Saarland University

Faculty of Mathematics and Computer Science

Bachelor'sThesis

Undecidability of the Post Correspondence Problem in Coq

Author
Edith Heiter

Advisor
Yannick Forster, M.Phil.
Prof. Dr. Gert Smolka

Reviewers
Prof. Dr. Gert Smolka
Prof. Bernd Finkbeiner, Ph.D.

Submitted: 07th August 2017

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig 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, 07th August, 2017

Abstract

We study, formalize, and verify reductions of the halting problem for Turing machines (Halt) to the Post correspondence problem (PCP). The formalizations and verifications are carried out with the interactive theorem prover Coq with constructive proofs not using excluded middle. We verify 7 reductions: Halt to SR (word problem for string rewriting), SR to MPCP (Post correspondence problem with fixed first domino) and MPCP to PCP. We present an alternative, direct reduction of Halt to MPCP, and reductions of SR to RSR (word problem for string rewriting with nonempty rules) as well as RSR to PCP. In addition, the reduction of Halt to SR includes a reduction of the reachability problem for Turing machines (Reach) to SR. We observe that the correctness of the reductions is argued rather informally in the literature and that the formal verification requires considerable elaboration and effort.

Acknowledgements

I express my sincere thanks to my advisors Yannick Forster and Professor Smolka for their systematic guidance and support. Yannick always had an open ear for all my questions, gave me new insights and led me to the right direction. I am grateful for his clear words and continuous feedback. Besides, he provided this appealing LaTeX-template. I would like to thank Professor Smolka for introducing me into the field of computational logic and offering me this thesis. I deeply appreciate his passionate participation and input.

I would also like to acknowledge Professor Finkbeiner for mentoring me during my Bachelor studies and reviewing this thesis.

Matthias always cared for me and cheered me up when I was distressed. His cooking skills nourished me during long Coq sessions, loosing myself in proofs. I am so grateful for his love, patience, joy and encouraging words! Finally I would like to express my profound gratitude to my parents, who always provide spiritual support when I need it and have confidence in me. Last but not least I would like to thank Insa for her endurance and motivation over the last year. Without her, I would not have had such a fun running routine along the river Saar.

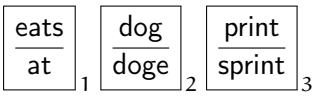
Contents

Chapter 1

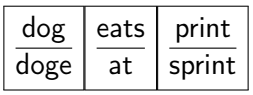
Introduction

Undecidability proofs are often based on reductions of undecidable problems that rely on the definition of Turing machines. Rice's Theorem states that every non-trivial property of the accepted language of a Turing machine is undecidable [10]. This leads to undecidable problems asking whether a Turing machine accepts all inputs, accepts a specific input (halting problem) or if it accepts a context-free language. The *correspondence decision problem*, stated by Emil L. Post in 1946 [14], is an undecidable problem which is machine independent. Today it is known as *Post correspondence problem* (PCP) and has many applications in undecidability proofs. For example, reductions of PCP prove problems related to context-free languages [10] or variants of specification formalisms [5, 16] undecidable.

An instance of PCP consists of a finite number of dominoes. The following example with three dominoes is adopted from Hesselink [9]:



The question is whether there is a finite sequence of these dominoes such that the concatenation of the top strings equals the concatenation of the bottom strings. One domino may be used several times in this sequence. Arranging the dominoes in the order $2, 1, 3$ describes the shortest solution for this instance. The concatenation of both rows yield the string dogeatsprint.



Since this example was easy to solve, it may seem surprising that in general it is not possible to decide algorithmically if an instance is solvable. The following instance uses only two different symbols and does not seem too difficult at first glance, but

needs considerable more thought.

$$\left[\frac{110}{1}\right]_1 \left[\frac{0}{111}\right]_2 \left[\frac{1}{01}\right]_3$$

This instance is indeed solvable with a sequence of 7 dominoes.[1]

Besides the general definition of PCP, there exist several restricted variants limiting the number of dominoes, the size of the alphabet, or the length of a solution. The version limiting the instance to only two dominoes is still decidable [4, 8]. While the question of undecidability for instances with three dominoes is still unsettled, using four or more dominoes leads to undecidability [13].

The undecidability of unrestricted PCP can be obtained in different ways. Emil Post presents a reduction of Post normal systems to PCP. A simplified version of his proof and a different reduction also based on normal systems can be found in Halava [7]. Many textbooks on computability theory reduce the halting problem (Halt) for Turing machines to PCP [10, 3]. In this thesis we formalize and verify the different approaches of reducing Halt to PCP from the literature.

The Halting Problem

Halt is the problem whether a Turing machine M eventually reaches a final state on a given input w. A reduction of Halt to PCP converts a Turing machine M and input w into a set of dominoes. The transformation described by a the reduction is correct, if in case the PCP instance is solvable, the machine M halts on input w and vice versa. We illustrate the idea of the reduction with an exemplary Turing machine M that accepts all strings over the alphabet $\{a, b\}$ containing an even number of a's. M has states $\{q_0, q_1, q_f\}$ with initial state q_0 and final state q_f. The symbol $\sqcup$ represents the blank and L, R moves of the machine head to the left or the right. The transition function δ is defined in Table 1.1. We are only interested in reaching the final state

q_i	$\delta(q_i, a)$	$\delta(q_i, b)$	$\delta(q_i, \sqcup)$
q_0	(q_1, a, R)	(q_0, b, R)	$(q_f, \sqcup, L)$
q_1	(q_0, a, R)	(q_1, b, R)	$(q_1, \sqcup, R)$

Table 1.1: Transition function for M.

and omit all outgoing transitions of q_f. The string aba contains an even number of a's and is accepted by M, eventually halting in the final state q_f. We represent the

[1] Take the dominoes in order 1311322 to obtain a solution. If this was still too easy, you might tackle the decision problem for the PCP instance $\left[\frac{1000}{0}\right]\left[\frac{01}{0}\right]\left[\frac{1}{101}\right]\left[\frac{00}{001}\right]$. The solution has at least 206 dominoes[21].

halting computation by configurations that include the current state to the left of the symbol under the machine head.

$$\mathbf{q_0}aba \vdash a\mathbf{q_1}ba \vdash ab\mathbf{q_1}a \vdash aba\mathbf{q_0} \vdash ab\mathbf{q_f}a \tag{1.1}$$

Since M accepts aba, the corresponding PCP instance should be solvable. This is achieved by providing dominoes emulating the computation of M. If we concatenate the strings of a domino sequence which is not yet a match, the resulting bottom row is one configuration ahead of the top row. The concatenated sequence in 1.2 represents the first two transitions of the computation in 1.1.

$$\frac{q_0aba \vdash aq_1ba \vdash \dots}{q_0aba \vdash aq_1ba \vdash abq_1a \vdash \dots} \tag{1.2}$$

Due to the infinite tape of a Turing machine, configurations can become indefinitely large. We cannot provide dominoes containing the current configuration at the top and the successor configuration at the bottom because a PCP instance is limited to finitely many dominoes. To solve this problem, we translate the transition function δ into dominoes and split configurations into two parts: A changing part including the state (indicated in bold) and symbols which do not change in the next transition.

$$ab\mathbf{q_1a} \vdash$$

Symbols that do not change are copied to the next configuration using dominoes $\boxed{\frac{a}{a}}$ and $\boxed{\frac{b}{b}}$. Dominoes representing the transition function cover the changing parts of a configuration. The domino $\boxed{\frac{\mathbf{q_1a}}{\mathbf{aq_0}}}$ for instance, corresponds to the transition $\delta(q_1, a) = (q_0, a, R)$. Adding the top and bottom strings of the dominoes $\boxed{\frac{a}{a}}\boxed{\frac{b}{b}}\boxed{\frac{\mathbf{q_1a}}{\mathbf{aq_0}}}$ to the sequence in 1.2, extends both rows by one configuration:

$$\frac{q_0aba \vdash aq_1ba \vdash ab\mathbf{q_1a}\dots}{q_0aba \vdash aq_1ba \vdash abq_1a \vdash ab\mathbf{aq_0}\dots}$$

We provide extra dominoes for all final states to complete a domino sequence to a solution. By defining these dominoes exclusively for final states, we ensure that a PCP solution always models a halting computation. However, we face the inauspicious situation that dominoes like $\boxed{\frac{a}{a}}$ already form trivial solutions of a PCP instance. To circumvent this problem and assure that the represented computation starts with the initial state and the given input word, we fix the first domino of the solution to the initial configuration (q_0aba). This restricted definition of PCP is called *modified Post correspondence problem* (MPCP). Halt is then reduced to MPCP and the undecidability of PCP follows from a reduction of MPCP.

String Rewriting

Instead of using MPCP as intermediate problem, Davis et al. [3] present a reduction of Halt to the word problem in string-rewriting systems. The word problem is the question whether a string x can be transformed into a string y using rewrite rules from a set R. Consider the instance of a word problem with the set of rules $R := \{ab/bb, ba/ab\}$, the initial string $x := aba$, and a target string $y := bbb$. In this case, a derivation from aba to bbb is possible. We substitute the bold parts of the strings according to the rules in R:

$$\mathbf{ab}a \Rightarrow b\mathbf{ba} \Rightarrow b\mathbf{ab} \Rightarrow bbb$$

The existence of a derivation from x to y for arbitrary R, x, and y is undecidable. This can be proven by a reduction of Halt similar to the reduction of Halt to MPCP. A reduction of SR to PCP completes this approach, but needs an intermediate step: The word problem for *restricted string-rewriting systems* (RSR) whose rewrite rules contain only nonempty strings, is used for the reduction to PCP.

