Saarland University

Faculty of Mathematics and Computer Science

Bachelor's Thesis

The Undecidability of Contextual Equivalence on PCF2 – Towards a Mechanisation in Coq

Author
Fabian Andreas Brenner

Supervisor
Prof. Dr. Gert Smolka

Advisors
Dr. Yannick Forster,
Dr. Dominik Kirst

Reviewers
Prof. Dr. Gert Smolka
Dr. Yannick Forster

Submitted: 20th August 2024

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 20th August, 2024

Abstract

PCF (Programming Computable Functions) is an idealised functional programming language introduced by Plotkin in 1977. It has been inspired by work of Scott as well as Platek and has proved to be of great importance in practice by serving as basis for the design of functional programming languages such as standard ML and Haskell. Nevertheless, the full abstraction problem, posing the question whether there exists a fully abstract model for PCF that is both concrete and independent of syntax, has been open for decades.

In 2001, Loader provided a negative answer to this problem by proving contextual equivalence undecidable for a severely restricted version of PCF called PCF_2. A solution of the full abstraction problem would entail the decidability of contextual equivalence on PCF – and in particular on PCF_2, which is why Loader's result excluded the existence of such a model. In Loader's undecidability proof, which is well-known to be full of intricate technical arguments and contains barely any examples, the undecidability of contextual equivalence on PCF_2 is deduced from that of string rewriting.

In this thesis, we point out that Loader's arguments can be turned into a chain of four many-one reductions and mechanise part of it in the Coq proof assistant in the setting of synthetic undecidability. To be precise, we mechanise observational equivalence – an equivalence relation used in Loader's proof agreeing with contextual equivalence, as well as all reductions but the first in the chain. Furthermore, we provide insightful examples and present nontrivial details that are left out in Loader's paper, making the result more accessible to non-expert readers and laying a foundation for future work mechanising the remaing reduction.

Acknowledgements

First and foremost, I would like to thank my advisors, Yannick and Dominik, for their extremely helpful support during this project. I am very grateful for the frequent and incredibly valuable feedback as well as the insightful discussions.

Of course, I also would like to thank Professor Smolka for introducing me to the field of constructive type theory and giving me the opportunity to write my Bachelor's thesis at his group.

Furthermore, I owe gratitude to Professor Groves for his exceptionally inspiring lectures, his active support, and numerous recommendation letters.

Moreover, I would like to acknowledge Professor Reineke for mentoring me during my Bachelor's studies.

I also thank all my friends for cheering my up when I was stressed. Especially, I would like to thank Ben, Christian, and Haoyi for proofreading parts of this thesis. Special thanks also goes to Janis for countless fruitful discussions and collaborations.

Last but not least, I owe my full gratitude to my family for their trust and unwavering support.

Contents

Chapter 1

Introduction

PCF (Programming Computable Functions) is an idealised functional programming language inspired by Platek [40] and introduced by Plotkin in 1977 [41] based on a famous unpublished manuscript written by Scott in 1969, which has eventually been published later [48]. It is an extension of the simply typed lambda calculus with a fixed point combinator at all types, natural numbers as base type and a match on natural numbers. In this thesis, we focus on an undecidability result of Loader [27], which provided a negative answer to a long-standing open problem concerning PCF called the full abstraction problem.

Historically, PCF has proved to be of high importance: Scott's work on PCF as model for computability and his axiomatic proof system based on PCF terms [48] formed the theoretical foundation of LCF (Logic for Computable Functions), an interactive theorem prover developed by Milner[34]. Furthermore, the design of functional programming languages such as standard ML [35] and Haskell [52] is inspired by PCF.[1]

Since the introduction of PCF, Computer scientists have been looking for decades for a – in some sense natural – model of PCF. In particular, the model should be fully abstract, i.e. two PCF terms should have the same presentation in the model if and only if they are contextually equivalent. Contextual equivalence is an equivalence relation on a calculus with two terms being in relation if and only if they have the same observational behaviour, i.e. in contexts the terms evaluate to booleans, they evaluate to the same boolean. The problem of finding a fully abstract model that is both concrete and independent of syntax is called the full abstraction problem. The first fully abstract model for PCF was given by Milner in 1977 [33], but was considered not satisfactory, as it was based on the syntax of PCF.[2] In the 1990s,

[1]"Higher-Order Computability" by Longley and Normann [29, p. 279]: "Indeed, PCF has in practice proved valuable as a basis for the design of functional programming languages such as Standard ML and Haskell."

[2]"Finitary PCF is not decidable" by Loader [27, p. 342]: "[...]it uses a term model construction

two fully abstract models for PCF have been developed using game semantics [1, 24] and Kripke logical relations [37]. However, it is not precisely defined what "concrete" and "independent of syntax" mean in this context, which is why one cannot easily point out which – if any – model is a good solution to the problem.

A criterion a sufficiently concrete model should fulfill – and which none of the above models does – is that for finitary parts of PCF, i.e. restrictions of PCF to a finite base type, the presentation in the model should be computable and the objects representing the terms should be finitely represented and belong to a decidable set. If a fully abstract model with this property existed, one could use it to decide if two terms of a finitary part of PCF have the same observational behaviour. To determine if such models can exist, Jung and Stoughton asked whether this decision problem is even decidable [25].

Surprisingly, a negative answer to this long-standing open problem has been provided: In 2001, Loader published a paper proving contextual equivalence undecidable even on a finitary part of PCF with strongly normalising reduction relation called PCF_2 [27]. In PCF_2, the base type is restricted to the booleans containing the standard boolean values as well as a third value representing an error. This result implies that no fully abstract model satisfying the previous criterion can exist, as such a model could be used to decide contextual equivalence on this severely restricted calculus. However, Loader's proof is well-known to be long and full of intricate technical arguments about the structure of certain PCF_2 terms.[3] Moreover, the original paper contains barely any examples illustrating the highly technical proof. In the proof, the undecidability of contextual equivalence on PCF_2 is deduced from that of string rewriting. String rewriting (SR), also called word problem for semi-Thue systems, is a well-known decision problem which has been introduced by Thue in 1914 [53] and proved undecidable independently by Post in 1947 [43] and Markov in 1948 [30, 31]. It is the problem, given an initial string and a target string, whether one can obtain the target string by rewriting the initial string with a fixed set of rewriting rules. For instance, if there is a rule (e, f), then a string of the form aec can be rewritten to afc.

In this thesis, we point out that Loader's arguments can be turned into a reduction chain consisting of four reductions. String rewriting does not have a similar structure as the contextual equivalence or an obvious relation to PCF_2 at all, which is why it is of particular ingenuity how to encode strings and rewriting rules into PCF_2. In fact, the main difficulty of the proof lies in establishing a connection be-

that does not tell us what mathematical structures are appropriate for modelling the calculus, and does not give useful techniques for reasoning about the model"

[3]"Higher-Order Computability" by Longley and Normann [29, p. 342]: "Whilst the theorem itself is of fundamental importance, the proof is long and technical, and consists of intricate syntactic arguments which in themselves shed little light on the nature of sequential functionals."

tween SR and certain PCF_2 terms, whereas the remaining reductions leading towards general contextual equivalence are rather straightforward.

The purpose of this thesis is both to give a formal account of Loader's proof, and introduce the reader to Loader's intricate technical arguments by providing insightful examples and explanations. Formalising such a highly technical proof on paper can easily cause errors in the details. Here, proof assistants like Coq [51], Lean [12], and Isabelle [36] come into play – computer programs designed to support in proving mathematical statements and verifying their correctness. Proof assistants can act as link between a full formalisation and the human reader: They provide both a level of informal reasoning by advanced automation as well as complex book-keeping of involved details beyond what is possible on paper with reasonable efforts. While one can be extremely sure that a theorem mechanised in a proof assistant is valid, this usually involves a lot of technical overhead as the proof needs to be given in full detail, including lemmas one would skip over in an informal proof (although they might not be easy to prove!). Even if there is no doubt that a result is correct, it can be worthwhile work to mechanise it, as this enforces choosing definitions very carefully to reduce technicalities as well as understanding the very details of the proofs, and thus may lead to simplifications of proofs and statements.

For our mechanisation, we work with the Coq proof assistant [51]. It is – as many proof assistants – based on intuitionistic type theory, which goes back to Martin-Löf [32]. To be precise, the formalism behind Coq is the Calculus of Inductive Constructions (CIC) [38, 39], which extends the Calculus of Constructions [9]. When inspecting the meta-theory carefully, it turns out every function definable in CIC is computable – a function defined in CIC is essentially an executable program. Furthermore, note that Coq's logic is intuitionistic, which means the law of excluded middle LEM $:= \forall P : \mathbb{P}.\ P \lor \neg P$ is not assumed.

There are two main approaches to formalising computability theory in a proof assistant: First, one could follow the textbook development and choose a concrete model of computation, e.g. Turing machines. While this approach is possible in proof assistants, it would require many technical results about the expressiveness of the specific model, e.g. mechanising each Turing machine used to prove a function computable. Second, if the proof assistant relies on a meta-theory only permiting to define computable functions – as for example Coq does – one can also use a synthetic approach, which goes back to Richman and Bridges [44, 8], and was developed further by Bauer [6, 7]. In such a setting, it is natural to treat all functions as the function space of computable functions. Thus, no details about a model of computation need to be considered, facilitating the work tremendously.

While this yields a straight-forward definition of decidability and reductions, it is more diffifult when turning to undecidability: The assumption that every predicate, i.e. a function of type $X \to \mathbb{P}$ for a type X, is decidable, is consistent in axiom-free Coq as observed by Werner [56]. Thus, in order to obtain synthetic undecidability results, one needs to assume further axioms if one defines undecidability as the negation of decidability as done by Bauer [6, 7]. Instead, to maintain an axiom-free setting, one can introduce a notion of synthetic undecidability relative to a base problem informally known not to be enumerable, e.g. the complement of a fixed Halting problem, and then call a problem undecidable if its decidability implies the enumerability of the base problem.

Forster, Kirst, and Smolka [19] introduced this approach and laid the foundations of synthetic undecidability in the Coq proof assistant. Since it is usually not possible with reasonable efforts to reduce directly from the Halting problem, it is desirable to maintain a library containing many undecidable problems, such that there is a large variety of problems to reduce from. And indeed, Forster et al. created such a library, namely the Coq Library of Undecidability Proofs [20], containing undecidability proofs for many decision problems in Coq. The work in this thesis towards a mechanisation of the undecidability of contextual equivalence in PCF_2 is in fact based on the Coq Library of Undecidability Proofs.

1.1 Contribution

To the best of our knowledge, we make the first contribution towards the mechanisation of Loader's proof of the undecidability of contextual equivalence on PCF_2. We mechanised PCF_2, contextual equivalence and several properties of these two notions. In particular, we mechanised that observational equivalence agrees with contextual equivalence, which is only postulated, but essential in Loader's proof. Furthermore, we turned Loader's proof into a reduction chain consisting of four many-one reductions and mechanised the reductions appearing in this chain, except for the first reduction, which starts from the complement of SR. We attempted to mechanise the forward direction of this reduction, but due to a lack of time, could not complete it. We did not mechanise the backward direction of this reduction, which is the most laborious part of the proof, as it requires numerous syntactical arguments, and whose mechanisation therefore would involve significant overhead.

However, we present the full proof on paper, providing additional explanations and details left out in Loader's proof. We also contribute insightful examples as well as technical observations regarding the encodings of rewriting systems into PCF_2, which in our opinion shed a little more light on this highly technical, intransparent result.

Overall, we contribute a foundation for future work aiming at a complete mechanisation of Loader's result, and an introduction to Loader's proof, making it more accesible to non-expert readers.

1.2 Overview

The structure of the thesis is the following: In Chapter 2, we introduce the type-theoretical background of this project and the synthetic approach to undecidability we have taken. Chapter 3 introduces PCF_2 as well as contextual equivalence and presents several useful results about these concepts – inter alia, the observational preorder is introduced, which is used in Loader's proof to characterise contextual equivalence. In Chapter 4, we introduce the decision problems in the reduction chain from the complement of string rewriting to contextual equivalence and present all but the first reduction. In Chapter 5, we proceed with the forward direction of the remaining reduction, whereas we discuss the backward direction in Chapter 6. Chapter 7 presents related systems in which contextual equivalence is in fact decidable and outlines key differences between these systems and PCF_2. Furthermore this is where related undecidability results are discussed. In Chapter 8, we summarise our work, add further comments on our Coq mechanisation and open problems as well as on possible future work.

Chapter 2

Preliminaries

In this chapter, we introduce basic definitions from Coq's type theory and synthetic undecidability to establish a formal basis for our work.

2.1 Constructive Type Theory

We work in the constructive type theory of the proof assistant Coq [51], which is based on CIC [39, 9] and goes back to Martin-Löf [32]. In this type theory, the universe of types is denoted by $\mathbb{T}$ and there is also an impredicative subuniverse of propositions, denoted by $\mathbb{P}$. For the type of functions from X to Y, the notation $X \to Y$ is used. There is also the dependent function type $\forall x \colon X.\ T(x)$, where the return type $T(x)$ of the function depends on the input x.

The following types are frequently used in our work:

$$\begin{aligned}
n \colon \mathbb{N} &::= 0 \mid S(n) && \text{(natural numbers)}\\
\mathcal{B} &::= \mathtt{t} \mid \mathtt{f} && \text{(booleans)}\\
L \colon \mathcal{L}(X) &::= \emptyset \mid x :: L \quad \text{where } X \colon \mathbb{T},\ x \colon X && \text{(lists)}\\
X + Y &::= L(x) \mid R(y) \quad \text{where } X, Y \colon \mathbb{T},\ x \colon X, y \colon Y && \text{(sum types)}\\
X \times Y &::= (x, y) \quad \text{where } X, Y \colon \mathbb{T},\ x \colon X, y \colon Y && \text{(product types)}\\
\mathcal{O}(X) &::= \mathsf{None} \mid \ulcorner x \urcorner \quad \text{where } X \colon \mathbb{T},\ x \colon X && \text{(option types)}
\end{aligned}$$

The common arithmetic operations on natural numbers are denoted by $(+, -, \cdot)$. Now consider two lists $A, B \colon \mathcal{L}(X)$ where $X \colon \mathbb{T}$. The concatenation of A and B is denoted by $A +\!\!+ B$ and the length of A by $|A|$. The proposition $x \in A$ means that x is contained in A. For a list with elements $x_1, \ldots, x_n$, the bracket notation $[x_1, \ldots, x_n]$ is usually used throughout this thesis.

In this type theory, there are also the usual propositions and propositional connectives: truth, falsity, conjunction $(\wedge)$, disjunction $(\vee)$ and implicaton $(\to)$. Negation of a proposition p $(\neg p)$ is defined as p implies falsity and equivalence $(\leftrightarrow)$ is defined as $p \leftrightarrow q := (p \to q) \wedge (q \to p)$.

There are also existential quantifiers in the type theory, modelled as proposition $\exists (p \colon X \to \mathbb{P}) \colon \mathbb{P}$. Existential quantifiers can only be eliminated when proving a proposition.

A computational alternative for $\exists$ in the sense that it can be eliminated in arbitrary contexts is the dependent pair type $\Sigma(p \colon X \to \mathbb{T}) \colon \mathbb{T}$, which does not underly the previously mentioned restriction as it is not a proposition. Note that it does not hold in general that $\exists p$ implies Σp. When writing "there exists" it is referred to $\exists$, and to Σ when writing "one can compute".

2.2 Synthetic Undecidability Theory

In this section, the basic notions of synthetic undecidability in Coq is formally introduced, following the work of Forster, Kirst, and Smolka [19], which goes back to Richman and Bridges [44, 8] as well as Bauer [6, 7]. To model decision problems, predicates are used, i.e. functions of type $X \to \mathbb{P}$ for a type X. First, the complement of a predicate is introduced.

Definition 2.1 *Let X be a type. For a predicate P, its complement $\overline{P}$ is defined by $\overline{P}\ x := \neg P\ x$ for $x \colon X$.*

Now, one can define decidability and enumerability following the observation that every function in axiom-free CIC is computable – this is because a function defined in CIC is essentially a program coming with an algorithm of how to compute it. This permits avoiding concrete models of computation such as Turing machines.

Definition 2.2 *Let $P \colon X \to \mathbb{P}$ be a predicate on type X.*

1. *P is called decidable iff there exists a function $f \colon X \to \mathcal{B}$ such that*

$$\forall x.\ P\ x \leftrightarrow f\ x = \mathtt{t}.$$

2. *P is called enumerable iff there exists a function $f \colon \mathbb{N} \to \mathcal{O}(x)$ such that*

$$\forall x.\ P\ x \leftrightarrow \exists n.\ f\ n = \ulcorner x \urcorner.$$

3. *X is called enumerable iff there exists a function $f \colon \mathbb{N} \to \mathcal{O}(x)$ such that*

$$\forall x.\ \exists n.\ f\ n = \ulcorner x \urcorner.$$

Next, undecidability is defined following the motivation from Chapter 1. Recall that in axiom-free Coq, the assumption that every predicate is decidable is consistent due to Werner [56]. To avoid assuming axioms in Coq or rolling back to a concrete model of computation and establish undecidability results with respect to this model, one can establish a notion of synthetic undecidability relative to a fixed base problem. The chosen base problem should be widely accepted to be undecidable, e.g. known to be undecidable in the usual computational models. We follow the development of the Coq Library of Undecidability Proofs by Forster et al. [20] and choose the complement of a fixed Halting problem Halt as base problem. It is known that the chosen Halting problem is undecidable and that its complement is not enumerable. This leads to the following definition:

Definition 2.3 (Undecidability) *A predicate* $P\colon X \to \mathbb{P}$ *on* X *is called undecidable iff the assumption that it was decidable implies* $\overline{\text{Halt}}$ *to be enumerable.*

With this definition, both v and $\overline{\text{Halt}}$ can be shown undecidable.

Lemma 2.4

1. $\overline{\text{Halt}}$ *is undecidable.*
2. Halt is undecidable.

The proof of (1) relies on decidable predicates on enumerable types also to be enumerable. As Halt and thus also $\overline{\text{Halt}}$ are defined on enumerable types, the result follows. One then can obtain (2) by using the following result, which relies on the fact that a decider for P can be used to construct a decider for $\overline{P}$.

Lemma 2.5 *Let* $P\colon X \to \mathbb{P}$ *be a predicate. If* $\overline{P}$ *is undecidable, then* P *is undecidable.*

Note that when proving a problem undecidable, it is uncommon to prove the defining implication directly. A common way to prove problems undecidable in textbook presentations of undecidability is by reducing from problems already proven undecidable. Thus, it is desirable for this synthetic version of undecidability to behave well with a notion of reductions such that this technique can also be used for synthetic undecidability – so the above result forms a basis for further undecidability results. Now, the definition of many-one reductions and the relevant lemmas are briefly presented.

Definition 2.6 (Many-one reductions) *A predicate* $P\colon X \to \mathbb{P}$ *is many-one reducible to* $Q\colon Y \to \mathbb{P}$ *iff there exists a function* $f\colon X \to Y$ *such that*

$$\forall x. P\ x \leftrightarrow Q\ (f\ x).$$

In this case we write $P \leqslant_m Q$.

Note that decidability transports backwards and undecidability forwards along reductions.

Lemma 2.7 *Let* $P : X \to \mathbb{P}$, $Q : Y \to \mathbb{P}$ *be predicates.*

1. *If* Q *is decidable and* $P \leqslant_m Q$*, then* P *is decidable.*
2. *If* P *is undecidable and* $P \leqslant_m Q$*, then* Q *is undecidable.*

Hereby, (1) follows easily by considering the composition of the decision function for Q with the reduction function, and (2) follows immediately from (1) as well as Definition 2.3.

Furthermore, many-one reductions are transitive – again by composition.

Lemma 2.8 *Let* $P : X \to \mathbb{P}$, $Q : Y \to \mathbb{P}$, $R : Z \to \mathbb{P}$ *be predicates. If* $P \leqslant_m Q$ *and* $Q \leqslant_m R$*, then* $P \leqslant_m R$.

A many-one reductions from P to Q induces a many-one reduction from $\overline{P}$ to $\overline{Q}$. The reduction function of the first reduction is also a valid for the second reduction.

Lemma 2.9 *Let* $P : X \to \mathbb{P}$, $Q : Y \to \mathbb{P}$ *be two predicates. If* $P \leqslant_m Q$ *holds, then* $\overline{P} \leqslant_m \overline{Q}$ *also holds.*

Note that these simple results about many-one reductions provide together with Lemma 2.4 a technique to show a vast variety of decision problems undecidable in the synthetic setting. And in fact, this is done in the Coq Library of Undecidability Proofs by Forster et al. [20]. This library contains undecidability results for various decision problems, i.e. reduction chains from the above mentioned halting problem and its complement. It may be observed that Lemma 2.7 (2) permits to start reductions from other undecidable problems than the base problem in order to show a problem undecidable, which in many cases tremendously facilitates the proofs: One may start the reduction from a problem with a more similar structure and less technical details than the formal semantics of Turing machines.

2.3 String Rewriting (SR)

Next, string rewriting is introduced, from which the reduction chain showing contextual equivalence on PCF_2 undecidable starts. String rewriting systems are also called semi-Thue systems and go back to Thue [53], shown undecidable in textbook decidability theory independently by Post in 1947 [43] and Markov in 1948 [30, 31]. A rewriting rule consists of a pair of strings (e, f) and permits to rewrite a string of the form aec into afc.

Formally, the booleans $\mathcal{B}$ are fixed as finite alphabet, where the boolean value `t` is identified with the PCF_2 term `true` – we proceed analogously with `f`. Strings (also called words) are modelled as lists $\mathcal{L}(\mathcal{B})$ over the alphabet $\mathcal{B}$. Rewriting rules are represented by pairs of strings $\mathcal{L}(\mathcal{B}) \times \mathcal{L}(\mathcal{B})$, and string rewriting systems, which consist of finitely many rewriting rules, by a list of rewriting rules $\mathcal{L}(\mathcal{L}(\mathcal{B}) \times \mathcal{L}(\mathcal{B}))$. For the sake of legibility, ab is used as notation for $a +\!\!+ b$, the concatenation of strings a, b. Now the application of a single rewriting rule $\Rightarrow_R$ and its reflexive, transitive closure $\Rightarrow_R^*$ are formally defined.

Definition 2.10 (String rewriting relation) *The binary relations on words $\Rightarrow_R$, $\Rightarrow_R^*$ are defined by the following rules:*

$$\frac{(e, f) \in R}{aec \Rightarrow_R afc} \qquad \frac{}{a \Rightarrow_R^* a} \qquad \frac{a \Rightarrow_R^* b \quad b \Rightarrow_R c}{a \Rightarrow_R^* c}$$

Note that one equivalently could define $\Rightarrow_R^*$ with $\Rightarrow_R$ and $\Rightarrow_R^*$ swapped in the premise of the third rule, as done in the Coq Library of Undecidability Proofs[20]. However, the above definition turns out to be useful in the reduction in Chapter 5 and Chapter 6 when doing induction on the derivation of certain words.

It has been proven by Davis [10] that there is a list of rules R over any alphabet with only two letters, such that the following problem, called string rewriting, word problem or reachablility problem for semi-Thue systems, is undecidable in textbook decidability theory: [1]

$$\mathrm{SR}_R(a\colon \mathcal{L}(\mathcal{B}), b\colon \mathcal{L}(\mathcal{B})) := a \Rightarrow_R^* b$$

Davis, who worked with Turing machines as computational model, took the following apporach: He argues that Turing machines can be translated into rewriting systems on some finite alphabet Γ in the sense that the Turing machine outputs a word on a given input if and only if the word is derivable in the system from the initial word, which corresponds to the input of the Turing machine. It then easily follows that for each recursively enumerable problem, there is a string rewriting system such that a word is contained in the problem if and only if it can be derived in the system from any word of a certain form – the proof is by translating the Turing machine enumerating the problem into a string rewriting system. Also note that any rewriting system on any finite alphabet Γ can be translated into a rewriting system on any binary alphabet preserving derivability. The result is then obtained by considering a string rewriting system corresponding to an undecidable, recursively enumerable problem and then turning this system into a system on the respective binary alphabet.

