypotheses on the well-formedness of patterns which is required by the locally-nameless approach the Coq embedding uses. Next, we have the global matching logic context on line 9 which is the theory (set of axioms) itself. Line 10 refers to the local matching logic context where we have a list of patterns identified by names, which are the local hypotheses, and everything below line 11, it is our matching logic goal.

In addition to this structured context, the ML proof mode includes several specialized Coq tactics designed for manipulating the ML proof state. These tactics go beyond simply mimicking Coq's functionalities. They provide operations specifically tailored for common ML reasoning tasks. Some examples include tactics to enter and exit the proof mode, perform equational reasoning within ML patterns, and apply lifted derived rules from the ML logic to the current proof state. More comprehensive details on how this proof mode was designed and implemented can be found in [15]. We show an illustration of the interactive proof mode using some proof mode tactics, we have the current proof state in example 2.2.

```

1 1 goal
2  $\Sigma$ : Signature
3  $\Gamma$ : Theory
4  $\varphi$  : Pattern
5 i : ProofInfo
6 H : well_formed  $\varphi$ 
7 H0 :  $\Gamma \vdash i \varphi$  using i
8 ----- (1/1)
9  $\Gamma \vdash$ 
10  $\varphi \dashrightarrow$  Top using i

```

Code 2.2: Example of a matching logic proof state

After introducing the implication (i.e. moving the left-hand side of the implication in the goal to the local matching logic context) by applying the tactic `mlIntro` we obtain the proof state in example 2.1.

Chapter 3

Formalization of ML Theories

This section focuses on the core of our research: formalizing Matching Logic (ML) theories within the Coq proof assistant. We make use of already-developed tools from earlier ML formalization efforts in Coq [2]. These resources consist of pre-established theorems, symbols, notations, and definitions. These well-established components allow us to describe ML theories in the Coq environment in an efficient manner. For simplicity, we present these theories using specifications that consist of a set of symbols, a set of notations, and a theory (set of axioms). This formalization lays the groundwork for the next step in our research; Stating and proving theorems that we present for each theory.

3.1 Supporting Theorems for Theories

While proving theorems within diverse theories, we encountered the necessity of auxiliary theorems to bridge gaps in the proof process. By introducing and proving these helper theorems, we were able to establish foundational results that paved the way for more intricate proofs within the theory. We present these theorems as follows:

Theorem 1. *We introduce the (TOPHOLDS) lemma, which states that $(\top)$ holds.*

$$\Gamma \vdash \top \quad (\text{TOPHOLDS})$$

Theorem 2. *This theorem states that if a pattern (φ) is provable that implies that*

(φ) is equivalent to top ($\top$).

$$\Gamma \vdash \varphi \rightarrow \Gamma \vdash \varphi \leftrightarrow \top$$

Proof. The proof idea is simple, first, we need to split the equivalence into two implications (i.e. $\varphi \rightarrow \top$ and $\top \rightarrow \varphi$), for the second implication, it is enough to introduce it and finish the proof by using the hypothesis (φ holds). $\square$

Theorem 3. *This theorem defines the usual conjunction with patterns (φ and $\top$).*

$$\Gamma \vdash \varphi \wedge \top \leftrightarrow \varphi$$

Proof. For the proof, we use the simple FOL reasoning. Firstly, by splitting the equivalence we obtain:

$$\begin{aligned} \Gamma \vdash \varphi \wedge \top &\rightarrow \varphi \\ \Gamma \vdash \varphi &\rightarrow \varphi \wedge \top \end{aligned}$$

we introduce the first implication, then we destruct the conjunction that creates two local hypotheses (φ and $\top$), and we finish the proof of the subgoal by assumption. We apply the same chain of thoughts for the second subgoal. $\square$

Theorem 4. *This theorem states that if we have inclusion in both directions between two patterns, here (φ and ψ), then these are equal.*

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

Proof. The general idea of the proof is to use the inclusion notation in (DEFINEDNESS) specification to unfold the ($\subseteq$), so we start by posing the (TOTALAND) lemma that states the following:

$$\Gamma \vdash \lfloor \varphi_1 \wedge \varphi_2 \rfloor \leftrightarrow \lfloor \varphi_1 \rfloor \wedge \lfloor \varphi_2 \rfloor \quad (\text{TOTALAND})$$

in addition, we know that:

$$\Gamma \vdash \varphi = \psi \leftrightarrow \lfloor \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi \rfloor$$

Therefore, by letting ($\varphi_1 = \varphi \rightarrow \psi$), and ($\varphi_2 = \psi \rightarrow \varphi$), and since equivalence is

commutative, we obtain our goal:

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

□

Theorem 5. *This theorem defines the propagation of the subset over the disjunction.*

$$\Gamma \vdash (\varphi_1 \vee \varphi_2) \subseteq \psi \leftrightarrow \varphi_1 \subseteq \psi \wedge \varphi_2 \subseteq \psi$$

Proof. In the same way, we use the (TOTALAND) lemma to prove this theorem by specializing it with the correct patterns; $(\varphi_1 \rightarrow \psi)$ as (φ_1) and $(\varphi_2 \rightarrow \psi)$ as (φ_2) in (TOTALAND). So, we have the following:

$$\Gamma \vdash [(\varphi_1 \rightarrow \psi) \wedge (\varphi_2 \rightarrow \psi)] \leftrightarrow [\varphi_1 \rightarrow \psi] \wedge [\varphi_2 \rightarrow \psi] \quad (3.1)$$

After unfolding $(\subseteq)$, we rewrite by (3.1) in 5 to replace the right-hand side of the statement above and splitting the equivalence into two implications we get two subgoals:

$$\begin{aligned} \Gamma \vdash [\varphi_1 \vee \varphi_2 \rightarrow \psi] &\rightarrow [(\varphi_1 \rightarrow \psi) \wedge (\varphi_2 \rightarrow \psi)] \\ \Gamma \vdash [(\varphi_1 \rightarrow \psi) \wedge (\varphi_2 \rightarrow \psi)] &\rightarrow [\varphi_1 \vee \varphi_2 \rightarrow \psi] \end{aligned}$$