1.1 Contribution

The reductions proving PCP undecidable that can be found in the literature are carried out informally and the authors strongly rely on the intuition of the reader when arguing about the correctness of their reductions. By formalizing and verifying the reductions of Hopcroft et al. [10] and Davis et al. [3], we reveal hidden invariants and clearly state the techniques needed to prove them correct. Beyond that, we present a reduction of SR to MPCP which is a simpler version of the reduction of RSR to PCP and can be found in Hesselink [9]. We further give a reduction of the *reachability problem* for Turing machines (Reach) to SR, which is hidden in the reduction of Halt to SR. Figure 1.1 states the different decision problems and reductions.

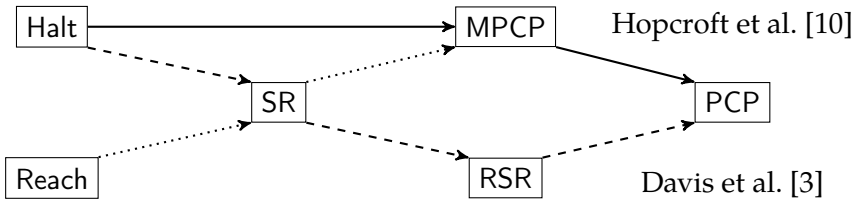


Figure 1.1: Overview illustrating the reductions verified in this thesis. The continuous line depicts the approach of Hopcroft et al. [10], the reductions from Davis et al. [3] are indicated with a dashed line. The dotted reduction of SR to MPCP can be found in Hesselink [9] and the reduction of Reach to SR can be extracted from Halt to SR.

1.2 Related Work

To the best of our knowledge, we present the first formalization of many-one reductions from the halting problem to string rewriting and PCP in constructive type theory. Asperti and Ricciotti [1; 2] formalize Turing machines in the interactive theorem prover Matita and verify a reduction of multi-tape to single-tape Turing machines. We adapt their formal definition of multi-tape Turing machines in our work. Xu et al. [20] prove the halting problem undecidable in Isabelle/HOL.

1.3 Outline

Chapter 1 contains the introduction.

Chapter 2 introduces string-rewriting systems, the Post correspondence problem, the modified Post correspondence problem, and single-tape Turing machines. Furthermore, it defines the notion of formal reductions and undecidability used throughout this thesis.

Chapter 3 presents the reduction of MPCP to PCP.

Chapter 4 presents the reduction of SR to MPCP.

Chapter 5 presents the reduction of Reach to SR.

Chapter 6 uses the results of Chapter 5 to present a reduction of Halt to SR.

Chapter 7 gives an alternative reduction of Halt to MPCP.

Chapter 8 outlines the reduction of SR to RSR and RSR to PCP.

Chapter 9 combines the results to state the undecidability of PCP and presents options for future work.

Chapter 2

Definitions

This chapter introduces formal reductions and undecidability as well as the decision problems used in this thesis. This includes the word problem in string-rewriting systems, the Post correspondence system, the modified Post correspondence problem, and the reachability and halting problem for Turing machines.

2.1 Reductions

Reductions are used to measure the relative hardness of two decision problems. We define decision problems as classes, which are unary predicates.

Definition 2.1 (Class) *Let* X *be a type. Then* $P : X \to \mathbb{P}$ *is a class over* X*. Elements* $x : X$ *are called* ***instances*** *of* P*.*

A reduction is a function that converts instances of one class into instances of a second class. The original instance is in the first class if and only if the resulting instance is in the second class.

Definition 2.2 (Reduction) *Assume two types,* X *and* Y*. A reduction of* $P : X \to \mathbb{P}$ *to* $Q : Y \to \mathbb{P}$ *is a function* $f : X \to Y$ *with*

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

Fact 2.3 *Reductions are transitive.*

We say that $P : X \to \mathbb{P}$ reduces to $Q : Y \to \mathbb{P}$ whenever there is a reduction $f : X \to Y$. The above definition formalizes the notion of many-one reductions. The reason for this is that all functions in Coq are total and also computable since our constructive development does not use further axioms.

2.2 String Rewriting

A central notion of this thesis is string rewriting which is the stepwise transformation of strings following rewrite rules. String-rewriting systems were first introduced by Axel Thue [18] and are also called semi-Thue systems. We define string-rewriting systems over a finite alphabet of symbols. Formally, a finite alphabet is represented by a finite type, a type whose elements are listable. We use **L** to denote list types. Assuming a finite alphabet Γ, we introduce the following terminology:

- a, b, c denote elements of Γ and are called symbols.
- x, y, z, u, v denote elements of type $\Gamma^* := \mathbf{L}\,\Gamma$ and are called strings.
- ax denotes cons and xy denotes concatenation.
- A rule is a pair of strings of type $\Gamma^* \times \Gamma^*$. We write u/v for a rule (u, v).
- A string rewriting system R is a finite subset of $\Gamma^* \times \Gamma^*$.

The single application of a rewrite rule and the reflexive transitive closure are defined inductively.

$$\frac{u/v \in R}{xuy \Rightarrow_R xvy} \qquad \frac{}{z \Rightarrow_R^* z} \qquad \frac{x \Rightarrow_R y \quad y \Rightarrow_R^* z}{x \Rightarrow_R^* z}$$

Fact 2.4 *$\Rightarrow^*$ is transitive.*

The word problem related to a string-rewriting system R asks whether a string y can be derived from a string x with rules from R.

Definition 2.5 *The word problem in string-rewriting systems is the class*

$$\mathsf{SR}\,(R, x, y) := x \Rightarrow_R^* y.$$

We also define the word problem for restricted string-rewriting systems (RSR) where R contains only rules (u/v) with u and v being nonempty strings. Formally, an instance of RSR carries a proof that no rule is empty.

Definition 2.6 *The word problem in restricted string-rewriting systems where x and y are strings over Γ is defined as*

$$\mathsf{RSR}\,(\{R : \mathbf{L}\,(\Gamma^* \times \Gamma^*) \mid \forall u\,v.\,(u/v) \in R \to u \neq [\,] \wedge v \neq [\,]\}, x, y) := x \Rightarrow_R^* y.$$

2.3 The Post Correspondence Problem

A PCP instance is a finite set of dominoes. We say an instance is solvable if there is a nonempty list of dominoes from the set, possibly containing duplicates, where the concatenated top and bottom rows are equal.

Example 2.7 *The PCP instance* $\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\}$ *has the solution* $\boxed{\frac{10111}{10}}\boxed{\frac{1}{111}}\boxed{\frac{1}{111}}\boxed{\frac{10}{0}}$.

Formally, we assume an alphabet Σ and define **dominoes** as pairs of strings and PCP instances as lists of dominoes.

$$\begin{aligned} \mathsf{domino} &:= \Sigma^* \times \Sigma^* \\ \mathsf{pcp} &:= \mathbf{L}\,\mathsf{domino} \end{aligned}$$

We use the variables P and S to refer to lists of dominoes, and d to denote a single domino.

Definition 2.8 (PCP match) *A list* S *is a match if the concatenation of the top strings equals the concatenation of the bottom strings.*

$$\begin{aligned} \mathsf{S}\ \textit{is a}\ \textbf{\textit{match}} &:= \mathsf{concat}\,(\mathsf{map}\,\pi_1\ \mathsf{S}) = \mathsf{concat}\,(\mathsf{map}\,\pi_2\ \mathsf{S}) \\ \mathsf{S}\ \textit{is a}\ \textbf{\textit{match for}}\ \mathsf{P} &:= \mathsf{S}\ \textit{is a match}\ \wedge\ \mathsf{S} \neq [\,] \wedge\ \mathsf{S} \subseteq \mathsf{P} \end{aligned}$$

We may abbreviate $\mathsf{concat}\,(\mathsf{map}\,\pi_1\ \mathsf{S})$ with $\mathsf{C}_1\,\mathsf{S}$, and $\mathsf{concat}\,(\mathsf{map}\,\pi_2\ \mathsf{S})$ with $\mathsf{C}_2\,\mathsf{S}$.

Definition 2.9 *The class*

$$\mathit{PCP}\,(\mathsf{P} : \mathsf{pcp}) := \exists\ \mathsf{S},\ \mathsf{S}\ \textit{is a match for}\ \mathsf{P}$$

characterizes all solvable PCP instances.

2.4 The Modified Post Correspondence Problem

Contrary to PCP instances, where the order of the dominoes is not relevant, an instance of the modified Post correspondence problem (MPCP) separates one domino from the remaining ones.

$$\mathsf{mpcp} := \mathsf{domino} \times \mathsf{pcp}$$

In addition, a match for an MPCP instance $(\mathtt{d}, \mathsf{P})$ must begin with domino $\mathtt{d}$, the first component of the pair. The definition of the MPCP class contains the notion of being a match for a PCP instance.