String rewriting is usually presented in a version where the rules R are not fixed, but provided as an additional argument. That version of string rewriting has been independently proven undecidable in textbook decidability theory by Post in 1947 [43]

[1]"Computability and Unsolvability" by Davis[10, p. 93]: "There exists a semi-Thue system whose alphabet consists of two letters and whose decision problem is recursively unsolvable."

and Markov in 1948 [30, 31]: Both proved the word problem for Thue systems – semi-Thue systems with symmetric rewriting rules – undecidable, which implies the undecidability of the word problem for semi-Thue systems. Post achieved this by reducing from the printing problem for Turing machines, while Markov reduced from Post canonical systems [42] – a more complex string rewriting problem.

In particular, the Coq Library of Undecidability Proofs[20] contains an undecidability result about string rewriting by Forster, Heiter, and Smolka [18]. However, the version from the library differs from the version presented above: It is the version where the rewriting rules are not fixed, but provided as argument. For the remainder of this thesis, the following is assumed.

Axiom 2.11 *There exists a list of rules* $\mathsf{R} = [(\mathsf{e}_1, \mathsf{f}_1), \ldots, (\mathsf{e}_\mathsf{N}, \mathsf{f}_\mathsf{N})]$ *over the alphabet* $\mathcal{B}$ *such that* $\overline{\mathrm{SR}}$ *is undecidable, where*

$$\mathrm{SR}\colon (\mathcal{L}(\mathcal{B}) \times \mathcal{L}(\mathcal{B})) \to \mathbb{P} \ := \ \mathrm{SR}_\mathsf{R}.$$

Because of Davis' results in textbook undecidability [10] discussed in Section 2.3, it is reasonable to assume Axiom 2.11.

Why exactly fixing a set of rules is important for our purposes, becomes clear in the reduction from SATIS to PS in Section 4.1. We remark that in Loader's original proof [27], it is not only a set of rewriting rules fixed, but also an initial word. As this turns out not to be necessary, the initial word remains in our development an argument of SR.

Chapter 3

Programming Computable Functions

In this chapter, we formally introduce PCF_2. In Section 3.1, the syntax is defined, establishing which form the types and terms of PCF_2 have. Afterwards, in Section 3.2, the typing rules of PCF_2 are given, categorising well-typed terms into booleans and functions, before we fix a semantics in Section 3.3 to assign meaning to the terms, i.e. specify the reduction relation. Furthermore, in Section 3.4 contexts are defined – terms with a hole, which one can fill with a term – as well as contextual equivalence, which is an equivalence relation with two terms being equivalent if and only if they are indistinguishable in their observational behaviour. Also, an alternative characterisation of contextual equivalence using logical relations is presented, which will later be useful in Loader's undecidability proof.

3.1 Syntax

The syntax of PCF_2 is divided into two parts, namely terms and types. PCF_2 is an extension of simply typed λ-calculus (STLC) with the booleans $\mathbb{B}$ as base type and a conditional.

Definition 3.1 (PCF_2) *The types* ty *and terms* tm *of PCF_2 are defined by by*

$$\begin{aligned} T_1, T_2 &: \mathsf{ty} ::= \mathbb{B} \mid T_1 \to T_2 \\ s, t, u &: \mathsf{tm} ::= \lambda x.s \mid s\ t \mid x \mid \texttt{if}\ s\ \texttt{then}\ t\ \texttt{else}\ u \mid \mathsf{true} \mid \mathsf{false} \mid \bot \end{aligned}$$

The types of PCF_2 consist of booleans ($\mathbb{B}$) and function types ($T_1 \to T_2$). Terms consist of lambda-abstractions ($\lambda x.\ s$), function applications ($s\ t$), variables (x), conditionals ($\texttt{if}\ s\ \texttt{then}\ t\ \texttt{else}\ u$) and boolean constants true, false and $\bot$.

Here, $\mathbb{B}$ comes with the three constants true, false and $\bot$. The third constant of $\mathbb{B}$, called $\bot$, may be unexpected as the booleans usually only consist of the two standard truth values. However, in this calculus, $\bot$ is an error constant at type $\mathbb{B}$, i.e. one can think of it as placeholder for errors, relating to full PCF's ability to diverge.

Note that on paper, named syntax is used to increase readibility and call the namespace of the variables var, whereas in Coq the de Bruijn indices are used for variables, which usually leads to less technical overhead. To be precise, we use in Coq the tool "Autosubst 2" for substitutions, which has been developed by Stark, Schäfer and Kaiser [50] and is based on "Autosubst" by Schäfer, Tebbi and Smolka [45]. Therefore, lambda-abstractions have in the mechanisation the form $\lambda.t$ rather than $\lambda x.t$. The key idea behind the de Bruijn representation is, instead of attaching to each λ a name and referring to it, variables are represented by a natural number called de Bruijn index that indicates the "distance" to its binder. For example, the term $\lambda x.\lambda y.y\ x$ can be represented by $\lambda.\lambda.$(`Var 0`) (`Var 1`). This technique has been developed by de Bruijn in 1972 [11].

In this work, parallel substitutions denoted by $\sigma\colon \mathsf{var} \to \mathsf{tm}$ are used, i.e. functions from variables to terms. The notation $t[\sigma]$ is used for the term resulting from applying σ to all free variables in t. Furthermore, we follow the Barendregt convention assuming that the bound variables chosen in our terms are always different from the free variables [5]. Thus, substitutions can be assumed to be capture avoiding. The notation $\sigma[s/x]$ is used for the substitution σ extended with the mapping of binder x to term s. The identity substitution mapping each variable to itself is denoted by *id*. The single point substitution replacing variable x by term t in term s is denoted by $s[t/x]$. Note that this is notation for $s[id[t/x]]$. Similarly, the substitution of finitely many variables $x_1, \ldots, x_n$ by terms $t_1, \ldots, t_n$ in term s is denoted by $s[t_1/x_1, \ldots, t_n/x_n]$ and is notation for $s[id[t_1/x_1, \ldots, t_n/x_n]]$.

Renamings of variables, which can be seen as a special case of substitutions where each variable is mapped to another variable, are on paper modelled as functions $r\colon \mathsf{var} \to \mathsf{var}$. The notation $t[r]$ is used for the term resulting from applying r to all free variables in t.

3.2 Typing

In this section, typing rules for PCF_2 are introduced to classify the well-typed terms into booleans and (simply typed) functions based on their syntax. To achieve this, we inductively define a typing judgement $\vdash$.

For $\Gamma \vdash t\colon\ T$, we say "term t has type T in (typing) context Γ". If a term t has some type in typing context Γ then it is called t well-typed in Γ.

Hereby, the typing context keeps track of the types of variables. On paper, we model typing contexts as lists of pairs of variables and types, where each pair (x, A) in the list indicates that variable x has type A in the corresponding context. For typing contexts, we use the suggestive notation $x\colon A$ for (x, A). Note that in the mechanisation, storing the variable can be admitted and typing contexts are modelled as lists of types with the convention that the nth free variable is assumed to have the

nth element of the context as type. For example, `Var` 2 refers in the term $\lambda.$`Var` 2 to the free variable with index 1, which would in context $[\mathbb{B} \to \mathbb{B}; \mathbb{B}]$ have type $\mathbb{B}$. If a variable refers to an index greater or equal than the length of the context, the term containing the variable has no type in the context.

Definition 3.2 (Typing for PCF$_2$) *The typing judgement* $\vdash : \mathcal{L}(\mathsf{var} \times \mathsf{ty}) \to \mathsf{tm} \to \mathsf{ty} \to \mathbb{P}$ *for PCF$_2$ is defined by the following rules:*

$$\frac{x\colon T_1 :: \Gamma \vdash t\colon\ T_2}{\Gamma \vdash \lambda x.t\colon\ T_1 \to T_2} \qquad \frac{\Gamma \vdash s\colon\ T_1 \to T_2 \quad \Gamma \vdash t\colon\ T_1}{\Gamma \vdash s\ t\colon\ T_2} \qquad \frac{(x, A) \in \Gamma}{\Gamma \vdash x\colon\ A}$$

$$\frac{\Gamma \vdash s\colon\ \mathbb{B} \quad \Gamma \vdash t\colon\ \mathbb{B} \quad \Gamma \vdash u\colon\ \mathbb{B}}{\Gamma \vdash \texttt{if}\ s\ \texttt{then}\ t\ \texttt{else}\ u\colon\ \mathbb{B}} \qquad \frac{}{\Gamma \vdash \bot\colon\ \mathbb{B}}$$

$$\frac{}{\Gamma \vdash \mathsf{true}\colon\ \mathbb{B}} \qquad \frac{}{\Gamma \vdash \mathsf{false}\colon\ \mathbb{B}}$$

The rules for lambda-abstractions and function applications follow standard presentations of the STLC. For the conditional, all subterms are expected to be of type $\mathbb{B}$ and then it is also assigned type $\mathbb{B}$. It is discussed in Section 3.3 why we decided not to use a more general typing rule. The three constants true, false, $\bot$ are unconditionally of type $\mathbb{B}$.

Terms having a type in the empty context are called closed. We sometimes refer to them as programs.

The following is an important lemma about typed terms and substitutions:

Lemma 3.3 *For typing context* Γ, *type* A, *term* t *with* $\Gamma \vdash t\colon\ A$ *and substitution* σ_1, σ_2, *we have*

$$(\forall x\ A.\ (x\colon A) \in \Gamma \ \to\ \sigma_1(x) = \sigma_2(x)) \ \to\ t[\sigma_1] = t[\sigma_2].$$

The proof is by induction on the typing judgement of t.

Definition 3.4 (Typed substitutions) *The typing judgement* $\sigma\colon \Gamma' \to \Gamma$ *for substitutions is defined by:*

$$\sigma\colon \Gamma' \to \Gamma \ :=\ \forall x\ A.\ (x\colon A) \in \Gamma \ \to\ \Gamma' \vdash \sigma(x)\colon\ A$$

The intuition behind $\sigma\colon \Gamma' \to \Gamma$ is that every variable having a type in context Γ is replaced by a term that has in context Γ' the same type.

The key lemma describing typed substitutions is the following. It is proven by induction on the typing judgement of t.

Lemma 3.5 *For type* A, *typing contexts* Γ, Γ', *substitution* σ *with* $\sigma\colon \Gamma' \to \Gamma$ *and term* t *with* $\Gamma \vdash t\colon\ A$, *it holds that* $\Gamma' \vdash t[\sigma]\colon\ A$.

Similarly, one can define a typing judgement for variable renamings.

Definition 3.6 (Typed renamings) *The typing judgement* $\sigma\colon \Gamma' \to \Gamma$ *for substitutions is defined by:*

$$\sigma\colon \Gamma' \to \Gamma \;:=\; \forall x\, A.\ (x\colon A) \in \Gamma \;\to\; (r(x)\colon A) \in \Gamma'$$

As for substitutions, the following lemma holds.

Lemma 3.7 *For type* A*, typing contexts* Γ*,* Γ'*, renaming* r *with* $r\colon \Gamma' \to \Gamma$ *and term* t *with* $\Gamma \vdash t\colon A$*, it holds that* $\Gamma' \vdash t[r]\colon A$*.*

3.3 Operational Semantics

Next, we define a semantics for PCF_2 to assign meaning to the terms. The semantics we give is operational, i.e. a reduction relation $\succ$ for PCF_2 describing the evaluation steps of terms is defined. One can think of this as execution rules of a programming language.

The semantics is call-by-name, i.e. arguments of a function do not need to be evaluated before the β-reduction substituting them into the function's body may be performed.

Definition 3.8 (Operational Semantics) *The reduction relation* $\succ\colon \mathsf{tm} \to \mathsf{tm} \to \mathbb{P}$ *is inductively defined by the following rules:*

$$\frac{s \succ s'}{s\ t \succ s'\ t} \qquad \frac{t \succ t'}{s\ t \succ s\ t'} \qquad \frac{}{(\lambda\, x.s)\ t \succ s[t\backslash x]} \qquad \frac{s \succ s'}{\lambda.s \succ \lambda s'}$$

$$\frac{s \succ s'}{\mathtt{if}\ s\ \mathtt{then}\ t\ \mathtt{else}\ u \succ \mathtt{if}\ s'\ \mathtt{then}\ t\ \mathtt{else}\ u} \qquad \frac{}{\mathtt{if}\ \mathsf{true}\ \mathtt{then}\ t\ \mathtt{else}\ u \succ t}$$

$$\frac{t \succ t'}{\mathtt{if}\ s\ \mathtt{then}\ t\ \mathtt{else}\ u \succ \mathtt{if}\ s\ \mathtt{then}\ t'\ \mathtt{else}\ u} \qquad \frac{}{\mathtt{if}\ \mathsf{false}\ \mathtt{then}\ t\ \mathtt{else}\ u \succ u}$$

$$\frac{u \succ u'}{\mathtt{if}\ s\ \mathtt{then}\ t\ \mathtt{else}\ u \succ \mathtt{if}\ s\ \mathtt{then}\ t\ \mathtt{else}\ u'} \qquad \frac{}{\mathtt{if}\ \bot\ \mathtt{then}\ t\ \mathtt{else}\ u \succ \bot}$$

$$\frac{}{\mathtt{if}\ (\mathtt{if}\ s\ \mathtt{then}\ t\ \mathtt{else}\ u)\ \mathtt{then}\ v\ \mathtt{else}\ w \succ \mathtt{if}\ s\ \mathtt{then}\ (\mathtt{if}\ t\ \mathtt{then}\ v\ \mathtt{else}\ w)\ \mathtt{else}\ (\mathtt{if}\ u\ \mathtt{then}\ v\ \mathtt{else}\ w)}$$

The reflexive and transitive closure of $\succ$ *is denoted by* $\succ^\star$*.*

Firstly, in every construct (including lambda-abstractions) it is possible to execute a reduction step of any subterm. Furthermore, β-reductions are added to the semantics, whereby the single point substitution replaces the variable by the argument of the function. For the conditional, rules are added for the cases when the condition is one of the three boolean constants: In the case it is true, the conditional reduces to the term after `then` and if it is false to the term after `else`. However, if the condition is $\bot$, the conditional reduces to $\bot$, following the intuition that $\bot$ represents non-termination and the evaluation of the conditional never terminates as its condition does not. Lastly, a rule is added enabling conditionals with another conditional in their condition to shift the latter into their `then` and `else` cases in a way preserving evaluation, i.e. both terms evaluate to the same term.

Note that the operational semantics presented above slightly differs from Loader's. Loader also adds η-expansion to the semantics, i.e. a variable f of type $A \to B$ may step to $\lambda x.\ f\ x$ and a variable x of type $\mathbb{B}$ may step to `if` x `then` true `else` false, both only in a context where they are not yet expanded. This makes defining the reduction relation a lot more complicated, as both the type of the terms and the information if they are expanded must be tracked. However, this does not affect the undecidability of contextual equivalence: Contextual equivalence only involves the evaluation behaviour of closed boolean terms, which is obviously uneffected by adding η-expansions to the reduction relation. This means that two terms in the calculus with η-expansions are contextually equivalent iff this is the case in the calculus without η-expansion. Thus, the former is decidable iff the latter is. In his paper, Loader even indicates that the precise reduction relation chosen is not essential in the proof.[1]

It is said that s is normal if there is no term t such that $s \succ t$. If $s \succ^{*} t$ holds and t is normal, it is said that t is the normal form of s. Note that the only normal forms of closed boolean terms are true, false, and $\bot$ – these are called boolean values. Moreover, it is said s evaluates to v if $s \succ^{*} v$ and v is a boolean value, and in this case we write $s \Downarrow t$. Observe that this notion of evaluation only makes sense for boolean terms.

Note that although the error constant $\bot$ relates to non-termination in full PCF, it is still a normal form in PCF_2 .

While the reduction relation is not deterministic, as e.g. in conditionals each subterm can reduce, it has the following three useful properties:

Fact 3.9 *Reduction on PCF_2 has the following properties:*

1. *Weak normalisation on closed boolean terms with computable normal form, i.e. for term s s.t. $\emptyset \vdash s \colon \mathbb{B}$, one can compute a boolean value v s.t. $s \Downarrow v$.*

[1]"Finitary PCF is not decidable" by Loader [27, p. 344]: "As the calculus is strongly normalising and Church-Rosser [...], there is no need here to be overly concerned with a precise presentation of the operational semantics."

2. *Church-Rosser property (also called confluent), i.e. for terms* s, t, u *s.t.* $s \succ^\star t$, $s \succ^\star u$, *there exists a term* v *s.t.* $t \succ^\star v$ *and* $u \succ^\star v$.
3. *Type preservation, i.e. for context* Γ, *term* s, *type* A *s.t.* $\Gamma \vdash s\colon A$ *and term* t *s.t.* $s \succ^\star t$, *it holds that* $\Gamma \vdash t\colon A$.

We did not mechanise the above stronger version of weak-normalisation (1) as well as the Church-Rosser property (2) in order to focus on the more interesting results.

The relatively short proof of type preservation (3) proceeds by proving the result first for single reduction steps $\succ$ and then lifting it to multiple steps $\succ^\star$. For single reduction steps, the result is proven by induction on the $s \succ^\star t$ and the using inversion on $\Gamma \vdash s\colon A$.

For type preservation to hold, it is important that all conditionals have type $\mathbb{B}$ since all conditionals can step to $\bot$, a term of type $\mathbb{B}$ when the condition is $\bot$, regardless of their other subterms. An alternative would be to make $\bot$ typeable at any type, then conditionals of any type could be allowed. However, this would lead to a more general calculus, which is not necessary, as even in this simplification contextual equivalence turns out to be undecidable.

Observe that the Church-Rosser property implies that normal forms are unique.

Note that (1) is a stronger property than just weak normalisation on closed boolean terms. It is essential that the normal form is computable such that functions can make a case disctinction on it in the following sense.

Lemma 3.10 *There is a function of the type*

$$\forall t.\ (\emptyset \vdash t\colon\ \mathbb{B}) \to\ (t \Downarrow \mathsf{true}\ +\ t \Downarrow \mathsf{false}\ +\ t \Downarrow \bot).$$

Reduction in PCF_2 is also strongly normalising on arbitrarily typed terms, i.e. for context Γ, term s, t, type A with $\Gamma \vdash s\colon A$ and $s \succ^\star t$, there exists a term u such that t evaluates to u. But this result is not necessary for our purposes.

3.4 Contexts and Contextual Equivalence

In the next section, we aim at introducing contexts on PCF_2, which can be seen as terms with a hole, and contextual equivalence on PCF_2. Firstly, in Section 3.4.1, contexts as well as an operation to fill a term into a context are defined. Furthermore, typing rules for contexts are given, which describe the type of the resulting term when filling a context with a term of a certain type. In Section 3.4.2, contextual equivalence for PCF_2 is defined, considering two terms equivalent if they evaluate to the same boolean in all appropriate contexts. Lastly, a characterisation of contextual equivalence using the so called observational preorder is given in Section 3.4.3 – a logical relation which is essential for Loader's proof.

3.4.1 Contexts

Intuitively, a context is a PCF_2 term with exactly one hole in it. Hereby, also the entire context can be a hole, which is denoted by $\bullet$.

Definition 3.11 (Contexts) *Contexts* ctxt *on* PCF_2 *are inductively defined by the following rules:*

$$C\colon \mathsf{ctxt} ::= \bullet \mid C\ s \mid s\ C \mid \lambda x.\ C \mid \texttt{if}\ C\ \texttt{then}\ s\ \texttt{else}\ t \mid \texttt{if}\ s\ \texttt{then}\ C\ \texttt{else}\ t \mid \texttt{if}\ s\ \texttt{then}\ t\ \texttt{else}\ C$$

where $s, t\colon \mathsf{tm}$.

Next, it is straightforward to define an inductive function filling a given PCF_2 term into a context:

Definition 3.12 (Context filling) *The filling of a* PCF_2 *term* s *into a context* C *is denoted by* $C[s]$*. It is defined by the following rules.*

$$\bullet[s] := s$$

$$\begin{array}{ll} (\lambda.C)[s] := \lambda.(C[s]) & (\texttt{if}\ C\ \texttt{then}\ t\ \texttt{else}\ u)[s] := \texttt{if}\ (C[s])\ \texttt{then}\ t\ \texttt{else}\ u \\ (t\ C)[s] := t\ (C[s]) & (\texttt{if}\ t\ \texttt{then}\ C\ \texttt{else}\ u)[s] := \texttt{if}\ t\ \texttt{then}\ (C[s])\ \texttt{else}\ u \\ (C\ t)[s] := (C[s])\ t & (\texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ C)[s] := \texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ (C[s]) \end{array}$$

Similarly, one can define the notion of composition of contexts.

Definition 3.13 (Context composition) *The composition of two contexts* C*,* C' *is denoted by* $C \circ C'$*. It is defined by the following rules.*

$$\bullet \circ C' := C'$$

$$\begin{array}{ll} (\lambda.C) \circ C' := \lambda.(C \circ C') & (\texttt{if}\ C\ \texttt{then}\ t\ \texttt{else}\ u) \circ C' := \texttt{if}\ (C \circ C')\ \texttt{then}\ t\ \texttt{else}\ u \\ (t\ C) \circ C' := t\ (C \circ C') & (\texttt{if}\ t\ \texttt{then}\ C\ \texttt{else}\ u) \circ C' := \texttt{if}\ t\ \texttt{then}\ (C \circ C')\ \texttt{else}\ u \\ (C\ t) \circ C' := (C \circ C')\ t & (\texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ C) \circ C' := \texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ (C \circ C') \end{array}$$

The following is a straightforward property about context composition.

Fact 3.14 *For a* PCF_2 *term* t *and contexts* C, C'*, it holds that* $(C \circ C')[t] = C[C[t]]$.

Now, we define a typing judgement $C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_2)$ for contexts. Hereby, $C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_2)$ can be thought of intuitively as that for any term t with $\Gamma \vdash t\colon T_1$, it holds that $\Gamma' \vdash C[t]\colon T_2$. This leads to the following definition.

Definition 3.15 (Typing for contexts) *For context* C*, typing contexts* Γ*,* Γ'*, types* T_1*,* T_2*,* T_3*, and PCF*$_2$ *terms* s t*, the typing judgement for contexts is defined by the following rules:*

$$\frac{}{\bullet\colon (\Gamma, T_1) \rightsquigarrow (\Gamma, T_1)} \qquad \frac{C\colon (\Gamma, T_1) \rightsquigarrow (T_2 :: \Gamma', T_3)}{\lambda\, x.C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_2 \to T_3)}$$

$$\frac{C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_2 \to T_3) \quad \Gamma' \vdash t\colon\ T_2}{C\ t\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_3)} \qquad \frac{C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_2) \quad \Gamma' \vdash t\colon\ T_2 \to T_3}{t\ C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', T_3)}$$

$$\frac{C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', \mathbb{B}) \quad \Gamma' \vdash s\colon\ \mathbb{B} \quad \Gamma' \vdash t\colon\ \mathbb{B}}{\texttt{if}\ C\ \texttt{then}\ s\ \texttt{else}\ t\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', \mathbb{B})} \qquad \frac{\Gamma' \vdash s\colon\ \mathbb{B} \quad C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', \mathbb{B}) \quad \Gamma' \vdash t\colon\ \mathbb{B}}{\texttt{if}\ s\ \texttt{then}\ C\ \texttt{else}\ t\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', \mathbb{B})}$$

$$\frac{\Gamma' \vdash s\colon\ \mathbb{B} \quad \Gamma' \vdash t\colon\ \mathbb{B} \quad C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', \mathbb{B})}{\texttt{if}\ s\ \texttt{then}\ t\ \texttt{else}\ C\colon (\Gamma, T_1) \rightsquigarrow (\Gamma', \mathbb{B})}$$

And with this definition, the following fact holds, matching the previously presented intuition for context typings.

Fact 3.16 *If* $\Gamma \vdash t\colon\ T$ *and* $C\colon (\Gamma, T) \rightsquigarrow (\Gamma', T')$*, then* $\Gamma' \vdash C[t]\colon\ T'$*.*

3.4.2 Contextual Equivalence (CE)

Finally, we introduce contextual equivalence and give an alternative characterisation using a logical relation, which is used in the proof of its undecidability.

Definition 3.17 (Contextual Equivalence) *Contextual equivalence in typing context* Γ *at type* T *is defined as a binary relation* $\equiv_c$ *on PCF*$_2$ *terms in the following way:*

$$\Gamma \vdash s \equiv_c t\colon T \ := \ \forall C\, v.\ C\colon (\Gamma', T) \rightsquigarrow (\emptyset, \mathbb{B}) \ \to \ (C[s] \Downarrow v \leftrightarrow C[t] \Downarrow v)$$

The decision problem whether two given closed terms are contextually equivalent is called CE.

