Paraconsistent Higher-Order Logic for Knowledge-Based Systems

Student
Thorbjørn Kamlarczyk Rasmussen

Supervisor
Jørgen Villadsen

Kongens Lyngby 2008
IMM-BSc-2008-19
In this report we discuss paraconsistent logic. In classical and intuitionistic logic everything follows from a contradiction. In a paraconsistent logic something limited and more reasonable than everything follows from a contradiction. In this report we concentrate our efforts on many valued paraconsistent first-order logic. We have formalized a many valued logic and a paraconsistent many valued logic in the generic theorem prover Isabelle.

The higher-order aspects of paraconsistent logic have been de-emphasized in this report since it turned out to be quite irrelevant to many authors of paraconsistent logics. We have also de-emphasized the use of paraconsistent logics in knowledge-based systems, since Isabelle is not very well suited for such a task.
This project was prepared at Informatics Mathematical Modelling, the Technical University of Denmark from February through June 2008. It was part of the requirements for acquiring the B.Sc. degree in engineering.

Lyngby, June 2008

Thorbjørn Kamlarczyk Rasmussen
## Contents

| Summary | i |
| Preface | iii |
| 1 Introduction | 1 |
| 2 Basics | 3 |
| 2.1 Paraconsistency | 3 |
| 2.2 Many Valued Logic | 4 |
| 2.3 Many Valued Paraconsistent First-Order Logic | 6 |
| 2.4 Many Valued Paraconsistent Higher-Order Logic | 7 |
| 2.5 Knowledge-Based Systems | 8 |
| 3 Logic of Partial Functions | 9 |
| 3.1 Model Theory | 10 |
| 3.2 Proof Theory | 12 |
4 Isabelle

4.1 Motivation ............................................... 15
4.2 Introduction to Isabelle ................................. 16
4.3 Formalization of $HRMI_m$ in Isabelle ............... 17
4.4 A proof in $HRMI_m$ ....................................... 18

5 Formalization of the Logic of Partial Functions in Isabelle 21

5.1 Basic Design .............................................. 21
5.2 Interactive Proofs ........................................ 23
5.3 Automated Proof Procedures ............................ 27

6 Conclusion ................................................... 29

A Definitions, Axioms and Rules in the Logic of Partial Functions 31

A.1 The Basic Connectives ................................. 33
A.2 Definitions of the Other Connectives .................. 35
A.3 Derived Rules for $\neg$, $\lor$ and $\land$ ................. 36
A.4 Derived Rules for $\rightarrow$, $\leftrightarrow$ and $\forall$ ....... 37
A.5 Rules for $\Delta$ ........................................... 38

B HRMI.thy ................................................... 39

C LPF.thy ..................................................... 41

C.1 Constants and Connectives ............................. 41
C.2 Axioms, Rules and Definitions ......................... 42
C.3  Derived Rules not Proven  .............................................. 43
C.4  Derived Rules  .......................................................... 44
C.5  Automated Proofs  ....................................................... 49
Chapter 1

Introduction

A knowledge-based system might contain inconsistencies. An inconsistency might get introduced to the system due to different sensors perceiving an event differently, due to one sensor perceiving an event wrongly at some time or in many other ways.

If a knowledge-based system uses classical or intuitionistic logic to reason, then we have to be absolutely sure the knowledge-base does not contain a single contradiction. If the knowledge-base is inconsistent we can infer everything in classical logic and the knowledge-based system is completely useless.

Knowledge-based systems often contain lots of information, so it is difficult to check whether a knowledge-base contains inconsistent information. It might even be the case that the inconsistent information is supposed to be part of the knowledge-base. For these reasons we would like the knowledge-based system to permit inference from inconsistent information in a reasonable manner.

Paraconsistent logic do permit inference from inconsistent information in a non-trivial way and paraconsistent logic is thus well suited for knowledge-based systems. It is not just in knowledge-based systems that paraconsistent logic seems to be relevant. We might have contradictory beliefs or theories we would like to formalize and reason about. We would again need a paraconsistent logic to do the reasoning satisfyingly.
The first section of this report introduces the basic concepts of paraconsistency, many valued logic and knowledge-based systems.

In the second section we introduce a many valued logic called the logic of partial functions.

The third section gives a brief introduction to the generic theorem prover Isabelle together with a formalization of the paraconsistent logic $HRMI_m$ in Isabelle.

We present a formalization of the logic of partial functions in Isabelle in the last section of this report.
Chapter 2

Basics

2.1 Paraconsistency

Inconsistent information leads in classical and intuitionistic logic to an explosion (everything being provable). If we can not guarantee that the information we reason about does not contain contradictions, then classical and intuitionistic logic become useless since everything is provable. We call a logic explosive if it, for all \( P \) and \( Q \), contains the following rule.

\[
\begin{array}{c}
P \quad \neg P \\
\hline
Q
\end{array}
\]

This rule is called ex contradictione quodlibet (ECQ). A logic in which ECQ is not valid for all \( P \) and \( Q \) are called paraconsistent\cite{15}. A paraconsistent logic limits what is provable from everything to something more reasonable.

It is easy to see that paraconsistent logics has to abandon some very intuitive principles. The following example highlights some of the difficulties.
In this derivation of ECQ we just used two rules, disjunction introduction \((P \vdash P \lor Q)\) and the disjunctive syllogism \((P, \lnot P \lor Q \vdash Q)\). It is clear that at least one of these rules can not hold for all \(P\) and \(Q\) in a paraconsistent logic even though they seem very intuitive.

Relevance logic, filter logic, non-truth functional logic and non-adjunctive logic are examples of paraconsistent logics.[15]. One of the simplest ways to create a paraconsistent logic is by introducing one or more additional truth values to create a many valued logic. Not all many valued logics are paraconsistent but some are. We present some famous many valued logics in Section 2.2 and a many valued paraconsistent logic in Section 2.3.

2.2 Many Valued Logic

In classical logic we have the law of the excluded middle, \(P \lor \lnot P\). The law of the excluded middle is has some problems. \((\frac{1}{0} = 1) \lor (\frac{1}{0} \neq 1)\) has the form \(P \lor \lnot P\) but we would not say that it is a tautology. The problem with the statement is that \(\frac{1}{0}\) is undefined and we are not able to express this in classical logic. Many valued logics do not just contain the truth values truth (T) and falsehood (F) but have one or more extra truth values. For all the many valued logics we discuss in this section the law of the excluded middle does not hold.

One of the first many valued logics was proposed by Łukasiewicz in 1920 [17]. Łukasiewicz’s logic is a three valued propositional calculus. It contains apart from T and F the truth value I. I is interpreted as intermediate or indeterminate. The interpretation of I comes from the idea that some statements are inherently neither true nor false (e.g. statements about the future).

When I is introduced the meaning of the propositional connectives has to be defined in a new way. In Łukasiewicz’s logic \(\rightarrow\) and \(\lnot\) are the basic connectives, also called the primitives. The other classical connectives are constructed from the primitives. We have the definitions \(P \lor Q \equiv (P \rightarrow P) \rightarrow Q\), \(P \land Q \equiv \lnot (\lnot P \lor \lnot Q)\) and \(P = Q \equiv (P \rightarrow Q) \land (Q \rightarrow P)\). The truth tables for the five connectives can be seen in Table 2.1.

One important property of Łukasiewicz’s three valued logic is that \(P \rightarrow P\) is a tautology. Unfortunately the classical definition of \(P \rightarrow Q\) begin equivalent to \(\lnot P \lor Q\) does not hold. This is a common problem in many valued logics.
2.2 Many Valued Logic

Other famous three valued logics interpret the additional truth value in other ways than Lukasiewicz did. Different interpretations of the meaning of the additional truth value alters the truth tables of the connectives. In Bochvars three valued logic [17] the I is interpreted as meaningless, so the truth tables for the connectives are very different from those of Lukasiewicz’s logic. Another way to interpret the extra truth value is as unknown or indeterminable. This is done in Kleenes strong three valued logic [17].

Four valued logics also exist. The usual interpretation of the four truth values are: truth (T), falsehood (F), contradictory knowledge (both truth and falsehood B) and a lack of knowledge (neither truth nor falsehood N).