Definition 2.10 *The class*

$$\mathsf{MPCP}\ (\mathtt{d}, \mathsf{P}) := \exists\,\mathsf{S}, (\mathtt{d} :: \mathsf{S})\ \textit{is a match for}\ (\mathtt{d} :: \mathsf{P})$$

characterizes all solvable MPCP instances.

Note that the list S is preceded by the first domino d of the MPCP instance before it is checked to be a match. Thus it might be empty if the first domino is a trivial match. Furthermore, we use the set notation in pictorial representations of MPCP instances, marking the special first domino with a thick domino border.

Example 2.11 *A match for the MPCP instance* $\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\}$*, which must start with* $\boxed{\frac{10111}{10}}$*, would be* $\boxed{\frac{10111}{10}}\left|\frac{1}{111}\right|\frac{1}{111}\left|\frac{10}{0}\right|$.

We call a list of dominoes a **partial match**, if the concatenated top row is a substring of the concatenated bottom row or vice versa. If suitable dominoes are available, a partial match can be completed to a match.

2.5 Turing Machines

The computational model of a Turing machine can be defined in many different ways. Turing machines may be deterministic or indeterministic, differ in the number of tapes and the movements of the machine head, and the tape may be infinite to both sides or only semi-infinite. In this thesis we adapt the formalization of of multi-tape Turing machines from Asperti and Ricciotti [2] to define symmetric single-tape Turing machines having a doubly-infinite tape. We start with the central concept of their Turing model, which is the definition of tapes.

A tape describes the content to the left, to the right, and under the tape head ↑. We assume the tape symbols to be from a fixed alphabet Σ, and use $a : \Sigma$ and lists $A, B : \mathbf{L}\,\Sigma$. We do not include the infinite number of blank cells to both sides of the tape content, which leads to the following kinds of tapes:

$$\underset{\uparrow}{\emptyset} \qquad \underset{\uparrow}{aA} \qquad \underset{\uparrow}{BaA} \qquad \underset{\uparrow}{Ba}$$

Only the third tape describes the situation where the head points to a symbol. Formally we define a type tape representing these four cases.

Definition 2.12 $\mathsf{tape} := \emptyset \mid \mathsf{leftof}\ a\ A \mid \mathsf{midtape}\ B\ a\ A \mid \mathsf{rightof}\ a\ B \quad (a : \Sigma)\,(A\ B : \Sigma^*)$

It turns out to be convenient providing the content to the left of the tape head in reversed oder. We use the superscript ^{R} to reverse a list. With this convention, the tape corresponding to $(\mathsf{rightof}\ a\ B)$ will be $(B^R \underset{\uparrow}{a}\ )$. The decision to represent the content with lists over Σ symbols accompanies the fact that we do not define a blank symbol and the tape head can not run over empty cells. To indicate that the head is not reading a symbol, we use an option type $\Sigma_\perp$ as input for the transition function.

A Turing machine M over a finite alphabet Σ, which represents the input and the tape alphabet, is defined as the quadruple

$$M := (Q, \delta, q_0, H)$$

consisting of the following components:

- A finite type of states Q.
- A transition function $\delta : Q \times \Sigma_\bot \to Q \times \Sigma_\bot \times \{L, N, R\}$, defined for all states and options $\Sigma_\bot$. An option either names a symbol from the alphabet, represented as $\lfloor a \rfloor$ or declares the current cell empty ($\bot$). The resulting triple $\delta(q, o) = (q', o', m)$ specifies
  - The new state q'.
  - An option o' indicating whether to write a new symbol on the tape or to leave the cell unchanged.
  - A *move* m of the tape head. L is a move to the *left*, R to the *right*, and N stands for *none* where the head remains stationary.
- The starting state $q_0 : Q$.
- A function indicating all halting states $H : Q \to \mathbb{B}$.

Note that limiting the moves of the tape head to L and R would suffice and agree with the standard definitions. Following [2], we adopt the moves from multi-tape Turing machines, to keep a uniform representation. Further, we do not apply the transition function δ to halting states and omit these values when defining transition functions for examples.

The following definitions and examples take place in the context of a Turing machine $M = (Q, \delta, q_0, H)$ over the alphabet Σ.

Effects of a Transition

To explain the effect of the transition function δ on tapes, we distinguish the case where a symbol is written on the tape from the case where the cell under the head remains unchanged. If we write a symbol $\lfloor b \rfloor$, the content of the cell under the tape head is changed to b and the head performs a specified move from $\{L, N, R\}$. If the writing option is $\bot$, either move does not have an effect on the empty tape. The situation with leftof and rightof tapes is different as a move further away from the tape content has not effect: A move to the left on $(\underset{\uparrow}{\,} aA)$ results in $(\underset{\uparrow}{\,} aA)$ and a move to the right on $(Ba\underset{\uparrow}{\,})$ results in $(Ba\underset{\uparrow}{\,})$.

A tape does not contain enough information to determine the result of the transition function. Therefore, we use **configurations** which are comprised of the current state and a tape.

$$\mathsf{conf} := Q \times \mathsf{tape}$$

Rather than $(q, \mathsf{midtape}\, B\, a\, A)$, we will use the pictorial representation $(q, B^R \underset{\uparrow}{a} A)$ of configurations. A sequence of successive configurations is called a computation of a Turing machine. We define a **step function** $\hat{\delta}$ which interprets the result of the transition function δ and computes the successor configuration.

$$\hat{\delta} : \mathsf{conf} \to \mathsf{conf}$$

While the successor state is directly given by δ, the tape is modified depending on the optional symbol and the move of the tape head.

Example 2.13 (Midtape move) *Assume a configuration* $(q, B \underset{\uparrow}{b} a A)$ *and* $\delta(q, \lfloor b \rfloor) = (q', \lfloor c \rfloor, R)$. *The step-function yields* $\hat{\delta}(q, B \underset{\uparrow}{b} a A) = (q', B c \underset{\uparrow}{a} A)$

Example 2.14 (Leftof move) *The configuration* $(q, \underset{\uparrow}{\ } a A)$ *describes the situation where the tape head points to an empty cell. Assume* $\delta(q, \bot) = (q', \bot, L)$. *In this case the step function yields* $\hat{\delta}(q, \underset{\uparrow}{\ } a A) = (q', \underset{\uparrow}{\ } a A)$ *which leaves the tape unchanged.*

Reachability of Configurations

We do not apply the step function to **final configurations** which contain a halting state. To identify those, we introduce the notation

$$H_c := H(\pi_1\, c) = \mathsf{true}.$$

The following inductive predicate relates two configurations if the second can be obtained by the repeated application of the step function to the first configuration.

Definition 2.15 (Reachability) *The configuration* c' *is reachable from* c *if* $c \vdash c'$.

$$\frac{}{c' \vdash c'} \qquad \frac{\hat{\delta}\, c \vdash c' \quad \neg H_c}{c \vdash c'}$$

Definition 2.16 *The class Reach states reachability of two configurations with respect to the transition function of a Turing machine* M.

$$\mathsf{Reach}(M, c_1, c_2) := c_1 \vdash_M c_2$$

We say a Turing machine halts, if it reaches a final configuration containing a halting state. The **halting problem** is the question whether a Turing machine halts, starting in configuration (q_0, t) for a given initial tape t.

Definition 2.17 *The class*

$$\mathsf{Halt}\,(M,t) := \exists\, c_f.\,(q_0,t) \vdash_M c_f \;\wedge\; \mathsf{H}_{c_f}$$

denominates all pairs of Turing machines M *and tapes* t *such that* M *halts on the initial configuration* (q_0,t)*.*

2.6 Undecidability

Coq's type theory is constructive and thus comes with a built-in notion of computability enabling straightforward definitions of decidable classes and computable reductions between classes. However, the concomitant notion of undecidability is useless since the assumption that every class is decidable is consistent and no class can be shown undecidable within Coq's type theory.

To circumvent this fact, we call a class Turing undecidable if the halting problem reduces to it. The internal notion of Turing undecidability then formalizes the external notion of undecidability and external undecidability proofs can be formalized in Coq. Since external undecidability proofs consist of reductions and the concomitant correctness proofs, they are well-suited for formalization in a constructive type theory.

Definition 2.18 *A class* $P : X \to \mathbb{P}$ *is* ***undecidable*** *if Halt reduces to* P*.*

This means there exists a function f such that $\forall\, M\, t.\, \mathsf{Halt}\,(M,t) \leftrightarrow P\,(f\,(M,t))$.

Chapter 3

Reducing MPCP to PCP

In this reduction we transform an MPCP instance (d, P) into a PCP instance consisting of a single list of dominoes. Recall the definitions of these two classes:

$$\begin{aligned} S \text{ is a match for } P &:= S \text{ is a match, } S \neq [\,] \text{ and } S \subseteq P \\ \mathsf{PCP}\ (P : \mathsf{pcp}) &:= \exists\, S,\ S \text{ is a match for } P \\ \mathsf{MPCP}\ (d, P) &:= \exists\, S, (d :: S) \text{ is a match for } (d :: P) \end{aligned}$$

Reducing MPCP to PCP means that the original MPCP instance is solvable if and only if the constructed PCP instance has a match. Since all matches for an instance (d, P) start with domino d, we need to make sure that all PCP matches also start with d. We cannot change the definition for a list to be a PCP match, such that the external condition to start with a special domino has to be internalized in the PCP dominoes itself.