Definition 3.18 (CE) *For type* T*, and terms* s, t *with* $\emptyset \vdash s\colon\ T$ *and* $\emptyset \vdash t\colon\ T$*, the predicate* CE is defined by

$$\mathrm{CE}(s, t, T) \ := \ \emptyset \vdash s \equiv_c t\colon T.$$

Intuitively, two terms are contextually equivalent if and only if they cannot be distinguished in their observational behaviour, i.e. they always evaluate to the same term in all contexts resulting in a closed term of type boolean. Here, only booleans are considered observational results as for functions, it is not "easy to observe" if they are equal.

Fact 3.19 *For each typing context* Γ *and type* A*, the contextual equivalence is an equivalence relation when restricted to PCF*$_2$ *terms of type* A *in context* Γ*.*

3.4.3 Characterisation of Contextual Equivalence

When proving contextual equivalence undecidable, the following logical relation called observational preorder turns out to be useful, as it can be used to establish a characterisation of contextual equivalence. In this subsection, we outline the proof of this characterisation. This is of particular interest since in Loader's paper [27], this is (together with few useful results to obtain it) only stated and not proven. For the work on the logical relations in this subsection, we took inspiration from the work of Dreyer et al. [14] as well as Ahmed [2].

First, observational preorder is defined on boolean terms typed in the empty context.

Definition 3.20 (Observational Preorder) *Observational preorder on booleans is defined in multiple layers.*

1. *First, it is inductively defined for boolean terms typed in the empty context via the binary relation $\leqslant_b$:*

$$s \leqslant_b t \ ::= \ C_1(s \Downarrow \bot) \mid C_2(\exists v.\ s \Downarrow v \ \wedge \ t \Downarrow v)$$

2. *It is then lifted to arbitrarily typed terms in the empty context by induction on the type.*

$$\begin{aligned} s \leqslant_c t \colon \mathbb{B} \quad &:= s \leqslant_b t \\ s \leqslant_c t \colon A \to B \quad &:= \textit{for all } s', t' \textit{ with } \emptyset \vdash s\colon\ A, \emptyset \vdash t\colon\ A \textit{ and } s' \leqslant_c t'\colon A, \\ &\quad\ \textit{it holds that } s\ s' \leqslant_c t\ t'\colon B. \end{aligned}$$

3. *Finally, it is defined for terms of any type in any context.*

$$\Gamma \vdash s \leqslant_o t\colon A \ := \textit{for all } \sigma \textit{ with } \sigma\colon \emptyset \to \Gamma \textit{ it holds that } \sigma(s) \leqslant_c \sigma(t)\colon A.$$

Next, we would like to show that observational preorder is indeed a preorder, i.e. it is reflexive and transitive. Note that most of the results about observational preorder presented here, may first be proven for closed terms and $\leqslant_c$ (for example by induction on the type) before it is easily lifted to open terms and $\leqslant_o$. However, for the proof of reflexivity, this is not possible as then the induction hypothesis is too weak in the case of Function types. If one tries to prove it for open terms by induction on the typing judgement, the induction still does not go through: In the λ-case, one needs to proove $\Gamma \vdash (\lambda x.e)\ s \leqslant_o (\lambda x.e)\ t\colon A \to B$ for arbitrary s, t such that $\emptyset \vdash s \leqslant_o t\colon A$ with only the assumption $x\colon A, \Gamma \vdash e \leqslant_o e\colon B$. As by the definition of $\leqslant_o$ on can only substitute x by the same term on both sides of $x\colon A \vdash e \leqslant_o e\colon B$, this proof does not go through. We therefore generalise $\leqslant_o$ in the following way:

Definition 3.21 (Strong Observational Preorder) *The strong observational preorder in typing context Γ at type T is defined as binary relation $\leqslant_o'$ on PCF$_2$ terms on closed terms in the same way as $\leqslant_o$ and on open terms as:*

$$\Gamma \vdash s \leqslant_o' t \colon A := \textit{for all } \sigma_1,\ \sigma_2 \textit{ with } \sigma_1 \colon \emptyset \to \Gamma,\ \sigma_2 \colon \emptyset \to \Gamma \textit{ and}$$
$$\forall x\, A.\ (x \colon A) \in \Gamma \ \to\ \sigma_1(x) \leqslant_c \sigma_2(x) \colon A,\ \textit{we have } \sigma_1(s) \leqslant_c \sigma_2(t) \colon A$$

One can then prove the following facts to obtain reflexivity of $\leqslant_o'$:

Lemma 3.22 *For each typing context Γ, type A, and terms s, t it holds that*

1. *If $s \succ s'$, $t \succ t'$ and $\Gamma \vdash s' \leqslant_o' t' \colon A$ then $\Gamma \vdash s \leqslant_o' t \colon A$.*
2. $\Gamma \vdash t \colon\ A \ \to\ \Gamma \vdash t \leqslant_o' t \colon A$.
3. $\Gamma \vdash s \leqslant_o' t \colon A \ \to\ \Gamma \vdash s \leqslant_o t \colon A$.
4. $\Gamma \vdash t \colon\ A \ \to\ \Gamma \vdash t \leqslant_o t \colon A$.

Proof First (1) is proven for the special case $\Gamma = \emptyset$ by induction on A and then lifted to the general case, afterwards. Next, one proves (2) by induction on the typing judgement, where (1) is used in the case of lambda-abstractions. One can then conclude (4) for the special case $\Gamma = \emptyset$, which we need to prove (3). Finally, (4) can be obtained in its general form by using (2) and (3). □

Now transitivity can easily derived by induction on the type for closed terms and then lifting the result to open terms. Note that in the induction case of function types, reflexivity is used.

Lemma 3.23 *For typing context Γ, type A and terms s, t, u with $\Gamma \vdash s \colon\ A$, $\Gamma \vdash t \colon\ A$ and $\Gamma \vdash u \colon\ A$, it holds that $\Gamma \vdash s \leqslant_o t \colon A \ \to\ \Gamma \vdash t \leqslant_o u \colon A \ \to\ \Gamma \vdash s \leqslant_o u \colon A$.*

The desired result now follows easily from Lemma 3.22 (4) and Lemma 3.23.

Fact 3.24 *For each typing context Γ and type A the observational preorder is a preorder when restricted to PCF$_2$ terms of type A in context Γ.*

Now, observational equivalence is introduced, which means that observational preorder holds in both directions.

Definition 3.25 (Observational Equivalence) *For terms s, t, typing context Γ and type T we define*

$$\Gamma \vdash s \equiv_o t \colon A \ :=\ \Gamma \vdash s \leqslant_o t \colon A \ \wedge\ \Gamma \vdash t \leqslant_o s \colon A$$

From Fact 3.24, it easily follows that observational equivalence is an equivalence relation.

Lemma 3.26 *For each typing context Γ and type A the observational equivalence is an equivalence relation when restricted to PCF$_2$ terms of type A in context Γ.*

We now establish the fact that observational preorder and contextual equivalence agree, which will be used later in Loader's undecidability proof.

Theorem 3.27 (Characterisation of observational equivalence) *For a type A, typing context Γ, terms s, t with $\Gamma \vdash s\colon A$, and $\Gamma \vdash t\colon A$, the following holds:*

$$\Gamma \vdash s \equiv_c t\colon A \;\leftrightarrow\; \Gamma \vdash s \equiv_o t\colon A$$

In order to prove this theorem, the following properties about observational preorder are useful:

Fact 3.28 *Observational preorder has the following properties:*

1. *For s, t with $\emptyset \vdash s \equiv_o t\colon \mathbb{B}$, it holds that $\forall v.\, s \Downarrow v \;\leftrightarrow\; t \Downarrow v$.*
2. *For f, g of type $A \to B$ in context Γ, it holds that*
$$\Gamma \vdash f \leqslant_o g\colon A \to B \quad\leftrightarrow\quad \forall x.\, \emptyset \vdash x\colon A \;\to\; \Gamma \vdash f\,x \leqslant_o g\,x\colon B.$$
3. *For s, s' with $\Gamma \vdash s\colon A$, it holds that $s \succ s' \;\to\; \Gamma \vdash s \leqslant_o s'\colon A$.*
4. *For s, t with $\Gamma \vdash s\colon A$, $\Gamma \vdash t\colon A$ and $C\colon (\Gamma, A) \rightsquigarrow (\Gamma', B)$, it holds that*
$$\Gamma \vdash s \leqslant_o t\colon A \;\to\; \Gamma' \vdash C[s] \leqslant_o C[t]\colon B.$$

Proof (1) can be proven directly using the definition of observational preorder on $\mathbb{B}$. One proves (2) and (3) by first considering $\Gamma = \emptyset$, in which (2) is proven by induction on A and (3) can be proven directly – before lifting them to the general case. Finally, (4) is proven directly by induction on the typing judgement of the context, as restricting Γ or Γ' would not simplify the proof. □

We have now all lemmas in hand to prove that observational preorder in both directions characterises contextual equivalence.

Proof (Theorem 3.27) For the forward direction, it suffices to prove $\Gamma \vdash s \leqslant_o t\colon A$, as $\equiv_c$ is symmetric. This is first proven for $\Gamma = \emptyset$ by induction on A, which is standard. Then it is lifted to an arbitrary Γ by simulating the substitution of closed terms for the free variables of s and t with a context of lambda-abstractions capturing the free variables of s and t. Hereby, the concept of context composition is needed: One context comes from the definition of contextual equivalence (the context in which the two terms should behave the same) and the other one is the context used to capture the free variables of the terms.

For the backward direction, one obtains for any C with $C\colon (\Gamma, A) \rightsquigarrow (\Gamma', \mathbb{B})$ that $\emptyset \vdash C[s] \equiv_o C[t]\colon \mathbb{B}$ by Fact 3.28 (4). Then, the claim follows by Fact 3.16 and Fact 3.28 (1). □

The following results about observational equivalence and contextual preorder that turn out to be useful in later chapters:

Lemma 3.29 *For typing contexts Γ, Γ', type A, term t with $\Gamma \vdash t\colon A$ and substitution σ_1, σ_2 with $\sigma_1\colon \Gamma' \to \Gamma$, $\sigma_2\colon \Gamma' \to \Gamma$, it holds that*

$$(\forall x.\ (x\colon A) \in \Gamma \ \to\ \Gamma' \vdash \sigma_1(x) \equiv_c \sigma_2(x)\colon A) \ \to\ \Gamma' \vdash t[\sigma_1] \equiv_c t[\sigma_2]\colon A.$$

The proof relies on the fact that the $\leqslant_o'$ is reflexive and implies $\leqslant_o$.

Lemma 3.30 *For typing context Γ, type A, terms t, t' with $\Gamma \vdash t\colon A$ and $t \succ^\star t'$, it holds that $\Gamma \vdash t \equiv_c t'\colon A$.*

The proof is by first lifting Fact 3.28 (3) to multiple steps using a standard induction on the steps. Then, a dual version of Fact 3.28 (3) is proven, where $\leqslant_o$ is replaced by $\geqslant_o$, using the same techniques as before. The result can then again be easily to multiple steps. Finally, the claim follows from these two results.

Lemma 3.31 *For typing context Γ and term t with $t\colon \emptyset \to \mathbb{B}$, it holds that*

1. $\Gamma \vdash \mathsf{true} \leqslant_o t\colon \mathbb{B} \ \leftrightarrow\ t \Downarrow \mathsf{true} \ \leftrightarrow\ \Gamma \vdash \mathsf{true} \equiv_c t\colon \mathbb{B}$
2. $\Gamma \vdash \mathsf{false} \leqslant_o t\colon \mathbb{B} \ \leftrightarrow\ t \Downarrow \mathsf{false} \ \leftrightarrow\ \Gamma \vdash \mathsf{false} \equiv_c t\colon \mathbb{B}$

This result follows immediately from the definition of $\leqslant_o$ and $\equiv_o$.

Given an inequality between two terms, it still holds after renaming the variables in the terms in the new context.

Lemma 3.32 *For typing context Γ, Γ', renaming r with $r\colon \Gamma' \to \Gamma$, type A and terms s, t with $\Gamma \vdash s \leqslant_o t\colon A$, it holds that $\Gamma' \vdash s[r] \leqslant_o t[r]\colon A$*

The proof is by using the definition of $\leqslant_o$ and considering the composition of the substitution obtained when proving the desired inequality with the renaming r.

Chapter 4

Decision problems appearing in Loader's Proof

Next, we explain the reduction chain in detail used to prove contextual equivalence on PCF_2 even for closed terms (CE) undecidable, following Loader's proof [27]:

$$\overline{\text{SR}} \leqslant_m \overline{\text{SATIS}} \leqslant_m \overline{\text{PS}} \leqslant_m \overline{\text{RPS}} \leqslant_m \text{CE}$$

According to transitivity of reductions, this chain induces a reduction from $\overline{\text{SR}}$ to CE. The undecidability of CE then follows from that of $\overline{\text{SR}}$, as undecidability is transported forwards along reductuctions and $\overline{\text{SR}}$ is undecidable. This leads to the following theorem.

Theorem 4.1 *Contextual equivalence for closed terms on PCF_2* (CE) is undecidable.

It remains to establish the reductions of the above chain. The first three of these are first established without the complements, which then induces the version with complements. So the reductions we actually present are the following:

$$\text{SR} \leqslant_m \text{SATIS} \leqslant_m \text{PS} \leqslant_m \text{RPS} \qquad \overline{\text{RPS}} \leqslant_m \text{CE}$$

In this chapter, all decision problems involved are formally defined and the proofs of all but one reduction are given, namely the reduction from string rewriting to satisfiability of words. In fact, SR is not only many-one reducible to SATIS but the two problem are even equivalent in the sense that for string a, b, it holds that $\text{SR}(a, b) \leftrightarrow \text{SATIS}(a, b)$. As this part of the proof, connecting string rewriting systems with terms in PCF_2, is by far the most intricate, we dedicate it both Chapter 5 and Chapter 6.

Note that from Theorem 4.1, the undecidability of contextual equivalence on PCF_2 for terms typed in arbitrary contexts follows.

Definition 4.2 (CE′) *For typing context* Γ*, type* T*, and terms* s*,* t *with* $\Gamma \vdash s\colon T$ *and* $\Gamma \vdash t\colon T$*, the predicate* CE′ is defined by

$$\text{CE}'(\Gamma, s, t, T) \;:=\; \Gamma \vdash s \equiv_c t\colon T.$$

Corollary 4.3 *Contextual equivalence on PCF$_2$ is undecidable for arbitrary typed terms.*

Proof It is easy to see that CE is many-one reducible to CE′ with the following reduction function:

$$f(s, t, T) \ := \ (\emptyset, s, t, T).$$

The correctness follows immediatly from the definitions of CE and CE′. Thus, the undecidability of CE′ follows from that of CE again as undecidability is transported forwards along reductions. □

In Section 4.3, it is also explained why Theorem 4.1 entails that contextual equivalence on PCF$_2$ is undecidable for arbitrary typed terms.

4.1 Reducing Satisfiability of Words to Preorder Systems

In this section, the second reduction of the chain is discussed.

$$\overline{\text{SR}} \leqslant_m \overline{\text{SATIS}} \leqslant_m \overline{\text{PS}} \leqslant_m \overline{\text{RPS}} \leqslant_m \text{CE}$$

Before giving the reduction, the two involved decision problems are defined, following Loader's original work [27]: In Section 4.1.1, the notion of satisfiability of words (SATIS) is defined, i.e. what it means for a PCF$_2$ term to satisfy an instance of SR, namely an initial string and a target string. Then, preorder sytems (PS) are introduced in Section 4.1.2, the problem whether there exists a PCF$_2$ term satisfying finitely many inequalities at once.

4.1.1 Satisfiability of Words (SATIS)

Since SATIS establishes a connection between string rewriting systems and PCF$_2$ terms, which have quite a different structure, we first explain how to encode words and rewriting rules into PCF$_2$.

Word Encodings

As the following PCF$_2$ type will regularly appear for the remainder of this thesis, the following notation is introduced:

$$\mathbb{T}(a) \ := \ \underbrace{\mathbb{B} \to \cdots \to \mathbb{B}}_{2|a|+2} \to \mathbb{B} \quad \text{where } a \text{ is a word.}$$

There are two kind of word encodings, namely true- and false-encodings.

Definition 4.4 (Word encoding) *Let $v \in [\mathsf{true}, \mathsf{false}]$. A function $Enc\colon \mathcal{L}(\mathcal{B}) \to \mathsf{tm}$ is a v-encoding iff for all words a, $\emptyset \vdash Enc(a)\colon\ \mathsf{T}(a)$ and*

$$\emptyset \vdash Enc(a) \leqslant_o \lambda x_1 \ldots x_{2|a|+2}.v\colon \mathsf{T}(a)$$

Intuitively, the above inequality means that the function *Enc*(a) only returns v and $\perp$. Hereby, each symbol of the word is represented by two of the function's arguments: The first symbol by the first two argments, the second by the next two and so on. The last two arguments of the function do not represent any symbol In particular, a word encoding may ignore the word's symbols and provide for each word any function of the correct type only returning v and $\perp$. Note that there is a duality between word encodings: Each true encoding can be transformed into a false encoding by returning for all words and arguments false if the original encoding would have returned true and $\perp$ otherwise. This check can easily be performed in PCF$_2$ with a conditional. In the same way, one can obtain a true encoding from a false encoding. We call such encodings duals. In his paper, Loader presents 16 word encoding together with their respective dual, so overall 32 encodings [27]. The list of these encodings is denoted by $\mathcal{E}$. All these encodings can be found in Appendix A. Note that most of Loader encodings – all but one encoding and its dual – ignore the symbols of the encoded words and only take the length of the word into account.

Now, some examples of word encodings are presented. To increase legibility, we do not give the explicit PCF$_2$ terms the encodings return for each word, but instead specify what these functions should evaluate to. It is rather obvious that functions with this behaviour can be constructed in PCF$_2$ by nesting conditionals.

- $\mathit{Const}_v(a)\ s_1\ \dots\ s_{2|a|}\ t_1\ t_2 = v$
  The constant encoding always returns v and never $\perp$, disregarding all arguments.

- $$\mathit{PosOdd}_v(a)\ s_1\ s_1'\ \dots\ s_{|a|}\ s_{|a|}'\ t_1\ t_2 = \begin{cases} v & \forall k.\ 1 \leqslant k \leqslant |a| \\ & \to (s_k \Downarrow \mathsf{true} \lor s_k \Downarrow \mathsf{false}) \\ \perp & \text{otherwise} \end{cases}$$
  This encoding returns v iff all the arguments at odd positions evaluate to true or false, disregarding the last two arguments t_1 and t_2.

- Let $a = a_1 \dots a_n$.
  $$\mathit{Word}_v(a)\ s_1\ s_1'\ \dots\ s_n\ s_n'\ t_1\ t_2 = \begin{cases} v & \forall k.\ 1 \leqslant k \leqslant |a| \to (s_k \Downarrow a_k \land s_k' \Downarrow a_k) \\ \perp & \text{otherwise} \end{cases}$$
  The *Word*-v encodings return v iff the arguments correspond exactly to the symbols of the encoded word, disregarding the last two arguments t_1 and t_2. These are the only two encodings taking the words' symbols into account.

Rule Encodings

Next, it is introduced how to encode rewriting rules with respect to a given word encoding `Enc`.

Definition 4.5 (Rule encoding) *A PCF$_2$ term* F *encodes a rule* (e, f) *w.r.t. a* v*-encoding*

Enc iff $\emptyset \vdash F\colon\ T(e) \to T(f)$ *and it is* $\leqslant$*-minimal s.t. fulfilling the following property: For all words* a, c, *it holds that*

$$\Gamma \vdash F(\lambda y_1 \dots y_{2|e|} ij.\mathit{Enc}(aec)\, x_1 \dots x_{2|a|} y_1 \dots y_{2|e|} z_1 \dots z_{2|c|} ij) y'_1 \dots y'_{2|f|} i'j' \\ \geqslant_o \mathit{Enc}(afc)\, x_1 \dots x_{2|a|} y'_1 \dots y'_{2|f|} z_1 \dots z_{2|c|} i'j' : \mathcal{B}$$

where $\Gamma := x_1 : \mathbb{B}, \dots, x_{2|a|} : \mathbb{B},\ y'_1 : \mathbb{B}, \dots, y'_{2|f|} : \mathbb{B},\ z_1 : \mathbb{B}, \dots, z_{2|c|} : \mathbb{B},\ i' : \mathbb{B}, j' : \mathbb{B}$.

Informally speaking, an encoding F of the rule (e, f) should satisfy the following: For each word of the form afc, it should simulate the behaviour of $\mathit{Enc}(afc)$ as well as possible in the following sense: If $\mathit{Enc}(afc)$ returns ν, F must also return ν in order to fulfill the inequality and if $\mathit{Enc}(afc)$ returns $\perp$, F should return $\perp$ as often as possible in order to fulfill the minimality condition. However, F has less information than $\mathit{Enc}(afc)$ – it only knows the arguments representing f and how Enc aec behaves with the argments for a and c alredy fixed. So in order to check if the arguments representing the whole word afc are chosen in such a way that ν should be returned, only the arguments representing f can directly be checked. In order to gain information about the arguments representing a and c, it can only be checked how $\mathit{Enc}(aec)$ behaves for any arguments representing e. Note that it depends on the concrete word encoding, what deductions can be made from this about the arguments representing a and c. Therefore, the behaviour of $\mathit{Enc}(afc)$ can generally speaking only be approximated in the sense that the rule encoding might evaluate to ν even though $\mathit{Enc}(afc)$ evaluates to $\perp$. While it is thus in general not possible to achieve an equality instead of the inequality, this will be the case for all the 32 encodings that will be considered in this paper. Encodings for which the equality is achieved are called exact encodings. However, the considered encodings being exact is of no importance for the reasoning in Loader's arguments or this thesis.

Next, an example for a rule encoding is presented. Again, we do not give the explicit PCF_2 term, but specify for all arguments what it should evaluate to, and claim that a function with this behaviour can be constructed in PCF_2 by nesting conditionals.

Let $e = e_1 \dots e_{|e|}$ and $f = f_1 \dots f_{|f|}$. Then F is a rule encoding for the rule $(e,\ f)$ and the $\texttt{Word}_\nu$ encoding

$$F\ g\ s_1 s'_1 \dots s_{|f|} s'_{|f|} t_1 t_2 = \begin{cases} \nu & \forall k.\ 1 \leqslant k \leqslant |f| \to\ s_k \Downarrow f_k\ \wedge\ s'_k \Downarrow f_k \\ & \wedge\ f\ e_1 e_1 \dots e_{|e|} e_{|e|} \perp\perp \Downarrow \nu \\ \perp & \text{otherwise} \end{cases}$$

Note that in this case, the last two arguments given to the function g do not matter, as the `Word` encoding ignores these arguments completely. Furthermore, one can clearly see that g – which is assumed to represent Enc aec for some words a and

c with their arguments already fixed – will return ν iff the arguments for a and c represent these words precisely, i.e. the encoding is exact.

The following uniqueness property follows immediately from the minimality property in the definition of rule encodings.

Lemma 4.6 *Let F_1 and F_2 be rule encodings for rule b/b' and word encoding Enc. Then it holds that $\emptyset \vdash F_1 \equiv_c F_2 \colon T(b) \to T(b')$.*

While it is clear that rule encodings are unique, it is more complicated to prove that for each word encoding and rule, a respective rule encoding exists. Note that due to a lack of time we did neither mechanise the proof of the existence of rule encodings, nor Lemma 4.7 or Lemma 4.8, which are used to establish this result. Instead, we assume the existence of rule (4.9) encodings as axiom. The proof makes use of a typical argument about finiteness and minimality. First the following finiteness property is established.

Lemma 4.7 *The contextual equivalence induces at each type only finitely many equivalence classes of closed terms.*

Proof The claim is proven by induction on the type. For $\mathbb{B}$, the only equivalence classes of closed terms are these of true, false and $\perp$. Now for $A \to B$, one can assume by the induction hypothesis that A and B have only finitely many, say a and b many, equivalence classes of closed terms. Each closed function f of type $A \to B$ must assign all closed, contextual equivalent terms s and t of type A also closed, contextual equivalent terms of type B. Otherwise, s and t would not be contextual equivalent. Following this logic, $A \to B$ has at most b^a many equivalence classes of closed terms, as each class of A can be assigned to one class of B. □

The proof proceeds by arguing that for two terms satisfying all the conditions of a rule encoding except the minimality, there is a term also satisfying these conditions that is smaller or equal than both terms with respect to the observational preorder. Furthermore, there is also at least one term satisfying these conditions.