We prove the first subgoal and the same chain of thoughts is applied for the second one. In the first subgoal, we apply the (FLOORMONOTONIC) lemma which is defined below:

$$\Gamma \vdash (\varphi_1 \rightarrow \varphi_2) \rightarrow [\varphi_1] \rightarrow [\varphi_2] \quad (\text{FLOORMONOTONIC})$$

Then, we move the left-hand side of the implication of the first subgoal into the local matching logic (i.e. introducing the implication), and we obtain:

$$\Gamma \vdash (\varphi_1 \rightarrow \psi) \wedge (\varphi_2 \rightarrow \psi)$$

Next, we introduce the conjunction in the proof. This creates two separate subgoals, which can be proven using similar reasoning steps. We perform another introduction and apply the local hypothesis that we get from the first implication introduction $(\varphi_1 \vee \varphi_2 \rightarrow \psi)$ in our current goal which is (ψ) ; From that, we get $(\varphi_1 \vee \varphi_2)$, we chose the left-hand side of the disjunction. Finally, we finish the proof by assumption since we have a similar local hypothesis that we obtained from the second implication

introduction (φ_1) as the current goal. $\square$

Theorem 6. *This theorem introduces the totality of patterns, such that if a pattern is total, then is equal to top ($\top$).*

$$\Gamma \vdash \lfloor \varphi \rfloor \leftrightarrow (\varphi = \top)$$

Proof. We begin the proof as usual, by splitting the equivalence into two parts, which we prove separately:

$$\Gamma \vdash \lfloor \varphi \rfloor \rightarrow \varphi = \top \quad (3.2)$$

$$\Gamma \vdash \varphi = \top \rightarrow \lfloor \varphi \rfloor \quad (3.3)$$

For the first subgoal (3.2), we unfold the equality, apply the (*FLOORMONOTONIC*), implication and conjunction introduction to get two other subgoals:

$$\Gamma \vdash \varphi \rightarrow \top \quad (3.4)$$

$$\Gamma \vdash \top \rightarrow \varphi \quad (3.5)$$

we introduce the implication for both (3.4 and 3.5), then finally, we finish the proof with (*TOPHOLDS*) lemma for (3.4) and assumption for (3.5). For the second subgoal (3.3), we unfold the equality, apply the (*FLOORMONOTONIC*), introduce the implication, destruct the conjunction in the local hypothesis, and apply the second local hypothesis ($\top \rightarrow \varphi$) to our goal, then we finish the proof by assumption with the (*TOPHOLDS*) lemma. $\square$

3.2 Simple Theories

We start with the basic types such as Boolean values and natural numbers.

3.2.1 Booleans Theory

We start by defining a set of symbols for the theory, which consists of the following:

- *Bool*: a symbol to define the sort/type,

- tt, ff : the two symbols are used to represent the usual boolean values (i.e true, and false),
- $!$: a symbol for the negation operator,
- $\&$: a symbol for the conjunction operator (i.e logical and operator),
- $-\&$: a symbol to introduce the short-circuit conjunction operator, this operator evaluates the left-hand side expression. If the result is false, then false is immediately returned. If the result is non false then the right-hand side is evaluated, and the logical result from that is returned.

Adding to that, we define a set of notations, that states that following the usual way of writing conjunction using the $(\&, -\&)$ operator between (φ_1) and (φ_2) is equivalent to applying $(\&, -\&)$ to both patterns. Finally, the specification includes a set of axioms that define the *Bool* sort, where the (INDUCTIVE DOMAIN) axiom clearly says that this sort has two constant values (tt) and (ff) ; From the (NO CONFUSION) axiom, we understand that there is no ambiguity between the two constructors (tt) and (ff) , and they are two distinct entities; The remaining axioms introduce rules that capture essential properties of these operators, such as the idempotence of conjunction, and rewriting rule for the negation operator. We point to the absence of the sorts in both (AXDEFANDTHENRIGHTFALSE) and (AXDEFANDTHENLEFTFALSE) axioms.

spec BOOL

Import: SORTS

Symbol: *Bool*, *tt*, *ff*, *!*, *&*, *-&*

Notation:

$$\varphi_1 \& \varphi_2 \equiv \& \varphi_1 \varphi_2$$

$$\varphi_1 \text{-}\& \varphi_2 \equiv \text{-}\& \varphi_1 \varphi_2$$

Axiom:

$$(\text{INDUCTIVE DOMAIN}): \quad \llbracket \text{Bool} \rrbracket = tt \vee ff$$

$$(\text{NO CONFUSION}): \quad tt \neq ff$$

$$(\text{AXDEFNEGTRUE}): \quad !tt = ff$$

$$(\text{AXDEFNEGFALSE}): \quad !ff = tt$$

$$(\text{AXDEFANDRIGHTTRUE}): \quad \forall x : \text{Bool}. x \& tt = x$$

$$(\text{AXDEFANDRIGHTFALSE}): \quad \forall x : \text{Bool}. x \& ff = ff$$

$$(\text{AXDEFANDLEFTTRUE}): \quad \forall x : \text{Bool}. tt \& x = x$$

$$(\text{AXDEFANDLEFTFALSE}): \quad \forall x : \text{Bool}. ff \& x = ff$$

$$(\text{AXDEFANDTHENRIGHTTRUE}): \quad \forall x : \text{Bool}. x \text{-}\& tt = x$$

$$(\text{AXDEFANDTHENRIGHTFALSE}): \quad \forall x. x \text{-}\& ff = ff$$

$$(\text{AXDEFANDTHENLEFTTRUE}): \quad \forall x : \text{Bool}. tt \text{-}\& x = x$$

$$(\text{AXDEFANDTHENLEFTFALSE}): \quad \forall x. ff \text{-}\& x = ff$$

endspec

Formalization Now, we show how to formalize this specification in Coq. it applies to all remaining theories.