To ensure that d starts every match, we transform all other dominoes in P such they do not fit at the beginning: A new symbol #, which is not in the original alphabet, is inserted to the left of each symbol in the top row and to the right of each symbol in the bottom row of all dominoes. Additionally, we define the domino d to start at the top and the bottom with #, and provide a final domino to compensate the extra # in the top row. With this modification, all MPCP matches starting with d can still be constructed and are simply interleaved with # symbols.

The idea of this reduction can be found in Hopcroft et al. [10]. They implicitly consider all dominoes to be pairs of nonempty strings, which is not the case in our definition but required for the correctness. For this reason, the deletion of empty dominoes is part of our reduction.

Before defining the reduction formally, we elaborate the underlying idea with examples. The most simple transformation of an MPCP instance (d, P) into a PCP

instance would add the first domino d to the remaining dominoes, creating the instance $(d :: P)$. The following instances demonstrate why this naive approach fails.

Example 3.1 *We transform a solvable MPCP instance* $\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\}$ *into the PCP instance* $\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\}$. *In this special case we observe the desired result that the MPCP match is the same for the PCP instance:*

$$\text{MPCP } \textit{match}: \boxed{\frac{10111}{10}}\boxed{\frac{1}{111}}\boxed{\frac{1}{111}}\boxed{\frac{10}{0}}$$

$$\text{PCP } \textit{match}: \boxed{\frac{10111}{10}}\boxed{\frac{1}{111}}\boxed{\frac{1}{111}}\boxed{\frac{10}{0}}$$

Example 3.2 *By choosing a different starting domino, we obtain the unsolvable MPCP instance*

$$\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\}.$$

If we proceed the same way as before to transform the above instace into

$$\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\},$$

we face the problem that the latter is indeed solvable, since it is equivalent to the instance from Example 3.1.

These examples illustrate the need to inhibit other dominoes from starting a match. Inserting the # symbol to the left of the symbols in the top row and to the right of each symbol in the bottom row, a domino $\boxed{\frac{a_1 a_2 \dots a_n}{b_1 b_2 \dots b_m}}$ is transformed into $\boxed{\frac{\#a_1\#a_2\dots\#a_n}{b_1\#b_2\#\dots b_m\#}}$. We abbreviate the application of this insertion process to each pair of a set of dominoes with $\overline{\#}$.

Example 3.3 *Convince yourself that no domino in the right set can be used to start a match.*

$$\overline{\#}\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\} = \left\{\boxed{\frac{\#1}{1\#1\#1\#}}, \boxed{\frac{\#1\#0\#1\#1\#1}{1\#0\#}}, \boxed{\frac{\#1\#0}{0\#}}\right\}$$

We also need to add a modified version of the designated first domino of an MPCP instance that can be used to start a PCP match. In addition, we provide a domino completing a match with the missing # at the top. These dominoes include another new symbol \$ to emphasize that they are only used at the very beginning or the end of a match. This simplifies formal proofs.

Example 3.4 *Transforming the MPCP instance* $\left\{\boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\}$ *into*

$$\left\{\boxed{\frac{\$\#1\#0\#1\#1\#1}{\$\#1\#0\#}}, \boxed{\frac{\#1}{1\#1\#1\#}}, \boxed{\frac{\#1\#0\#1\#1\#1}{1\#0\#}}, \boxed{\frac{\#1\#0}{0\#}}, \boxed{\frac{\#\$}{\$}}\right\}$$

ensures that a match can only start with the modified first domino $\boxed{\frac{\$\#1\#0\#1\#1\#1}{\$\#1\#0\#}}$*:*

$\frac{\$\#1\#0\#1\#1\#1}{\$\#1\#0\#}$	$\frac{\#1}{1\#1\#1\#}$	$\frac{\#1}{1\#1\#1\#}$	$\frac{\#1\#0}{0\#}$	$\frac{\#\$}{\$}$

3.1 Definition of PCP Dominoes

To state the formal reduction, we fix an MPCP instance (d, P) over alphabet Σ for the remainder of this chapter. The resulting type of the PCP instance is defined as Γ, including the new symbols # and \$.

$$\Gamma ::= \# \mid \$ \mid s : \Sigma$$

Definition 3.5 (Functions inserting # symbols) *We define the functions* $\#_L$ *and* $\#_R$*, inserting # to the left or the right of all symbols of a string* A *over* Σ*.*

$$\begin{aligned}
\#_L &: \Sigma^* \to \Gamma^* \\
\#_L\ A &:= \mathsf{concat}\ (\mathsf{map}\ (\lambda s.\ [\#; s])\ A) \\
\\
\#_R &: \Sigma^* \to \Gamma^* \\
A\,\#_R &:= \mathsf{concat}\ (\mathsf{map}\ (\lambda s.\ [s; \#])\ A) \\
\\
\overline{\#} &: \mathsf{pcp}_\Sigma \to \mathsf{pcp}_\Gamma \\
\overline{\#} &:= \mathsf{map}\ (\lambda p.\ (\#_L\ (\pi_1\ p),\ (\pi_2\ p)\,\#_R))
\end{aligned}$$

We use the $\#_R$ *operation in postfix notation. The function* $\overline{\#}$ *applies both operations to a list of dominoes.*

The functions $\#_L$ and $\#_R$ have no effect on empty dominoes like $\boxed{\frac{}{}}$. This fact is problematic, since an empty domino could be used as a trivial match for a PCP instance.

Example 3.6 *The original intention of altering the dominoes such that no domino can start a match, is not met when we have a domino* $\boxed{\frac{}{}}$ *in the set:*

$$\overline{\#}\left\{\boxed{\frac{}{}}, \boxed{\frac{1}{111}}, \boxed{\frac{10111}{10}}, \boxed{\frac{10}{0}}\right\} = \left\{\boxed{\frac{}{}}, \boxed{\frac{\#1}{1\#1\#1\#}}, \boxed{\frac{\#1\#0\#1\#1\#1}{1\#0\#}}, \boxed{\frac{\#1\#0}{0\#}}\right\}$$

To resolve this problem we delete empty dominoes before applying $\overline{\#}$ using the notation $\mathsf{del}_{\square}$. Next we define the reduction of MPCP to PCP.

Definition 3.7 (Reduction) *The function* f' *modifies* d *as a starting domino and provides a domino to complete a match. All dominoes in* P *are interleaved with* # *symbols using* $\overline{\#}$*. This function is separated from the reduction, because it is also used to transform an MPCP match into a PCP match.*

$$f' : \mathsf{mpcp}_\Sigma \to \mathsf{pcp}_\Gamma$$

$$f'(d, P) := \left[\frac{\$ :: (\#_L\ (\pi_1\ d))}{\$ :: \# :: ((\pi_2\ d)\ \#_R)}\right] :: \overline{\#}\ (\mathsf{del}_{\square}\ P) +\!\!+ \left[\frac{\#\$}{\$}\right]$$

Since d *might be needed in the middle of a match it is added to the set of dominoes* P *which yields the reduction* f*.*

$$f : \mathsf{mpcp}_\Sigma \to \mathsf{pcp}_\Gamma$$

$$f(d, P) := f'(d, (d :: P))$$

Example 3.8 *Applying the reduction to an unsolvable MPCP instance, the deletion of empty dominoes and the # symbols ensure that the resulting PCP instance is unsolvable too.*

$$f\left(\left[\frac{10}{0}\right], \left\{\left[\frac{}{}\right], \left[\frac{1}{111}\right], \left[\frac{10111}{10}\right]\right\}\right) = \left\{\left[\frac{\$\#1\#0}{\$\#0\#}\right], \left[\frac{\#1\#0}{0\#}\right], \left[\frac{\#1\#0\#1\#1\#1}{1\#0\#}\right], \left[\frac{\#1}{1\#1\#1\#}\right], \left[\frac{\#\$}{\$}\right]\right\}$$

Before stepping into the correctness proof, we discuss how to transform a match of type pcp_Γ into a match for the original MPCP instance of type mpcp_Σ. Since the # symbols have been inserted in an interleaving manner, eliminating those again does not affect the match itself.

Definition 3.9 *The function* $\mathsf{del}_{\#\$}$ *removes all* # *and* \$ *symbols, returning a string over* Σ*.*

$$\begin{aligned} \mathsf{del}_{\#\$} &: \Gamma^* \to \Sigma^* \\ g &: \mathsf{pcp}_\Gamma \to \mathsf{pcp}_\Sigma \\ g &:= \mathsf{map}\ (\lambda\,(x, y)\,.\,(\mathsf{del}_{\#\$}\ x\,,\ \mathsf{del}_{\#\$}\ y)) \end{aligned}$$

Since $\mathsf{del}_{\#\$}\left[\frac{\#\$}{\$}\right]$ results in the empty domino $\left[\frac{}{}\right]$, we use $\mathsf{del}_{\square}$ on top of g to receive only dominoes that can be found in the original MPCP instance.