Lemma 4.8 *Let Enc be a ν encoding and (e, f) a rewriting rule.*

1. *Let F_1, F_2 of type $T(e) \to T(f)$ in the empty context such that*

$$\Gamma \vdash \mathit{Enc}(afc)x_1 \dots x_{2|a|}y'_1 \dots y'_{2|f|}z_1 \dots z_{2|c|}i'j' \leqslant_o$$
$$G\ (\lambda y_1 \dots y_{2|e|}ij.\ \mathit{Enc}(aec)x_1 \dots x_{2|a|}y_1 \dots y_{2|e|}z_1 \dots z_{2|c|}ij)y'_1 \dots y'_{2|f|}i'j' : \mathbb{B}$$

where $\Gamma := x_1 : \mathbb{B}, \dots, x_{2|a|} : \mathbb{B},\ y'_1 : \mathbb{B}, \dots, y'_{2|f|} : \mathbb{B},\ z_1 : \mathbb{B}, \dots, z_{2|c|},\ i' : \mathbb{B}, j' : \mathbb{B}$ holds for all word a, c, and both $G = F_1$ and $G = F_2$. Then there exists F of type $T(e) \to T(f)$ in the empty context such that $\emptyset \vdash F \leqslant_o F_1 : T(e) \to T(f)$, $\emptyset \vdash F \leqslant_o F_2 : T(e) \to T(f)$ and the above inequality holds for $G = F$ and all words a, c.

2. *There exists a term* F *of type* $\mathsf{T}(e) \to \mathsf{T}(f)$ *in the empty context such that the above inequality holds for* $\mathsf{G} = \mathsf{F}$. *There is a rule encoding for* (e, f) *and Enc.*

Proof 1. Let F be defined in the following way:

$$\mathsf{F} := \lambda g\, y_1' \,\ldots\, y_{2|t|}'\, i'\, j'.\, h(\mathsf{F}_1\; g\, y_1' \,\ldots\, y_{2|f|}'\, i'\, j')(\mathsf{F}_2\; g\, y_1' \,\ldots\, y_{2|f|}'\, i'\, j'), \text{ where}$$

$$h\,x\,y := \begin{cases} \mathsf{true} & x \Downarrow \mathsf{true} \wedge y \Downarrow \mathsf{true} \\ \mathsf{false} & x \Downarrow \mathsf{false} \wedge y \Downarrow \mathsf{false} \\ \bot & \text{otherwise} \end{cases}$$

Note that h can easily be translated into PCF$_2$ by nesting conditionals. Also note that F is by the specification of h smaller or equal than F_1 and F_2. It still respects the above inequality: If F_1 returns true and F_2 returns false or the other way around, the only way both can satisfy the inequality is that the left hand side of the inequality evaluates to $\bot$. So the inequality also holds for F, as it returns $\bot$ on these arguments. In all other cases when F_1 and F_2 do not agree, one of them evaluates to $\bot$ and so does F, which must then also fulfill the inequality.

2. The term $\mathsf{F} := \lambda g\, y_1' \,\ldots\, y_{2|f|}'\, i'\, j'.\nu$ has the required properties: It is obvious that it has the appropriate type in the empty context. Furthermore, note that when plugging it in the relevant inequality, the right hand side always evaluate to ν. According to Lemma 3.30 and Lemma 3.23 it then suffices to prove

$$\Gamma \vdash \mathit{Enc}(afc)x_1 \ldots x_{2|a|}y_1' \ldots y_{2|f|}'z_1 \ldots z_{2|c|}i'j' \leqslant_o \nu \colon \mathbb{B}.$$

This inequality easily follows from Definition 4.4. □

To obtain a minimal encoding for the rule (e, f) with respect to a word encoding *Enc*, only finitely many terms need to be considered: There are only finitely many equivalence classes of closed terms at all types and all terms in one class are contextually equivalent. As rule encodings are only unique up to contextual equivalence, it suffices to consider one term of each equivalence class at type $\mathsf{T}(e) \to \mathsf{T}(f)$. For each of these finitely many terms, excluded middle is used to analyse whether it has all the required properties which a rule encoding of the rule (e, f) should have except for the miniality. It is clear from Lemma 4.8 (2) that there is at least one term fulfilling these conditions. If there is exactly on such term, the minimality follows immediately. Otherwise, Lemma 4.8 (1) is used to obtain a minimal term by choosing an arbitrary of the terms fulfilling these conditions as the preliminary minimal element. Then Lemma 4.8 (1) is applied to the current preliminary minimal element and each of the relevant terms, updating the prelimary minimal element after each application. It is clear that the resulting term also satisfies the minimality property and thus is a rule encoding.

Corollary 4.9 *Let Enc be a* ν *encoding and* (e, f) *a rewriting rule. Then there exists a rule encoding* F *of* (e, f) *with respect to Enc.*

Note that this result does not imply that rule encodings are computable – the proof uses representants of equivalence classes without computing them. According to Loader, it is possible but tedious to compute rule encodings for all of the fixed word encodings and not necessary for this proof.[1]

Now, it is defined what it means for a term to satisfy a word:

Definition 4.10 *A term* t *is said to satisfiy a word* b *with respect to word encoding Enc and initial word* a *iff* t *is normal and for all rule encodings* $F_1, \dots, F_N$ *of the rules* $(e_1, f_1), \dots, (e_N, f_N)$ *with respect to Enc, it holds that*

$$x_1 : \mathbb{B}, \dots, x_{2|b|+2} : \mathbb{B} \vdash t[F_1/r_1, \dots, F_N/r_N, \mathit{Enc}(a)/w] \geqslant_o \mathit{Enc}(b)\ x_1 \dots x_{2|b|+2} : \mathbb{B},$$

where $r_1, \dots, r_N, w$ *are fixed variables.*

Note that the above inequality holds for all rule encodings iff this is the case for any rule encodings of the appropriate rules, as rule encodings for the same rule and word encoding are contextually equivalent. Note further that for a ν-encoding, the term ν trivially satisfies each word. This is obviously not desirable, as it is intended to characterise when a word is derivable from an initial word. To avoid this, one can say a term needs to satisfy the word with respect to all 32 of Loader's encodings, which contain both true and false encodings.

The following is a lemma typing the substitution introduced in the previous definition. It will be important in Section 5.2.

Lemma 4.11 *For word encoding Enc, rule encodings* $F_1, \dots, F_N$ *of the rewriting rules* $(e_1, f_1), \dots, (e_N, f_N)$ *with respect to Enc, and word* a*, it holds that*

$$\mathit{id}[F_1/r_1, \dots, F_N/r_N, \mathit{Enc}(a)/w] : x_1 : \mathbb{B}, \dots, x_{2|b|+2} \to \Gamma,$$

where $\Gamma := x_1 : \mathbb{B}, \dots, x_{2|b|+2} : \mathbb{B},\ r_1 : T(e_1) \to T(f_1), \dots, r_N : T(e_N) \to T(f_N),\ w : T(a).$

We finally have all notions at hand to present the definition of SATIS:

Definition 4.12 (SATIS)

$$\mathrm{SATIS}(a, b) \ := \ \exists t.\ \Gamma \vdash t : \ \mathcal{B} \ \wedge \ \forall \mathit{Enc} \in \mathcal{E}.\ t \text{ satisfies } b \text{ w.r.t. } \mathit{Enc} \text{ and } a$$

where $\Gamma := x_1 : \mathbb{B}, \dots, x_{2|b|+2} : \mathbb{B},\ r_1 : T(e_1) \to T(f_1), \dots, r_N : T(e_N) \to T(f_N),\ w : T(a)$

[1]"Finitary PCF is not decidable" by Loader [27, p. 347]: "It is possible, although tedious, to calculate the required encoding of rules."

4.1.2 Preorder Systems (PS)

Next, preorder systems are introduced, the decision problem to which SATIS is reduced to.

Definition 4.13 *A well-formed preorder system for type* T *consists of a list of pairs of PCF*$_2$ *terms of the following form:*

$$[(s_1, u_1), \dots, (s_n, u_n)]$$

where for all k *with* $1 \leqslant k \leqslant n$, *it holds that* $x : T \vdash s_k \colon \mathbb{B}$ *and* $\emptyset \vdash u_k \colon \mathbb{B}$.

For each pair (s, u) with $x : T \vdash s\colon \mathbb{B}$ and $\emptyset \vdash u\colon \mathbb{B}$, one can pose the question whether there exists a closed term of type T solving the inequality $\emptyset \vdash s[t/x] \geqslant_o u\colon \mathbb{B}$. The decision problem arising from the above definition is whether all the inequalities induced by the pairs in the system are uniformly solvable, i.e. whether there exists a closed term of type T solving all of them at once. This leads to the following definition:

Definition 4.14 (PS) *For a well-formed preorder system* S *of type* T, *the predicate* PS is defined by

$$\mathrm{PS}(S, T) \;:=\; \exists t.\, \emptyset \vdash t\colon\ T \;\wedge\; \forall (s, u) \in S.\, \emptyset \vdash s[t/x] \geqslant_o u\colon \mathbb{B}.$$

Now, the reduction from SATIS to PS is presented.

Lemma 4.15 SATIS is many-one reducible to PS, i.e. SATIS $\leqslant_m$ PS.

Proof Before giving the reduction function, rule encodings for all finitely many previously fixed rules and all of Loader's 32 word encodings are obtained. In order to achieve this, one can use Lemma 4.8 (2), as the current goal is a proposition (namely that SATIS is many-one reducible to PS). The rule encoding of rule (e_k, f_k) and word encoding *Enc* obtained in this way is denoted by $F_k(\mathit{Enc})$.

Now the reduction function can be defined:

$$f(a, b) \;:=\; (S_{a,b}, (T(e_1) \to T(f_1)) \dots (T(e_N) \to T(f_N)) \to T(a) \to T(b))$$

where $S_{a,b}$ is a preorder system containing all pairs of the following form:

$$(x\ F_1(\mathit{Enc})\ \dots\ F_N(\mathit{Enc})\ \mathit{Enc}(a)\ v_1\ \dots\ v_{2|b|+2},\ \mathit{Enc}(b)\ v_1\ \dots\ v_{2|b|+2})$$

where the v_k range over $[\mathsf{true}, \mathsf{false}, \bot]$ for $1 \leqslant k \leqslant 2|b| + 2$ and *Enc* over all 32 of Loader's word encodings. From the definition of word and rule encodings, it is clear that all terms have the appropriate types and thus each $S_{a,b}$ is well-formed.

It is left to show the correctness of the reduction function: Assume $(a, b) \in$ SATIS. Then there exists a term t such that

$$x_1 : \mathbb{B}, \ldots, x_{2|b|+2} : \mathbb{B}, r_1 : T(e_1) \to T(f_1), \ldots, r_N : T(e_N) \to T(f_N), w : T(a) \vdash t\colon\ \mathbb{B}$$

and t satisfies b with respect to a and all 32 word encodings.
Then $t' := \lambda x_1 \ldots x_{2|w|+2}\ r_1 \ldots r_N\ w.\ t$ is a solution to $S_{a,b}$: From the type of t, it is clear that t' has the correct type. Now take any pair (l, r) in $S_{a,b}$. Then $l = x\ F_1(\mathit{Enc}) \ldots F_N(\mathit{Enc})\ \mathit{Enc}(a)\ v_1 \ldots v_{2|b|+2}$ and $r = \mathit{Enc}(b)\ v_1 \ldots v_{2|b|+2}$ for some word encoding *Enc* and some $v_1, \ldots, v_N$ in $\{\mathsf{true}, \mathsf{false}, \bot\}$. It is clear that $l[t'/x]$ steps to and is thus according to Lemma 3.30 equivalent to

$$t[v_1/x_1, \ldots, v_{2|b|+2}/x_{2|b|+2}, F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w].$$

The fact that $\emptyset \vdash l \geqslant_o r\colon \mathbb{B}$ holds, follows then from the inequality obtained by the fact that t satisfies b with respect to a and *Enc*.

Conversely, if s is a solution to $S_{a,b}$, then $\emptyset \vdash s\colon\ (T(e_1) \to T(f_1)) \ldots (T(e_N) \to T(f_N)) \to T(a) \to T(b)$. Then $s' := s\ r_1 \ldots r_N\ \mathit{Enc}(a)\ x_1 \ldots x_{2|b|+2}$ is the witness proving $(a, b) \in$ SATIS: From the type of s, it is clear that s' has the correct type. It remains to show that for all word encoding *Enc*, s' satisfies b w.r.t. *Enc*. First note that according to Lemma 4.6 and Lemma 3.29, it suffices to prove the relevant inequality only for the fixed rule encoding. Now let $t_1, \ldots, t_{2|b|+2}$ be closed terms of base type. It then suffices to prove:

$$\emptyset \vdash s\ F_{\mathit{Enc}}(1) \ldots F_{\mathit{Enc}}(N)\ \mathit{Enc}(a)\ t_1 \ldots t_{2|b|+2} \geqslant_o \mathit{Enc}(b)\ t_1 \ldots t_{2|b|+2} \colon \mathbb{B}$$

As each of the t_k for $1 \leqslant k \leqslant 2|b| + 2$ steps to some $v_k \in [\mathsf{true}, \mathsf{false}, \bot]$. Accoring to the definition of $S_{a,b}$, it contains the pair

$$(x\ F_1(\mathit{Enc}) \ldots F_N(\mathit{Enc})\ \mathit{Enc}(a)\ v_1 \ldots v_{2|b|+2},\ \mathit{Enc}(b)\ v_1 \ldots v_{2|b|+2}).$$

Now the relevant inequality follows from Lemma 3.30, Lemma 3.23 and the fact that s solves $S_{a,b}$ – and thus in particular the inequality induced by the above pair. □

Note that obtaining rule encodings in the descried way is only possible, as the set of rules is fixed: Otherwise, one would receive the rules to encode as argument of the reduction function. But then, the goal would not be a proposition anymore, so the existence of rule encodings would not suffice, instead it would be necessary to compute them.

Now the reduction $\overline{\text{SATIS}} \leqslant_m \overline{\text{PS}}$ in the chain immediately follows from Lemma 2.9.

Corollary 4.16 $\overline{\text{SATIS}}$ *is many-one reducible to* $\overline{\text{PS}}$*, i.e.* $\overline{\text{SATIS}} \leqslant_m \overline{\text{PS}}$.

4.2 Reducing Preorder Systems to Restricted Preorder Systems

Next in the reduction chain comes the reduction from preorder systems to restricted preorder systems, the latter of which will be introduced in this section, before the reduction is given.

$$\overline{\mathrm{SR}} \leqslant_m \overline{\mathrm{SATIS}} \leqslant_m \overline{\mathrm{PS}} \leqslant_m \overline{\mathrm{RPS}} \leqslant_m \mathrm{CE}$$

They have a very similar structure as the preorder systems presented before, but the right hand side will be restricted.

4.2.1 Restricted Preorder Systems (RPS)

Definition 4.17 *A well-formed restricted preorder system for type T consists of a list of pairs of PCF$_2$ terms of the following form:*

$$[(s_1, b_1), \ldots, (s_n, b_n)]$$

where for all k with $1 \leqslant k \leqslant n$, we have $x : T \vdash s_k : \mathbb{B}$ and $b_k : \mathcal{B}$.

RPS is defined in a similar way as PS:

Definition 4.18 (RPS) *For a well-formed restricted preorder system S of type T, we define*

$$\mathrm{RPS}(S, T) \; := \; \exists t. \emptyset \vdash t : \; T \wedge \forall (s, b) \in S. \emptyset \vdash s[t/x] \geqslant_o \tilde{b} : \mathbb{B}.$$

where $\tilde{b}$ is the embedding of b into PCF$_2$, i.e. $\mathtt{t}$ is mapped to true and $\mathtt{f}$ to false.

Next, the reduction from PS to RPS is presented, which relies on Lemma 3.30, which implies that inequalities holding for the normal form of a term also hold for the term itself.

Lemma 4.19 PS is many-one reducible to RPS, i.e. PS $\leqslant_m$ RPS.

Proof The reduction function is the following: Let $S = [(s_1, u_1), \ldots, (s_n, u_n)]$ be a well-formed preorder system. Recall that u_k for $1 \leqslant k \leqslant n$ is a closed term of type boolean. Thus, due to Lemma 3.10, one can check for each k with $1 \leqslant k \leqslant n$ whether u_k evaluates to true, false or $\bot$. If it evaluates to true, (s_k, true) is added to the resulting restricted preorder system, if it evaluates to false, (s_k, false) is added, and if it evaluates to $\bot$, nothing is added.

It remains to show the correctness, namely that the initial system is solvable iff the restricted system is solvable. Note that it suffices to show the slightly stronger statement that a term t is a solution to the initial system iff it is a solution for the restricted system.

Now assume t solves the initial preorder system. Then for each k with $1 \leqslant k \leqslant n$, it holds that $\emptyset \vdash s_k[t/x] \geqslant_o u_k \colon \mathbb{B}$. Then the same inequalities hold for t with u_k in normal form as from Lemma 3.30 it follows that each term is equivalent to its normal form. In particular, t also solves the resulting restricted system.

Conversely, assume t solves the resulting system. It remains to show that for k with $1 \leqslant k \leqslant n$ it holds $\emptyset \vdash s_k[t/x] \geqslant_o u_k \colon \mathbb{B}$. One now makes a case distinction on the normal form of u_k using Lemma 3.10: If u_k evaluates to $\perp$, then the respective inequality holds trivially. Otherwise, it evaluates to true or false, and it follows that $(s_k, \mathtt{t})$ or $(s_k, \mathtt{f})$ is in the restricted system, respectively. As t solves this system and a term is acoording to Lemma 3.10 equivalent to its normal form, the desired inequality follows. Thus, t also solves the original system, which closes the proof. □

Now the reduction $\overline{\text{PS}} \leqslant_m \overline{\text{RPS}}$ in the chain immediately follows from Lemma 2.9.

Corollary 4.20 $\overline{\text{PS}}$ *is many-one reducible to* $\overline{\text{RPS}}$*, i.e.* $\overline{\text{PS}} \leqslant_m \overline{\text{RPS}}$*.*

4.3 Reducing $\overline{\text{RPS}}$ to Contextual Equivalence

Now, the reduction from $\overline{\text{RPS}}$ to CE is discussed, which is last in the reduction chain.

$$\overline{\text{SR}} \leqslant_m \overline{\text{SATIS}} \leqslant_m \overline{\text{PS}} \leqslant_m \overline{\text{RPS}} \leqslant_m \text{CE}$$

In the proofs, $\equiv_o$ is used instead of $\equiv_c$. In Section 3.4.3, it has been shown that both notions are equivalent.

Finally, $\overline{\text{RPS}}$ is reduced to CE. It will only be left to prove the undecidability of SATIS by reducing from SR in order to conclude undecidability of CE.

Lemma 4.21 $\overline{\text{RPS}}$ *is many-one reducible to* CE*.*

Proof The reduction function is defined in the following way:

$$f([(s_1, b_1), \dots, (s_n, b_n)], T) := (\lambda x.\ G_n\ s_1\ \dots\ s_n, \lambda x.\ \perp, T \to \mathbb{B})$$

where G_n is a program taking n arguments of base type with base type as return type s.t. for closed boolean terms $t_1, \dots, t_n$, it holds that $G_n\ t_1\ \dots\ t_n \Downarrow$ true iff for all k with $1 \leqslant k \leqslant n$, it holds that $t_k \Downarrow b_k$, and $G_n\ t_1\ \dots\ t_n \Downarrow \perp$, otherwise. Such a G_n can be constructed by nesting conditionals.

It remains to show the correctness, namely that a well-formed restricted preorder system $[(s_1, b_1), \dots, (s_n, b_n)]$ of type T is not solvable iff $\emptyset \vdash \lambda x.\ G_n\ s_1\ \dots\ s_n \equiv_c \lambda x.\ \perp \colon T \to \mathbb{B}$.

Assume the system is not solvable. Then there is no closed term t of type T such that all for all k with $1 \leqslant k \leqslant n$, it holds that $\emptyset \vdash s_i[t/x] \geqslant_o \tilde{b_i} \colon \mathbb{B}$. Since a closed boolean term is greater or equal than true iff it evaluates to true (the same holds for false), it follows that for all closed terms t of type T substituted for x, at least one s_k does not evaluate to b_k. Thus, according to the specification of G_n, $G_n\ s_1\ \dots\ s_n$ will evaluate to $\bot$ for x substituted by any closed term t of type T. According to Lemma 3.30, it follows that $\emptyset \vdash (\lambda x.\ G_n\ s_1\ \dots\ s_n)\ t \equiv_c (\lambda x.\ \bot)\ t\colon \mathbb{B}$ for all closed terms t of type T.

Conversely, assume that $\emptyset \vdash \lambda x.\ G\ s_1\ \dots\ s_n \equiv_c \lambda x.\ \bot\colon T \to \mathbb{B}$. This means that according to Lemma 3.30 and the defintion of $\leqslant_o$ on closed terms of type $\mathbb{B}$, for all closed terms t of type T substituted for x, $G\ s_1\ \dots\ s_n$ evaluates to $\bot$. By the specification of G, it follows that at least one $s_k[t/x]$ does not evaluate to b_k. Thus, the restricted preorder system is not solvable. □

Note that our definition of RPS slightly differs from Loader's development: Instead of restricted systems of inequalities, Loader considers systems of equalities that are restricted in the same way– so in Definition 4.18, $\geqslant_o$ would be replaced by $\equiv_o$. However, according to Lemma 3.31, this is equivalent to our definition. Our defintion is beneficial for the following reasons: First, in the reduction from PS to RPS, no redundant inequality needs to be proven in the forward direction. And second, in the reduction from RPS to CE, only the evaluation behaviour of the s_k's in the system is important, and this is completely captured by the inequalities, no equivalence is necessary.

Chapter 5

Equivalence of SR and SATIS: Forward direction

Next, we aim at a reduction from SR to SATIS, which then implies a reduction from $\overline{\text{SR}}$ to $\overline{\text{SATIS}}$ – the last remaing reduction in the chain to

$$\overline{\text{SR}} \leqslant_m \overline{\text{SATIS}} \leqslant_m \overline{\text{PS}} \leqslant_m \overline{\text{RPS}} \leqslant_m \text{CE}$$

In order to achieve this reduction, it is proven that the two probems are even equivalent, i.e. for strings a and b, we have $\text{SR}(a, b) \leftrightarrow \text{SATIS}(a, b)$. From this, a reduction from SR to SATIS follows with the identity function as reduction function. In this chapter, we focus on the easier forward direction.

In Section 5.1, we give examples of how terms satisfying words are constructed from the derivation of the respective word to foster understanding of which parts of these terms represent which part of the derivation. For the sake of readibility, we leave out some technical details in the example that play no role in the forward, but only in the backward direction. Afterwards, we present Loader's proof of the forward direction in full detail in Section 5.2. Recall, that we have not completed a mechanisation of the proof.

5.1 Examples of terms satisfying words

In this section, focus on explaining how terms satisfying a word are constructed and which inequalities must be shown, not at proving the respective inequalities – the latter is done in detail in Section 5.2. We start by explaining the notational simplifications made for the example, compared to the remainder of this thesis. As explained for word encodings in Section 4.1.1, each symbol of a word is represented by two boolean variables in the terms. For this section only, each symbol is represented by one instead of two variables. Furthermore, there are – again as for word encodings in Section 4.1.1 – two additional boolean variables that represent no part of the word, and which we also omit for now. So for the derivation of word b, the term we construct in this example only contains $|b|$ boolean variables instead of $2|b| + 2$. Why the double representation of symbols and these so-called "control variables" representing no part of the word are needed for the backward direction, becomes clear in Chapter 6. Further note that in order to keep the simplifications presented in this section consistent with the formal definitions given in Section 4.1.1, the latter would need to be adapted for this section only. In particular, the rewriting rules R we fixed need to be changed to the two rules considered in this example, and the definition of T must be modified in the following way:

$$T(a) := \underbrace{\mathbb{B} \to \cdots \to \mathbb{B}}_{|a|} \to \mathbb{B}$$

The boolean arguments we removed also need to be removed in the definition of SATIS(4.12) and the definition of a term satisfying a word (4.10), as well as in the definition of word and rule encodings. The inequality for a term t to satisfy a word b with respect to initial word a and word encoding *Enc* is now the following:

$$x_1 : \mathbb{B}, \ldots, x_{|b|} : \mathbb{B} \vdash t[F_1/r_1, F_2/r_2, \mathit{Enc}(a)/w] \geqslant_o \mathit{Enc}(b)\ x_1 \ldots x_{|b|} : \mathbb{B}$$

for each of Loader's word encodings *Enc* and all rule encodings F_1, F_2 of the rules from the example with respect to *Enc*.