As an example of a four valued logic we have Belnaps four valued logic [1](this is also a paraconsistent logic). The truth values in Belnaps four valued logic can be arranged after the amount of knowledge the truth values contain (B at the top and N at the bottom) to obtain an approximation lattice. The truth values can also be arranged after the degree of truth in the truth values (T at the top and F at the bottom) to obtain a logical lattice. We are thus able to arrange the truth values in two related partial orders which each form a lattice. If we introduce an operator that reverses the partial order of one of the lattices, but keeps the partial order of the other lattice, we obtain a logic based on bilattices. Logics based on bilattices can have an arbitrary amount of truth values as long as we keep the two partial orderings.

Another many valued logic called the logic of partial functions is presented in Section 3 and in Section 5 we present an implementation of the logic of partial functions in the generic theorem prover Isabelle.
2.3 Many Valued Paraconsistent First-Order Logic

In this section we describe the logic $RMI_m[3][2]$. We present a Hilbert style proof system for $RMI_m$ ($HRMI_m$) in Section 4.3 where we also try to formalize $HRMI_m$ in the generic theorem prover Isabelle.

$RMI_m$ tries to combine classical logic, da Costa’s paraconsistent logic and relevance logic. Paraconsistent logics build on da Costa’s ideas have three truth values. In three valued paraconsistent logics any two contradictory propositions are equivalent. Since $RMI_m$ tries to incorporate ideas from relevance logic as well this is not satisfactory. $RMI_m$ therefore has an infinity of truth values, and contradictory propositions with different truth values are considered to be irrelevant to each other. The truth values of $RMI_m$ are named $\top, \bot, I_1, I_2, I_3, \ldots$. Here $\top$ is interpreted as truth, $\bot$ as falsehood and $I_1, I_2, I_3, \ldots$ are the paradoxical truth values.

$RMI_m$ has two basic connectives $\sim$ and $\otimes$. The connectives are defined as.

$$\sim \top = \bot, \sim \bot = \top, \sim I_k = I_k \quad (k = 1, 2, \ldots)$$

$$P \otimes Q = \begin{cases} 
\bot & \text{if } P = \bot \text{ or } Q = \bot, \\
I_k & \text{if } P = Q = I_k, \\
\top & \text{otherwise}
\end{cases}$$

The connectives $\sim$ and $\otimes$ constitute the entire language of $RMI_m$. The evaluation function $v$ for the language of $RMI_m$ is defined as $v(\top) = \top, v(\bot) = \bot, \ldots$ and $v(\sim \phi) = \sim v(\phi)$ and $v(\phi \otimes \psi) = v(\phi) \otimes v(\psi)$ for all formulas $\phi$ and $\psi$.

We write entailment in $RMI_m$ for $\mathcal{T}$ and $\phi$ as $\mathcal{T} \vdash \phi$. We have $\mathcal{T} \vdash \psi$ if and only if we for all $\psi \in \mathcal{T}$ have $v(\psi) \neq \bot$ and this leads to $v(\phi) \neq \bot$.

In Section 4.3 we use the connective $\rightarrow$. $\rightarrow$ is defined as $\phi \rightarrow \psi \equiv \sim(\phi \otimes \sim \psi)$. From this definition we can derive the following.

$$P \otimes Q = \begin{cases} 
\top & \text{if } P = \bot \text{ or } Q = \top, \\
I_k & \text{if } P = Q = I_k, \\
\bot & \text{otherwise}
\end{cases}$$

We stop the description of $RMI_m$ at this point but much more can of course be said of $RMI_m[3][2]$. 
2.4 Many Valued Paraconsistent Higher-Order Logic

A lot of the paraconsistent logics used in computer science are intended to solve tasks such as automated reasoning from an inconsistent knowledge-base, belief revision etc[15]. Such tasks are traditionally solved in first-order predicate logic. It is thus no surprise that most paraconsistent logics are also first-order logics. Among the paraconsistent higher-order logics that have been developed we very briefly present one by Villadsen[18] in this section.

In Villadsen's logic we have a countably infinite number of truth values. One truth value for each proper constant introduced in our theory. A proper constant is either a proposition, a property or a relation.

¬ and ∧ are primitives in the logic and disjunction are defined using these. We have usual definition of disjunction $P \lor Q \equiv \neg(\neg P \land \neg Q)$. The truth tables for ¬, ∧ and ∨ in a four valued version of Villadsen's logic can be seen in Table 2.2.

Villadsen introduces two different biimplication connectives. One is equivalent to strong equality and it is called ⇔. Strong equality(=) is true whenever $Q$ and $P$ have the exact same truth value for $P = Q$. $P = Q$ is false in all other cases. This makes strong equality a non monotone connective. The second biimplication connective(⇔) are used to define the implication connective →. The definition is $P \rightarrow Q \equiv P \leftrightarrow P \land Q$. The truth tables for = and → can be seen in Table 2.2.

<table>
<thead>
<tr>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \rightarrow Q$</th>
<th>$P = Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$\neg P$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Table 2.2: Truth tables for connectives in a four valued version of Villadsen's logic.

Villadsen defines many more connectives and he bases his logic on the typed λ-calculus. The result is a paraconsistent higher-order logic.
2.5 Knowledge-Based Systems

A knowledge-based system consists of a knowledge-base and method of reasoning about the information in the knowledge-base [16]. The knowledge-base is a set of sentences. The sentences represent all the known information in the system.

Knowledge-based systems have at least two methods, one used to query the knowledge-based system for what is known and one to add new sentences to the system. When a query is given to the knowledge-based system an answer must be given back. This is often done by deriving new sentences relevant to the query and from these find a satisfactory answer which then is returned.

A general problem with knowledge-based systems is that it is very hard to find a good algorithm that can figure out which sentences from the knowledge-base the system should use when it has to derive new sentences.

We have chosen not to create a knowledge-based system. This is largely because we have chosen to use the generic theorem prover Isabelle. Since Isabelle is a theorem prover we would not be able to create a method to add new sentences to the system while it is running. Everything in the knowledge-base would then be background knowledge. We could of course manually add sentences to the implemented system while it is not running, but this approach would be unsatisfying since we then have to alter a large part of the code every time we add a new sentence.
The logic of partial functions (LPF) \cite{5,8,4} is a three valued first-order logic designed to prove program correctness. The rest of this report is based on LPF and we present a formalization of LPF in the generic theorem prover Isabelle in Section 5. In this section we explain the basic model and proof theory of the LPF.

We have chosen to work with LPF since it is a three valued logic so it has many similarities to a lot of paraconsistent logics even though LPF is not a paraconsistent logic. In fact the proof theory LPF contains the rule ECQ. LPF has a nice well presented natural deduction proof system that is easy use and makes LPF attractive. Furthermore the proof system of LPF resembles the natural deduction proof system of classical first-order logic, so we hoped that the construction of automated proof procedures would be a somewhat easy task. This turned out not to be such a easy task after all, but it was one of the motives for choosing LPF.
3.1 Model Theory

The motivation behind LPF is that partial functions\(^1\) that give undefined results often occur in programs. In order to prove properties about such programs we therefore need a logic that can handle undefined terms in a satisfying manner. As an simple example of a recursively defined function that might give problematic results we have the subtraction function $\text{subp}(i, j)[8]$ defined as

$$
\text{subp}(i, j) = \begin{cases} 
it = j & \text{then } 0 \\
\text{else } \text{subp}(i+1, j) + 1
\end{cases}
$$

$subp(i, j)$ runs forever when $i$ is greater than $j$ and we could call the result of such behavior undefined. If $i \leq j$ then we have the result $\text{subp}(i, j) = j - i$. We could formalize this in the following way.

$$
\forall i, j. i \leq j \rightarrow \text{subp}(i, j) = j - i
$$

If the antecedent is false then $\text{subp}(i, j)$ is undefined and we would not be able to reason about the above statement in e.g. classical first-order logic. We could of course formalize the function in other ways or choose a lazy evaluation of the above statement. We could imagine more complicated cases where such corrections would be harder to do. It would therefore be nice to be able to reason with undefined terms.

At least three truth values are necessary to reason about problems such as the above. In LPF we have exactly three truth values. The truth values are truth ($T$), falsehood ($F$) and undefinedness ($I$). The undefined truth value are seen as a gap in our knowledge.