Example 3.10 *The composition of* $\mathsf{del}_{\square}$ *and* g *yields a match for the original MPCP instance from Example 3.4.*

$$\mathsf{del}_{\square}\left(g\left\{\left[\frac{\$\#1\#0\#1\#1\#1}{\$\#1\#0\#}\right], \left[\frac{\#1}{1\#1\#1\#}\right], \left[\frac{\#1}{1\#1\#1\#}\right], \left[\frac{\#1\#0}{0\#}\right], \left[\frac{\#\$}{\$}\right]\right\}\right) = \left\{\left[\frac{10111}{10}\right], \left[\frac{1}{111}\right], \left[\frac{1}{111}\right], \left[\frac{10}{0}\right]\right\}$$

3.2 Correctness Proof

In this section we show the reduction f to be correct by proving the equivalence

$$\mathsf{MPCP}\,(\mathsf{d},\mathsf{P}) \;\leftrightarrow\; \mathsf{PCP}\,(\mathsf{f}\,(\mathsf{d},\mathsf{P})).$$

Recall the notation $\mathsf{C}_1\,A := \mathsf{concat}\,(\mathsf{map}\;\pi_1\;A)$ and $\mathsf{C}_2\,A := \mathsf{concat}\,(\mathsf{map}\;\pi_2\;A)$.

Fact 3.11 (Equivalences) *Let A be a string over Σ and S a list of dominoes.*

a) $\mathsf{C}_1\;(\overline{\#}\,A) = \#_L\;(\mathsf{C}_1\,A)$

b) $\mathsf{C}_2\;(\overline{\#}\,A) = (\mathsf{C}_2\,A)\;\#_R$

c) $\#_L\,A +\!\!+ [\#] = [\#] +\!\!+ A\,\#_R$

d) $\mathsf{C}_1\;(\mathsf{del}_{\square}\;S) = \mathsf{C}_1\;S$

e) $\mathsf{C}_2\;(\mathsf{del}_{\square}\;S) = \mathsf{C}_2\;S$

Transforming MPCP Matches to PCP Matches

We use the function f' to transform a match for the MPCP instance (d,P) into a match for the PCP instance $\mathsf{f}\,(\mathsf{d},\mathsf{P})$.

Lemma 3.12 *If $(\mathsf{d}::S)$ is a match, then the list $\mathsf{f}'\,(\mathsf{d},S)$ is a match as well.*

Proof By equations (a) to (e) of Fact 3.11

Lemma 3.13 *If $\mathsf{MPCP}\,(\mathsf{d},\mathsf{P})$, we can construct a match for the PCP instance $\mathsf{f}\,(\mathsf{d},\mathsf{P})$.*

Proof Let S be a list such that $(\mathsf{d}::S)$ is a match for (d,P). By definition it holds that $(\mathsf{d}::S) \subseteq (\mathsf{d}::\mathsf{P})$ and $\mathsf{C}_1\,(\mathsf{d}::S) = \mathsf{C}_2\,(\mathsf{d}::S)$. By Lemma 3.12 we know that $\mathsf{f}'\,(\mathsf{d},S)$ is a match. $\mathsf{f}'\,(\mathsf{d},S) \subseteq \mathsf{f}'\,(\mathsf{d},\mathsf{d}::\mathsf{P})$ follows from the assumption $(\mathsf{d}::S) \subseteq (\mathsf{d}::\mathsf{P})$. □

Transforming PCP Matches to MPCP Matches

In the other proof direction we have a match $S : \mathtt{pcp}_\Gamma$ for the PCP instance $\mathsf{f}\,(\mathsf{d},\mathsf{P})$ and need to provide a list $S' : \mathtt{pcp}_\Sigma$ such that $(\mathsf{d}::S')$ is a match. We aim to identify the first domino of S as the modified domino d. Remember the intention of inserting #-symbols to prevent dominoes that are different from d from starting a match.

Lemma 3.14 *If S is a match for $\mathsf{f}\,(\mathsf{d},\mathsf{P})$, then* $\boxed{\frac{\$\#_L\,(\mathsf{fst}\;\mathsf{d})}{\$\#\;\;(\mathsf{snd}\;\mathsf{d})\,\#_R}}$ *is the first domino of S.*

Proof Since a match cannot be empty, we have $\mathsf{s}::S' \subseteq \mathsf{f}\,(\mathsf{d},\mathsf{P})$. We continue by case analysis on $\mathsf{s} \in \mathsf{f}\,(\mathsf{d},\mathsf{P})$.

$s = \boxed{\frac{\$\#_L\,(\mathsf{fst}\,d)}{\$\#\,(\mathsf{snd}\,d)\,\#_R}}$ The claim holds.

$s \in \overline{\#}\,(\mathsf{del}_{[]}\,(d, P))$ Let $s = (A, B)$. If $A = a :: A$ and $B = b :: B$, the first domino is $\boxed{\frac{\#\,a(\#_L\,A)}{b\,\#\,(B\,\#_R)}}$, which is contradictory. If either $A = [\,]$ or $B = [\,]$, we prove $C_1\ S = a :: \# :: A +\!\!+ C_2\ S$ and $\# :: a :: A +\!\!+ C_1\ S = C_2\ S$ to be contradictory for all $a \in \Sigma$. This is done by induction on S.

$s = \boxed{\frac{\#\$}{\$}}$ Contradiction because $\# \neq \$$. □

Now we are close to stating a list that is an MPCP match when preceded by the domino d. The last step removes the additional symbols # and \$ and the resulting empty dominoes using $\mathsf{del}_{[]}$ and $g : \mathsf{pcp}_\Gamma \to \mathsf{pcp}_\Sigma$.

Fact 3.15 *Let A be a string over Σ and $S : \mathrm{pcp}_\Gamma$.*

a) $C_1\,S = C_2\,S \to C_1\,(g\,S) = C_2\,(g\,S)$

b) $\mathsf{del}_{\#\$}\,(A\,\#_R) = A$

c) $\mathsf{del}_{\#\$}\,(\#_L\,A) = A$

Lemma 3.16 *If the PCP instance $f\,(d, P)$ has a match, then there is a match for (d, P).*

Proof Let $d = (d_1, d_2)$. By Lemma 3.14 it holds that the match for the PCP instance must be of the form $\boxed{\frac{\$\#_L\,d_1}{\$\#\ \ d_2\,\#_R}} :: S$. We prove that the list $\mathsf{del}_{[]}\,(g\,S)$ is a match for (d, P). By definition of MPCP we need to show that $(d :: \mathsf{del}_{[]}\,(g\,S))$ is a match for $(d :: P)$.

- $d :: \mathsf{del}_{[]}\,(g\,S) \subseteq d :: P$: All dominoes in S must be in $f\,(d, P)$ because S is a match for this instance. By Fact 3.15 (b) and (c), g reverses the $\overline{\#}$ operation and the domino $\boxed{\frac{\#\$}{\$}}$ is eliminated with $\mathsf{del}_{[]}$.
- $C_1\,(d :: \mathsf{del}_{[]}\,(g\,S)) = C_2\,(d :: \mathsf{del}_{[]}\,(g\,S))$: By Fact 3.11 (d) and (e) it suffices to prove $d_1 +\!\!+ C_1\,(g\,S) = d_2 +\!\!+ C_2\,(g\,S)$. This follows the assumption that $\boxed{\frac{\$\#_L\,d_1}{\$\#\ \ d_2\,\#_R}} :: S$ is a match and Fact 3.15. □

Theorem 3.17 *MPCP reduces to PCP.*

$$\forall\, d\, P.\, \mathsf{MPCP}\,(d, P) \leftrightarrow \mathsf{PCP}\,(f\,(d, P))$$

Proof Follows with Lemmas 3.13 and 3.16. □

Altogether, the correctness proof of this reduction uses several technical facts, most of them are proven by list induction. All other lemmas only use case analysis and the equivalences established in Facts 3.11 and 3.15.

In our reduction we insert the \$ symbol at the beginning of the initial domino $\left[\frac{\$\#_L(\pi_1\ d)}{\$\#\ (\pi_2\ d)\,\#_R}\right]$. In [10] it is only used at the end of the final domino. This is possible since they assume all dominoes to be pairs of non empty strings. We might have dominoes with one empty string in our PCP instance:

$$f\left(\left\{\left[\frac{1}{01}\right], \left[\frac{0}{-}\right]\right\}\right) = \left\{\left[\frac{\#1}{\#0\#1\#}\right], \left[\frac{\#1}{0\#1\#}\right], \left[\frac{\#0}{-}\right], \left[\frac{\#\$}{\$}\right]\right\}$$

A match for this PCP instance is $\left[\frac{\#0}{-}\middle|\frac{\#1}{\#0\#1\#}\middle|\frac{\#\$}{\$}\right]$, whereas the original MPCP instance is not solvable. Lemma 3.14 would not be provable without this extra \$ symbol in front.

Chapter 4

Reducing String Rewriting to MPCP

A reduction of the word problem in string-rewriting systems to MPCP transforms a set of rules, the initial string x, and the target string y into a designated first domino and a list of dominoes. The dominoes are designed such that each row of an MPCP match depicts a rewrite sequence from x to y.