For defining the set of symbols, we declare an inductive type named **Symbols** that belongs to the type class **set**, where the **Inductive** command allows defining types by cases on the form of the inhabitants of the type, we need just to introduce the constructors as follows:

```

1 Inductive Symbols: Set :=
2 | sBool
3 | sTrue
4 | sFalse
5 | sAnd
6 | sNeg
7 | sAndThen.
    
```

After defining the symbols, the next step is to inject them into the signature using the (*inj*) class member. Definitions extend the global environment by associating names with terms, it can be seen as a way to give meaning to a name. From the Coq code below, we can divide these definitions into two categories, in one hand, we define patterns from symbols using the `patt_sym` function. For example, we construct the `mlTrue` that represents the boolean constant `True` from the `sTrue` symbol. On another hand, we define functions for operators (conjunction and negation) as functions that have patterns as argument and as a return value. As an illustration, the function `mlBAnd` which refers to the normal logical conjunction, is defined by the application of the conjunction symbol (`&`) to `phi` and `psi`, and similarly for the negation and short-circuit conjunction.

```

1  Definition mlBool      := patt_sym (inj sBool).
2  Definition mlTrue     := patt_sym (inj sTrue).
3  Definition mlFalse    := patt_sym (inj sFalse).
4  Definition mlsBAnd    := patt_sym (inj sAnd).
5  Definition mlsBNeg    := patt_sym (inj sNeg).
6  Definition mlsBAndThen := patt_sym (inj sAndThen).
7
8  Definition mlBAnd ( phi psi : Pattern) : Pattern :=
9    (mlsBAnd $ phi) $ psi.
10 Definition mlBNeg (phi : Pattern) : Pattern :=
11   mlsBNeg $ phi.
12 Definition mlBAndThen ( phi psi : Pattern) : Pattern :=
13   (mlsBAndThen $ phi) $ psi.
14
```

Code 3.1: Symbols definitions for boolean theory

(*inj*) is a member of the `Syntax` class, and it is used to inject the symbols into a matching logic signature. It says, that there are symbols in the signature, and we can access them with for example `inj sTrue`, where `sTrue` is defined above.

```

1 Class Syntax :=
2   { inj : Symbols -> symbols;
3     imported_sorts : Sorts_Syntax.Syntax;
4   }.
5
```

Code 3.2: Class syntax

The following Coq module introduces custom notations to enhance the readability and clarity of ML specifications and proofs. Instead of writing long function applications, we can now use symbols like `&&ml` for conjunction and `!b` for negation. This makes ML expressions easier to read and write, just like using `(+)` instead of a function to add numbers.

```
1 Module Notations.  
2   Notation "phi '&&ml' psi" := (mlBAnd phi psi)  
3     (at level 40, left associativity) : ml_scope.  
4   Notation "'!b' phi" := (mlBNeg phi)  
5     (at level 60) : ml_scope.  
6   Notation "phi 'andThen' psi" := (mlBAndThen phi psi)  
7     (at level 40, left associativity) : ml_scope.  
8 End Notations.  
9
```

Code 3.3: Notations module for boolean theory

We define a new section for the axioms, where we import the notations (defined above in a separate module 3.3) that will be used later to define the axioms. We introduce an inductive type `AxiomName`, which enumerates the names of the axioms for the booleans theory. In fact, each constructor within this inductive type represents a specific axiom, such as `AxDefNegTrue` for the axiom of the negation of `True`, and so on.

```
1 Section axioms.  
2   Import Notations.  
3   Open Scope ml_scope.  
4   Context  
5     {Σ : Signature}  
6     {syntax : Syntax}.  
7  
8   Inductive AxiomName :=  
9     | AxFunTrue  
10    | AxFunFalse  
11    | AxFunAnd  
12    | AxFunNeg  
13    | AxNoConfusion  
14    | AxInductiveDomain  
15    | AxDefNegTrue
```

```

16 | AxDefNegFalse
17 | AxDefAndRightTrue
18 (*..... *)
19 .
20

```

Code 3.4: Axioms section for boolean theory

The below Coq code shows how we define each axiom from the booleans specification. The `axiom` function takes an `AxiomName` as input, which identifies the specific axiom being defined. Based on this name, the function constructs a corresponding pattern that captures the essence of the axiom. Moreover, we use a straightforward representation while defining these axioms for bound variables, known as de Bruijn encoding. This approach was used in [2] to define the syntax of ML, and it is known in the literature as *locally nameless syntax* that relies on this encoding. We refer the reader to [2, 14, 16, 17, 18] for more details.

```

1 Definition axiom (name : AxiomName) : Pattern :=
2   match name with
3   | AxFunTrue => ex mlBool , mlTrue =ml b0
4   | AxFunFalse => ex mlBool , mlFalse =ml b0
5   | AxFunAnd =>
6     all mlBool , all mlBool , ex mlBool , b1 &&ml b2 =ml b0
7   | AxFunNeg => all mlBool , ex mlBool , !b b1 =ml b0
8   | AxNoConfusion => !(mlTrue =ml mlFalse)
9   | AxInductiveDomain => [| mlBool |] =ml (mlTrue or mlFalse)
10  | AxDefNegTrue => !b mlTrue =ml mlFalse
11  | AxDefNegFalse => !b mlFalse =ml mlTrue
12  | AxDefAndRightTrue =>
13    all mlBool, b0 &&ml mlTrue =ml b0
14    (*..... *)
15    .
16

```

Code 3.5: Axioms section for booleans theory

We follow the same approach to formalize the syntax of the remaining theories in Coq. The implementation of the theory in Coq can be found in [19].

Proved Theorems

Theorem 7. *This theorem states that double negation (applying the negation symbol “!” twice) on a boolean variable (x) has no effect and leaves the variable unchanged. In other words, regardless of the initial truth value of x (tt for true or ff for false), applying negation twice flips the value back to its original state.*

$$\Gamma \vdash \forall x : Bool. !(!x) = x$$

Proof. We start proving this theorem with an introduction of the universal quantifier, and a simplification; We pose the (INDUCTIVE DOMAIN), and we rewrite with it in the hypothesis that we obtain after introducing the implication in the goal. From that, we eliminate the disjunction and obtain two sub-goals to prove with two hypotheses:

$$\begin{array}{ll} x = tt & \text{(hypothesis 1)} \\ x = ff & \text{(hypothesis 2)} \end{array}$$

- **Case 1:** We have $(!(!x) = x)$; From (hypothesis 1), we obtain $(!(!tt) = tt)$, we rewrite by (AXDEFNEGTRUE) axiom; $(!(ff) = tt)$, then by (AXDEFNEGFALSE), and hence $(tt = tt)$ which is our goal.
- **Case 2:** We apply the same chain of thoughts with (hypothesis 2), but we use the (AXDEFNEGFALSE) axiom, then (AXDEFNEGTRUE) axiom to rewrite, and arrive to $(ff = ff)$

□

Theorem 8 and **Theorem 9** deal with conjunction (&) “logical and”. These theorems tell us that in the Booleans theory, applying the “logical and” to (tt) “logical true” and any boolean variable (x) preserves the value of (x), however, applying the “logical and” to (ff) “logical false” and any boolean variable (x) results always in (ff).