It is also introduced what variables have which type, such that we do not need to give appropriate typing contexts all of the time – this would be a tremendous notational blow up. Each x_l for $l : \mathbb{N}$ is a boolean variable. The variable x_l represents the lth symbol of the word the term it appears in should satisfy. The variable r_k represent an encoding of the kth rewriting rule (e_k, f_k) for $i : \mathbb{N}$ and is assumed to have type $T(e_k) \to T(f_k)$. Note that the two rewriting rules in the example are different to those fixed before. The encoding of the respective initial word a is represented by the variable w, which is expected to have type $T(a)$. Consequently, the typing contexts from the observational preorder are also omitted when they are clear from the terms.

Recall that we work with the booleans $\mathcal{B}$ as underlying alphabet. Consider the initial word a, target word b, and rewriting rules R:

$$a := \texttt{tt},\ b := \texttt{ff},\ R := [(\texttt{t}, \texttt{ff}), (\texttt{tf}, \texttt{f})]$$

Note that the target word is derivable from the initial word using the above rules in the following way, where applications of $(\texttt{t}, \texttt{ff})$ are highlighted in blue and of $(\texttt{tf}, \texttt{f})$ in green:

$$\texttt{tt} \Rightarrow_R \texttt{tff},\ \texttt{tff} \Rightarrow_R \texttt{ff}$$

Now a term satisfying the target word b with respect to any word encoding *Enc* (not only Loader's 32 ones) is constructed. Let F_1 be a rule encoding for $(\texttt{t}, \texttt{ff})$ and

Word	Term	Proof obligation
$\mathtt{tt}$	$w\, x_1\, x_2$	$\mathit{Enc}(a)\, x_1\, x_2 \geqslant_o \mathit{Enc}(\mathtt{tt})\, x_1\, x_2$
$\mathtt{tff}$	$r_1\, (\lambda y_1.\, w\, x_1\, y_1)\, x_2\, x_3$	$F_1\, (\lambda y_1.\, \mathit{Enc}(a)\, x_1\, y_1)\, x_2\, x_3 \geqslant_o \mathit{Enc}(\mathtt{tff})\, x_1\, x_2\, x_3$
$\mathtt{ff}$	$r_2\, (\lambda y_1'\, y_2'.\, r_1\, (\lambda y_1.\, w\, y_1'\, y_1)\, y_2'\, x_2)\, x_1$	$F_2\, (\lambda y_1'\, y_2'.\, F_1\, (\lambda y_1.\, \mathit{Enc}(a)\, y_1'\, y_1)\, y_2'\, x_2)\, x_1 \geqslant_o \mathit{Enc}(\mathtt{ff})\, x_1\, x_2$

Figure 5.1: Example of terms satisfying words

Derived words	Inequality chain
$\mathtt{tt}$	$F_2\, (\lambda y_1'\, y_2'.\, F_1\, (\lambda y_1.\, \mathit{Enc}(\mathtt{tt})\, y_1'\, y_1)\, \underline{y_2'\, x_2})\, x_1$
$\mathtt{t}\underline{\mathtt{ff}}$	$\geqslant_o F_2\, (\lambda y_1'\, y_2'.\, \mathit{Enc}(\mathtt{tff})\, y_1'\, \underline{y_2'\, x_2})\, \underline{x_1}$
$\underline{\mathtt{f}}\mathtt{f}$	$\geqslant_o \mathit{Enc}(\mathtt{ff})\, \underline{x_1}\, x_2$

Figure 5.2: Proof that t_3 satisfies $\mathtt{ff}$ by stepwise following the derivation of $\mathtt{ff}$

F_2 for $(\mathtt{tf}, \mathtt{f})$ with respect to *Enc* – these are needed for the inequalities to be shown for terms to satisfy a word. In order to construct appropriate terms, we follow the derivation of the target word from the initial word, i.e. terms satisfying words are constructed in the order the words appear in the derivation of b. Figure 5.1 displays in one place all these terms, which words they satisfy, and which inequalities need to be proven (with the substitution at the left hand side already performed).

Note that the terms are completely agnostic about the concrete symbols of the word, but they only consider the length of the words and at which position and in which order the rewriting rules are applied. Handling the concrete symbols is completely outsourced to the word encodings (and the corresponding rule encodings) – even among the encodings, the Word_v-encodings are the only two taking the symbols and not only the length of the word into account.

Now define

$$t_1 := w\, x_1\, x_2$$

and observe that with the variables of the above described types, it has type $\mathbb{B}$. The relevant inequality it must fulfill to satisfy a is the following:

$$t_1[F_1/r_1, F_2/r_2, \mathit{Enc}(a)/w] \geqslant_o \mathit{Enc}(a)\, x_1\, x_2$$

As performing the substitution at the left hand side yields that both sides are actually equal, the inequality follows by reflexivity, namely Lemma 3.22 (4), and thus, t_1 satisfies the initial word a.

Now t_1 is used to construct a term satisfying $\mathtt{tff}$, the next word in the derivation. Hereby, the part that comes from t_1 is underlined in blue. The first argument of w remains x_1 since the first symbol of the old word has not been replaced, and the second argument of w is renamed to y_1, since the second symbol of the old word is replaced by the applied rule and the variables y_i represent the positions where a rewriting has been performed. So if the first symbol would have been replaced instead of the second, the blue part would instead be $w\ y_1\ x_2$.

$$t_2 := r_1\ (\lambda y_1.\ \underline{w\ x_1\ y_1})\ x_2\ x_3$$

As the derivation step uses the first rule, the corresponding variable r_1 is used in the term. To this variable, which is assumed to have type $T(\mathtt{t}) \to T(\mathtt{ff})$ in the relevant context, arguments are assigned such that a term of type $\mathbb{B}$ is obtained: As first argument, a function of type $T(\mathtt{t}) = \mathbb{B} \to \mathbb{B}$ is assigned, taking one argument of type $\mathbb{B}$ and returning the renamed version of t_1. Finally, the remaining arguments of r_1 are the free variables representing the two newly inserted symbols of the new word – in this case x_2 and x_3 a the second and third are newly inserted. If instead the first symbol would have been replaced by the same rule, following the same logic, these arguments would be x_1 and x_2 instead.

After the substitution in the term $t_2[F_1/r_1, F_2/r_2, \mathit{Enc}(\mathtt{a})/w]$ has been performed, the inequality to be shown for t_2 to satisfy $\mathtt{tff}$ is the following:

$$F_1(\lambda y_1.\ \mathit{Enc}(\mathtt{a})\ x_1\ y_1)\ x_2\ x_3 \ \geqslant_o\ \mathit{Enc}(\mathtt{tff})\ x_1\ x_2\ x_3$$

In the same manner, starting from t_2, which satisfies $\mathtt{tff}$, a term satisfying the target word $\mathtt{b}$ can be constructed. The part of the term that comes from the term satisfying the previous word in the derivation is again highlighted in blue.

$$t_3 := r_2\ (\lambda y'_1\ y'_2.\ \underline{r_1\ (\lambda y_1.\ w\ y'_1\ y_1)\ y'_2\ x_2})\ x_1$$

Here, the second rule is used and thus, the number of variables representing the newly inserted part of the word is one instead of two, as this part consists of one symbol instead of two. Similarly, the function applied as first argument r_2 takes two arguments instead of one, since the part of the word being replaced consists of two symbols instead of one. The variable renaming in the part derived from t_2 is done in the same way as before: First, note that only the x_i are (possibly) renamed, and never the variables representing the symbols that have been replaced (in this case y_1), which also indicate at which positions the previous rewriting has taken place. As the first two symbols of the old word are replaced, x_1 and x_2 in t_2 are replaced by y'_1 and y'_2. Since the first symbol has not been changed by the first rewriting, the x_1 in t_2 is contained in the part derived from t_1, and thus, under the lambda-abstraction introducing y_1. Therefore, it is important to give the variables representing the symbols that have been replaced always fresh variable names to avoid variable capturing. Moreover, the third symbol in $\mathtt{tff}$ is unchanged by this rewriting, but as the length of the word has changed – namely decreased by one –, it is still renamed into x_2.

Again after performing the substitution in $t_3[F_1/r_1, F_2/r_2, \mathit{Enc}(a)/w]$, the inequality to be shown for t_2 to satisfy `tff` is the following:

$$F_2\ (\lambda y_1'\ y_2'.\ F_1\ (\lambda y_1.\ \mathit{Enc}(a)\ y_1'\ y_1)\ y_2'\ x_2)\ x_1\ \geqslant_o\ \mathit{Enc}(b)\ x_1\ x_2.$$

Figure 5.2 depicts how this inequality is obtained. Hereby, symbols that are substituted in the next derivation step are highlighted in the colour of the corresponding rule, while symbols that have been inserted by the previous derivation step are underlined in the corresponding colour. Variables representing these symbols are marked in the same way.

5.2 Proof of the forward direction

In the same manner as in Section 5.1, one can in general construct a term satisfying a derivable word. To present the proof appropriate to the definitions presented in Chapter 4 and the backward direction in Chapter 6, the notational simplifications made in Section 5.1 are now reverted and the rewriting rules fixed in Section 2.3 are considered again, instead of the rules from the example.

From the following lemma, the forward direction can immediately be derived. It shows an even stronger statement: The term t constructed does not depend on the word encoding. Thus, Loader's word encodings play no role in the forward direction. While the concrete word encodings are irrelevant for the forward direction as already observed by Loader, they are necessary for the backward direction, as discussed in Chapter 6.

Lemma 5.1 *If* $\mathrm{SR}(a, b)$ *holds, then there exists a term* t *with* $\Gamma_b \vdash t\colon \mathbb{B}$ *satisfying* b *with respect to* a *and any word encoding, where*
$\Gamma_b := x_1 : \mathbb{B},\ \ldots,\ x_{2|b|+2} \colon \mathbb{B},\ r_1 : T(e_1) \to T(f_1),\ \ldots,\ r_N : T(e_N) \to T(f_N),\ w : T(a).$

Proof As $\mathrm{SR}(a, b)$ holds, b is derivable from a with the fixed rewriting rules R. The proof is by induction on this derivation $a \Rightarrow_R^* b$.

First consider the base case $a = b$. The witness can be chosen in the following way: $t := w\ x_1\ \ldots\ x_{2|b|+2}$. It is easy to see that in context Γ_b, the term t has type $\mathbb{B}$. It is left to show that t satisfies b with respect to *Enc* and a:

For all word encodings *Enc*, rule encodings $F_1, \ldots, F_N$ of $(e_1, f_1), \ldots, (e_N, f_N)$ with respect to *Enc*, it holds that

$$\begin{aligned} & t[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w] \\ = {} & \mathit{Enc}(a)\ x_1\ \ldots\ x_{2|b|+2} \\ = {} & \mathit{Enc}(b)\ x_1\ \ldots\ x_{2|b|+2} \end{aligned}$$

where the first equality holds as the substitution is performed for the term t, and the second equality holds as $a = b$ holds. From Lemma 4.11 and Lemma 3.5, one can obtain $x_1 : \mathbb{B}, \ldots, x_{2|b|+2} \vdash t[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w] : \mathbb{B}$. It follows from Lemma 3.22 (4) that

$$x_1 : \mathbb{B}, \ldots, x_{2|b|+2} \vdash t[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(b)/w] \geqslant_o \mathit{Enc}(b)\ x_1\ \ldots\ x_{2|b|+2} : \mathbb{B}$$

which finishes the base case.

Now consider the inductive step, namely the case where $a \Rightarrow_R^* c$ and $c \Rightarrow_R b$. As inductive hypothesis, one can assume there exists a term t such that $\Gamma_c \vdash t : \mathbb{B}$ and t satisfies c with respect to *Enc* and a. From $c \Rightarrow_R b$, one knows that $c = d_1 e_k d_2$, $b = d_1 f_k d_2$ for some words d_1, d_2 and some $1 \leqslant k \leqslant N$. Now define

$$\begin{aligned} t' &:= r_k\ (\lambda y_1\ \ldots\ y_{2|e_k|} ij.\ \sigma(t))\ x_{2|d_1|+1}\ \ldots\ x_{2(|d_1|+|f_k|)+2} \text{ where} \\ \sigma &:= \mathit{id}[y_1/x_{2|d_1|+1},\ \ldots,\ y_{2|b|}/x_{2(|d_1|+|e_k|)}, \\ &\qquad x_{2(|d_1|+|f_k|)+1}/x_{2(|d_1|+|e_k|)+1},\ \ldots,\ x_{2|c|}/x_{2|b|},\ i/x_{2|c|+1},\ j/x_{2|c|+2}] \end{aligned}$$

Here, the substitution σ renames the variables of the term satisfying the last word in the derivation as described in Section 5.1: The variables of the symbols that have been replaced are substituted by the variables $y_1, \ldots, y_{2|b|}$ bound to the lambda-abstraction in t' and the variables of unchanged symbols at the right of the replaced part in the old word are substituted in such a way that the variable in the new word represents the same symbol. The latter is only important if $|e_k| \neq |f_k|$. Apart from that, the control variables in t are replaced by i and j, which are also bound by the lambda-abstraction in t'. This was not necessary in the example since we decided to omit these.

It is again not difficult to check that t' has type $\mathbb{B}$ in context Γ_c. Now take any word encodings *Enc*, rule encodings $F_1, \ldots, F_N$ of $(e_1, f_1), \ldots, (e_N, f_N)$ with respect to *Enc*. Now, it is shown that t' satisfies b with respect to *Enc* and a:

Note that it holds that

$$\begin{aligned} & t'[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w] \\ &= F_k\ (\lambda y_1\ \ldots\ y_{2|e_k|} ij.\ (t[\sigma])[\sigma'])\ x_{2|d_1|+1}\ \ldots\ x_{2(|d_1|+|f_k|)+2} \\ &= F_k\ (\lambda y_1\ \ldots\ y_{2|e_k|} ij.\ t[\sigma''])\ x_{2|d_1|+1}\ \ldots\ x_{2(|d_1|+|f_k|)+2} \end{aligned}$$

where

$$\begin{aligned} \sigma' &:= \mathit{id}[F_1/r_1, \ldots, F_N/r_N,\ \mathit{Enc}(a)/w] \\ \sigma'' &:= \mathit{id}[F_1/r_1, \ldots, F_N/r_N,\ \mathit{Enc}(a)/w,\ y_1/x_{2|d_1|+1}, \ldots, y_{2|e_k|}/x_{2(|d_1|+|e_k|)}, \\ &\qquad x_{2(|d_1|+|f_k|)+1}/x_{2(|d_1|+|e_k|)+1}, \ldots, x_{2|c|}/x_{2|b|},\ i/x_{2|c|+1},\ j/x_{2|c|+2}] \end{aligned}$$

The first equality holds by performing the substitution and the second holds as for any variable x it holds that either σ and σ' behave both as *id* or one of them substitutes in a close value and the other behaves as *id*.

Also observe that from the fact that t satisfies c with respect to any word encoding and a together with the fact that $\sigma'' \colon \Gamma \to x_1 : \mathbb{B}, \dots, x_{2|c|+2} : \mathbb{B}$, the following inequality also holds according to Lemma 4.11 and Lemma 3.32:

$$\Gamma \vdash \sigma''(t) \geqslant_o \mathit{Enc}(c)\, x_1\ \dots\ x_{2|d_1|}\, y_1\ \dots\ y_{2|e_k|}\, x_{2(|d_1|+|b|)+1}\ \dots\ x_{2|c|}\ i\, j \colon \mathbb{B}$$

where

$$\Gamma := x_1 : \mathbb{B}, \dots, x_{2|d_1|} : \mathbb{B},\ y_1 : \mathbb{B}, \dots, y_{2|f_i|} : \mathbb{B},\ x_{2(|d_1|+|b|)+1} : \mathbb{B},\ \dots x_{2|c|} : \mathbb{B},\ i \colon \mathbb{B},\ j \colon \mathbb{B}.$$

Putting these two observations together, it follows

$$\begin{aligned} &x_1 : \mathbb{B}, \dots, x_{2|b|+2} \vdash\ t'[F_1/r_1, \dots, F_N/r_N, \mathit{Enc}(a)/w] \\ \geqslant_o\ &F_k\ (\lambda y_1\ \dots\ y_{2|e_k|} ij.\ \mathit{Enc}(c)\, x_1\ \dots\ x_{2|d_1|}\, y_1\ \dots\ y_{2|e_k|}\, x_{2(|d_1|+|b|)+1}\ \dots\ x_{2|c|}\ i\, j) \\ &x_{2|d_1|+1}\ \cdots\ x_{2(|d_1|+|f_i|)+2} \\ \geqslant_o\ &\mathit{Enc}(b)\, x_1\ \dots\ x_{2|b|+2} \end{aligned}$$

where the first inequality is obtained from Fact 3.28 (4) with the above observation by interpreting t' as context $\sigma''(t)$ is plugged in, and the second inequality follows by the defining inequality of rule encodings (4.5) after renaming the variables in it using Lemma 3.32. Thus, t' satisfies b with respect to Enc. □

As the term constructed in Lemma 5.1 does not depend on a word encoding, but satisfies b for all word encodings, the forward direction can easily deduced.

Corollary 5.2 (Forward direction) *If* SR(a, b) *then* SATIS(a, b).

Proof This immediately follows from Lemma 5.1 since $\mathcal{E}$ only contains word encodings. □

Chapter 6

Equivalence of SR and SATIS: Backward Direction

In this chapter, the focus lies on the backward direction of the equivalence of SR and SATIS, namely that for strings a and b, it holds that SATIS$(a, b) \to$ SR(a, b). This is where the most technicalities come in: Fix an initial word a as well as a target word b, and assume SATIS(a, b) holds. Thus, there exists a term t with typing

$$x_1 : \mathbb{B}, \dots, x_{2|b|+2} : \mathbb{B}, r_1 : T(e_1) \to T(f_1), \dots, r_N : T(e_N) \to T(f_N), w : T(a) \vdash t : \mathbb{B}$$

that satisfies b with resepct to a and all of Loader's word encodings. It remains to deduce that b is derivable from a using the fixed rules R. In Section 6.3, it will turn out this is fairly straightforward if t is in the form of the terms constructed in Chapter 5. A priori, one however barely knows anything about the structure of t. One thing that is known is that t must be normal according to the definition of a term satisfying a word (4.10). However, this does not suffice to deduce the derivability of b. Therefore, several simplifications are presented in Section 6.2 to obtain a new term still satisfying the same word, but with restricted structure. To make these structural reductions possible, the concrete word encodings of Loader $\mathcal{E}$ (introduced in Section 4.1.1 and listed in Appendix A) are important: If too few or not meaningful encodings would be considered, it would not be possible to obtain a term of the desired structure, e.g. if $\mathcal{E}$ only contained true-encodings, then true would satisfy every word with respect to the considered encodings (as explained in Section 4.1.1) and the backward direction would even be wrong. In Section 6.1, a technique to make deductions about the structure of certain subterms of t is described, which is both useful to obtain reduced terms and to obtain the derivability of b.

The definitions and proofs in this chapter all originate from Loader's work [27]. Note that we also did not mechanise the proofs and defintions presented here. This chapter has two main purposes: First, it should give an informal overview of the structural simplifications made to a term satisfying a word. This should prepare the interested reader for Loader's full reasoning about the simplifications [27], which is long and full of intricate technical arguments about the structure of certain PCF_2 terms. [1] Second, it is formally presented how to obtain the derivability of the target word, assuming that the term satisfying it has all previous simplifications applied. Hereby, the proof is given in more detail compared to Loader's proof, in which some nontrivial steps are left out. This may provide a basis for future mechanisations of the backward direction.

In this chapter, several notions of subterms will be introduced, namely spinal and rib subterms. The following depicts these subterms at the example of a term constructed in Section 5.1, which is now presented without notational simplifications. The spinal subterms correspond exactly to derivation steps, which is why each term is a spinal subterm of itself – it corresponds to the last derivation step. The innermost spinal subterm, which corresponds to the start of the derivation – the initial word – is called coccyx. Here, the spinal subterms corresponding to previous derivation steps are highlighted using braces – the whole term, which is as explained also a spinal subterm is not highlighted for the sake of legibility. The rib subterms, which are highlighted in green, are the boolean arguments of the spinal subterms – in this case all of these are boolean variables. Each rib subterm consists of exactly one such boolean argument, so in the example, there are 16 rib subterms.

$$t := r_2(\lambda y'_1 y'_2 y'_3 y'_4 i' j'.\, \underbrace{r_1(\lambda y_1 y_2 y_3 y_4 ij.\ \overbrace{w\, y'_1 y'_2 y_1 y_2 ij}^{\text{spinal subterm (coccyx)}}\ )\, y'_3 y'_4 x_3 x_4 i' j'}_{\text{spinal subterm}})\, x_1 x_2 x_5 x_6$$

Note that the above term is canonical in the following sense: The coccyx consists only of an application of the variable w, which represents the encoding of the initial word, to several boolean arguments – this property is called *reduced spine*. Furthermore, none of the rib subterms, which should encode at which position rewriting has been done, contains the variables w or r_k for some $1 \leqslant k \leqslant N$, which should represent the initial word and which rule has been applied, respectively – so t has *reduced ribs*. Apart from that, each of the rib subterms consists of a single boolean variable respecting certain patterns, e.g. variables with odd indices are only provided at arguments at an odd position when counting from the left – it is said that t has *sane ribs*. Moreover, each spinal subterm corresponds to one derivation step and there are no redundant spinal subterms – it is said that t is *chain reduced*. Each of the boolean variables appears exactly once in the term – thus, t is *linear*.

[1]"Higher-Order Computability" by Longley and Normann [29, p. 342]: "Whilst the theorem itself is of fundamental importance, the proof is long and technical, and consists of intricate syntactic arguments which in themselves shed little light on the nature of sequential functionals."

6.1 Descent funtions

Note that if t is of the form $w\ t_1\ \dots\ t_{2|a|+2}$ or $r_k(\lambda y_1\ \dots\ y_{2|e_k|+2}.s)t_1\ \dots\ t_{2|f_k|+2}$ for some $1 \leqslant k \leqslant N$, then deductions about the t_l for $1 \leqslant l \leqslant 2|f_k|+2$ can be made using Loader's word encodings: In the first case, by considering the *FixTrue*$_v$ encodings, one can deduce that none of the t_l is the constant false or $\bot$. This follows by the inequality obtained by the fact that t satisfies b with respect to the initial word and *Fixt*$_v$. Similarly, it can be deduced by considering the *FixFalse*$_v$ encodings that none of the t_l is the constant true or $\bot$ – so none of the t_l can be a constant at all. In the second case, the same deductions can be made by inspecting the rule encodings of (e_k, f_k) with respect to the relevant word encodings in the same manner as done in the example in Section 4.1.1.

However, when reasoning in detail about the structure of t, one also needs to apply a similar reasoning not only on the outside of the term, but also at positions deeper inside it, i.e. if $t = r_k(\lambda y_1\ \dots\ y_{2|e_k|+2}.s)t_1\ \dots\ t_{2|f_k|+2}$, this reasoning should also be applicable to s. To achieve this, one can introduce so-called descent functions.

Definition 6.1 (Descent functions) *Let Enc be a word encoding,* (e, f) *a rule and* F *a rule encoding of* (e, f) *with respect to Enc. Then* $g_1, \dots, g_{2|e|+2}$ *are called descent functions for* F *iff* $\emptyset \vdash g_l\colon\ T(e)$ *for all* $1 \leqslant l \leqslant 2|e| + 2$ *and*

$$\emptyset \vdash F \leqslant_o \lambda f\ x_1 \dots x_{2|f|+2}.f\ (g_1\ x_1\ \dots\ x_{2|f|+2}) \dots (g_{2|e|+2}\ x_1\ \dots\ x_{2|f|+2})\colon \mathbb{B}$$

The existence of descent functions can be shown by arguing about rule encodings as in the example in Section 4.1.1. In most cases, appropriate constant functions do the job.