We start with a domino where the initial string x at the bottom exceeds the top component. In all partial matches, the bottom row will be longer than the top row, determining the upcoming dominoes. To simulate one rewrite step, we provide *rewrite dominoes* $\boxed{\frac{u}{v}}$ for each rule u/v which tie together the current and the next string. Since a rewrite rule might be applied in the midst of a string we need to transfer unchanged symbols from the current to the next string. This is done by *copy dominoes* having the same symbol from the alphabet at the top and the bottom component. To prevent rewrite dominoes from covering two successive strings, which is not possible in string rewriting, we insert an additional symbol $\star$ between consecutive strings. If a partial match exceeds the top row by the target string y, a single domino with y in the top component is defined to complete the match.

The general idea of this reduction can be found in [9], where the question of rewriting to the empty string is reduced to MPCP.

Example 4.1 *The set of rules* $R{:=}\{ab/ba, aa/ab\}$ *characterizes a string-rewriting system over the alphabet* $\{a, b\}$*. The derivation*

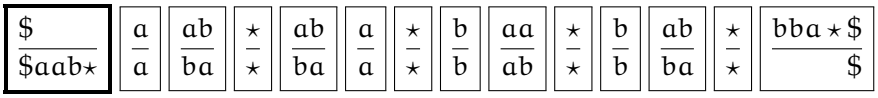
$$aab \Rightarrow aba \Rightarrow baa \Rightarrow bab \Rightarrow bba$$

illustrates that $(aab) \Rightarrow_R^* (bba)$ *holds. To obtain an MPCP instance, we define the first domino, the last domino, copy dominoes for the symbols* a, b *and the separator* $\star$*, and dominoes representing rewrite rules as follows:*

$$\left\{ \boxed{\frac{\$}{\$aab\star}}, \boxed{\frac{bba\star\$}{\$}}, \boxed{\frac{a}{a}}, \boxed{\frac{b}{b}}, \boxed{\frac{\star}{\star}}, \boxed{\frac{ab}{ba}}, \boxed{\frac{aa}{ab}} \right\}$$

The \$ symbol ensures that the first and the last domino cannot be used inside a match. The MPCP match below corresponds to the rewriting sequence from aab *to* bba*:*

$$\mathtt{aab} \Rightarrow \quad \mathtt{aba} \quad \Rightarrow \quad \mathtt{baa} \quad \Rightarrow \quad \mathtt{bab} \quad \Rightarrow \quad \mathtt{bba}$$



4.1 Definition of MPCP Dominoes

For the remaining of this chapter, we assume an arbitrary string-rewriting system $R \subseteq \Sigma^* \times \Sigma^*$ and strings x, y in Σ^*. The alphabet of the MPCP instance to construct, consists of all symbols from Σ, the separator $\star$, and \$:

$$\Gamma := \$ \mid \star \mid (a : \Sigma)$$

Definition 4.2 (Reduction) *A string-rewriting instance* (R, x, y) *is converted into a MPCP instance with dominoes containing* x *and* y *to start and end a solution, respectively. Then, there are dominoes to copy symbols from* Σ *and to simulate rewrite rules from* R*.*

$$f : \mathbf{L}\,(\Sigma^* \times \Sigma^*) \times \Sigma^* \times \Sigma^* \to \mathsf{mpcp}_\Gamma$$

$$f\,(R, x, y) := \left\{ \boxed{\frac{\$}{\$x\star}}, \boxed{\frac{y\star\$}{\$}}, \boxed{\frac{\star}{\star}} \right\} \cup \left\{ \boxed{\frac{a}{a}} \,\middle|\, a : \Sigma \right\} \cup \left\{ \boxed{\frac{u}{v}} \,\middle|\, u/v \in R \right\}$$

4.2 Correctness Proof

We prove that the MPCP instance $f\,(R, x, y)$ is solvable if and only if x rewrites to y with rules in R.

$$x \Rightarrow_R^* y \leftrightarrow \mathsf{MPCP}\,(f\,(R, x, y))$$

Since for all symbols $a : \Sigma$ there are dominoes $\boxed{\frac{a}{a}}$, we abbreviate the domino representation of a string $z := z_1 z_2 \ldots z_n$. Instead of enumerating $\boxed{\frac{z_1}{z_1}}\boxed{\frac{z_2}{z_2}} \ldots \boxed{\frac{z_n}{z_n}}$, we will use the notation $\boxed{\frac{z}{z}}$.

Lemma 4.3 *If* $x \Rightarrow_R^* y$*, then there is some* $A \subseteq f\,(R, x, y)$ *such that*

$$(C_1\,A) +\!\!+\, y +\!\!+\, [\star] = x +\!\!+\, [\star] +\!\!+\, (C_2\,A)$$

Proof By induction on the derivation $\Rightarrow^*$.

$x = y$ Take $A := [\,]$.

$x \Rightarrow_R z$ Let $x = x_1 u x_2$ and $z = x_1 v x_2$ with $u/v \in R$. By the inductive hypothesis we
$z \Rightarrow_R^* y$ have a list $A' \subseteq f(R, x, y)$ with $(C_1\ A') +\!\!+ y +\!\!+ [\star] = x_1 v x_2 +\!\!+ [\star] +\!\!+ (C_2\ A')$.
We define $A := \boxed{\frac{x_1}{x_1}} :: \boxed{\frac{u}{v}} :: \boxed{\frac{x_2}{x_2}} :: \boxed{\frac{\star}{\star}} :: A'$ and have $x_1 u x_2 \star :: (C_1\ A') +\!\!+ [y] +\!\!+ [\star] = x_1 u x_2 +\!\!+ [\star] +\!\!+ x_1 v x_2 \star :: (C_2\ A)$. □

Lemma 4.4 *If $x \Rightarrow_R^* y$, then the MPCP instance $f(R, x, y)$ is solvable.*

Proof By Lemma 4.3 we have a list $A \subseteq f(R, x, y)$ with $(C_1\ A) +\!\!+ y +\!\!+ [\star] = x +\!\!+ [\star] +\!\!+ (C_2\ A)$. The list $\boxed{\frac{\$}{\$x\star}} :: A +\!\!+ \boxed{\frac{y\star\$}{\$}}$ is a match and contains only dominoes in $f(R, x, y)$. □

The opposite direction of the correctness statement requires that a solvable MPCP instance $f(R, x, y)$ implies that y can be derived from x with rules from R. The proof is slightly more involved and needs some insight how MPCP solutions are composed. By construction of the reduction, it is possible that two consecutive strings in a solution list A which are separated by $\star$ encode zero, one, or several rewrite steps. Consider the rules $R := \{ab/ba, aa/ab\}$ and the following solution for the MPCP instance $f(R, aaaa, baba)$:

$$\boxed{\frac{\$}{\$aaaa\star}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{aa}{ab}}\ \boxed{\frac{\star}{\star}}\ \boxed{\frac{aa}{ab}}\ \boxed{\frac{ab}{ba}}\ \boxed{\frac{\star}{\star}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{\star}{\star}}\ \boxed{\frac{ab}{ba}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{\star}{\star}}\ \boxed{\frac{baba\star\$}{\$}}$$

When concatenating these dominoes, it becomes obvious that the third string is the result of rewriting $aaab$ with both rules in R, whereas the fourth, $abba$, equals the previous.

$$\frac{\$aa\mathbf{aa}\star\mathbf{aab}\star abba\star\mathbf{ab}ba\star baba\star\$}{\$aaaa\star aa\mathbf{ab}\star\mathbf{abba}\star abba\star\mathbf{ba}ba\star\$}$$

As a consequence, the two forthcoming Lemmas 4.6 and 4.7 use $\Rightarrow^*$ instead of $\Rightarrow$ to establish the connection between string rewriting and the MPCP match. Additionally, the proofs are done by size induction on the length of the solution list, because one string might be represented by more than one domino.

Fact 4.5 *Let A and B be two strings which contain only Σ symbols and C, D strings over Γ. If $A +\!\!+ C = B +\!\!+ [\star] +\!\!+ D$, then $B = A +\!\!+ B'$ for some B'.*

Proof By induction on A with a generalized claim for all B.

$A = [\,]$ We have $B = [\,] +\!\!+ B$.

$A = a :: A$ If $B = [\,]$, $a :: A +\!\!+ C = \star :: D$ is contradictory because $a : \Sigma$, so $a \neq \star$. If $B = b :: B$, we have $a = b$ and use the inductive hypothesis to obtain B' and prove $b :: B = a :: A +\!\!+ B'$. □

Lemma 4.6 *Let z and w be strings over Σ and $A \subseteq f\,(R, x, y)$. If*

$$C_1\,A = z +\!\!+ [\star] +\!\!+ w +\!\!+ C_2\,A$$

then either $z \Rightarrow_R^* y$ and $w = [\,]$, or $z \Rightarrow_R^* m$, $C_1\,B = z$, $C_2\,B = m$, and $A = B +\!\!+ \left[\frac{\star}{\star}\right] +\!\!+ A'$ hold for some A', B and string m.

Proof By size induction on A and a generalized claim for all z and w. Since $A = [\,]$ is contradictory, we continue with $A = \left[\frac{d_1}{d_2}\right] :: A$ and a case analysis on z.