The connectives of LPF are made as generous as possible, so whenever we have enough information a result that is not undefined is given. So for $P \lor Q$ we get truth whenever at least one of $P$ or $Q$ are true, false if both are false and undefined otherwise. Negation turns truth into falsehood and vice versa but undefinedness stays undefined. From disjunction and negation we can define the other usual connectives: conjunction, implication and biimplication. $P \land Q$ is defined as $\neg(\neg P \lor \neg Q)$ and $P \rightarrow Q$ is defined as $\neg P \lor Q$. $P \iff Q$ is defined as $(P \rightarrow Q) \land (Q \rightarrow P)$. The truth tables for the connectives $\neg$, $\land$, $\lor$, $\rightarrow$ and $\iff$ in LPF can be seen in Table 3.1.

It can be seen from truth tables that $P \rightarrow P$, $P \iff P$ and $\neg P \lor P$ are not tautologies in LPF, but many classical identities such as $\neg \neg P \equiv P$ and the associative and commutative laws for $\lor$ and $\land$ still hold. We are in fact going to derive the associative and commutative laws for disjunction and conjunction

\(^1\)Partial functions - functions that are only defined for some part of a larger domain
### 3.1 Model Theory

<table>
<thead>
<tr>
<th>$P \backslash Q$</th>
<th>$P$</th>
<th>$\neg P$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \rightarrow Q$</th>
<th>$P \leftrightarrow Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$I$</td>
<td>$I$</td>
<td>$I$</td>
<td>$I$</td>
<td>$I$</td>
<td>$I$</td>
<td>$I$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

Table 3.1: Truth tables for connectives in the LPF

in our Isabelle implementation of LPF.

That $P \rightarrow P$ and $P \leftrightarrow P$ are not tautologies does not match our intuitive understanding of these connectives. The problem arises from our mechanical definition of $P \rightarrow Q$ as $\neg P \lor Q$. Other many valued logics are designed to keep $P \rightarrow P$ as a tautology.

The truth values of LPF can be partially ordered. LPF has the partial ordering function $\sqsubseteq$ where we have $I \sqsubseteq F$, $I \sqsubseteq T$ and $T$ and $F$ are not comparable. All the connectives defined up till this point have been monotone with respect to this ordering.

We have the usual quantifiers, $\forall$ and $\exists$. The quantifiers in LPF have the same properties as in classical first-order logic such as $\forall x. P(x) \equiv \neg \exists x. \neg P(x)$ and conventions about free and bound variables are the same. We in fact define $\forall$ using $\exists$ in LPF. As in classical first-order logic we can treat the universal quantifier, $\forall$, as a generalized conjunction and the existential quantifier, $\exists$, as a generalized disjunction.

A constant symbol for truth called $\text{True}$ are also part of LPF. $\text{True}$ is used to defined another constant symbol representing falsehood. We have the definition $\neg \text{True} \equiv \text{False}$.

We might sometimes need to distinguish between the defined and the undefined expressions. The $\Delta$ connective performs this task. If $P$ is defined then the result of $\Delta P$ is $T$ and if $P$ is not defined then the result is $F$. $\Delta$ is a non-monotone connective in LPF. We also define another connective, $\delta P \equiv P \lor \neg P$. $\delta P$ thus has the truth value $T$ when $P$ is defined and $I$ when $P$ is undefined. The $\delta$ connective is monotone unlike the $\Delta$ connective.

There are two equality predicates in LPF. One called strong equality($==$) and the other called weak equality($=$). For weak equality the result of $s = t$ is $I$ if either or both $s$ or $t$ are undefined. For strong equality the result of $s == t$ is true if both $s$ and $t$ are undefined, and false otherwise. Strong and weak equality behave as normal when no undefined truth values are involved. Strong equality
is thus non monotone and weak equality is monotone.

The connectives have the following order of precedence (in descending order) $\Delta$, $\delta$, $\neg$, $\wedge$, $\lor$, $\rightarrow$, $\leftrightarrow$. The quantifiers $\exists$ and $\forall$ have same precedence which also is the lowest precedence of all connectives and quantifiers. All the binary connectives are right associative. $P \rightarrow Q \wedge R \wedge Q \lor S$ is therefore understood as $(P \rightarrow ((Q \wedge (R \wedge Q)) \lor S))$ in LPF.

### 3.2 Proof Theory

LPF uses a natural deduction proof style. Natural deduction proofs are easy to read. All inference rules that are valid in LPF are also valid in classical first-order logic. The opposite is not true. Some inference rules from classical first-order logic are not valid in LPF. The most important of the rules that are not valid in LPF is the law of the excluded middle. The law of the excluded middle states in classical first-order logic that $\neg P \lor P$ is a tautology. In LPF this is clearly not the case since $\neg P \lor P$ evaluates to $I$ whenever $P$ has the truth value $I$. Since the law of the excluded middle does not hold in LPF we can not use the proof by contradiction principle[7].

Most of the classical deduction rules holds in LPF. Examples of valid deduction rules in LPF that are also valid in classical first-order logic are the two disjunctive introduction rules and the disjunctive elimination rule.

\[
\begin{array}{c}
\frac{P}{P \lor Q} \\
\frac{Q}{P \lor Q}
\end{array}
\]

\[
\begin{array}{c}
\vdots \\
\vdots \\
\frac{P \lor Q \quad R \quad R}{R}
\end{array}
\]

We explain the disjunctive elimination rule since the notation might be unfamiliar. The rule states that if we have $P \lor Q$, and $P$ leads to $R$, and $Q$ leads to $R$ then we can conclude $R$. $P \lor Q$, $P$ leads to $R$ and $Q$ leads to $R$ are called the assumptions or the premises and $R$ are called the conclusion.
All the natural deduction rules for the two primitives of LPF, $\neg$ and $\lor$, together with the rules for weak equality can be seen in Appendix A.1.

We also have deduction rules for our quantifiers. For example we have the existential quantifier elimination rule.

\[
\frac{[P(y/x)]}{\exists x. P(x) \quad Q} \quad Q
\]

In this rule $P(x)$ is a formula with $x$ occurring freely and $P(y/x)$ is the formula $P$ with all occurrences of $x$ substituted by $y$. The elimination rule for the existential quantifier also comes with a proviso saying that $y$ must be arbitrary and bound for the rule to be valid. There are four deduction rules for the existential quantifier and they can all be seen in Appendix A.1.

We defined $\land$, $\rightarrow$, $\leftrightarrow$, $\delta$ and $\forall$ using the other connectives and quantifiers. We therefore include these definitions as bidirectional rules in our natural deduction proof system so we can use these connectives and the universal quantifier. These definitions as rules can be seen in Appendix A.2.

We can of course make proofs of many well known principles from classical first-order logic in LPF. In Section 5 we therefore formalize LPF in the generic theorem prover Isabelle, and try to find proofs to the derived rules listed in Appendix A.3 and Appendix A.4.
Chapter 4

Isabelle

4.1 Motivation

A lot of different theorem provers exist today. Among the most used are Mizar, HOL, Coq, Metamath and Isabelle [19]. Of the popular provers only Isabelle and Metamath allows us to define our own logic. Since it is part of the project goal to implement an object logic we have to choose either Isabelle or Metamath.

Metamath has a low level of build-in automation compared to Isabelle [19]. The 'Isar' (Intelligible semi-automated reasoning)[11] proof language of Isabelle together with Proof General [6] allows users of the Isabelle system to construct interactive proofs in a graphical environment in an fairly intuitive manner. The Isabelle theorem prover therefore seems to be the easiest to learn of the two. Isabelle is also the most popular and widely used of the two provers.

Because of Isabelle’s popularity, accessibility, graphical environment and the possibility of creating our own object logic we have chosen Isabelle for this project.

We could of course choose to implement everything in a programming language like ML in which Isabelle itself is built, but this would be to big a task for this project.
4.2 Introduction to Isabelle

In this project the 2007 distribution of the Isabelle theorem Prover has been used. The developed files have not been tested on the new 2008 distribution.

Isabelle is build in Standard ML[9]. Users of the Isabelle system is allowed to create extensions to the system in ML.

The underlying logic of Isabelle is an intuitionistic higher-order logic which is implemented in a theory called Pure.

There are three basic constructs (\(\Rightarrow\), \(\Rightarrow\) and \(!!\)) in Pure. \(\Rightarrow\) is used to separate premises from conclusions in theorems. \(\Rightarrow\) is used to make definitions and \(!!\) is the universal quantifier. When you design an object logic you have to define the rules using these constructs.

Since we are defining our own object logic we only have access to the automatic proof procedures that have been developed for Pure, which is very few. One very important feature build into the Pure framework is higher-order unification. So whenever you think it is possible to unify two sentences, Isabelle can check if it is possible to unify the sentences automatically. Since we do not have many powerful automatic proof procedures offered we have to make long proofs using the apply command many times or alternatively develop our own automatic proof procedures.