Theorem 8.

$$\Gamma \vdash \forall x : Bool. x \& tt = x \tag{3.6}$$

Theorem 9.

$$\Gamma \vdash \forall x : Bool. x \& ff = ff$$

Proof. They are a direct consequence from (AXDEFANDRIGHTTRUE), and (AXDEFANDRIGHTFALSE) respectively.

As an example, we show the proof of **theorem 8**, we have $(\forall x : Bool.x \& tt = x)$; By the introduction of the universal quantifier we obtain $(x \& tt = x)$; We pose the (AXDEFANDRIGHTTRUE) axiom as a local hypothesis ... (1), and we specialize it with the variable (x) that we introduced previously; We finish the proof by using (1) which is the same as the goal. $\square$

Theorem 10. *The commutative property of conjunction ($\&$) in Booleans theory is the subject of the **theorem 10**. The commutative property says that, if the operation is commutative, the order in which we carry out a certain operation on two operands has no bearing on the result.*

$$\Gamma \vdash \forall x, y : Bool.x \& y = y \& x$$

Proof. Equivalently, we introduce the universal quantifier for (x) and (y) , simplify the statement, and introduce the implication from the goal; We rewrite in the hypotheses by the (INDUCTIVE DOMAIN) for both variables, therefore, we need to prove four cases separately as follows:

- **Case 1:** We have $(x = tt)$ and $(y = tt)$; Then $tt \& tt = tt \& tt$ holds based on the (REFLEXIVITY) of the equality.
- **Case 2:** We have $(x = tt)$, $(y = ff)$; Then $(tt \& ff = ff \& tt)$; We can start by introducing a hypothesis $(ff \& tt = ff)$ (1) which follows from specializing the (AXDEFANDLEFTTRUE) with $(y = ff)$. We rewrite the right-hand side of our goal using (1), and we obtain $(tt \& ff = ff)$. Then we pose the (AXDEFANDLEFTTRUE) axiom, we eliminate the universal quantifier by specializing it with $(y = ff)$, and finally apply it.
- **Case 3:** $(x = ff)$, $(y = tt)$; Then $(ff \& tt = tt \& ff)$; Similar to the second case, we introduce $(tt \& ff = ff)$ (2) as a hypothesis, by rewriting the right-hand side of the equality with (2), we obtain $(ff \& tt = ff)$, which holds from applying the (AXDEFANDLEFTFALSE) axiom after specializing it with $(y = tt)$.
- **Case 4:** We have $(x = ff)$, $(y = ff)$; Then by rewriting, we obtain $(ff \& ff = ff \& ff)$ which holds based on the (REFLEXIVITY) of the equality.

$\square$

3.2.2 Natural Numbers Theory

We present here the second specification for the natural numbers in the same way. We have a set of symbols that contains a symbol for the type, a symbol to denote the pattern zero (0), a symbol for the successor construct (s), and a symbol for the addition operator ($+$). Similarly, we have a notation for the addition operator, and a set of axioms, where the (INDUCTIVE DOMAIN) axiom states that $\llbracket Nat \rrbracket$ is the smallest set built from (0) and (s) applied to a natural number, which are the only two constructors of (Nat). The (NO CONFUSION) axioms specify that (0) is not the successor of any natural numbers and two numbers are equal only if the successors of these two numbers are equal. Finally, the (AXFUNADD) axiom states that for any two natural numbers you add, there is always a unique result that is also a natural number; The (AXDEFADDID) captures the identity property of zero for addition; The last axiom says that adding a natural number (x) to the successor of another natural number (y) (written as ($s\ y$)) is equal to the successor of adding (x) to (y) (written as $s\ (x + y)$).

```

spec NAT
  Import: SORTS
  Symbol: Nat, s, 0, +
  Notation:  $\varphi_1 + \varphi_2 \equiv +\varphi_1 \varphi_2$ 
  Axiom:
    (INDUCTIVE DOMAIN):       $\llbracket Nat \rrbracket = \mu X. 0 \vee s\ X$ 
    (NO CONFUSION1):          $\forall x : Nat. 0 \neq s\ x$ 
    (NO CONFUSION2):          $\forall x, y : Nat. s\ x = s\ y \rightarrow x = y$ 
    (AXFUNADD):                $\forall x, y : Nat, \exists z : Nat. x + y = z$ 
    (AXDEFADDID):              $\forall x : Nat. x + 0 = x$ 
    (AXDEFADD):                $\forall x, y : Nat. x + s\ y = s\ (x + y)$ 
endspec
    
```

The implementation of the theory in Coq can be found here [20].