$z = [\,]$ We have $d_1 +\!\!+ C_1A = \star :: w +\!\!+ d_2 +\!\!+ C_2A$ and proceed by case analysis on the type of the domino $d \in f\,(R, x, y)$.

- $\left[\frac{\$}{\$x\star}\right]$ Contradiction because $\$ \neq \star$.
- $\left[\frac{y \star \$}{\$}\right]$ Then $y = [\,]$, $w = [\,]$ and the left hand side holds.
- $\left[\frac{u}{v}\right]$ The upper string u must be empty, otherwise it conflicts with the $\star$ symbol. The claim follows from the inductive hypothesis with $z := [\,]$ and $w := w +\!\!+ v$. Either the left hand side holds trivially, or $[\,] \Rightarrow_R^* (v +\!\!+ m)$ can be proven with $[\,] \Rightarrow_R^* m$ and $\varepsilon/v \in R$.
- $\left[\frac{\star}{\star}\right]$ With $B := [\,]$ and $m := [\,]$ the right hand side, $[\,] \Rightarrow_R^* [\,]$, holds.
- $\left[\frac{a}{a}\right]$ Contradiction because $a \neq \star$.

$z = b :: z$ The assumption is $d_1 +\!\!+ C_1A = b :: z +\!\!+ \star :: w +\!\!+ d_2 +\!\!+ C_2A$. We do case analysis on $d \in f\,(R, x, y)$.

- $\left[\frac{\$}{\$x\star}\right]$ Contradiction because $\$ \neq b$.
- $\left[\frac{y \star \$}{\$}\right]$ This domino implies $y = b :: z$ and $w = [\,]$ and the left hand side stating $b :: z \Rightarrow_R^* b :: z$ holds.

$\left[\frac{u}{v}\right]$ Fact 4.5 yields $b :: z = u +\!\!+ z'$ for some z'. If the inductive hypothesis with $z{:=}z$ and $w{:=}w +\!\!+ v$ results in $z \Rightarrow_R^* y$ and $v = [\,]$, then $u +\!\!+ z \Rightarrow_R^* y$ follows. Otherwise $u +\!\!+ z \Rightarrow_R^* v +\!\!+ m$ follows from $z \Rightarrow_R^* m$.

$\left[\frac{\star}{\star}\right]$ Contradiction because $\star \neq b$.

$\left[\frac{a}{a}\right]$ We have $a = b$ and use the inductive hypothesis with $z := z$ and $w := w +\!\!+ [b]$. If $w +\!\!+ [b] = [\,]$ we prove this contradictory. If $z \Rightarrow_R^* m$ the right hand side, $b :: z \Rightarrow_R^* b :: m$ is obvious. □

Lemma 4.7 *Let z be a string over Σ and A a list of dominoes with $A \subseteq f\,(R, x, y)$. If $\left[\frac{-}{z\star}\right] :: A$ is a match, then $z \Rightarrow_R^* y$.*

Proof We start by size induction on the length of A. From Lemma 4.6 with $(w{:=}[\,])$ we get two cases. In the first, the assumption $z \Rightarrow_R^* y$ equals the claim. In the second, where $z \Rightarrow_R^* m$ and $A = B +\!\!+ \left[\frac{\star}{\star}\right] +\!\!+ A'$, we conclude $C_1\,A' = m +\!\!+ \star :: C_2\,A'$. By transitivity of $\Rightarrow^*$ it suffices to show $z \Rightarrow_R^* m$ and $m \Rightarrow_R^* y$. The latter follows from the inductive hypothesis. □

Lemma 4.8 *If the MPCP instance $f\,(R, x, y)$ is solvable, then $x \Rightarrow_R^* y$.*

Proof Follows from Lemma 4.7. □

Theorem 4.9 *The word problem in string-rewriting systems reduces to MPCP.*

$$\forall\, R\, x\, y.\, \mathsf{SR}\,(R, x, y) \leftrightarrow \mathsf{MPCP}\,(f\,(R, x, y))$$

Proof By Lemmas 4.4 and 4.8. □

What we have seen in this correctness proof is that it is straightforward to simulate a single rewrite with dominoes from the constructed MPCP instance. The main difficulty is the other direction where we are given the match and need to work out its structure. Lemma 4.6 assumes that at least the separator symbol $\star$ precedes the bottom row of a match, which yields a sufficient inductive hypothesis to establish the rewrite connection between sequential strings.

One might replace the \$ symbol by the separator $\star$ when arguing informally about the correctness of the reduction, as it is done in [9]. However, in the formal correctness proof it is beneficial that the initial and the final domino include a symbol that is not used in between the strings. This simplifies to prove the usage of a final domino in the midst of a match contradictory.

Chapter 5

Reducing Reachability of Turing Configurations to String Rewriting

The reduction of the halting problem to string-rewriting systems is based on the translation of the transition function for Turing machines to rewrite rules. This construction can be stated as independent reduction of Reach to SR. The reduction of Halt to SR is discussed in Chapter 6. We adopt the general reduction idea from Davis et al. [3] which reduce the halting problem for nondeterministic Turing machines. Some definitions are customized to fit our notion of deterministic machines which can perform a move of the tape head and write a symbol in one step.

Recall the definition of the class $\mathsf{Reach}\,(M, c_1, c_2) := c_1 \vdash_M c_2$. A reduction to SR converts a machine M and configurations c_1 and c_2 into an SR instance with a set of rules R and strings x and y such that

$$\mathsf{Reach}\,(M, c_1, c_2) \leftrightarrow \mathsf{SR}\,(R, x, y)$$

The main concept behind the reduction is to consider configurations as strings and translate the transition function into rewrite rules. A rewrite rule substitutes those parts of a configuration which must be changed to obtain the successor configuration. One rewrite of a configuration simulates exactly one $\hat{\delta}$-step. The question whether c_2 is reachable from c_1 is converted to the question whether the representation of c_1 rewrites to c_2.

For the remaining of this chapter we fix a Turing machine $M := (Q, \delta, q_0, H)$ over the finite alphabet Σ. To transform configurations into strings, we use an encoding function $\langle\cdot\rangle$ stated in Table 5.1. It assumes a symbol $a : \Sigma$ and strings A and B over Σ. We use the delimiter symbols $(\!|$ and $|\!)$ to mark the left and the right end of the tape content. The state q is inserted to the left of the symbol under the machine head. If there is no symbol to read, the state occurs to the left of the respective tape delimiter. This encoding leads to the underlying type Γ of the string-rewriting

tape	$\emptyset$	leftof	midtape	rightof
c	$(q, \emptyset)$	$(q, \underset{\uparrow}{a}A)$	$(q, B\underset{\uparrow}{a}A)$	$(q, B\underset{\uparrow}{a})$
$\langle c \rangle$	$q(\!(\!)$	$q(\!(aA)\!)$	$(\!(BqaA)\!)$	$(\!(Baq)\!)$

Table 5.1: Encoding function for configurations.

system R. It is comprised of the alphabet Σ, states of type Q and the separators $(\!(\,|\,)\!)$.

$$\Gamma := q : Q \mid a : \Sigma \mid (\!(\mid)\!)$$
$$\langle\cdot\rangle : \mathsf{conf} \to \Gamma^*$$

Lemma 5.1 *$\langle\cdot\rangle$ is injective.*

With the type of the string-rewriting system fixed, we can characterize the reduction of Reach to SR formally:

$$f : (\mathsf{TM} \times \mathsf{con} \times \mathsf{con}) \to (\mathbf{L}\,(\Gamma^* \times \Gamma^*) \times \Gamma^* \times \Gamma^*)$$

$$\forall\, M\ c_1\ c_2.\, \mathsf{Reach}\,(M, c_1, c_2) \leftrightarrow \mathsf{SR}\,(f\,(M, c_1, c_2))$$

We already know how the second and third component of $f\,(M, c_1, c_2)$ are defined. The initial string $x := \langle c_1 \rangle$ and the target string $y := \langle c_2 \rangle$. The next step is to specify rewrite rules simulating the application of $\hat{\delta}$ on configurations.