An object logic which implements a formal theory has to contain connectives, well-formed formulas, axioms and rules of inference in Isabelle.

A lot of different object-logics have been developed for Isabelle[12]. The best developed and documented of these are HOL[10] (higher-order logic) and ZF[13] (Zermelo-Fraenkel set theory). Many other logics have are also developed and are distributed together with Isabelle.

The Isabelle theories called IFOL and FOL[13] has been the most relevant to this project. IFOL is an implementation of intuitionistic first-order logic with a Gentzen-style natural deduction proof system. The Isabelle theory of FOL is an implementation of classical first-order logic. FOL is build upon IFOL. Since LPF is similar to classical first-order logic in many ways IFOL and FOL have been a source of inspiration for the developed theory of LPF.
4.3 Formalization of $HRMI_m$ in Isabelle

As an example of how to implement a (very simple) theory into Isabelle we have implemented $RMI_m$ with a Hilbert-style proof system[3][2]. In Section 2.3 we explained the basics of $RMI_m$. The Hilbert-style proof system has been presented and proven complete by Avron[3].

In Table 4.1 the nine axioms of the Hilbert-style system is shown.

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Name</th>
<th>Axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Identity</td>
<td>$A \to A$</td>
</tr>
<tr>
<td>T</td>
<td>Transitivity</td>
<td>$(A \to B) \to (B \to C) \to A \to C$</td>
</tr>
<tr>
<td>P</td>
<td>Permutation</td>
<td>$(A \to B \to C) \to B \to A \to C$</td>
</tr>
<tr>
<td>R1</td>
<td>Residuation 1</td>
<td>$(A \to B \to C) \to A \otimes B \to C$</td>
</tr>
<tr>
<td>R2</td>
<td>Residuation 2</td>
<td>$(A \otimes B \to C) \to A \to B \to C$</td>
</tr>
<tr>
<td>C</td>
<td>Contraction</td>
<td>$A \to A \otimes A$</td>
</tr>
<tr>
<td>M</td>
<td>Mingle</td>
<td>$A \otimes A \to A$</td>
</tr>
<tr>
<td>N1</td>
<td>Contraposition</td>
<td>$(A \to \neg B) \to B \to \neg A$</td>
</tr>
<tr>
<td>N2</td>
<td>Double Negation</td>
<td>$\neg \neg A \to A$</td>
</tr>
<tr>
<td>F</td>
<td>Falsehood</td>
<td>$\bot \to A$</td>
</tr>
</tbody>
</table>

Table 4.1: The nine axioms of the Hilbert-style proof system for $RMI_m$

The Hilbert-style proof system of $RMI_m$ has just one rule, modus ponens. Modus ponens has the form.

$$
\frac{A \quad A \to B}{B}
$$

The Isabelle code for the implemented theory can be seen in Appendix B.

The first thing we do in the Isabelle code is to name our theory $HRMI$. Then we load the Pure theory and we are now ready to begin declaring our own theory.

In $HRMI$ we declare one type called $\sigma$. We use this type when we declare the three connectives used in $HRMI$. The connectives we declare are: a unary connective called intensional negation ($\neg$), the binary connective conjunction ($\otimes$) and the binary connective implication ($\to$). A unary connective has of course the type $\sigma \Rightarrow \sigma$. The binary connectives are given the types $[\sigma, \sigma] \Rightarrow \sigma$. We give the intensional negation connective the highest precedence of the three connectives and
we associate the symbol $\sim$ with it in HRMI. The intensional negation connective is implemented in Isabelle using the following piece of code.

\[
\text{Not} :: " \circ \Rightarrow \circ" \quad (" \sim" [40] 40)
\]

Conjunction is given the second highest precedence, right associativity and the symbol $\&$. Implication is given the lowest precedence, right associativity and the symbol $\rightarrow$. The Isabelle code for conjunction and implication looks similar to the code for intensional negation.

After we have defined the connectives we set up some pretty printing of the different symbols.

Then we define the axioms and modus ponens in HRMI. We use the short names mentioned in Table 4.1 to name the implemented axioms. The modus ponens rule is given the name MP. In HRMI we have defined the modus ponens rule in the following way

\[
\text{MP} :: "[| P \rightarrow Q; P |] ==> Q"
\]

Everything to the right of $\Rightarrow$ are the premises. The modus ponens rule has two premises and we separate these with $;$. The premises are grouped using $[| |]$. $[| P \rightarrow Q; P |] ==> Q$ is just an abbreviation for $(P \rightarrow Q ==> (P ==> Q))$. After we have declared the rules $P$ and $Q$ are seen as schematic variables by Isabelle and they are renamed by the system to $?P$ and $?Q$ respectively. Schematic variables are free variables that can be instantiated when we are proving theorems. The axioms are declared in the same way as modus ponens but they are much simpler since they have no assumptions.

Our implemented theory follows the standard way of implementing a theory in Isabelle. First types are declared, then constants(connectives and other predefined symbols) and at last rules and axioms.

### 4.4 A proof in $\text{HRMI}_m$

The last part of the code in Appendix B contains a simple backward proof of $\neg\neg P \rightarrow P \otimes P$ in our implemented version of $\text{HRMI}_m$. We present the proof here to explain how proofs are made interactively in Isabelle.
We start the proof by stating the lemma we want to prove. This is done with the Isabelle code.

\begin{verbatim}
lemma "\sim\sim P \rightarrow P \& P"
\end{verbatim}

Since we are not interested in using our proved theorem again we do not name it. Isabelle gives the following response to the above command.

\begin{verbatim}
proof (prove): step 0

goal (1 subgoal):
  1. \sim\sim P \rightarrow P \& P
\end{verbatim}

From the output we see that we have one subgoal to prove before we are done with the proof. We try to apply the modus ponens rule with the command \texttt{apply(rule MP)}. We get the response

\begin{verbatim}
proof (prove): step 1

goal (2 subgoals):
  1. ?P \rightarrow \sim\sim P \rightarrow P \& P
  2. ?P
\end{verbatim}

We now have two subgoals to prove. The next rule we apply is used on the first subgoal. We could of course choose another subgoal to work on (with the command \texttt{prefer} or \texttt{defer}), but we stick with first subgoal. ?P is a meta-variable. That means we assign any value we would like to it. None of the axioms seems obvious to use so we try the modus ponens rule again and we get the response.

\begin{verbatim}
proof (prove): step 2

goal (3 subgoals):
  1. ?P3 \rightarrow ?P \rightarrow \sim\sim P \rightarrow P \& P
  2. ?P3
  3. ?P
\end{verbatim}

Now the transitivity axiom seems to fit the first subgoal and we try to apply it with the command \texttt{apply(rule T)}. Isabelle tries to unify the first subgoal with the transitivity axiom and we get the response
proof (prove): step 3

goal (2 subgoals):
1. $\sim \sim P \rightarrow ?B6$
2. $?B6 \rightarrow P \& P$

The unification was successful so the first subgoal was proven. The first subgoal looks like it could unify with the double negation axiom so we apply it and get the response.

proof (prove): step 4

goal (1 subgoal):
1. $P \rightarrow P \& P$

The unification was a success and we can now apply the contraction axiom to finish the proof.

proof (prove): step 5

goal:
No subgoals!

We have no more subgoals to prove and we can use the command done to finish the proof. Isabelle shows us what we have proven.

lemma: $\sim \sim ?P \rightarrow ?P \& ?P$

The proof is now finished.
Chapter 5

Formalization of the Logic of Partial Functions in Isabelle

5.1 Basic Design

The implementation of the LPF theory has been inspired by the build-in theory of IFOL of the Isabelle system. The Isabelle code of the LPF theory can be seen in Appendix C.

First we declare one type called $o$. We use this type to declare all the connectives. We declare all the connectives described in Section 3 except strong equality which we do not include since no rules, except its definition deals with this connective. The connectives and quantifiers we do implement are: $\neg$, $\lor$, $\land$, $\rightarrow$, $\delta$, $\leftrightarrow \forall$ and $\exists$. We define these connectives in the same way as we defined the connectives of $HRMI_m$ in Isabelle. As an example we have $\rightarrow$ which we define in following way in Isabelle.

```
"op --> " :: "[o,o] \Rightarrow o"
```

We make sure that we get the correct precedence values. We also assign a symbol to each implemented connective. The symbols and the precedence values can be seen in Table 5.1.
We have also implemented two constant symbols. These are \texttt{False} and \texttt{True}. We use these Isabelle constants to represent the symbols \texttt{False} and \texttt{True} in LPF.