Proved Theorems

Theorem 11. *This theorem articulates a fundamental property of addition for natural numbers, commonly known as the identity property of addition from the right side. It asserts that adding zero to any natural number (x) results in the number itself.*

$$\Gamma \vdash \forall x : \text{Nat}. x + 0 = x$$

Proof. **Theorem 11** is a direct consequence of the pre-defined axiom (AXDEFADDID) of the Nat theory.

we have $(\forall x : \text{Nat}. x \& x + 0 = x)$; By introduction of the universal quantifier we obtain $(x + 0 = x)$; We pose the (AXDEFADDID) axiom as a local hypothesis ... (1), and we specialize it with the variable (x) that we introduced previously; Then we simplify the substitutions, and finally, we finish the proof by using (1) which is the same as the goal. $\square$

Theorem 12. *Similarly to the previous theorem 11, it defines the identity property of addition on the left side.*

$$\Gamma \vdash \forall x : \text{Nat}. 0 + x = x$$

Proof. For **theorem 12**, the proof idea would be to use the Peano Induction theorem [1].

$$\Gamma \vdash 0 \in X \wedge (\forall y : \text{Nat}. y \in X \rightarrow (s y) \in X) \rightarrow \forall x : \text{Nat}. x \in X \quad (\text{Peano Induction})$$

Therefore, all instances of (Peano Induction) can then be proved from the pattern theorem, by letting $(\psi \equiv \exists z : \text{Nat}. z \wedge \varphi(z))$ be the pattern of all (z) such that $(\varphi(z))$ holds [13]. Here, the pattern $(\varphi(z) = 0 + z = z)$. In this case, our initial pattern $(\varphi(x))$ “goal” holds, if and only if we can prove that $(x \in \psi)$ which we obtain by using the set variable substitution rule, where (ψ) is substituted for (X) , and we get:

$$\begin{aligned} \Gamma \vdash 0 \in (\exists z : \text{Nat}. z \wedge \varphi(z)) \wedge \\ (\forall y : \text{Nat}. y \in (\exists z : \text{Nat}. z \wedge \varphi(z)) \rightarrow (s y) \in (\exists z : \text{Nat}. z \wedge \varphi(z))) \\ \rightarrow \forall x : \text{Nat}. x \in (\exists z : \text{Nat}. z \wedge \varphi(z)) \quad (3.7) \end{aligned}$$

In general, the proof idea is to assume two assertions (basic and induction case) and prove them separately:

$$\Gamma \vdash 0 \in (\exists z : \text{Nat}. z \wedge 0 + z = z) \quad (3.8)$$

$$\Gamma \vdash (\forall y : \text{Nat}. y \in (\exists z : \text{Nat}. z \wedge 0 + z = z) \rightarrow (sy) \in (\exists z : \text{Nat}. z \wedge 0 + z = z)) \quad (3.9)$$

We start with **assertion 3.8**, we pose the helper lemma (MEMBERSHIPEXISTS2) with the adequate pattern, apply the universal generalization (i.e. UNIVERSALGENERALIZATION), and an all functional substitution (i.e. ALLFUNCTIONALSUBST) where we substitute (x) with the pattern Zero (0).

$$\Gamma \vdash (\exists x. x \in \varphi) \rightarrow x \in (\exists. \varphi) \quad (\text{MEMBERSHIPEXISTS2})$$

$$\Gamma \vdash \varphi \text{ implies } \Gamma \vdash \forall x. \varphi \quad (\text{UNIVERSALGENERALIZATION})$$

$$\Gamma \vdash (\exists x. \psi = x) \rightarrow (\forall. \varphi) \rightarrow \varphi[x/\psi] \quad (\text{ALLFUNCTIONALSUBST})$$

So, we obtain after sorted and simple simplification:

$$\Gamma \vdash (\exists z : \text{Nat}. 0 \in (z \wedge 0 + z = z)) \rightarrow 0 \in (\exists z : \text{Nat}. z \wedge 0 + z = z) \quad (3.10)$$

We apply (3.10) in (3.8). Finally, we finish the proof by using the exists functional substitution to prove the left-hand side of (3.10), which results in proving a side condition of the pattern (0) being a functional pattern (i.e. the pattern can be matched by exactly one element); It is trivial and can be proven using the (INDUCTIVE DOMAIN) axiom of the theory.

We use the same proof steps for the **assertion 3.9**. Now we dive into proving the main goal 12, first, we apply (3.8) and 3.9 in (3.7), then we introduce the universal quantifier, and specialize (3.8) with x . We simplify the substitutions to obtain:

$$\Gamma \vdash x \in (\exists z : \text{Nat}. z \wedge 0 + z = 0)$$

We introduce the **hypothesis 3.11**, and apply it in the previous state where we substitute the pattern $(\exists z : \text{Nat}. z \wedge 0 + z = 0)$ with x . To conclude, we destruct the conjunction and complete the proof by assumption from the local context.

$$\Gamma \vdash \varphi \in (\exists x. \psi) \rightarrow \psi[x/\varphi] \quad (3.11)$$

□

3.3 Parametric Theories

In this second part, we delve into the parametric types. A parameterized type (sort) is one that relies on other type values.

3.3.1 Sum Sort Theory

Starting with the sum sort theory, where given two sorts (s_1) and (s_2) we define a new sort $(s_1 \oplus s_2)$, called the sum(coproduct) of (s_1) and (s_2) .

```

spec SUM  $\{s_1, s_2\}$ 
  Import: SORTS
  Symbol:  $\oplus, \iota_1, \iota_2, \epsilon_1, \epsilon_2$ 
  Notation:  $s_1 \oplus s_2 \equiv \oplus s_1 s_2$ 
  Axiom:
    (INJECT LEFT):       $\forall x : s_1, \exists y : s_1 \oplus s_2, \iota_1(x) = y$ 
    (INJECT RIGHT):      $\forall x : s_2, \exists y : s_1 \oplus s_2, \iota_2(x) = y$ 
    (EJECT LEFT):        $\forall x : s_1 \oplus s_2, \exists y : s_1, \epsilon_1(x) \subseteq y$ 
    (EJECT RIGHT):       $\forall x : s_1 \oplus s_2, \exists y : s_2, \epsilon_2(x) \subseteq y$ 
    (INVERSE INJEJ1):    $\forall x : s_i, \epsilon_i(\iota_i x) = x, i = 1, 2$ 
    (INVERSE INJEJ2):    $\forall x : s_{3-i}, \epsilon_i(\iota_{3-i} x) = \perp, i = 1, 2$ 
    (COPRODUCT):         $\llbracket s_1 \oplus s_2 \rrbracket \subseteq (\iota_1 \llbracket s_1 \rrbracket) \vee (\iota_2 \llbracket s_2 \rrbracket)$ 
endspec
    
```

The implementation of the theory in Coq can be found here [21].

Proved Theorems

Theorem 13 and **theorem 14** state that the functions (INJECT_) are injectives.