Lemma 6.2 (Descent existence) *For each of Loader's word encodings, rule* (b, b') *and rule encoding* F*, there exist descent functions.*

The next two results are not relevant in the proof of the backward direction assuming the relevant term has all structural reductions applied, but only in Loader's proof that these reductions are in fact possible [27]. However, we still present them in this thesis, as they point out the main purpose of descent functions, namely making deductions about subterms buried inside t. The following lemma shows how this can be done if these subterms are in some sense independent of the remaining term.

Lemma 6.3 (Constant descent) *Let Enc be a word encoding,* (e_k, f_k) *with* $1 \leqslant k \leqslant N$ *a rule and* s *a term with* $\Gamma \vdash r_k(\lambda y_1\ \dots\ y_{2|e_k|+2}.\ s)t_1\ \dots\ t_{2|f_k|+2}\colon\ \mathbb{B}$*. If* $y_1\ \dots\ y_{2|e_k|+2}$ *do not occur in* s*, then*

$$\Gamma \vdash r_k\ (\lambda y_1\ \dots\ y_{2|e_k|+2}.\ s)t_1\ \dots\ t_{2|f_k|+2} \leqslant_o s\colon \mathbb{B}$$

where $\Gamma := x_1 : \mathbb{B}, \dots, x_{2|b|+2} : \mathbb{B},\ r_1 : T(e_1) \to T(f_1), \dots, r_N : T(e_N) \to T(f_N),\ w\colon T(a).$

Descent functions are used to examine the behaviour of subterms in a certain class of subterms, namely these buried inside. These subterms will now be formally introduced. They are intended to represent a derivation step each, which is indicated by the respective variable r_k or w, which represents an application of the rule (e_k, f_k) or the encoding of the initial word, respectively.

Definition 6.4 (Spinal sub-terms) *Spinal subterms are defined in the following way:*

- *A term* s *is a spinal sub-term of itself.*
- *Spinal sub-terms of* s *are also spinal sub-terms of* $r_k(\lambda y_1 \ldots y_{2|e_k|+2}.s)t_1 \ldots t_{2|f_k|+2}$ *where* $1 \leqslant k \leqslant N$ *and* r_k *is the variable representing* (e_k, f_k)

The following lemma now shows how descent functions allow making deductions about spinal subterms in general.

Lemma 6.5 (Repeated descent) *Let Enc be a word encoding,* t *a term with*

$$x_1 : \mathbb{B}, \ldots, x_{2|b|+2} : \mathbb{B},\ r_1 : T(e_1) \to T(f_1), \ldots, r_N : T(e_N) \to T(f_N),\ w : T(a) \vdash t : \ \mathbb{B}$$

and σ *a substitution of closed terms for* x_l *where* $1 \leqslant l \leqslant |b| + 2$*. Assume for a value* v *it holds that*

$$\emptyset \vdash t[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w] \equiv_c v : \mathbb{B}$$

and for every spinal subterm of t *of the form* $r_k(\lambda y_1 \ldots y_{2|e_i|+2}.s)\ t_1 \ldots t_{2|f_i|+2}$ *for some* $1 \leqslant k \leqslant N$*, there are descent functions for* F_k *with*

$$\sigma(y_m) = g_m\ (t_1[\sigma]) \ldots (t_{2|f_k|+2}[\sigma])\ \textit{for each}\ 1 \leqslant m \leqslant 2|e_k| + 2$$

Then it holds for each spinal subterm u *of* t *that*

$$\emptyset \vdash u[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w] \equiv_c v : \mathbb{B}$$

6.2 Structural reductions

The aim of this section is to prepare the correctness proof of the backward direction by constructing a new term t' also satisfying b with respect to a and all of Loader's word encodings, but having the structure of the terms presented in Chapter 5, which allows deducing the derivation of b from a as discussed in Section 6.3. In order to obtain such a term, several structural redutions are introduced such that for each term satisfying a word with all previous reductions applied, there is a term with the same properties that is also reduced with respect to the next reduction. We do not present Loader's proofs [27] of how these terms are obtained in detail, but rather focus on giving an overview of all the relevamt simplifications and exmplaining the high-level ideas.

First, spinal reduction is introduced, stating a term's coccyx has a certain canonical form.

Definition 6.6 (Spine reduction) *For a term* t*, the following is defined:*

- *The coccyx of* t *is the unique spinal sub-term of* t *that does not have the form*

$$r_k(\lambda y_1 \ \dots \ y_{2|e_k|+2}.s)t_1 \ \dots \ t_{2|f_k|+2}$$

- *A term is said to have reduced spine if its coccyx has the form* $w \ t_1 \ \dots \ t_{2|a|+2}$

The spinal reduction makes sure that the coccyx corresponds to the beginning of the derivation and thus represents the initial word. Note that the the terms constructed in Chapter 5 have reduced spine: They are constructed following the structure of the derivation, which always starts by constructing a term s satisfying the initial word of the form $s := w \ x_1 \ \dots \ x_{2|a|+2}$. Since terms corresponding to previous derivation steps are spinal subterms of terms satisfying words coming later in the derivation, it is obvious from the structure of s that it is the coccyx. Furthermore, it is also easy to see that all terms having s as coccyx have reduced spine.

In Loader's proof, it is pointed out that the coccyx of a term satisfying a word can neither be a boolean constant, a boolean variable, nor of the form `if` x `then` s `else` t where x is a boolean variable. However, the coccyx could for example have the form `if` $w \ t_1 \ \dots \ t_{2|a|+2}$ `then` s `else` t or `if` $r_i \ f \ t_1 \ \dots \ t_{2|f_i|+2}$`then` s `else` t. Loader showed that in this case the coccyx can be replaced by $w \ t_1 \ \dots \ t_{2|a|+2}$ respectively $r_i \ f \ t_1 \ \dots \ t_{2|f_i|+2}$ – the conditional can be removed – and the resulting term still satisfies the same word. Note that the coccyx must be a normal as all terms satisfying words are normal by definition. This severly restricts the form the coccyx can have.

While the boolean arguments of the terms constructed in Chapter 5 were only variables, they underly so far no restriction at all. So in particular, the variables w or any r_k for $1 \leqslant k \leqslant N$, which encode the derivation of the word, i.e. which rule is applied at which derivation step, could appear in the boolean arguments representing the symbols of the word, which is unintended behaviour.

Definition 6.7 (Rib sub-terms) *For a spine reduced term* t*, the following is introduced:*

- *If* $t = w \ t_1 \ \dots \ t_{2|a|+2}$*, its rib sub-terms are* $t_1, \dots, t_{2|a|+2}$
- *If* $t = r_k(\lambda y_1 \ \dots \ y_{2|e_k|+2}.s)t_1 \ \dots \ t_{2|f_k|+2}$ *for some* $1 \leqslant k \leqslant N$*, then its rib sub-terms are* $t_1, \dots, t_{2|f_i|+2}$ *as well as the rib sub-terms of* s

Note that for the above definition, it is important that t has reduced spine, otherwise the two previous cases may not be exhaustive, and thus the spinal subterms not well-defined.

A term is said to have reduced ribs if the variables representing the initial word and the rewriting rules do not occur in the boolean arguments of w or any r_k for $1 \leqslant k \leqslant N$, which consist of arguments representing the symbols of the word and two technical additional arguments.

Definition 6.8 (Reduced ribs) *A term* t *with reduced spine is said to have reduced ribs if neither* w *nor any* r_i *for* $1 \leqslant i \leqslant N$ *occurs in the rib sub-terms of* t.

The idea to make term t satisfying a word b with respect to initial word a and all of Loader's word encodings rib reduced is the following: If only ν-encodings´would be considered, all occurences of w and any r_k with appropriate arguments in the rib subterms applied could be replaced by ν: First note that ν-encodings and rule encodings with respect to ν-encodings only return $\perp$ or ν. For the first, this is by definition and for the latter, this holds as otherwise, the minimality condition in the definition of rule encodings would not be fulfilled. Thus, one obtains

$$x_1 \ \ldots \ x_{2|b|+2} \vdash \mathit{Enc}(a)\ s_1 \ \ldots \ s_{2|a|+2} \leqslant_o \nu\colon \mathbb{B} \text{ and}$$
$$x_1 \ \ldots \ x_{2|b|+2} \vdash F_k\ f\ t_1 \ \ldots \ t_{2|f_k|+2} \leqslant_o \nu\colon \mathbb{B}$$

for rule encoding F_k of (e_k, f_k) and appropriate closed terms as arguments. According to Fact 3.28 (4), this would only make the resulting term bigger with respect to the observational preorder and thus, it would still satisfy b. However, both true and false encodings are considered, so it is not clear with which value ro replace the occurences of w and r_k – both at the same time is obviously not possible. The following lemma, using that Loader's word encodings consist of 16 pairs of dual encodings resolves this problem:

Lemma 6.9 *If* t *is spine and rib reduced and satisfies* b *with respect to initial word* a *and* ν*-encoding Enc. Then* t *also satisfies* b *with respect to initial word* a *and the dual of Enc.*

This lemma states that the above described procedure still works, in spite of the fact that both true- and false-encodings are considered. Applying the procedure for $\nu = \mathsf{true}$ yields a rib and spine reduced term t' satisfying all of Loader's 16 true-encodings. As the remaining encodings are the duals of the 16 considered encodings, the desired result can be obtained by Lemma 6.9.

Note that for Lemma 6.9, it is crucial that the term t is spine and rib reduced. If this would not be necessary, it would be possible to prove that true satisfies b with respect to a and all of Loader's 32 encodings, which is clearly not spine reduced and satisfies b with respect to a all true-encodings. This is obviously absurd and would even make the backward direction of the reduction from SR to SATIS wrong.

Next, a classification of the rib subterms into different classes is introduced.

Definition 6.10 (Classification) *Consider terms of the form* $w\ t_1\ \dots\ t_{2|a|+2}$ *and* $r_k(\lambda y_1\ \dots\ y_{2|e_k|+2}.s)t_1\ \dots\ t_{2|f_k|+2}$ *for* $1 \leqslant k \leqslant N$. *The arguments are classified in the following way:*

- t_{2l-1} *for* $1 \leqslant l \leqslant |a| + 1$ *are odd arguments*
- t_{2l} *for* $1 \leqslant l \leqslant |a| + 1$ *are even arguments*
- $t_1 \dots t_{2|a|}$ *are positional arguments*
- $t_{2|a|+1}, t_{2|a|+2}$ *are control arguments*

Variables (both bound and the free variables) are classified in the same manner.

It turns out that using the *PosOdd*$_\nu$, *PosEven*$_\nu$, *ConOdd*$_\nu$ and *ConEven*$_\nu$ encodings, for a rib and spine reduced term satisfying a word with respect to all of Loader's encodings, a new term can be constructed that is also rib and spine reduced, satisfies the same word with respect to all of Loader's encodings and respects the above classification in the following sense: Boolean variables of any class only appear in arguments of the same class.

Moreover, Loader shows that for a spine and rib reduced term respecting the above classification and satisfying a word with respect to all 32 considered encodings, each rib subterm contains exactly one variable of the appropriate class and is even equivalent to this variable. This leads to the following definition.

Definition 6.11 (Rib sanity) *A term* t *is said to have sane ribs if each of its rib subterms is a variable of the same class.*

While the previous simplifications aimed at obtaining a term with coccyx and ribs of a certain canonical structure, the following reduction establishes that each spinal subterm truly corresponds to one step in the derivation of the word. In the following way, redundant applications of r_k not corresponding to a derivation step could have been added to satisfying terms, as explained by Loader [27]:

Assume for words c_1 and c_2, ν-encoding *Enc*, rule encoding F_k of (e_k, f_k) with $1 \leqslant k \leqslant N$ and closed boolean terms $s_1, \dots, s_{2|c_1|}, t'_1, \dots, t'_{2|f_k|}, u_1, \dots, u_{2|c_2|}, i', j'$, it holds that

$$\emptyset \vdash F_k(\lambda t_1 \dots t_{2|e_k|} ij.\mathit{Enc}(a)\, s_1 \dots s_{2|c_1|} t_1 \dots t_{2|e_k|} u_1 \dots u_{2|c_2|} ij) t'_1 \dots t'_{2|f_k|} i'j' \equiv_c \nu \colon \mathbb{B}.$$

It then follows by the fact that $\emptyset \vdash \mathit{Enc}(a)\ s_1 \dots s_{2|c_1|} t_1 \dots t_{2|e_k|} u_1 \dots u_{2|c_2|} ij \leqslant_o \nu \colon \mathbb{B}$ (immediately from Definition 4.4) and Fact 3.28 (4) that

$$\emptyset \vdash F_i(\lambda y''_1 \dots y''_{2|e_k|} i''j''.s) t'_1 \dots t'_{2|b'|} i'j' \equiv_c \nu \colon \mathbb{B}$$

where p denotes the left hand side of the above equiality and $y''_1, \dots, y''_{2|e_k|}, i'', j''$ are free in s.

It clearly follows by Lemma 3.26 that $\emptyset \vdash p \equiv_c p' \colon \mathbb{B}$ where p' denotes the left hand side of the latter equality.

Now, the control variables come in to track and then remove possible redundant rib subterms in the following way.

Definition 6.12 (Chain reduction) *A term t is chain reduced iff for each spinal subterm in the form $r_k(\lambda y_1 \ \dots \ y_{2|e_k|+2}.s)t_1 \ \dots \ t_{2|f_k|+2}$ for $1 \leqslant k \leqslant N$, it holds that $t_{2|f_k|+2} = y_{2|e_k|+2}$.*

In a chain reduced term with all previous reduction applied, one can observe that even control variables have unique occurences: Each even control variable bound by a lambda-abstraction in any of the rib subterms appears at least once as the term is chain reduced – namely as boolean argument in the rib subterm it is bound to. Because the term has sane spine, there is only one position left an even control variable can appear in, namely as argument of the coccyx as argument in even control position. But as Loader showed that a term satisfying a word b must contain all $2|b| + 2$ free boolean variables, the free even control variable must take up this place.

In this setting, it can be deduced using the Lin_ν-encodings that all variables have unique occurences. Interestingly, no new term need to be constructed, but this is simply a statement holding for each term with all the previous reductions applied.

Lemma 6.13 (Linearity) *If t satisfies b and has all the previous reductions applied, then each x_l for $1 \leqslant l \leqslant 2|b| + 2$ occurs exactly once in t. In this case, t is said to be linear.*

The following lemma summarises all previously discussed structural restrictions of a term satisfying a word:

Lemma 6.14 *Assume there exists a term t satisfying b with resepct to a and all of Loader's word encodings with*

$$x_1 \colon \mathbb{B}, \dots, x_{2|b|+2} \colon \mathbb{B},\ r_1 \colon T(e_1) \to T(f_1), \dots, r_N \colon T(e_N) \to T(f_N),\ w \colon T(a) \vdash t \colon\ \mathbb{B}$$

Then there exists a term t' also satisfying b with resepct to a and all of Loader's word encodings with the same type as t and with all previous simplifications applied, i.e. t' is spine reduced, rib reduced, has sane ribs, is chain reduced, and is linear.

6.3 Correctness Proof

We now have all notions in hand to present Loader's proof of the backward direction and thus obtain a many-one reduction from SR to SATIS. Recall that we have not mechanised the proofs presented here. As this chapter aims, inter alia, at preparing the reader for Loader's paper, the proofs of the results in this section, namely Lemma 6.17 and Theorem 6.18, are very close to Loader's proofs – almost literal citations from Loader [27]. This is not the case for the sections coloured in blue. These are remarks and further explanation of details left out by Loader, which should support future work on the mechanisation of this result.

The following lemma will be useful in the inductive step in the proof of the backward direction.

Lemma 6.15 *For each word encoding Enc of Loader's encodings except for the Lin_ν encodings, it follows that*

$$\emptyset \vdash F\ g \leqslant_o \mathit{Enc}\ f\colon T(f)$$

where F is a rule encoding of the rule (e, f) and g is a closed term of type $T(e)$.

In the proof of the above lemma, it is verified $\lambda g.\ \mathit{Enc}\ f$ fulfills the inequation in the definition of rule encodings. As a rule encoding is a minimal term satisfying it, the claim follows.

Furthermore, the following result showing the existence of descent functions with certain additional properties will be crucial in the inductive step of the backward direction:

Lemma 6.16 (Descent completeness) *Take a ν-encoding of Loader's word encodings, a rewriting rule (e, f), a rule encoding F of (e, f) with respect to Enc, words $c = d_1 e d_2$, $c' = d_1 f d_2$ and terms s_l for $1 \leqslant k \leqslant 2|c| + 2$ with*

$$\emptyset \vdash \mathit{Enc}(c)\ s_1\ \dots\ s_{2|c|+2} \equiv_c \nu\colon \mathbb{B}.$$

Then there exist closed terms of boolean type t_l with $1 \leqslant l \leqslant 2|f| + 2$ and descent functions $g_1, \dots, g_{2|e|+2}$ for F such that

$$\emptyset \vdash s_m \equiv_c g_n\ t_1\ \dots\ t_{2|f|+2}\colon \mathbb{B}$$

holds for $2|d_1| + 1 \leqslant m \leqslant 2(|d_1| + |e|)$, $n = m - 2|d_1|$ and $2|c| + 1 \leqslant m \leqslant 2|c| + 2$, $n = m - 2|c|$, as well as

$$\emptyset \vdash \mathit{Enc}(c')\ s_1\ \dots\ s_{2|d_1|}\ t_1\ \dots\ t_{2|f|}\ s_{2(|d_1|+|e|+1)}\ \dots\ s_{2|c|}\ t_{2|f|+1}\ t_{2|f|+2} \equiv_c \nu\colon \mathbb{B}.$$

The following lemma essentially covers the main reasoning in the base case of the backward direction. It will be used to prove that a term consisting of only one spinal subterm with all the previous reductions applied and satisfying a word, is actually in the canonical form of the term satisfying the initial word from Chapter 5.

Lemma 6.17 *For a term* $t = w\, x_{\rho(1)}\ \dots\ x_{\rho(2|a|)}\ x_{2|b|+1}\ x_{2|b|+2}$ *satisfying a word* b *with respect to initial word* a *and all of Loader's word encodings, where* ρ *is a permutation of the numbers* $1, \dots, 2|a|$*, the following holds: The permutation* ρ *is the identity permutation and it holds that* $t = w\, x_1\ \dots\ x_{2|a|}\ x_{2|b|+1}\ x_{2|b|+2}$.

Proof The reasoning relies on inspecting the inequalities of the form

$$x_1 : \mathbb{B}, \dots, x_{2|b|+2} : \mathbb{B} \vdash t[F_1/r_1, \dots, F_N/r_N, \mathit{Enc}(a)/w] \geqslant_o \mathit{Enc}(b)\ x_1 \dots x_{2|b|+2} : \mathbb{B}.$$

obtained from the definition of satisfiability of words and considering appropriate substitutions of closed variables for the free variables of t. If a substitution σ is chosen in such a way that the right hand side evaluates to v with respect to a specific word encoding, this must also be the case for the left hand side for the inequality to hold. But as the left hand side is equal to

$$\mathit{Enc}(b)\ \sigma(x_{\rho(1)})\ \dots\ \sigma(x_{\rho(2|a|)})\ \sigma(x_{2|b|+1})\ \sigma(x_{2|b|+2}),$$

this allows making deductions about the positions of the x_l in t for $1 \leqslant l \leqslant 2|b|$.

Let $1 \leqslant p, q \leqslant |a|$ be given. Consider the substitution with $\sigma(x_{2q-1}) = \sigma(x_{2q}) :=$ true and $\sigma(x) :=$ false for all other boolean variables x. By considering the PosEq_v encodings, it follows that

$$\sigma(x_{\rho(2p-1)}) \Downarrow \mathsf{true} \quad \leftrightarrow \quad \sigma(x_{\rho(2p)}) \Downarrow \mathsf{true}.$$

It follows by class separation that

$$x_{\rho(2p-1)} = x_{2p-1} \quad \leftrightarrow \quad x_{\rho(2p)} = x_{2p}. \tag{6.1}$$

Let $1 \leqslant p, q < |a|$ be given. Consider the substitution with $\sigma(x_{2q}) = \sigma(x_{2q+1}) :=$ true and $\sigma(x) :=$ false for all other boolean variables x. By considering the PosCh_v encodings, it follows that

$$\sigma(x_{\rho(2p)}) \Downarrow \mathsf{true} \quad \leftrightarrow \quad \sigma(x_{\rho(2p+1)}) \Downarrow \mathsf{true}.$$

It follows by class separation that

$$x_{\rho(2p)} = x_{2p-1} \quad \leftrightarrow \quad x_{\rho(2p+1)} = x_{2p}. \tag{6.2}$$

Consider the substitution with $\sigma(x_1) := \perp$ and $\sigma(x) :=$ true for all other boolean variables x. Let $1 < p, q \leqslant |a|$ be given. By considering the PosCh_v encodings, it follows that

$$x_{\rho(p)} \neq x_1. \tag{6.3}$$

So it follows $x_{\rho(1)} = x_1$. These three equations imply that ρ is the identity function and thus, the claim follows. □

Note that in the reasoning of the previous lemma, it was essential that each symbol is represented by two and not only by one variable. This allowed making deductions (6.2) about the order of the variables representing adjacent symbols in the word relative to each other due to the word encoding $PosCh_v$.

Theorem 6.18 (Backward direction) *If* SATIS(a, b) *holds, then* SR(a, b).

Proof As SATIS(a, b) holds, there exists a term t satisfying b with resepct to a and all of Loader's word encodings with

$$x_1 : \mathbb{B},\ \ldots,\ x_{2|b|+2} : \mathbb{B},\ r_1 : T(e_1) \to T(f_1),\ \ldots, r_N : T(e_N) \to T(f_N),\ w : T(a) \vdash t :\ \mathbb{B}$$

Now assume that t' is a term with the same properties and all the previous reductions applied, which can be assumed due to Lemma 6.14.The proof is by induction on the number of spinal subterms of t' satrting at 1, as no term has 0 spinal subterms. For the base case, consider the case that t' has only one rib subterm, namely itself. As t' is rib reduced, has sane spine, and is linear, it is in the following form: $t' = w\ x_{\rho(1)}\ \ldots\ x_{\rho(2|a|)}\ x_{2|b|+1}\ x_{2|b|+2}$ where ρ is a permutation of the numbers $1, \ldots, 2|a|$. Next, applying Lemma 6.17 shows that ρ is actually the identity function.

Now note that as t' satisfies a with respect to the $Word_{\mathsf{true}}$ encoding, one obtains the following by considering the a substitution σ with $\sigma(x_{2l}) = \sigma(x_{2l+1}) := a_l$, where a_l is the kth letter of a for $1 \leqslant k \leqslant |a|$, and $\sigma(x) := \bot$ for all other boolean variables x.

$$\emptyset \vdash Word_{\mathsf{true}}(b)\ a_1\ a_1\ \ldots\ a_{|a|}\ a_{|a|}\ \bot\ \bot \geqslant_o Word_{\mathsf{true}}(a)\ a_1\ a_1\ \ldots\ a_{|a|}\ a_{|a|}\ \bot\ \bot : \mathbb{B}$$

Together with

$$Word_{\mathsf{true}}(a)\ a_1\ a_1\ \ldots\ a_{|a|}\ a_{|a|}\ \bot\ \bot \Downarrow \mathsf{true},$$

which follows by the definition of $Word_{\mathsf{true}}$, Fact 3.28 (3), and Lemma 3.23, one obtains:

$$\emptyset \vdash Word_{\mathsf{true}}(b)\ a_1\ a_1\ \ldots\ a_{|a|}\ a_{|a|}\ \bot\ \bot \geqslant_o \mathsf{true} : \mathbb{B}.$$

Using Lemma 3.31 (1) and the definition of $Word_{\mathsf{true}}$, it can be deduced that $b = a$, so a is obviously derivable.