The full implementation of the connectives and the constants can be seen in Appendix \texttt{C.1}.

Most of the basic rules from Appendix \texttt{A.1} are straight forward to implement in the same manner as we did with the rule and axioms of \texttt{HRMI}_m. As an example we show the formalization of the rule \texttt{disjE} in Isabelle.

\[
\begin{array}{c}
\frac{[P] \quad [Q]}{P \lor Q \quad R \quad R \quad \text{[disjE]}}
\end{array}
\]

We implement this rule as

\begin{verbatim}
lemma disjE: " [| P | Q; P ==> R; Q ==> R |] ==> R"
\end{verbatim}

A few of the basic rules are more interesting to implement than \texttt{disjE}. One of these are \texttt{exE}.
5.2 Interactive Proofs

\[ [P(y/x)] \]
\[
\vdots
\]
\[ \exists x. P(x) \] \[ Q \] \[ \text{exE} \]

In addition to this we have that \( y \) must be arbitrary and bound. Luckily Isabelle makes sure that \( y \) is arbitrary for us. We thus only have to make sure \( y \) is bound. We can bind \( y \) by using the universal quantifier from the Pure theory (\( \forall \)) to bind \( y \). In Isabelle the rule exE thus looks like.

```isabelle
lemma disjE: " [| EX x. P(x); \( \forall \) x. P(x) ==> Q |] ==> Q"
```

The rest of the Isabelle versions of the basic rules can be seen in C.2.

The definitions of the other connectives from Appendix A.2 have been implemented as rules. We have tried to implement the rules as definitions using the \( == \) operator, but Isabelle seems to have problems with unification in our theory when definitions are involved. We have thus made two rules for every definition, one in each direction. As an example we have the definition `and_defn`.

```isabelle
\[ \neg(\neg P \lor \neg Q) \] \[ P \land Q \] \[ \text{and_defn} \]
```

`and_defn` is implemented in Isabelle as the following two rules

```isabelle
lemma and_defnD: " \( \neg \) (\neg P \mid \neg Q) ==> P \& Q"
lemma and_defnU: " P \& Q ==> \( \neg \) (\neg P \mid \neg Q)"
```

See Appendix C.2 for the full list of implemented definitions.

We have now described the core of the system and in the following sections we will see what we can do in it.

5.2 Interactive Proofs

We have tried to find proofs for all the derived rules described by Barringer el al.[4] in LPF. The derived rules can be seen in Appendix A.3, Appendix A.4
and Appendix A.5.

Unfortunately we have not been able to prove all of the derived rules, but most have been proven. The rules which we have not been able to derive can be seen in Appendix A.3 and Appendix A.4 (the rules are marked with a ∗). In the Isabelle code the rules we have not proven can be seen in Appendix C.3. We have chosen to put these rules into our Isabelle theory as basic rules, since some of these rules are needed in order to prove a few of the other derivable rules. The rules negConjI1, negConjI2 and negConjE are examples of rules which we have not been able to prove but which are very useful when we have to derive De Morgan’s laws for conjunction.

Most of the derivable rules have been proven and we present and explain two of the proofs in this section. The first proof is of the conjunction introduction rule conjI

\[
\begin{array}{c}
P \\
Q \\
\hline
P \land Q
\end{array}
\]  
\text{conjI}

In the proof of conjI we make a mixed forward and backwards proof in Isabelle. We start by stating what we want to prove in Isabelle.

\[
\text{lemma conjI: assumes p: } \neg \neg P \\
\text{ assumes q: } \neg \neg Q \\
\text{ shows } \neg \neg (P \land Q)
\]

We get the response

1. \( P \land Q \)

We try to use the and_defn definition with the command apply(rule and_defnD). The result is:

1. \( \neg (\neg P \lor \neg Q) \)

It seems like a good idea to split our subgoal into two with the negDisjI rule and we do this with the command apply(rule negDisjI). Then we get the result
1. \( \neg\neg P \)
2. \( \neg\neg Q \)

It is now obvious to that the first subgoal can unify with our premise \( P \) after we have removed the double negation. We could also introduce the double negation to our premise and then unify in a forward style proof. We choose to make the forward proof with the command `apply(rule p [THEN dNegI])`. We then get

1. \( \neg\neg Q \)

We can now finish the proof with a command similar to the one we just used. We use the command `apply(rule q [THEN dNegI])` to finish the proof. We then get the response

```
```

The proof is now finished and we can use the rule `conjI` in other derivations. The full Isabelle code can be seen in Appendix C.4.

The second proof we present in this section is of one of the distributive laws for disjunction `orAssD`.

\[
\begin{array}{c}
(P \lor Q) \lor R \\
\hline
P \lor (Q \lor R)
\end{array}
\]

\textbf{orAssD}

We write the following command in Isabelle to begin the proof

```
lemma orAssD: "'(P | Q) | R ==> P | (Q | R)"
```

It seems like a good idea to use the rule `disjE` so we can construct part of the left side in the premises. We use the rule `disjE` and then we unify immediately afterwards in one step with the command `apply(rule disjE, assumption)`. We then get the response.

1. \[|(P | Q) | R; P | Q|] ==> P | Q | R
2. \[|(P | Q) | R; R|] ==> P | Q | R

We now have \( P | Q \) in the premises and it would be nice to use this when we unify a disjunction. We thus try to introduce another disjunction with the command `apply(rule disjE)`. Isabelle gives us the following response
We can now try to unify again with the command `apply(assumption)`. Isabelle chooses to unify with the first assumption \((P | Q) | R\). This was not what we had in mind so we can tell Isabelle to go back and try again with the command `back`. This gives us the wanted unification. We could also tell Isabelle to replace \(?P4\) with \(P\) and \(?Q4\) with \(Q\). Another solution is to apply more than one rule at a time and then let Isabelle backtrack automatically until the right unification is found. It is clear that once we have replaced \(?P4\) with \(P\) and \(?Q4\) with \(Q\) the first subgoal is solved and we go on to solve the second subgoal. In the second subgoal \(?P4\) has been replaced by \(P\) and we can easily solve this subgoal by removing the right part of the conclusion so the conclusion becomes \(P\). We choose the method that was presented last and use the command `apply(assumption, rule disjI1, assumption)`. The response is.

1. \[(P | Q) | R; P | Q; Q) => P | Q | R\]
2. \[(P | Q) | R; R) => P | Q | R\]

It is easy to isolate \(Q\) in the first subgoal. We do this with the command `apply(rule disjI2, rule disjI1, assumption)` and get the response.

1. \[(P | Q) | R; P | Q; Q) => P | Q | R\]

It is again easy to solve this subgoal by isolating the \(R\). This is done with the command `apply(rule disjI2, rule disjI2, assumption)`. There are now no more subgoals and the proof is finished. We can thus use the derived rule in future proofs. When we ask Isabelle to show us the derived rule we get the following response.


We have now shown the proofs for two of the derived rules from Appendix A.3 and Appendix A.4. In the two proofs we have used the forward proof style mixed with the backward proof style in Isabelle. We have also shown how to use the backtracking functionality of Isabelle to simplify proofs. The rest of the proofs in Isabelle for the derived rules use similar methods and they can be seen in Appendix C.4.
5.3 Automated Proof Procedures

The well developed and documented theories in Isabelle (like HOL\[10\] and ZF\[13\]) have lots of tactics and tacticals implemented. These make it easy to solve the trivial parts of large proofs, so we only have to focus on the hard parts. Some of the very powerful procedures implemented are resolution tactics, simplification tactics etc. A tactic in Isabelle is a theorem proving function which tries to find a proof of a given subgoal according to the implemented function. Tacticals are functions which combine several different tactics\[14\].

Tactics and tacticals can in Isabelle be coded directly in ML. We have not developed any such proof procedures in ML in this project since we do not have a large enough knowledge of how the underlying system works. Such a knowledge can be obtained by reading the *The Isabelle Reference Manual*\[14\], but we have not had the time to get to know the foundations of Isabelle well enough to code proof procedures in ML.