Theorem 13.

$$\Gamma \vdash \forall x, y : s_1. \iota_1(x) = \iota_1(y) \rightarrow x = y$$

Theorem 14.

$$\Gamma \vdash \forall x, y : s_2. \iota_2(x) = \iota_2(y) \rightarrow x = y$$

Proof. We take as an example of proof **theorem 14**.

First of all, we introduce both (x) and (y) with the elimination of the universal quantifier, after performing one sorted simplification, one simplification, and introducing the implication from the goal, we have as a hypothesis $(\iota_2(x) = \iota_2(y))$ (1),

we need to prove $(x = y)$.

A first step would consist of posing the (INVERSE INJEJ1) axiom with $(i = 2)$ and specializing it with (x) to obtain $(\epsilon_2(\iota_2 x) = x)$ (2); We rewrite in (2) by (1), so $(\epsilon_2(\iota_2 y) = x)$; To facilitate the rewrite in the goal, we perform (mlSymmetry) in (2), and thus $(\epsilon_2(\iota_2 y) = y)$. As a last step of the proof, we repose the (INVERSE INJEJ1) axiom with $(i = 2)$, and specialize it this time with (y) to obtain $(\epsilon_2(\iota_2 y) = y) \dots$ (3); We complete the proof by assumption since the goal is identical to our local hypothesis (3). $\square$

Theorem 15. *The following theorem states that any pattern of the sum type $(s_1 \oplus s_2)$ is either constructed from $(\iota_1 \llbracket s_1 \rrbracket)$ or $(\iota_2 \llbracket s_2 \rrbracket)$.*

$$\Gamma \vdash \llbracket s_1 \oplus s_2 \rrbracket = (\iota_1 \llbracket s_1 \rrbracket) \vee (\iota_2 \llbracket s_2 \rrbracket)$$

Proof. For **theorem 15**, we need to show $((\iota_1 \llbracket s_1 \rrbracket) \vee (\iota_2 \llbracket s_2 \rrbracket) \subseteq \llbracket s_1 \oplus s_2 \rrbracket)$, therefore from the (COPRODUCT) axiom and this assertion, we can prove the theorem using the helper theorem 4. First, let's assume and prove the following assertion separately:

$$\iota_1 \llbracket s_1 \rrbracket \vee (\iota_2 \llbracket s_2 \rrbracket) \subseteq \llbracket s_1 \oplus s_2 \rrbracket \quad (3.12)$$

By applying the helper theorem 5 to (3.12) we get :

$$(\iota_1 \llbracket s_1 \rrbracket) \subseteq \llbracket s_1 \oplus s_2 \rrbracket \wedge (\iota_2 \llbracket s_2 \rrbracket) \subseteq \llbracket s_1 \oplus s_2 \rrbracket$$

We introduce the conjunction of the main goal, by creating two subgoals which we prove separately. To seek simplicity, we prove only the first subgoal, and the same chain of thoughts is applied to the second one.

$$(\iota_1 \llbracket s_1 \rrbracket) \subseteq \llbracket s_1 \oplus s_2 \rrbracket \quad (3.13)$$

$$(\iota_2 \llbracket s_2 \rrbracket) \subseteq \llbracket s_1 \oplus s_2 \rrbracket \quad (3.14)$$

Then we introduce the **hypothesis 3.15** into the local matching logic context of the current proof state,

$$\Gamma \vdash \varphi \subseteq \psi \leftrightarrow (\forall x. x \in \varphi \rightarrow x \in \psi) \quad (3.15)$$

And we apply it in (3.13) to obtain:

$$\forall x. (x \in (\iota_1 \llbracket s_1 \rrbracket)) \rightarrow x \in \llbracket s_1 \oplus s_2 \rrbracket$$

Then by rewriting with anti-unif $\exists$ -subst:

$$\forall x. (x \in (\iota_1 \llbracket s_1 \rrbracket)) \rightarrow (\iota_1 x) \in \llbracket s_1 \oplus s_2 \rrbracket$$

Then by sorted quantifier, we get:

$$\forall x. (x \in (\iota_1 \llbracket s_1 \rrbracket)) \rightarrow \exists y. (y \in \llbracket s_1 \oplus s_2 \rrbracket) \wedge (\iota_1 x) = y$$

Which follows directly from (INJECT LEFT) axiom. □

3.3.2 Product Sort Theory

We define a new sort called the product(sort) denoted by $(s_1 \otimes s_2)$ that depends on two other type values (s_1) and (s_2) .

```

spec PROD  $\{s_1, s_2\}$ 
  Import: SORTS
  Symbol:  $\otimes, \langle \_, \_ \rangle \pi_1, \pi_2,$ 
  Notation:
     $s_1 \otimes s_2 \equiv \otimes s_1 s_2$ 
     $\langle x, y \rangle \equiv \langle \_, \_ \rangle x y$ 
  Axiom:
    (PAIR):  $\forall x : s_1, \forall y : s_2, \exists z : s_1 \otimes s_2, \langle x, y \rangle = z$ 
    (PROJECT LEFT):  $\forall x : s_1 \otimes s_2, \exists y : s_1, \pi_1(x) = y$ 
    (PROJECT RIGHT):  $\forall x : s_1 \otimes s_2, \exists y : s_2, \pi_2(x) = y$ 
    (INJECTION):  $\langle x_1, x_2 \rangle = \langle y_1, y_2 \rangle \rightarrow x_1 = y_1 \wedge x_2 = y_2$ 
    (INVERSE PAIRPROJ1):  $\forall x_1 : s_1, \forall x_2 : s_2, \pi_i \langle x_1, x_2 \rangle = x_i, i = 1, 2$ 
    (INVERSE PAIRPROJ2):  $\forall x : s_1 \otimes s_2, \langle \pi_1 x, \pi_2 x \rangle = x$ 
endspec
    
```

The implementation of the theory in Coq can be found here [22].