For the inductive step, suppose t' has $n + 1$ spinal subterms. As t' is rib reduced, has sane spine, and is linear, it is in the following form:

$$t' = r_k\ (\lambda y_1\ \ldots\ y_{2|e_k|+2}.\ s)\ x_{\rho(1)}\ \ldots\ x_{\rho(2|f_k|)}\ x_{2|f_k|+1}\ x_{2|f_k|+2}$$

for some $1 \leqslant k \leqslant N$, where ρ is an injection from the numbers $1, \ldots, 2|f_k|$ into the numbers $1, \ldots, 2|e_k|$.

The inductive hypothesis states that for all terms t'' with n spinal subterms and words c such that

$$x_1 : \mathbb{B}, \ldots, x_{2|c|+2} : \mathbb{B}, r_1 : T(e_1) \to T(f_1), \ldots, r_N : T(e_N) \to T(f_N), w : T(a) \vdash t'' \colon \mathbb{B}$$

and t'' satisfies c with respect to all of Loader's word encodings and initial word a, it holds that $SR(a, c)$.

From Lemma 6.15 and the definition of SATIS, one can obtain that for each word encoding *Enc* of Loader's encodings except for the Lin_v encodings, it holds that

$$\emptyset \vdash Enc(b)\ x_{\rho(1)}\ \ldots\ x_{\rho(2|f_k|)}\ x_{2|f_k|+1}\ x_{2|f_k|+2} \geqslant_o Enc(b)\ x_1\ \ldots\ x_{2|b|+2} \colon \mathbb{B} \tag{6.4}$$

From this inequality, one can obtain by the same reasoning used in Lemma 6.17 that Equation (6.1) holds for $1 \leqslant p \leqslant |f_k|$, $1 \leqslant q \leqslant |b|$, Equation (6.2) for $1 \leqslant p < |f_k|$, $1 \leqslant q < |b|$, as well as Equation (6.3) for $1 < p \leqslant |f_k|$. These three equations imply that $x_{\rho(1)}, \ldots, x_{\rho(2|f_k|)}$ are consecutive variables within $x_1, \ldots, x_{2|b|}$ starting with an odd index. So there exists a m with $1 \leqslant m \leqslant |b| - |f_k|$ such that

$$x_1,\ \ldots,\ x_{2|b|} = x_1,\ \ldots\ x_{2m},\ x_{\rho(1)},\ \ldots,\ x_{\rho(2|f_k|)},\ x_{2(m+|f_k|)+1},\ \ldots,\ x_{2|b|}.$$

Consider a substitution σ with $\sigma(x_{2p-1}) = \sigma(x_{2p}) := b_p$ for $1 \leqslant p \leqslant |b|$ where b_p is the pth letter of b, and $\sigma(x) := \bot$ for all other boolean variables x. It follows from Equation (6.4) that b has the form $b = d_1 f_k d_2$ with $|d_1| = m$, $|d_2| = |b| - |f_k| - m$.

Now it suffices to show that there exists a term with n spinal subtermssatisfying $c := d_1 e_k d_2$ with respect to all of Loader's word encodings and the initial word a. Then the derivability of $d_1 e_k d_2$ will follow by the induction hypothesis, and thus the derivability of $b = d_1 f_k d_2$ will follow, as it can be derived from $c = d_1 e_k d_2$ by an application of the rewriting rule (e_k, f_k). As every spinal subterm represents one derivation step, s would be a canonical candidate for such a term satisfying c. Furthermore, s has n spinal subterms, as t' has by assumption $n+1$ spinal subterms.

Note that because t' is linear, the variables $x_{\rho(1)}, \ldots, x_{\rho(2|f_k|)}, x_{2|f_k|+1}, x_{2|f_k|+2}$ do not appear in s, but only the variables

$$x_1, \ldots, x_{2m},\ y_1, \ldots, y_{2|e_k|+2},\ x_{2(m+|f_k|)+1}, \ldots, x_{2|b|}.$$

As the variable r_k appearing in t' is expected to have type $T(e_k) \to T(f_k)$ in the appropriate typing context, the function applied to r_k has return type $\mathbb{B}$. Consequently, it holds that $\Gamma \vdash s \colon \mathbb{B}$ where

$$\begin{aligned}\Gamma := x_1 : \mathbb{B},\ \ldots,\ x_{2k} : \mathbb{B},\ y_1 : \mathbb{B},\ \ldots,\ y_{2|e_i|} : \mathbb{B},\ x_{2(k+e_i)+1} : \mathbb{B},\ x_{2|b|} : \mathbb{B},\\ y_{2|e_i|+1} : \mathbb{B},\ y_{2|e_i|+2} : \mathbb{B},\ r_1 : T(e_1) \to T(f_1),\ \ldots, r_N : T(e_N) \to T(f_N),\ w : T(a)\end{aligned}$$

But as in the definition of a term to satisfy a word, the typing context is different, the variables in s need to be renamed: It is now shown that $s[\mathrm{ren}]$ satisfies $d_1 e_k d_2$ with respect to all of Loader's word encodings and the initial word a, where $\mathrm{ren}(y_l) := x_{2m+l}$ for $1 \leqslant l \leqslant 2|e_k|$, $\mathrm{ren}(y_{2|e_k|+l}) := x_{2|c|+l}$ for $1 \leqslant l \leqslant 2$, and $\mathrm{ren}(x) := x$ otherwise. This means ren renames the variables in s such that they agree with the variables in the typing context in the definition of SATIS. As it obviously holds that $\sigma\colon \Gamma' \to \Gamma$ for

$$\Gamma' := x_1\colon \mathbb{B},\ \ldots,\ x_{2|c|+2},\ r_1\colon \mathrm{T}(e_1) \to \mathrm{T}(f_1),\ \ldots, r_N\colon \mathrm{T}(e_N) \to \mathrm{T}(f_N),\ w\colon \mathrm{T}(a),$$

it follows that $\Gamma' \vdash s[\mathrm{ren}]\colon\ \mathbb{B}$.

Now let *Enc* be a ν-encoding from Loader's word encodings and let F_k be a rule encoding of rule (e_k, f_k) with respect to *Enc* for $1 \leqslant k \leqslant N$. It remains to show that $x_1\colon \mathbb{B},\ \ldots,\ x_{2|c|+2} \vdash (s[\mathrm{ren}])[F_1/r_1,\ \ldots,\ F_N/r_N,\ \mathit{Enc}(a)/w] \geqslant_o \mathit{Enc}(c)\, x_1\ \ldots\ x_{2|c|+2}\colon \mathbb{B}$. According to Lemma 3.32 and the fact that for all terms t, it holds that

$$(s[\mathrm{ren}])[F_1/r_1,\ \ldots,\ F_N/r_N,\ \mathit{Enc}(a)/w] = (t[F_1/r_1,\ \ldots,\ F_N/r_N,\ \mathit{Enc}(a)/w])[\mathrm{ren}],$$

it suffices to show that

$$\begin{aligned} &\Gamma'' \vdash s[F_1/r_1,\ \ldots,\ F_N/r_N,\ \mathit{Enc}(a)/w] \\ \geqslant_o\ &\mathit{Enc}(c)\ x_1\ \ldots x_{2k}\ y_1\ \ldots\ y_{2|e_i|}\ x_{2(k+e_i)+1}\ x_{2|b|}\ y_{2|e_i|+1}\ y_{2|e_i|+2}\colon \mathbb{B}, \end{aligned}$$

$$\begin{aligned} \text{where } \Gamma'' := {} & x_1\colon \mathbb{B},\ \ldots,\ x_{2k}\colon \mathbb{B},\ y_1\colon \mathbb{B},\ \ldots,\ y_{2|e_k|}\colon \mathbb{B}, \\ & x_{2(m+e_k)+1}\colon \mathbb{B},\ x_{2|b|}\colon \mathbb{B},\ y_{2|e_k|+1}\colon \mathbb{B},\ y_{2|e_k|+2}\colon \mathbb{B} \end{aligned}$$

Now take any τ substituting in closed terms of boolean type for the variables in Γ''. It remains to show the above inequality in the empty context with τ applied on both sides, which simplifies to:

$$\begin{aligned} &\emptyset \vdash (s[F_1/r_1,\ \ldots,\ F_N/r_N,\ \mathit{Enc}(a)/w])[\tau] \\ \geqslant_o\ &\mathit{Enc}(c)\ s_1\ \ldots s_{2k}\ t_1\ \ldots\ t_{2|e_i|}\ s_{2(m+e_k)+1}\ s_{2|b|}\ t_{2|e_k|+1}\ t_{2|e_k|+2}\colon \mathbb{B}. \end{aligned}$$

where $s_l := \tau(x_l)$ for $1 \leqslant l \leqslant 2m\ \vee\ 2(m+e_k)+1 \leqslant l \leqslant 2|b|$ and $t_l := \tau(y_l)$ for $1 \leqslant l \leqslant 2|e_k| + 2$.

Now, a case distinction is made of what τ applied to the right hand side evaluates to, which is possible due to Lemma 3.10: Note that from the definition of word encodings, this can only be $\bot$ or ν. In the first case, the desired inequality holds trivially, so one only needs to consider the latter, in which one can obtain by Fact 3.28 (3) the following:

$$\emptyset \vdash \mathit{Enc}(c)\ s_1\ \ldots\ s_{2k}\ t_1\ \ldots\ t_{2|e_k|}\ s_{2(m+e_k)+1}\ \ldots\ s_{2|b|}\ t_{2|e_k|+1}\ t_{2|e_k|+2} \equiv_c \nu\colon \mathbb{B} \tag{6.5}$$

Now consider the descent functions $g_1, \ldots, g_{2|e_k|+2}$ for F_k with respect to *Enc* and the terms u_l for $1 \leqslant l \leqslant 2|f_k| + 2$ obtained from Lemma 6.16. It then follows by Lemma 3.23 and Lemma 3.31 (1) that

$$\emptyset \vdash \mathit{Enc}(b)\ s_1\ \ldots\ s_{2m}\ u_1\ \ldots\ u_{2|f_k|}\ s_{2(m+e_k)+1}\ \ldots\ s_{2|b|}\ u_{2|f_k|+1}\ u_{2|e_k|+2} \equiv_c v \colon \mathbb{B}$$

and as t' satisfies b with respect to a and *Enc*, one also obtains by Lemma 3.31 (1)

$$\emptyset \vdash (t[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w])[\sigma] \equiv_c v \colon \mathbb{B}$$

where $\sigma(x_l) := s_l$ for $1 \leqslant l \leqslant 2m \ \vee\ 2(m + e_k) + 1 \leqslant l \leqslant 2|b|$, $\sigma(x_l) := u_{l-2m}$ for $2m + 1 \leqslant l \leqslant 2(m + |f_k|) - l$, $\sigma(x_{2|b|+1}) := u_{2|f_k|+1}$ and $\sigma(x_{2|b|+2}) := u_{2|f_k|+2}$. For all other variables x, it holds that $\sigma(x) := x$. From Definition 6.1 and the properties of the u_l for $1 \leqslant l \leqslant 2|f_k|$ given by descent completeness (6.16), it holds that

$$\emptyset \vdash (s[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w])[\tau] \geqslant_o v \colon \mathbb{B}$$

Thus, it follows from Equation (6.5) and Lemma 3.23that

$$\begin{aligned} &\emptyset \vdash (s[F_1/r_1, \ldots, F_N/r_N, \mathit{Enc}(a)/w])[\tau] \\ &\geqslant_o \mathit{Enc}(c)\ s_1\ \ldots s_{2m}\ t_1\ \ldots\ t_{2|e_k|}\ s_{2(m+e_k)+1}\ \ldots\ s_{2|b|}\ t_{2|e_k|+1}\ t_{2|e_k|+2} \colon \mathbb{B}, \end{aligned}$$

which shows that in each case, the desired inequality holds and thus, $\sigma(s)$ satisfies c with respect to all of Loader's word encodings. □

Now, equivalence between SR and SATIS follows directly from Corollary 5.2 and Theorem 6.18.

Corollary 6.19 *For words* a, b, *it holds that* $\mathrm{SR}(x) \leftrightarrow \mathrm{SATIS}(x)$.

From Corollary 6.19, the following reduction can easily be obtained with the identity function as reduction function.

Corollary 6.20 SR is many-one reducible to SATIS.

Now, the desired reduction easily follows from Lemma 2.9.

Corollary 6.21 $\overline{\mathrm{SR}}$ *is many-one reducible to* $\overline{\mathrm{SATIS}}$.

This completes the reduction chains from SR to CE presented in Chapter 4 and thus deduces the undecidability of CE from the undecidability of SR.

Chapter 7

Related Work

Loader's result may be considered particularly surprising, as contextual equivalence is decidable in related calculi. In 1998, Loader has shown contextual equivalence for PCF_1 decidable [26] – an even more restricted version of PCF than PCF_2. Schmidt-Schauß provided a shorter proof of this result in 1999 [47].

Apart from that, contextual equivalence for STLC with product and sum types, which is in some sense more general than PCF_2, is also decidable. This relies on the fact that for this calulus, $\beta\eta$-equivalence – the usual reduction relation up to η-conversions – is decidable, which first has been shown in 1995 by Ghani for non-empty sum types [21]. It has been followed by results of Altenkirch, Dybjer, Hofmann, and Scott in 2001 [3] and Balat, Di Cosmo, and Fiore in 2004 [4], showing contextual equivalence decidable for non-empty sum types, using Grothendieck logical relations introduced by Fiore and Simpson [17] and invovling techniques from category theory. In 2017, Scherer showed both $\beta\eta$- and contextual equivalence in presence of the empty type decidable by showing that both equivalence relations agree on this calculus [46].

In sections Section 7.1 and Section 7.2, we briefly introduce these two calculi, sketch which techniques have been used to obtain these results, and why they are not applicable to PCF_2. Finally, we discuss related undecidability results from the theory of programming languages in Section 7.3.

7.1 Decidability of PCF_1

PCF_1 is both an extension of STLC with the unit type as base type and a match operation on it, as well as a restriction of PCF_2 with a base type containing only one proper, i.e. non-error, constant instead of two.

Definition 7.1 (PCF_1) *The types* ty *and terms* tm *of PCF_1 are defined by*

$$\begin{aligned} T_1, T_2 : \mathsf{ty} &::= \texttt{Unit} \mid T_1 \to T_2 \\ s, t : \mathsf{tm} &::= \lambda x.s \mid s\ t \mid x \mid \texttt{match}\ s\ \texttt{with}\ t \mid () \mid \bot \end{aligned}$$

Here, `Unit` comes with the two constants $()$ and $\bot$. The standard unit constant is denoted by $()$, whereas $\bot$ denotes as in PCF_2 an error constant. Hereby, the match on PCF_1 can be interpreted as a degenerated version of PCF_2's conditional: As `Unit` contains only one proper constant, there is only one case possible.

The reduction rules for `match` are the following:

$$\frac{}{\mathtt{match}\ ()\ \mathtt{with}\ t \succ t} \qquad \frac{s \succ s'}{\mathtt{match}\ s\ \mathtt{with}\ t \succ \mathtt{match}\ s'\ \mathtt{with}\ t}$$

$$\frac{}{\mathtt{match}\ \bot\ \mathtt{with}\ t \succ \bot} \qquad \frac{t \succ t'}{\mathtt{match}\ s\ \mathtt{with}\ t \succ \mathtt{match}\ s\ \mathtt{with}\ t'}$$

The reduction rules for the remaining constructs correspond to those for PCF_2 given in Section 3.3. As for PCF_2, the reduction relation on PCF_1 is strongly normalising, Church-Rosser and type preserving.

Next, we sketch Schmidt-Schauß's proof that contextual equivalence on PCF_1 is decidable:

Using the same reasoning as in Lemma 4.7, one obtains that at each type, there are only finitely many equivalence classes of closed terms induced by the contextual equivalence. A key lemma in Schmidt-Schauß's proof is the following:

Lemma 7.2 *Representatives of all finitely many equivalence classes induced by contextual equivalence at each type and the empty context are computable.*

From this lemma, the decidability of contextual equivalence on closed terms can straightforwardly be deduced. Note that Schmidt-Schauß uses the following extensionality property of contextual equivalence, which for PCF_2 can be deduced by Fact 3.28 (2): Closed terms s and t of type $A \to B$ are contextually equivalent iff for all closed terms u of type T_1, $s\ u$ and $t\ u$ are contextually equivalent.

Theorem 7.3 *On PCF_1, contextual equivalence on closed terms is decidable at every type.*

Proof Let T be a type, s, t be PCF_1 terms with $\emptyset \vdash s\colon\ T$ and $\emptyset \vdash t\colon\ T$. The proof is by induction on T. For $T = \mathtt{Unit}$, it holds that – as for $\mathbb{B}$ in PCF_2 – that s and t are contextually equivalent iff they have the same normal form. The latter is decidable using Lemma 3.10, which can also be proven for PCF_1 as its reduction is strongly normalising and thus in particular weakly normalising and type preserving. If $T = A \to B$, it remains – according to the above mentioned extensionality property – to show that for all closed terms u of type T_1, $s\ u$ and $t\ u$ are contextually equivalent. It suffices to show the latter for one representative of each equivalence class of closed terms at type T_1, instead of showing it for all closed terms u of type T_1. Then the claim follows by Lemma 7.2 and the inductive hypothesis, as the number of equivalence classes of closed terms at each type is finite. □

Schmidt-Schauß only introduces contextual equivalence on closed terms, but we think his result can be lifted to a setting involving arbitrary contexts Γ in the following way: In Loader's decidability proof of contextual equivalence for PCF_1 [26], he defines the observational preorder on PCF_1 analogously as done for PCF_2 – but only on closed terms. However, using the same techniques as in Section 3.4.3, it should be possible to extend it to open terms and prove it equivalent to contextual equivalence in arbitrary contexts. Thus, one would obtain the following extensionality property, which for PCF_2 immediately comes from Definition 3.20 and allows lifting the result to arbitrary contexts Γ. Terms s and t with $\Gamma \vdash s\colon T$ and $\Gamma \vdash t\colon T$ are contextually equivalent iff this is the case for all $\sigma(s)$ and $\sigma(t)$, where σ is a substitution of closed terms of the appropriate types for the free variables in s and t. Assuming the above mentioned extensionality property, we suspect contextual equivalence on PCF_1 in any context and at every type can be deduced to be decidable in the following way:

Let Γ be a typing context, T be a type, and s, t be PCF_1 terms with $\Gamma \vdash s\colon T$ and $\Gamma \vdash t\colon T$. It should be decided whether $\Gamma \vdash s \equiv_c t\colon T$ holds. Using the extensionality property, it becomes clear that this is the case iff $\sigma(s)$ and $\sigma(t)$ are contextually equivalent for all substitutions σ of closed terms of the appropriate types for the free variables in s and t. It suffices to check the latter for all substitutions σ such that σ substitutes each variable contained in Γ by a representative of any equivalence class of closed terms at this type. As there are only finitely many such σ, the representatives are computable according to Lemma 7.2, and all $\sigma(s)$, $\sigma(t)$ are closed terms of type T, this decidability result follows from Theorem 7.3.

Note, given that Lemma 7.2 also held for PCF_2, the proof of Theorem 7.3 and our following observation could easily be adapted to PCF_2. However, Lemma 7.2 breaks down when trying to extend to PCF_2. Even though it does hold for PCF_2 that there are only finitely many equivalence classes of closed terms at each type (see Lemma 4.7), no representatives of these can be computed. Schmidt-Schauß states that this is the case, since the observational preorder on PCF_2 does not have greatest elements at all types [47]. Instead, there are even at no type greatest elements: Note that accoding to Lemma 3.30, it suffices to reason about the normal forms at each type. In PCF_2, true and false are incomparable at the base type, i.e. neither $\emptyset \vdash \mathsf{true} \leqslant_o \mathsf{false}\colon \mathbb{B}$ nor $\emptyset \vdash \mathsf{false} \leqslant_o \mathsf{true}\colon \mathbb{B}$ holds. As $\bot$ is certainly not a greatest element at base type, it follows that there is no such element. Analogously, one can obtain that there are no greatest elements at any function type: Functions returning true and false on the same arguments are also incomparable and functions returning $\bot$ at some arguments are certainly no greatest elements. The observational preorder on PCF_1, the definition of which is similar to Definition 3.20 and can be found in Loader's paper [26], has greatest elements at each type: $()$ is a greatest element at type `Unit` and for function types, constant functions returning

() are greatest elements.

7.2 Decidability of STLC with sum and product types

Next, consider an extension of the simply typed lambda calculus with the empty and the unit type as base types: The types are extended by sums and products and the terms by injections, pairs as well as matches on sums and products.

Definition 7.4 (Simply typed λ-calculus with sums and products) *The types* ty *and terms* tm *of STLC with sums and products are defined by*

$$\begin{aligned} T_1, T_2 : \mathsf{ty} &::= T_1 \to T_2 \mid \texttt{Unit} \mid \texttt{Empty} \mid T_1 + T_2 \mid T_1 \times T_2 \\ s, t : \mathsf{tm} &::= \lambda x.\ s \mid s\ t \mid \mathit{Var}\ n \mid () \mid \mathsf{inj}_1 s \mid \mathsf{inj}_2 s \mid (s, t) \end{aligned}$$

Here, `Unit` comes only with the standard constant () – there is no native error constant. Nevertheless, PCF_2 can be seen as a restriction of this calculus, since there is a natural way to embed it in the latter: Now identify $\mathbb{B}$ with the sum type $\texttt{Unit} + \texttt{Unit} + \texttt{Unit}$, which we call $\mathbb{B}'$ and has exactly three closed normal forms. The boolean constants true, false as well as $\perp$ are identified with one of these normal forms each – we call them true$'$, false$'$ and $\perp'$. The conditional is mapped to a match on that sum type, where in the case of $\perp'$, the match returns $\perp'$.

However, contextual equivalence is not preserved by this embedding: A match in the extended STLC can map $\perp'$ to any term of the appropriate type, while a conditional in PCF_2 must evaluate to $\perp$ on input $\perp$. So there are functions in STLC with sums and products behaving in a way no function in PCF_2 can. This means there are contexts in STLC with sums and products with no correspondence in PCF_2, which suggests that there could be terms being contextually equivalent in PCF_2, but not in the extended STLC. And in fact, this intuition will turn out to be correct.

The technique used in Scherer's paper is the following: In the first part, the same reasoning can be carried out for PCF_2: It is shown that βη-equivalence, the usual reduction relation up to η-conversions, is decidable. This decidability result uses that each typed term has a unique, computable normal form, and that it is decidable whether two terms are equal up to η-conversions. As each term is βη-equivalent to its normal form, this establishes a decision procedure for the βη-equivalence of typed terms.

The second part of Scherer's proof – and this is where an adjustment to PCF_2 breaks down – is to show that βη-equivalence and contextual equivalence coincide on STLC with sums and products. We show that this is not the case for PCF_2 by providing a counterexample. Before, observe the following, which is a special case of a result proven by Schmidt-Schauß for PCF_1 [47]:

Lemma 7.5 *In PCF_2, each program* f *with $\emptyset \vdash f\colon\ \mathbb{B} \to \mathbb{B}$ is either constant or strict in its argument, i.e.* $f \perp \Downarrow \perp$.

The proof relies on the intuition that the only way for f to make a case distinction on the argument, i.e. for f not to be constant, is to use a conditional on the argument. But as conditionals reduce to $\perp$ if the argument does, one then obtains $\mathsf{f} \perp \Downarrow \perp$. The above lemma does not hold for each closed term of type $\mathbb{B}' \to \mathbb{B}'$ in the extended STLC, because – as exlpained before – the match on $\mathbb{B}'$ has no such restriction of always returning $\perp'$ on the input $\perp'$.

Now consider the terms

$$t_b := \lambda f.\ \texttt{if}\ (f\perp)\ \texttt{then}\ (f\ b)\ \texttt{else}\ (f\ b)$$

for $b \in \{\texttt{true}, \texttt{false}\}$. It obviously holds have $\emptyset \vdash t_b\colon\ (\mathbb{B} \to \mathbb{B}) \to \mathbb{B}$ and t_b is in normal form for $b \in \{\texttt{true}, \texttt{false}\}$. t_{true} and t_{false} are not $\beta\eta$-equivalent, as their normal forms (the terms themselves) are not equal up to η-conversions.

However, t_{true} and t_{false} are contextually equivalent: According to Fact 3.28 (2) and Lemma 3.30, it suffices to prove that the following equivalence holds for all closed f of type $\mathbb{B} \to \mathbb{B}$:

$$\emptyset \vdash t_{\mathsf{true}}\ f \equiv_c t_{\mathsf{false}}\ f\colon \mathbb{B}$$

But this again follows from Lemma 3.30: By a case distinction on Lemma 7.5 and Lemma 3.10, it becomes clear that $t_{\mathsf{true}}\ f$ and $t_{\mathsf{false}}\ f$ always evaluate to the same boolean.