When working in the Isar language of Isabelle, which we have done throughout this project, it is also not advised to code directly in ML\[9\]. Instead the Isar language provides us with commands which gives us some power to automate proofs. We will describe these commands in this section and show how to use them to create somewhat automated proofs in our theory of LPF.

The Isar language has four special symbols used to create something like automatic proofs\[9\]. The four symbols are used to structure the proofs. The first is \(\&\), which is used to make sequential combinations of rules. As an example we have the command

\[
\text{apply(\text{rule conjI}, \text{rule disjE})}
\]

This command first applies the rule \text{conjI} to the first subgoal and then it tries to apply the rule \text{disjE} to the new first subgoal.

The second symbol is \(|\). The command

\[
\text{apply(\text{rule conjI} \mid \text{rule disjE})}
\]

first attempts to use the rule \text{conjI} on the first subgoal. If \text{conjI} succeeds we have a new subgoal. But if the first rule fails the other rule (\text{disjE}) is used and the result of this is given back to the user.
is used to report success if the tried rule failed. If the rule \texttt{conjI} fails, then the command \texttt{apply(rule conjI)} will report a success and the current non altered subgoal is returned.

The last of the symbols is +. When the command \texttt{apply(rule conjI)+} is given, the rule \texttt{conjI} is applied till it fails. If the rule can not be applied at least once the command will fail.

We can use these four symbols to structure our proofs but unfortunately the Isabelle system does not compare the current proof state to parent proof states. It is thus very easy to create an infinite loop with these commands if we try to make to prove to much at once.

When automated proof procedures are created we have to consider which rules are safe to use and which rules we have to use with care\cite{11}. Safe rules can be applied backwards without the loss of information. Safe rules thus transforms a proof state to something which are logically equivalent. In LPF double negation introduction(\(P \vdash \neg \neg P\)) and elimination(\(\neg \neg P \vdash P\)) rules are examples of safe rules. Some rules loose information when they are applied and they are therefore called unsafe. When an unsafe rule is applied to a proof state it is thus not certain that the proof can be completed after the rule has been applied even if it was provable before. Examples of unsafe rules in LPF are the two disjunction introduction rules \((P \vdash P \lor Q)\) and \((P \vdash Q \lor P)\).

As an example of some very simple automatization in LPF we have proven that \((P \lor Q \lor R \lor S \lor T) \land (P \lor S \lor T)\) follows from \(S \lor T\). It is quite obvious that we have to use the rules \texttt{conjI} and \texttt{disjI2} in order to make the proof. To make the a proof we can thus use the command.

\[
\texttt{apply(assumption | rule conjI | rule disjI2)+}
\]

This completes the proof in one step. The full Isabelle code can be seen in Appendix C.5.
In this project we have discussed many valued and paraconsistent logic and we have formalized $HRMI_m$ and LPF in Isabelle.

We have not obtained a satisfactory level of automatization for LPF in Isabelle and we have not been able to prove some of the important derived rules in LPF.
Appendix A

Definitions, Axioms and Rules in the Logic of Partial Functions

We use the following conventions in this section:

$P$, $Q$ and $R$ are logical expressions.

$x$ and $y$ are variables over the proper elements in a universe.

$s$ and $t$ are terms.

$P(x)$ is a formula in which $x$ occurs freely.

$P(y/x)$ is the formula $P$ with all occurrences of $x$ substituted by $y$.

$P[t/x]$ is the formula $P$ with all occurrences of $x$ substituted by $t$.

For $\text{negAllE}$ and $\text{exE}$ we have the proviso that $y$ must be arbitrary and bound.

For $\text{allI}$ and $\text{negExI}$ we have the proviso that $y$ must be arbitrary.

The names of the rules used in this section are the same as the names used to in the Isabelle implementation in Appendix C.

The rules that go in both directions are split up into two rules in the implementation in Isabelle. $U$ is put as a suffix for the upwards version and $D$ is
added to the name for the downwards version. In Isabelle we therefore have two rules corresponding to the definition of $\land$ called \texttt{and\_defn}. One rule is called \texttt{and\_defnU} and the other is called \texttt{and\_defnD}.

The * marks the derived rules that we have not been able to derive ourselves in Isabelle, but which are listed by Barringer et al.[4] as derivable.
A.1 The Basic Connectives
Definitions, Axioms and Rules in the Logic of Partial Functions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{P}{P \lor Q} )</td>
<td>disjI1</td>
</tr>
<tr>
<td>( \frac{Q}{P \lor Q} )</td>
<td>disjI2</td>
</tr>
<tr>
<td>( \frac{[P]}{P \lor Q} )</td>
<td>disjE</td>
</tr>
<tr>
<td>( \frac{[Q]}{P \lor Q} )</td>
<td>disjE</td>
</tr>
<tr>
<td>( \frac{\neg P \quad \neg Q}{-(P \lor Q)} )</td>
<td>negDisjI</td>
</tr>
<tr>
<td>( \frac{\neg(P \lor Q)}{\neg P} )</td>
<td>negDisjE1</td>
</tr>
<tr>
<td>( \frac{\neg(P \lor Q)}{\neg Q} )</td>
<td>negDisjE2</td>
</tr>
<tr>
<td>( \frac{P}{\neg \neg P} )</td>
<td>dNegI</td>
</tr>
<tr>
<td>( \frac{\neg \neg P}{P} )</td>
<td>dNegE</td>
</tr>
<tr>
<td>( \frac{[P(y/x)]}{\exists x.P(x)} )</td>
<td>exI</td>
</tr>
<tr>
<td>( \frac{Q}{\exists x.P(x)} )</td>
<td>exE</td>
</tr>
<tr>
<td>( \frac{\neg \exists x.P(x)}{\neg \exists x_0.P_0(x_0)} )</td>
<td>negExI</td>
</tr>
<tr>
<td>( \frac{\neg \exists x_0.P_0(x_0)}{\neg \exists x_0.P_0(x_0)} )</td>
<td>negExE</td>
</tr>
<tr>
<td>( \frac{P}{\neg P} )</td>
<td>notE</td>
</tr>
<tr>
<td>( \frac{\neg P}{s = t} )</td>
<td>eqContr</td>
</tr>
<tr>
<td>( \frac{\neg True}{P} )</td>
<td>negTrue</td>
</tr>
<tr>
<td>( \frac{\neg True}{P} )</td>
<td>trueI</td>
</tr>
<tr>
<td>( \frac{s = t}{s = s} )</td>
<td>eqRefIx1</td>
</tr>
<tr>
<td>( \frac{s = t}{s = s} )</td>
<td>eqRefIx2</td>
</tr>
<tr>
<td>( \frac{\neg (s = t)}{s = s} )</td>
<td>negEqRefIx1</td>
</tr>
<tr>
<td>( \frac{\neg (s = t)}{s = t} )</td>
<td>negEqRefIx2</td>
</tr>
<tr>
<td>( \frac{s = s \quad t = t}{s = t \lor \neg (s = t)} )</td>
<td>eqTwoVal</td>
</tr>
</tbody>
</table>

Table A.1: Rules for the basic connectives \( \lor \) and \( \neg \) and the existential quantifier
A.2 Definitions of the Other Connectives

\[ \neg(\neg P \lor \neg Q) \quad \text{and_defn} \]
\[ P \land Q \]

\[ (P \rightarrow Q) \land (Q \rightarrow P) \quad \text{bii_defn} \]
\[ P \leftrightarrow Q \]

\[ P \lor \neg P \quad \text{del_defn} \]
\[ \delta P \]

\[ \neg P \lor Q \quad \text{imp_defn} \]
\[ P \rightarrow Q \]

\[ \neg \exists x. \neg P(x) \quad \text{all_defn} \]
\[ \forall x. P(x) \]

\[ \neg \text{True} \quad \text{fal_defn} \]
\[ \text{False} \]

Table A.2: Definitions of $\land$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\delta$ and False
A.3 Derived Rules for $\neg$, $\lor$ and $\land$

\[
\begin{array}{ll}
\frac{P \land Q}{P} & \text{conjI} \\
\frac{P}{P \land Q} & \text{conjEi} \\
\frac{P \land Q}{Q} & \text{conjE2} \\
\frac{\neg P}{\neg (P \land Q)} & \text{negConjI1} \\
\frac{\neg Q}{\neg (P \land Q)} & \text{negConjI2} \\
\frac{\neg (P \land Q)}{R \quad R} & \text{negConjE} \\
\frac{P \lor Q}{Q \lor P} & \text{orComm} \\
\frac{P \lor (Q \lor R)}{(P \lor Q) \lor R} & \text{orAss} \\
\frac{P \lor (Q \land R)}{(P \lor Q) \land (P \lor R)} & \text{orAndDist} \\
\frac{\neg (P \lor Q)}{\neg P \land \neg Q} & \text{orDeM} \\
\frac{P \land (Q \lor R)}{(P \land Q) \lor (P \land R)} & \text{andOrDist} \\
\frac{\neg (P \land Q)}{\neg P \lor \neg Q} & \text{andDeM}
\end{array}
\]