Proved Theorems

Theorem 16. *This theorem essentially asserts that every element of a product type can be uniquely constructed with $\langle _, _ \rangle$ applied to its constituent parts from the respective types s_1 and s_2 .*

$$\Gamma \vdash \forall x : s_1 \otimes s_2, \exists y : s_1, \exists z : s_2. x = \langle y, z \rangle$$

Proof. We start with the introduction of the universal quantifier, we get $(\exists y : s_1, \exists z : s_2. x = \langle y, z \rangle)$, then we make two sorted quantifier simplification, and one simplification for substitutions, and we pose the following axioms as a local hypothesis for later use:

$$\forall x : s_1 \otimes s_2, \exists y : s_1, \pi_1(x) = y \tag{3.16}$$

$$\forall x : s_1 \otimes s_2, \exists y : s_2, \pi_2(x) = y \tag{3.17}$$

$$\forall x : s_1 \otimes s_2, \langle \pi_1 x, \pi_2 x \rangle = x \tag{3.18}$$

We eliminate the universal quantifier of the hypothesis (3.16) by specializing it with (x) where we obtain $(\exists y : s_1, \pi_1(x) = y)$; We destruct (3.16) as (y) in order to extract a new witness and eliminate the existential quantifier, we end up with $(\pi_1(x) = y)$. We apply the same chain of thoughts with the hypothesis (3.17), where we get $(\pi_2(x) = z)$ by destructing the existential quantifier as a new fresh variable (z) . Next, we specialize our existential goal with (y) and (z) , thus $(x = \langle y, z \rangle)$; We obtain $(x = \langle \pi_1(x), \pi_2(x) \rangle)$ by rewriting by both hypotheses (3.16) and (3.17) in the goal. Finally, this equality follows by the (INVERSE PAIRPROJ2) axiom. $\square$

Chapter 4

Discussion

In this thesis, we explored the formalization of matching logic (ML) theories within the Coq proof assistant. Our focus was on representing mathematical domains and data types, such as booleans and natural numbers, and extending these formalizations to more complex types like sum and product sorts, but also defining and proving properties of these theories.

The results from our efforts affirm the expressive power and flexibility of the ML embedding in Coq, where we were able to use various notations, definitions, and theorems in defining these specifications and reasoning about their properties, such as using the member of the Syntax class (*inj*) to inject the symbols into a matching logic signature and access them for not only the definition of the axioms but also for stating the related theorems. Moreover, one notable outcome was the ability to handle parametric theories such as sum and product sorts, and effectively discuss the proof of the defined associated theorems.

The significance of these findings lies in their contribution to the broader field of formal verification and theorem proving. By formalizing and proving ML theories within Coq, this work provides a template for future research in the area, particularly in enhancing the scope and efficiency of formal verification tools. Additionally, these results contribute to the theoretical foundation necessary for the application of ML in other domains, such as the $\mathbb{K}$ framework, which is used for defining and verifying the semantics of programming languages [23].

It is also important to mention that several challenges and limitations were encountered despite the successes. First, we found it challenging to handle certain proofs due to a lack of supporting lemmas, and repetitively proving side conditions

about well-formedness and mu-freeness of patterns manually. In addition, one of the challenges was to define supporting theorems on the object level such as the Peano Induction theorem which is for now admitted in our current formalization. However, it could be completed in the future. Also, we encountered issues with some proof mode tactics of the Coq embedding not functioning as expected, requiring manual modifications that ideally should have been automatic.

Chapter 5

Related Work

Matching Logic (ML) [1] has emerged as a prominent formalism in the field of formal reasoning. Its core strength lies in its ability to specify and reason about mathematical structures by leveraging patterns and pattern matching. This inherent flexibility has made ML a compelling choice for defining a wide range of logical systems. This research delves into this domain by examining relevant work, with a particular focus on the Iris framework.

The Iris framework is a higher-order separation logic framework implemented and verified entirely within the Coq proof assistant. Separation logic itself is an extension of Hoare logic, specifically designed to facilitate modular and scalable reasoning about programs that manipulate pointers. The formalization of Iris logic, as presented in [24], employs a “shallow embedding” approach [25]. In this approach, logical connectives are directly defined as functions operating on the semantic model of propositions within Coq. This eliminates the need for extending the signature of the object logic, and formalizes new types like lists, natural numbers, finite sets, finite maps, etc, but also allows the reuse of the data types of the meta-logic, here, Coq. Furthermore, building upon this foundation, [26] introduced a dedicated proof mode within Iris as a Coq extension. This proof mode provides a suite of specialized tactics tailored for interactive proofs in higher-order separation logic.

Furthermore, we explore the work by [25] on mechanizing Linear Logic within the Coq proof assistant. Their approach involves formalizing the linear predicates where they define a new inductive type in Coq to represent linear predicates. This type essentially acts as a set containing elements built from specific cases. These cases encompass the core linear connectives of the logic, (Implies, One, Plus, Times,

Top, With, and Zero). Notably, this allows any Coq variable to be used as a linear proposition or predicate. In the second step, they establish a formal syntax for the linear connectives. They then define their operational behavior by encoding the corresponding left and right rules from sequent calculus.

Chapter 6

Summary

6.1 Conclusion

As part of a wider research effort seeking a solid foundation for machine-checked, interactive reasoning about systems and languages based on matching logic (ML), this thesis delved into the formalization of ML theories within the Coq proof assistant. These theories represent fundamental mathematical concepts like booleans, natural numbers, and parametric data structures (product and sum types). In this work, we provided the approach on how we were able to formalize these specifications in Coq, leveraging the embedding of ML within this proof assistant, which gives us the essential concepts, a large library of notations, theories, and different supporting lemmas and theorems. Moreover, we did not stop at definitions, we further constructed and formally proved various theorems specific to each of these theories with the help of the ML proof system and proof mode which offer specialized tactics to manipulate the proof states itself and reach our goal.

This work describes a significant step forward in establishing a robust framework for machine-checked reasoning with ML, it provides formal definitions, notations, and theorems for core mathematical domains and data types, all accompanied by Coq proofs. These meticulously constructed formalizations within Coq pave the way for future exploration in several directions. Researchers can now build upon this foundation to develop more intricate theorems and explore the application of ML to reason about increasingly complex systems and languages expressed in the $\mathbb{K}$ framework which is based on ML [2].