This shows that for PCF$_2$, the two discussed equivalences do not agree and in particular, not all contextually equivalent terms are $\beta\eta$-equivalent. However, the converse holds: For PCF$_2$, all $\beta\eta$-equivalent terms are contextually equivalent. This follows from Lemma 3.30 and the fact that terms that are equal up to η-conversions are contextually equivalent.

7.3 Related Undecidability Results

We now briefly discuss other undecidability results from the theory of programming languages, which have been mechanised in Coq and contributed to the Coq Library of Undecidability Proofs [20].

Firstly, there has been work done on several decision problems related to System F, an extension of the simply typed lambda calulus with universal quantification over types, allowing for polymorphism. To be precise, in 2018, Dudenhefner and Rehof mechanised an undecidability proof of type inhabitation on System F by reducing from Hilbert's tenth problem, the solvability of Diophantine equations [16]. Type inhabitation is the problem of whether, given a type T and a typing context Γ, there exists a term of type T in context Γ. The proof is known to be simpler than previous approaches by Löb and Urzyczyn [28, 54] and thus particularly well-suited for mechanisation. Furthermore, Dudenhefner also mechanised the undecidability of typability and type checking for System F in 2021 [15]. Typability denotes the problem of whether, given a term, there exists a type T and a typing context Γ such that the term has type T in context Γ. While typability asks for the existence of any valid typing of a given term, type checking is the problem whether, given a term t, a type T and a typing context Γ, t has type T in context Γ. Dudenhefner came up with a direct reduction from semi-unification to typability, concluding the undecidability of typability from that of semi-unification. He then deduced the undecidability of type checking by a folklore reduction from typability. The first undecidability result of type checking and typability in System F goes back to Wells in 1994 [55]. However, his apporach involved a reduction from type checking to typability relying on heavy machinery. Compared to Wells' apporach, Dudenhefner's approach is relatively simple and direct, which made the mechanisation more feasible.

Apart from that, Spies and Forster mechanised the undecidability of higher-order unification in the simply typed lambda-calulus [49]. Higher-order unification denotes the problem whether, given two typed terms, there exists a substitution such that both terms become convertible when applying it. They established the undecidability of higher-order unification by reducing from the solvability of Diophantine equations, following a proof by Dowek [13]. Moreover, they sharpened the result by proving second-order and third-order unification already undecidable, following proofs of Goldfarb [22] and Huet [23].

Chapter 8

Conclusion

In this chapter, we briefly summarise our work, comment on our Coq mechanisation and finally mention remaining open problems.

We have discussed the undecidability of the contextual equivalence on PCF_2, implying that no solution for the full abstraction problem exists, following Loader's proof [27]. To be precise, we have mechanised the observational as well as contextual equivalence on PCF_2 and have proven that both agree. Furthermore, we have mechanised all but one reductions appearing in Loader's proof, namely the reductions from $\overline{\text{SATIS}}$ to $\overline{\text{PS}}$ (4.16), from $\overline{\text{PS}}$ to $\overline{\text{RPS}}$ (4.20), and from $\overline{\text{RPS}}$ to CE (4.21). We have discussed the remaining reduction – from $\overline{\text{SR}}$ to $\overline{\text{SATIS}}$ – in detail. In particular, we have given formal account of the forward direction on paper, and have seen examples of how derivation of words are encoded into PCF_2 terms. Due to a lack of time, we have not completed the mechanisation of the forward direction, but we have conserved our attempt. Regarding the backward direction, which we did not mechanise, we have provided an informal overview of how satisfying terms may be brought into a structurally simplified from, and have presented in detail, how then the respective derivability result is obtained.

8.1 Coq Mechanisation

The complete Coq mechanisation is available at

```
https://www.ps.uni-saarland.de/~brenner/bachelor/coq/toc.html
```

The digital version of this thesis contains clickable references to the online documentation for all mechanised results. The files have been compiled using version 8.18.0 of Coq and the structure of the mechanisation is the following:

All our files are located in the folde *PCF2*. The folder external contains the part of the Coq Library of Undecidability Proofs [20] we need for our work. Hereby, the definition of string rewriting has been changed accordingly, as explained in Section 2.3. The remaining files are diretly located in the folder PCF2:

File	Spec	Proof	Σ
preliminaries.v	39	115	154
pcf2_system.v	100	323	423
pcf2_contexts.v	10	57	67
pcf2_utils.v	63	173	236
CE.v	2	22	24
PS.v	11	0	11
RPS.v	11	0	11
SATIS.v	5	33	38
CE_facts.v	123	537	660
PS_facts.v	7	11	18
RPS_facts.v	7	11	18
SATIS_facts.v	29	142	171
co_RPS_CE.v	14	184	198
PS_RPS.v	11	71	82
SATIS_PS.v	64	468	532
SR_SATIS_forward.v	27	262	289
undecidability.v	15	36	51
	538	2445	2983

Table 8.1: Lines of code in the respective files

- *preliminaries.v*: Results about lists and standard functions used in this thesis
- *core.v, core_axioms.v, pcf_2.v, unscoped.v, unscoped_axioms.v*: Generated by Autosubst 2 regarding PCF$_2$
- *pcf_2_system.v, pcf_2_contexts.v, pcf_2_utils.v*: Introducing central notions and results about PCF$_2$
- *CE.v, PS.v, RPS.v, SATIS.v*: Definition of respective decision problem and useful related lemmas
- *CE_facts.v, PS_facts, RPS_facts, SATIS_facts*: Useful properties related to respective decision problem
- *co_RPS_CE.v, PS_RPS.v, SATIS_PS.v*: Respective reduction
- *SR_SATIS_forward.v*: Conservation of attempt to mechanise forward direction of reduction from SR to SATIS
- *undecidability.v*: undecidability proof of $\overline{\mathsf{SR}}$ and conclusion of undecidability of CE

Table 8.1 depicts how many lines of code the respective files of our mechanisation contains, separated in specifications (Spec) and proofs (Proof).

Note that we assumed the following axioms in the Coq development:

1. Stronger version of weak normalisation and Church-Rosser property of reduction in PCF_2 (3.9)
2. Existence of rule encodings (4.9)
3. Equivalence between SR and SATIS(6.19)
4. Existence of a set of rules such that SR_R is undecidable (2.11)

The axioms (1) are standard results about PCF_2 and are assumed such that we could focus on the more interesting results.

Axiom (2), which can be derived from a result in the metatheory of PCF_2 and a standard argument about finiteness as described in Section 4.1.1, is assumed due to the lack of time. Recall that the argument about finiteness makes use of the law of excluded middle LEM – so in order to get rid of axiom (2) in the proposed way, LEM must be assumed.

The third axiom (3) is assumed to obtain the desired undecidability result in Coq. It assumes the equivalence between SR and SATIS presented in Chapter 5 and Chapter 6, which we did not mechanise.

Axiom (4) is assumed, as in the Coq Library of Undecidability Proofs [20], which this work is based on, there is no undecidability result about the version of string rewriting presented in Section 2.3.

8.2 Open problems

There are two sorts of remaining problems.

On the one hand, there are still gaps in our mechanisation, which remain to be filled: The existence of rule encodings remains to be shown. Furthermore, the computable version of weak normalisation as well as the Church-Rosser property of the reduction relation are still open. The forward direction of the reduction from SR to SATIS also remains uncompleted – our attempt could serve as basis for future work. Finally, the backward direction of the same remains open. In particular, the simplifications of the structure of terms satisfying words with respect to all of Loader's word encodings remains to be mechanised, which we have given an overview of in Section 6.2.

On the other hand, it is left to connect this work to the Coq Library of Undecidability Proofs [20], i.e. to prove the base problem of the reduction chain presented here undecidable in the synthetic setting. In order to achieve this, it remains to mechanise the undecidability of the particular version of string rewriting used in this thesis, e.g. by following Davis' approach mentioned in Section 2.3.

Appendix A

Appendix

We now present Loader's 32 word encodings by condensing dual encodings to one general ν-encoding each (for $\nu \in [\mathsf{true}, \mathsf{false}]$) to avoid repetitions. Apart from that, we do not give explicite PCF$_2$ terms for the encodings, but define their behaviour mathematically by specifying for which arguments ν is returned – for all other arguments, $\perp$ is returned. Again, appropriate PCF$_2$ terms can easily be constructed by nesting conditionals. Let $a = a_1 \dots a_n$ be a word of length n.

1. If $\forall i,\ 1 \leqslant i \leqslant |a|.\ s_i \Downarrow a_i\ \wedge\ s'_i \Downarrow a_i$, then

$$\mathit{Word}_\nu(a)\ s_1\ s'_1\ \dots\ s_{|a|}\ s'_{|a|}\ t_1\ t_2 = \nu.$$

2. For any $s_1,\ \dots,\ s_{2|a|+2}$

$$\mathit{Const}_\nu(a)\ s_1\ \dots\ s_{2|a|+2} = \nu$$

3. If $\forall i,\ 1 \leqslant i \leqslant |a|\ \rightarrow\ (s_i \Downarrow \mathsf{true}\ \vee\ s_i \Downarrow \mathsf{false})$, then

$$\mathit{PosOdd}_\nu(a)\ s_1\ s'_1\ \dots\ s_{|a|}\ s'_{|a|}\ t_1\ t_2 = \nu.$$

4. If $\forall i,\ 1 \leqslant i \leqslant |a|\ \rightarrow\ (s'_i \Downarrow \mathsf{true}\ \vee\ s'_i \Downarrow \mathsf{false})$, then

$$\mathit{PosEven}_\nu(a)\ s_1\ s'_1\ \dots\ s_{|a|}\ s'_{|a|}\ t_1\ t_2 = \nu.$$

5. If $(t_1 \Downarrow \mathsf{true}\ \vee\ t_1 \Downarrow \mathsf{false})$, then

$$\mathit{ConOdd}_\nu(a)\ s_1\ s'_1\ \dots\ s_{|a|}\ s'_{|a|}\ t_1\ t_2 = \nu.$$

6. If $(t_2 \Downarrow \mathsf{true}\ \vee\ t_2 \Downarrow \mathsf{false})$, then

$$\mathit{ConEven}_\nu(a)\ s_1\ s'_1\ \dots\ s_{|a|}\ s'_{|a|}\ t_1\ t_2 = \nu.$$

7. If $\forall i,\ 1 \leqslant i \leqslant |a|+1 \ \rightarrow\ (s_i \Downarrow \text{true} \ \wedge\ (s'_i \Downarrow \text{true} \vee s'_i \Downarrow \text{false})) \vee s_i \Downarrow \text{false}$, then
$$\mathit{EvenSimpTrue}_v(a)\ s_1\ s'_1\ \ldots\ s_{|a|+1}\ s'_{|a|+1} = v.$$

8. If $\forall i,\ 1 \leqslant i \leqslant |a|+1 \ \rightarrow\ (s_i \Downarrow \text{false} \ \wedge\ (s'_i \Downarrow \text{true} \vee s'_i \Downarrow \text{false})) \vee s_i \Downarrow \text{true}$, then
$$\mathit{EvenSimpFalse}_v(a)\ s_1\ s'_1\ \ldots\ s_{|a|+1}\ s'_{|a|+1} = v.$$

9. If $\forall i,\ 1 \leqslant i \leqslant |a|+1 \ \rightarrow\ (s'_i \Downarrow \text{true} \ \wedge\ (s_i \Downarrow \text{true} \vee s_i \Downarrow \text{false})) \vee s'_i \Downarrow \text{false}$, then
$$\mathit{OddSimpTrue}_v(a)\ s_1\ s'_1\ \ldots\ s_{|a|+1}\ s'_{|a|+1} = v.$$

10. If $\forall i,\ 1 \leqslant i \leqslant |a|+1 \ \rightarrow\ (s'_i \Downarrow \text{false} \ \wedge\ (s_i \Downarrow \text{true} \vee s_i \Downarrow \text{false})) \vee s'_i \Downarrow \text{true}$, then
$$\mathit{OddSimpFalse}_v(a)\ s_1\ s'_1\ \ldots\ s_{|a|+1}\ s'_{|a|+1} = v.$$

11. If $\forall i,\ 1 \leqslant i \leqslant 2|a|+2 \ \rightarrow\ s_i \Downarrow \text{true}$, then
$$\mathit{FixTrue}_v(a)\ s_1\ \ldots\ s_{2|a|+2} = v.$$

12. If $\forall i,\ 1 \leqslant i \leqslant 2|a|+2 \ \rightarrow\ s_i \Downarrow \text{false}$, then
$$\mathit{FixFalse}_v(a)\ s_1\ \ldots\ s_{2|a|+2} = v.$$

13. If $(t_2 \Downarrow \text{true} \wedge (\forall i,\ 1 \leqslant i \leqslant 2|a| \rightarrow\ (s_i \Downarrow \text{true} \ \vee\ s_i \Downarrow \text{false})) \wedge (t_1 \Downarrow \text{true} \ \vee\ t_1 \Downarrow \text{false}))\ \vee\ t_2 \Downarrow \text{false}$, then
$$\mathit{Chain}_v(a)\ s_1\ \ldots\ s_{2|a|}\ t_1\ t_2 = v.$$

14. If $\forall i\, v_1\, v_2,\ 1 \leqslant i \leqslant n \ \rightarrow\ s_i \Downarrow v_1 \ \rightarrow\ s'_i \Downarrow v_2 \ \rightarrow\ v_1 = v_2 \in [\text{true},\ \text{false}]$, then
$$\mathit{PosEq}_v(a)\ s_1\ s'_1\ \ldots\ s_{|a|}\ s'_{|a|}\ t_1\ t_2 = v.$$

15. If $\forall i\, v_1\, v_2,\ 1 \leqslant i \leqslant n-1 \ \rightarrow\ s_{2i} \Downarrow v_1 \ \rightarrow\ s_{2i+1} \Downarrow v_2 \ \rightarrow\ v_1 = v_2 \in [\text{true},\ \text{false}]$, then
$$\mathit{PosChain}_v(a)\ s_1\ \ldots\ s_{2|a|}\ t_1\ t_2 = v.$$

16. If $\exists i\ 1 \leqslant i \leqslant 2|a|+2 \ \wedge\ s_i \Downarrow \text{true} \ \wedge\ (\forall j,\ 1 \leqslant j \leqslant 2|a|+2 \ \rightarrow\ i \neq j \ \rightarrow\ s_j \Downarrow \text{false})$, then
$$\mathit{Lin}_v(a)\ s_1\ \ldots\ s_{2|a|+2} = v.$$

Bibliography

[1] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full Abstraction for PCF. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1994. URL `https://doi.org/10.1007/3-540-57887-0_87`.

[2] Amal Ahmed. Oplss 2023: Logical relations, 2023.

[3] Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 303–310. IEEE Computer Society, 2001. URL `https://doi.org/10.1109/LICS.2001.932506`.

[4] Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 64–76. ACM, 2004. URL `https://doi.org/10.1145/964001.964007`.

[5] Hendrik Pieter Barendregt. *The Lambda Calculus - Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985. ISBN 978-0-444-86748-3.

[6] Andrej Bauer. First Steps in Synthetic Computability Theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006. ISSN 1571-0661. URL `https://doi.org/10.1016/j.entcs.2005.11.049`. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI).

[7] Andrej Bauer. On fixed-point theorems in synthetic computability. *Tbilisi Mathematical Journal*, 10(3):167 – 181, 2017. URL `https://doi.org/10.1515/tmj-2017-0107`.

[8] Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1987.

[9] Thierry Coquand and Gérard P. Huet. The Calculus of Constructions. *Inf. Comput.*, 76(2/3):95–120, 1988. URL `https://doi.org/10.1016/0890-5401(88)90005-3`.

[10] Martin D. Davis. *Computability and Unsolvability*. McGraw-Hill Series in Information Processing and Computers. McGraw-Hill, 1958.

[11] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972. ISSN 1385-7258. URL `https://doi.org/10.1016/1385-7258(72)90034-0`.

[12] Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. URL `https://doi.org/10.1007/978-3-030-79876-5_37`.

[13] Gilles Dowek. Higher-Order Unification and Matching. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1009–1062. Elsevier and MIT Press, 2001. URL `https://doi.org/10.1016/b978-044450813-3/50018-7`.

[14] Derek Dreyer, Simon Spies, Lennard Gäher, Ralf Jung, Jan-Oliver Kaiser, Hoang-Hai Dang, David Swasey, and Jan Menz. Semantics of type systems lecture notes, 2022.

[15] Andrej Dudenhefner. The Undecidability of System F Typability and Type Checking for Reductionists. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–10. IEEE, 2021. URL `https://doi.org/10.1109/LICS52264.2021.9470520`.

[16] Andrej Dudenhefner and Jakob Rehof. A Simpler Undecidability Proof for System F Inhabitation. In Peter Dybjer, José Espírito Santo, and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs, TYPES 2018, June 18-21, 2018, Braga, Portugal*, volume 130 of *LIPIcs*, pages 2:1–2:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. URL `https://doi.org/10.4230/LIPIcs.TYPES.2018.2`.

[17] Marcelo P. Fiore and Alex K. Simpson. Lambda Definability with Sums via

Grothendieck Logical Relations. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 1999. URL `https://doi.org/10.1007/3-540-48959-2_12`.

[18] Yannick Forster, Edith Heiter, and Gert Smolka. Verification of PCP-Related Computational Reductions in Coq. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2018. URL `https://doi.org/10.1007/978-3-319-94821-8_15`.

[19] Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 38–51. ACM, 2019. URL `https://doi.org/10.1145/3293880.3294091`.

[20] Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq Library of Undecidable Problems. In *CoqPL 2020 The Sixth International Workshop on Coq for Programming Languages*, New Orleans, United States, January 2020. URL `https://hal.science/hal-02944217`.

[21] Neil Ghani. βη-equality for coproducts. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, pages 171–185, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49178-1. URL `https://doi.org/10.1007/BFb0014052`.

[22] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981. URL `https://doi.org/10.1016/0304-3975(81)90040-2`.

[23] Gérard P. Huet. The Undecidability of Unification in Third Order Logic. *Inf. Control.*, 22(3):257–267, 1973. URL `https://doi.org/10.1016/S0019-9958(73)90301-X`.

[24] J. M. E. Hyland and C.-H. Luke Ong. On Full Abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000. URL `https://doi.org/10.1006/inco.2000.2917`.

[25] Achim Jung and Allen Stoughton. Studying the Fully Abstract Model of PCF

within its Continuous Function Model. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 1993. URL `https://doi.org/10.1007/BFb0037109`.

[26] Ralph Loader. Unary PCF is decidable. *Theor. Comput. Sci.*, 206(1-2):317–329, 1998. URL `https://doi.org/10.1016/S0304-3975(98)00048-6`.

[27] Ralph Loader. Finitary PCF is not decidable. *Theor. Comput. Sci.*, 266(1-2): 341–364, 2001. URL `https://doi.org/10.1016/S0304-3975(00)00194-8`.

[28] M. H. Lob. Embedding first order predicate logic in fragments of intuitionistic logic. *J. Symb. Log.*, 41(4):705–718, 1976. URL `https://doi.org/10.2307/2272390`.

[29] John Longley and Dag Normann. *Higher-Order Computability*. Theory and Applications of Computability. Springer, 2015. ISBN 978-3-662-47991-9. URL `https://doi.org/10.1007/978-3-662-47992-6`.

[30] A. Markov. On the impossibility of certain algorithms in the theory of associative systems. *Journal of Symbolic Logic*, 13(3):170–171, 1948. URL `https://doi.org/10.2307/2267871`.

[31] A. Markov. Impossibility of certain algorithms in the theory of associative systems. *Journal of Symbolic Logic*, 16(3):215–215, 1951. URL `https://doi.org/10.2307/2266407`.

[32] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in proof theory*. Bibliopolis, 1984. ISBN 978-88-7088-228-5.

[33] Robin Milner. Fully abstract models of typed lambda-calculi. *Theor. Comput. Sci.*, 4(1):1–22, 1977. URL `https://doi.org/10.1016/0304-3975(77)90053-6`.

[34] Robin Milner. LCF: A way of doing proofs with a machine. In Jirí Becvár, editor, *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 146–159. Springer, 1979. URL `https://doi.org/10.1007/3-540-09526-8_11`.

[35] Robin Milner, Mads Tofte, and Robert Harper. *Definition of standard ML*. MIT Press, 1990. ISBN 978-0-262-63132-7.

[36] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer*

Science. Springer, 2002. ISBN 3-540-43376-7. URL `https://doi.org/10.1007/3-540-45949-9`.

[37] Peter W. O'Hearn and Jon G. Riecke. Kripke Logical Relations and PCF. *Inf. Comput.*, 120(1):107–116, 1995. URL `https://doi.org/10.1006/inco.1995.1103`.

[38] Christine Paulin-Mohring. Inductive definitions in the system Coq rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993. URL `https://doi.org/10.1007/BFb0037116`.

[39] Christine Paulin-Mohring. Introduction to the Calculus of Inductive Constructions. In Bruno Woltzenlogel Paleo and David Delahaye, editors, *All about Proofs, Proofs for All*, volume 55 of *Studies in Logic* (*Mathematical logic and foundations*). College Publications, January 2015. URL `https://inria.hal.science/hal-01094195`.

[40] Richard Alan Platek. *Foundations of recursion theory*. Stanford University, 1966.

[41] Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977. URL `https://doi.org/10.1016/0304-3975(77)90044-5`.

[42] Emil L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943. ISSN 00029327, 10806377. URL `https://doi.org/10.2307/2371809`.

[43] Emil L. Post. Recursive Unsolvability of a problem of Thue. *J. Symb. Log.*, 12 (1):1–11, 1947. URL `https://doi.org/10.2307/2267170`.

[44] Fred Richman. Church's thesis without tears. *J. Symb. Log.*, 48(3):797–803, 1983. URL `https://doi.org/10.2307/2273473`.

[45] Steven Schäfer, Tobias Tebbi, and Gert Smolka. Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2015. URL `https://doi.org/10.1007/978-3-319-22102-1_24`.

[46] Gabriel Scherer. Deciding equivalence with sums and the empty type. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, page 374–386, New York, NY, USA, 2017. Association

for Computing Machinery. ISBN 9781450346603. URL https://doi.org/10.1145/3009837.3009901.

[47] Manfred Schmidt-Schauß. Decidability of behavioural equivalence in unary PCF. *Theor. Comput. Sci.*, 216(1-2):363–373, 1999. URL https://doi.org/10.1016/S0304-3975(98)00024-3.

[48] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theor. Comput. Sci.*, 121(1&2):411–440, 1993. URL https://doi.org/10.1016/0304-3975(93)90095-B.

[49] Simon Spies and Yannick Forster. Undecidability of higher-order unification formalised in coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 143–157, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370974. URL https://doi.org/10.1145/3372885.3373832.

[50] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. Autosubst 2: Reasoning with Multi-sorted de Bruijn Terms and Vector Substitutions. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 166–180. ACM, 2019. URL https://doi.org/10.1145/3293880.3294101.

[51] The Coq Development Team. The Coq Proof Assistant, June 2024. URL https://doi.org/10.5281/zenodo.11551307.

[52] The Haskell Development Team. Haskell 2010 Language Report, 2010. URL https://www.haskell.org/definition/haskell2010.pdf.

[53] A. Thue. *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln*. Skrifter (Norske videnskaps-akademi. I–Mat.-naturv. klasse). J. Dybwad, 1914.

[54] Pawel Urzyczyn. Inhabitation in Typed Lambda-Calculi (A Syntactic Approach). In Philippe de Groote, editor, *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 373–389. Springer, 1997. URL https://doi.org/10.1007/3-540-62688-3_47.

[55] J. B. Wells. Typability and type-checking in the second-order lambda-calculus are equivalent and undecidable. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 176–

185. IEEE Computer Society, 1994. URL `https://doi.org/10.1109/LICS.1994.316068`.

[56] Benjamin Werner. Sets in Types, Types in Sets. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software, Third International Symposium, TACS '97, Sendai, Japan, September 23-26, 1997, Proceedings*, volume 1281 of *Lecture Notes in Computer Science*, pages 530–346. Springer, 1997. URL `https://doi.org/10.1007/BFb0014566`.