Table A.3: Derived rules for $\neg$, $\lor$ and $\land$
### A.4 Derived Rules for →, ↔ and ∀

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([P])</td>
<td>(P \rightarrow Q) (\neg P)</td>
</tr>
<tr>
<td>(Q)</td>
<td>(P \rightarrow Q) (\neg P)</td>
</tr>
<tr>
<td>(P \leftrightarrow Q)</td>
<td>(\neg P) (P \rightarrow Q) (\text{vac1}^*)</td>
</tr>
<tr>
<td>(P \leftrightarrow Q)</td>
<td>(P \rightarrow Q) (\text{vac2}^*)</td>
</tr>
<tr>
<td>(P \rightarrow Q) (\neg P)</td>
<td>(\text{vac1}^*)</td>
</tr>
<tr>
<td>(P \rightarrow Q) (\text{vac2}^*)</td>
<td></td>
</tr>
<tr>
<td>(P \leftrightarrow Q)</td>
<td>((P \leftrightarrow Q) \leftrightarrow R) (P \leftrightarrow (Q \leftrightarrow R)) (\text{biiAss}^*)</td>
</tr>
<tr>
<td>(\forall x. P(x))</td>
<td>(\forall x. P(x)) (\neg P) (\text{negAllI}^*)</td>
</tr>
<tr>
<td>(\neg \forall x. P(x))</td>
<td>(\neg \forall x. P(x)) (\forall x. P(x)) (\text{negAllE}^*)</td>
</tr>
</tbody>
</table>

Table A.4: Derived rules for →, ↔ and ∀
### A.5 Rules for $\Delta$

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{P}{\Delta P}$</td>
<td>delI1</td>
<td>$\frac{\neg P}{\Delta P}$</td>
</tr>
<tr>
<td>$[P] \quad [\neg P]$</td>
<td>delE</td>
<td>$[\Delta P] \quad [\Delta P]$</td>
</tr>
<tr>
<td>$\vdots \quad \vdots$</td>
<td>$Q \quad Q$</td>
<td>$Q \quad \neg Q$</td>
</tr>
<tr>
<td>$\Delta P \quad Q \quad Q$</td>
<td></td>
<td>$\neg Q$</td>
</tr>
<tr>
<td>$\frac{\neg \Delta P \quad [\neg \Delta P]}{\Delta P \quad \neg Q}$</td>
<td>negDelE*</td>
<td>$\frac{\neg \Delta P}{\Delta P \quad \neg Q}$</td>
</tr>
</tbody>
</table>

Table A.5: Rules for the non-monotone connective $\Delta$
Appendix B

HRMI.thy
theory HRMI
imports Pure
begin
typedcl
o
consts
Trueprop :: o ⇒ prop (- 5)
consts
False :: o
op −− (infixr −→ 10)
op & (infixr & 35)
Not :: o ⇒ o (¬ - [40] 40)
notation (xsymbols)
Not (¬ - [40] 40) and
op & (infixr ∙ 35) and
op −− (infixr −→ 25) and
False (⊥)
axioms

I : A → A
T : (A → B) → ((B → C) → (A → C))
P : (A → (B → C)) → (B → (A → C))
R1 : (A → (B → C)) → (A & B → C)
R2 : (A & B → C) → (A → (B → C))
C : A → A & A
M : A & A → A
N1 : (A → ~B) → (B → ~A)
N2 : ~ ~A → A
F : False → A

MP : [ P → Q; P ] ⇒ Q
lemma ~P → P & P
apply(rule MP)
apply(rule MP)
apply(rule T)
apply(rule N2)
apply(rule C)
done
end
theory LPF
imports Pure
begin

global

classes term
defaultsort term

typedecl o

judgment
Trueprop :: o ⇒ prop

C.1 Constants and Connectives

Constants and Connectives

consts
True :: o
False :: o

op = :: ['a, 'a] ⇒ o

(infixl = 50)
\textbf{C.2 Axioms, Rules and Definitions}

Axioms, Rules and Definitions

Basic Rules

axioms

\begin{align*}
disjI1 & : \quad P \implies P \mid Q
\end{align*}
\begin{align*}
disjI2 & : \quad Q \implies P \mid Q \\
disjE & : \quad \[ P \mid Q; \quad P \implies R; \quad Q \implies R \] \implies R \\
negDisjI & : \quad \neg(P \mid Q) \implies \neg(P \mid Q) \\
negDisjE1 & : \quad \neg(P \mid Q) \implies \neg P \\
negDisjE2 & : \quad \neg(P \mid Q) \implies \neg Q \\
dNegI & : \quad P \implies \neg\neg P \\
dNegE & : \quad \neg\neg P \implies P \\
exI & : \quad P(x) \implies (EX \; x. \; P(x)) \\
exE & : \quad \[ EX \; x. \; P(x); \quad \exists! x. \; P(x) \implies Q \] \implies Q \\
negExI & : \quad \neg(EX \; x. \; P(x)) \implies \neg(P(x)) \\
negExE & : \quad \neg(EX \; x. \; P(x)) \implies \neg Q \\
notE & : \quad \[ P; \quad \neg P \] \implies Q \\
eqSubs & : \quad \[ s = t; \quad P(s) \] \implies P(t) \\
eqContr & : \quad \neg(t = t) \implies P \\
eqCons & : \quad t = t \\
negTrue & : \quad \neg True \implies P \\
trueI & : \quad True \\
eqReflx1 & : \quad (t = s) \implies (t = t) \\
eqReflx2 & : \quad (t = s) \implies (s = s) \\
negEqReflx1 & : \quad \neg(t = s) \implies (t = t) \\
negEqReflx2 & : \quad \neg(t = s) \implies (s = s) \\
eqTwoVal & : \quad \[ s = s; \quad t = t \] \implies (s = t) \mid \neg(s = t)
\end{align*}

Definitions of the Other Connectives

\begin{align*}
\text{axioms} \\
\text{fal-defnD} & : \quad \neg True \implies False \\
\text{fal-defnU} & : \quad False \implies \neg True \\
\text{and-defnD} & : \quad \neg(P \mid \neg Q) \implies P \land Q \\
\text{and-defnU} & : \quad P \land Q \implies \neg(P \mid \neg Q) \\
\text{imp-defnD} & : \quad \neg P \mid Q \implies P \rightarrow Q \\
\text{imp-defnU} & : \quad P \rightarrow Q \implies \neg P \mid Q \\
\text{bii-defnD} & : \quad (P \rightarrow Q) \land (Q \rightarrow P) \implies P \leftrightarrow Q \\
\text{bii-defnU} & : \quad P \leftrightarrow Q \implies (P \rightarrow Q) \land (Q \rightarrow P) \\
\text{all-defnD} & : \quad \neg(EX \; x. \; \neg P(x)) \implies (ALL \; x. \; P(x)) \\
\text{all-defnU} & : \quad (ALL \; x. \; P(x)) \implies \neg(EX \; x. \; \neg P(x)) \\
\text{del-defnD} & : \quad P \mid \neg P \implies \neg P \\
\text{del-defnU} & : \quad \neg P \implies P \mid \neg P
\end{align*}

C.3 Derived Rules not Proven

Derived Rules not Proven

\begin{align*}
\text{axioms}
\end{align*}
negConjI1 : \( \sim P \implies \sim (P \& Q) \)

negConjI2 : \( \sim Q \implies \sim (P \& Q) \)

negConjE : \[ \sim (P \& Q); \sim P \implies R; \sim Q \implies R \] \( \implies R \)

impE : \[ P \implies Q; \sim P \] \( \implies Q \)

biiAssD : (P \iff Q) \iff R \implies P \iff (Q \iff R)

biiAssU : P \iff (Q \iff R) \implies (P \iff Q) \iff R

allE : \[ \forall x. P(x); P(x) \implies Q \] \( \implies Q \)

negAllI : \[ \forall x. \sim P(x) \] \( \implies \sim (\forall x. P(x)) \)

negAllE : \[ \sim (\forall x. P(x)); \sim P(x) \implies Q \] \( \implies Q \)

negDelI : \[ \sim P \implies Q; \sim P \implies \sim Q \] \( \implies \sim \sim P \)

negDelE : \[ \sim \sim P \implies Q; \sim \sim P \implies \sim Q \] \( \implies \sim P \)