6.2 Future Work

This thesis has made significant progress in formalizing ML theories within the Coq proof assistant. We have successfully laid the groundwork for reasoning about basic concepts like booleans, natural numbers, sum types, and product types. However, various areas of possible future work remain open:

- Reasoning about $\mathbb{K}$'s builtins implementation since ML serves as a basis of the $\mathbb{K}$ framework, and our work lies down the fundamentals.
- Formalizing additional ML theories like function types, lists, vectors, and sets. For each theory, we could reason about its properties by defining and proving relevant theorems, this would enrich the overall framework with a broader range of reasoning capabilities.
- Expanding the ML proof mode by defining new tactics, to reduce the repetitive steps during the theorem-proving process.
- Improving the existing tactics within the ML proof mode. For instance, tactics like `m1Specialize` and `m1Intro` could be enhanced to support functional patterns and sorted quantifiers. This would provide users with more powerful tools to tackle more intricate proofs.

Acknowledgements

I would like to extend my thanks to Runtime Verification Inc. and the ADA Foundation for financially supporting this project. Additionally, I am grateful to Dániel Horpácsi for his supervision, and Péter Berczky for his assistance, and guidance.

Bibliography

- [1] Xiaohong Chen, Dorel Lucanu, and Grigore Roşu. “Matching logic explained”. In: *Journal of Logical and Algebraic Methods in Programming* 120 (2021), p. 100638.
- [2] Péter Bereczky et al. “Mechanizing Matching Logic In Coq”. In: *Electronic Proceedings in Theoretical Computer Science* 369 (Sept. 2022), 17–36. ISSN: 2075-2180. DOI: 10.4204/eptcs.369.2.
- [3] Chris Hathhorn, Chucky Ellison, and Grigore Roşu. “Defining the undefinedness of C”. In: *SIGPLAN Not.* 50.6 (2015), 336–345. ISSN: 0362-1340. DOI: 10.1145/2813885.2737979.
- [4] Denis Bogdanas and Grigore Roşu. “K-Java: A Complete Semantics of Java”. In: *SIGPLAN Not.* 50.1 (2015), 445–456. ISSN: 0362-1340. DOI: 10.1145/2775051.2676982.
- [5] Daejun Park, Andrei Stănescu, and Grigore Roşu. “KJS: a complete formal semantics of JavaScript”. In: *SIGPLAN Not.* 50.6 (2015), 346–356. ISSN: 0362-1340. DOI: 10.1145/2813885.2737991.
- [6] Dwight Guth. “A formal semantics of Python 3.3”. PhD thesis. University of Illinois at Urbana-Champaign, 2013.
- [7] Feng Wang et al. “Krust: A formal executable semantics of Rust”. In: *2018 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE. 2018, pp. 44–51.
- [8] Feng Wang et al. “KRust: A Formal Executable Semantics of Rust”. In: *2018 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. 2018, pp. 44–51. DOI: 10.1109/TASE.2018.00014.
- [9] Xiaohong Chen and Grigore Rosu. “Technical Report: Applicative Matching Logic: Semantics of K”. In: (2020).

- [10] Joseph R Shoenfield. *Mathematical logic*. AK Peters/CRC Press, 2018.
- [11] Dexter Kozen. “Results on the propositional μ -calculus”. In: *Theoretical computer science* 27.3 (1983), pp. 333–354.
- [12] Zoltán Ésik. “Completeness of Park induction”. In: *Theoretical Computer Science* 177.1 (1997), pp. 217–283.
- [13] Xiaohong Chen and Grigore Roşu. “Matching μ -logic”. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE. 2019, pp. 1–13.
- [14] Arthur Charguéraud. “The locally nameless representation”. In: *Journal of automated reasoning* 49 (2012), pp. 363–408.
- [15] Jan Tušil, Péter Bereczky, and Dániel Horpácsi. “Interactive Matching Logic Proofs in Coq”. In: *International Colloquium on Theoretical Aspects of Computing*. Springer. 2023, pp. 139–157.
- [16] Andrew D Gordon. “A mechanisation of name-carrying syntax up to alpha-conversion”. In: *HOL Users’ Group Workshop*. Springer. 1993, pp. 413–425.
- [17] Xavier Leroy. “A locally nameless solution to the POPLmark challenge”. PhD thesis. INRIA, 2007.
- [18] Conor McBride and James McKinna. “Functional pearl: i am not a number–i am a free variable”. In: *Proceedings of the 2004 ACM SIGPLAN Workshop on Haskell*. 2004, pp. 1–9.
- [19] High-Assurance Refactoring Project. *Formalization of Matching Logic Theories in Coq - Bools theory*. Accessed on 11th May, 2024. 2024. URL: <https://github.com/harp-project/AML-Formalization/pull/412>.
- [20] High-Assurance Refactoring Project. *Formalization of Matching Logic Theories in Coq - Nat theory*. Accessed on 11th May, 2024. 2024. URL: <https://github.com/harp-project/AML-Formalization/pull/425>.
- [21] High-Assurance Refactoring Project. *Formalization of Matching Logic Theories in Coq - Sum sort theory*. Accessed on 11th May, 2024. 2024. URL: <https://github.com/harp-project/AML-Formalization/pull/430>.

- [22] High-Assurance Refactoring Project. *Formalization of Matching Logic Theories in Coq - Product sort theory*. Accessed on 11th May, 2024. 2024. URL: <https://github.com/harp-project/AML-Formalization/pull/429>.
- [23] Xiaohong Chen et al. “Towards a trustworthy semantics-based language framework via proof generation”. In: *International Conference on Computer Aided Verification*. Springer. 2021, pp. 477–499.
- [24] Ralf Jung et al. “Iris from the ground up”. In: *Submitted to JFP* (2017).
- [25] Martin Wildmoser and Tobias Nipkow. “Certifying machine code safety: Shallow versus deep embedding”. In: *International Conference on Theorem Proving in Higher Order Logics*. Springer. 2004, pp. 305–320.
- [26] Robbert Krebbers, Amin Timany, and Lars Birkedal. “Interactive proofs in higher-order concurrent separation logic”. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. 2017, pp. 205–217.