C.4 Derived Rules

Derived Rules

lemma conjI: assumes p:P
    assumes q:Q
    shows P \& Q
apply(rule and-defnD)
apply(rule negDisjI)
apply(rule p [THEN dNegI])
apply(rule q [THEN dNegI])
done

lemma conjE1: assumes P \& Q
    shows P
apply(insert assms)
apply(rule dNegE)
apply(rule negDisjE1)
apply(rule and-defnU)
apply(assumption)
done

lemma conjE2: assumes P \& Q
shows \( Q \)
apply\((\text{insert assms})\)
apply\((\text{rule dNegE})\)
apply\((\text{rule negDisjE2})\)
apply\((\text{rule and-defnU})\)
apply\((\text{assumption})\)
done

lemma \text{orComm}: \( P \mid Q \implies Q \mid P \)
apply\((\text{rule disjE})\)
apply\((\text{assumption})\)
apply\((\text{rule disjI2})\)
apply\((\text{assumption})\)
apply\((\text{rule disjI1})\)
apply\((\text{assumption})\)
done

lemma \text{andComm}: \( P \& Q \implies Q \& P \)
apply\((\text{rule conjI})\)
apply\((\text{rule conjE2})\)
apply\((\text{assumption})\)
apply\((\text{rule conjE1})\)
apply\((\text{assumption})\)
done

lemma \text{orAssD}: \((P \mid Q) \mid R \implies P \mid (Q \mid R)\)
apply\((\text{rule disjE, assumption})\)
apply\((\text{rule disjE})\)
apply\((\text{assumption, rule disjI1, assumption})\)
apply\((\text{rule disjI2, rule disjI1, assumption})\)
apply\((\text{rule disjI2, rule disjI2, assumption})\)
done

lemma \text{orAssU}: \( P \mid (Q \mid R) \implies (P \mid Q) \mid R \)
apply\((\text{rule disjE})\)
apply\((\text{rule orComm})\)
apply\((\text{assumption})\)
apply\((\text{rule disjE})\)
apply\((\text{assumption, rule disjI1, rule disjI2, assumption})\)
apply\((\text{rule disjI2})\)
apply\((\text{assumption})\)
apply\((\text{rule disjI1})\)
apply\((\text{rule disjI1})\)
apply\((\text{assumption})\)
done
lemma andAssD: \((P \& Q) \& R \Rightarrow P \& (Q \& R)\)
  apply(rule conjI)
  apply(rule conjE1)
  apply(rule conjE1)
  apply(assumption)
  apply(rule conjI)
  apply(rule conjE2)
  apply(rule conjE1)
  apply(assumption)
  apply(rule conjE2)
  apply(assumption)
  apply(assumption)
done

lemma andAssU: \(P \& (Q \& R) \Rightarrow (P \& Q) \& R\)
  apply(rule conjI)
  apply(rule conjI)
  apply(rule conjE1)
  apply(assumption)
  apply(rule conjE1)
  apply(rule conjE2)
  apply(assumption)
  apply(rule conjE2)
  apply(assumption)
  apply(assumption)
done

lemma orAndDistD: \(P \mid (Q \& R) \Rightarrow (P \mid Q) \& (P \mid R)\)
  apply(rule conjI)
  apply(rule disjE)
  apply(rule orComm)
  apply(assumption)
  apply(rule disjI2)
  apply(rule conjE1)
  apply(assumption)
  apply(rule disjI1, assumption)
  apply(rule disjE)
  apply(rule orComm)
  apply(assumption)
  apply(rule disjI2)
  apply(rule conjE2, assumption)
  apply(rule disjI1, assumption)
done

lemma orAndDistU: \((P \mid Q) \& (P \mid R) \Rightarrow P \mid (Q \& R)\)
  apply(rule disjE)
  apply(rule conjE1, assumption)
  apply(rule disjI1, assumption)
apply (rule disjE)
apply (rule orComm)
apply (rule conjE2, assumption)
apply (rule disjI2, rule conjI, assumption, assumption)
apply (rule disjI1, assumption)
done

lemma andOrDistD: P & (Q | R) \implies (P & Q) | (P & R)
apply (rule disjE)
apply (rule orComm)
apply (rule conjE2)
apply (assumption)
apply (rule disjI2)
apply (rule conjI)
apply (rule conjE1, assumption)
apply (assumption)
apply (rule disjI1)
apply (rule conjI)
apply (rule conjE1, assumption)
apply (assumption)
done

lemma andOrDistU: (P & Q) | (P & R) \implies P & (Q | R)
apply (rule andComm)
apply (rule conjI)
apply (rule disjE)
apply (assumption)
apply (rule disjI1)
apply (rule conjE2, assumption)
apply (rule disjI2)
apply (rule conjE2, assumption)
apply (rule disjE)
apply (assumption)
apply (rule conjE1, assumption)
apply (rule conjE1, assumption)
done

lemma orDeMD: \neg (P | Q) \implies \neg P & \neg Q
apply (rule and-defnD)
apply (rule negDisjI)
apply (rule dNegI)
apply (rule negDisjE1, assumption)
apply (rule dNegI)
apply (rule negDisjE2, assumption)
done

lemma orDeMU: \neg P & \neg Q \implies \neg (P | Q)
apply(rule negDisjI)
apply(rule conjE1, assumption)
apply(rule conjE2, assumption)
done

lemma andDeMU: ~P | ~Q =⇒ ~(P & Q)
  apply(rule disjE)
  apply(assumption)
  apply(rule negConjI1, assumption)
  apply(rule negConjI2, assumption)
done

lemma andDeMD: ~(P & Q) =⇒ ~P | ~Q
  apply(rule conjE2)
  apply(rule andOrDistU)
  apply(rule negConjE)
  apply(assumption)
  apply(rule disjI1, rule conjI, assumption, assumption)
  apply(rule disjI2, rule conjI, assumption, assumption)
done

lemma impI: [ P =⇒ Q; ~P ] =⇒ P → Q
  apply(rule imp-defnD)
  apply(rule conjE2)
  apply(rule andOrDistU)
  apply(rule disjE)
  apply(rule del-defnU)
  apply(assumption)
  apply(rule disjI2)
  apply(rule conjI, assumption, assumption)
  apply(rule disjI1)
  apply(rule conjI, assumption, assumption)
done

lemma vac1: Q =⇒ P → Q
  apply(rule imp-defnD)
  apply(rule disjI2, assumption)
done

lemma vac2: ~P =⇒ P → Q
  apply(rule imp-defnD, rule disjI1, assumption)
done

lemma bitComm: P ↔ Q =⇒ Q ↔ P
  apply(rule bit-defnD)
apply (rule andComm)
apply (rule bii-defnU, assumption)
done

lemma allI: \[ \forall x. P(x) \] \implies ALL x. P(x)
apply (rule all-defnD)
apply (rule negExI)
apply (rule dNegI)
apply (assumption)
done

lemma contrp : P \implies Q \implies \neg Q \implies \neg P
apply (rule imp-defnD)
apply (rule conjE2)
apply (rule andOrDistU)
apply (rule disjE)
apply (rule imp-defnU)
apply (assumption)
apply (rule disjI2)
apply (rule conjI)
apply (assumption, assumption)
apply (rule disjI1)
apply (rule conjI)
apply (assumption, rule dNegI, assumption)
done

lemma delI1: P \implies \nega P
apply (rule del-defnD, rule disjI1, assumption)
done

lemma delI2: \neg P \implies \nega P
apply (rule del-defnD, rule disjI2, assumption)
done

lemma delE: [ \nega P; P \implies Q; \nega P \implies Q] \implies Q
apply (rule disjE)
apply (rule del-defnU)
apply (assumption)
done

C.5 Automated Proofs

Automated Proofs
lemma \( S \mid T \implies (P \mid Q \mid R \mid S \mid T) \& (P \mid S \mid T) \)
  apply (assumption | rule conjI | rule disjI2) +
  done
end
Bibliography