5.1 Definition of Rewrite Rules Simulating a Transition

Example 5.2 *Consider the configuration sequence of the Turing machine defined in Table 1.1 which accepts the string* aba*.*

$$\underset{\uparrow}{a}ba \;\vdash\; a\underset{\uparrow}{b}a \;\vdash\; ab\underset{\uparrow}{a} \;\vdash\; aba\underset{\uparrow}{\ } \;\vdash\; ab\underset{\uparrow}{a}$$
$$q_0 \qquad q_1 \qquad q_1 \qquad q_0 \qquad q_f$$

The upcoming definitions will provide rules such that the following rewrite sequence holds.

$$(\!(q_0aba)\!) \Rightarrow (\!(aq_1ba)\!) \Rightarrow (\!(abq_1a)\!) \Rightarrow (\!(abaq_0)\!) \Rightarrow (\!(abq_fa)\!)$$

Recall the type of the transition function $\delta : Q \times \Sigma_\perp \to Q \times \Sigma_\perp \times \{L, N, R\}$. Because δ is not applied to halting states, we partition the set of states Q into $Q_H := \{q : Q \mid H\,q = \mathrm{true}\}$ and $Q_{\overline{H}} := Q \setminus Q_H$. For all states q_1 in $Q_{\overline{H}}$ we determine the result of $\delta\,(q_1, \perp)$ and $\delta\,(q_1, \lfloor a \rfloor), \forall\, a \in \Sigma$, and provide rewrite rules which interpret δ the same way

$\delta(q_1, \bot) = (q_2, \text{write}, \text{move})$

u	v	u	v	write	move
$q_1 (\!\vert$	$q_2 (\!\vert$	$c\ q_1)\!\rangle$	$q_2\ c)\!\rangle$	$\bot$	L
$q_1 (\!\vert$	$q_2 (\!\vert$	$q_1)\!\rangle$	$q_2)\!\rangle$	$\bot$	N
$q_1 (\!\vert\!)$	$q_2 (\!\vert\!)$	$q_1)\!\rangle$	$q_2)\!\rangle$	$\bot$	R
$q_1 (\!\vert\ c$	$(\!\vert\ q_1\ c$			$\bot$	R
$q_1 (\!\vert$	$q_2 (\!\vert b$	$c\ q_1)\!\rangle$	$q_2\ c\ b)\!\rangle$	$\lfloor b \rfloor$	L
$q_1 (\!\vert$	$(\!\vert q_2\ b$	$q_1)\!\rangle$	$q_2\ b)\!\rangle$	$\lfloor b \rfloor$	N
$q_1 (\!\vert$	$(\!\vert b\ q_2$	$q_1)\!\rangle$	$b\ q_2)\!\rangle$	$\lfloor b \rfloor$	R

$\delta(q_1, \lfloor a \rfloor) = (q_2, \text{write}, \text{move})$

u	v	u	v	write	move
$(\!\vert q_1\ a$	$q_2 (\!\vert a$	$c\ q_1\ a$	$q_2\ c\ a$	$\bot$	L
		$q_1\ a$	$q_2\ a$	$\bot$	N
		$q_1\ a$	$a\ q_2$	$\bot$	R
$(\!\vert q_1\ a$	$q_2 (\!\vert b$	$c\ q_1\ a$	$q_2\ c\ b$	$\lfloor b \rfloor$	L
		$q_1\ a$	$q_2\ b$	$\lfloor b \rfloor$	N
		$q_1\ a$	$b\ q_2$	$\lfloor b \rfloor$	R

Table 5.2: The upper rewrite rules (u/v) simulate a transition where the machine does not read a symbol and the head is at the left or the right end of the tape. The rules below represent transitions where the head points to a symbol. The rules are defined for all symbols $c : \Sigma$.

as $\hat{\delta}$ does on configurations. We will refer to these rewrite rules defined in Table 5.2 as **transition rules**.

Note that there are possibly several rules for for one δ-transition. The result of $\delta(q_1, \bot) = (q_2, \bot, R)$ for instance, leads to three transition rules which cover the different tapes:

$(q_1 \langle\!| \,/\, q_2 \langle\!|)$ If the tape is empty, we only change the state itself and do not move right between the tape delimiters.

$(q_1 \langle\!| \, c \,/\, \langle\!| \, q_2 \, c)$ This rule can be used whenever the tape is not empty, since it is provided for every $c : \Sigma$.

$(q_1 |\!\rangle \,/\, q_2 |\!\rangle)$ If the state occurs at the right end of the tape content, the move has no effect because empty cells are not included in the tape.

Definition 5.3 (Transition Rules) *We combine all rules which translate the transition function δ in a set Δ. Let the process of choosing the corresponding transition rules from Table 5.2 be abbreviated by* rules*.*

$$\Delta := \bigcup_{q \in Q \setminus Q_H} \text{rules}(\delta(q, \bot)) \quad \cup \quad \bigcup_{\substack{q \in Q \setminus Q_H \\ a : \Sigma}} \text{rules}(\delta(q, \lfloor a \rfloor))$$

Example 5.4 *The set of rules*

$$\{(q_0 a / b q_1),\ (q_1 b / b q_1),\ (q_1 a / b q_0),\ (a q_0 |\!\rangle / q_f a |\!\rangle)\}$$

which represents some of the transition rules for the Turing machine defined in 1.1 suffices to establish the rewrite sequence

$$\langle\!| q_0 a b a |\!\rangle \Rightarrow \langle\!| a q_1 b a |\!\rangle \Rightarrow \langle\!| a b q_1 a |\!\rangle \Rightarrow \langle\!| a b a q_0 |\!\rangle \Rightarrow \langle\!| a b q_f a |\!\rangle$$

Definition 5.5 (Reduction) *The reduction f is defined as* $f(M, c_1, c_2) := (\Delta_M, \langle c_1 \rangle, \langle c_2 \rangle)$.

5.2 Correctness Proof

To prove our transformation of the transition function into rewrite rules correct, we need two lemmas, one for each proof direction. We prove that one rewrite $\Rightarrow_\Delta$ simulates one $\hat{\delta}$ application and vice versa.

Lemma 5.6 *If c is not a final configuration, then $\langle c \rangle \Rightarrow_\Delta \langle\, \hat{\delta}\, c \,\rangle$*

Proof Let $c = (q, t)$. First we do case analysis on tape t and then a case analysis on the outcome of $\delta(q, \bot)$ or $\delta(q, \lfloor a \rfloor)$ depending on the tape. The combinations of the new symbols $\Sigma_\bot$ and a move from $\{L, N, R\}$ leaves us with six cases for each tape. In each case there is exactly one applicable rule in Δ which is used to convert $\langle c \rangle$ into the successor configuration $\langle\, \hat{\delta}\, c \,\rangle$. □

Example 5.7 *Consider a configuration* $(q_1, \underset{\uparrow}{a}B)$ *with* q_1 *being a non-halting state. Let the result of the transition* $\delta(q_1, \lfloor a \rfloor)$ *be* $(q_2, \lfloor b \rfloor, L)$*. We are looking for a rewrite rule in* Δ *to transform the string representation* $(\!| q_1 a B |\!)$ *into the successor configuration* $q_2(\!| bB |\!)$*. For this transition we have rules* $((\!| q_1 a / q_2 (\!| b)$ *and* $(c q_1 a / q_2 c b)$, $\forall c : \Sigma$ *in* Δ *(see Table 5.2). Since we are at the left end of the tape, only the former rule is applicable.*

The next lemma states that applying a transition rule to a configuration $\langle c \rangle$ invariably leads to the successor configuration. In the proof we match the left hand side of a transition rule u/v to the encoding of a configuration $\langle c \rangle$. Since both, u and $\langle c \rangle$, contain one state, we conclude that they must be equal. Hence u/v and $\hat{\delta}\, c$ rely on the same δ-transition and yield equal successor configurations.

Lemma 5.8 *If* $\langle c \rangle \Rightarrow_\Delta z$ *then* $z = \langle \hat{\delta}\, c \rangle$ *and* c *is not a final configuration.*

Proof Let $c = (q, t)$. By the definition of $\Rightarrow$ we have $\langle c \rangle = x\,u\,y$ and $z = x\,v\,y$ for some strings x and y over Γ, and $u/v \in \Delta$. We continue by case analysis on t.

$\underset{\uparrow}{\emptyset}$ We have $\langle (q, \underset{\uparrow}{\emptyset}) \rangle = q(\!|\!|) = x\,u\,y$. We do a case analysis on $u/v \in \Delta$.

- $u/v \in \mathsf{rules}(\delta(q', \lfloor c \rfloor))$: Contradiction because $q(\!|\!| \neq x\,q'\,c\,y$ for all states q' and symbols c.
- $u/v \in \mathsf{rules}(\delta(q', \bot))$: By case analysis on the outcome of this transition we obtain the concrete transition rules. In order to shorten the proof we assume $\delta(q', \bot) = (\hat{q}, \lfloor a \rfloor, R)$ and $(u/v) = (q'(\!| / (\!| a \hat{q})$. From $q(\!|\!| = x\,q'(\!|\,y$ we conclude $q = q'$, $x = [\,]$ and $y = |\!)$. Now, $\delta(q, \bot) = (\hat{q}, \lfloor a \rfloor, R)$, and $\langle \hat{\delta}\, c \rangle$ is equal to $(\!| a \hat{q} |\!)$. The claim $\neg H_c$ follows from the fact that all rules in Δ are only defined for non halting states.

$\underset{\uparrow}{a}A$ We have $q(\!| aA |\!) = x\,u\,y$ and use the same technique as in the niltape case.

$B^R a \underset{\uparrow}{\ }$ We have $(\!| B^R a q |\!) = x\,u\,y$ and use the same technique as in the niltape case.

$B^R \underset{\uparrow}{a} A$ With $(\!| B^R q a A |\!) = x\,u\,y$, we do a case analysis on $u/v \in \Delta$.

- $u/v \in \mathsf{rules}(\delta(q', \bot))$: Contradiction because the state q is not to the left of a tape delimiter.
- $u/v \in \mathsf{rules}(\delta(q', \lfloor c \rfloor))$ for some q' and c. Let $\delta(q', \lfloor c \rfloor) = (\hat{q}, \lfloor b \rfloor, R)$ and $(u/v) = (q'c / b\hat{q})$. From $x\,q'c\,y = (\!| B^R q a A |\!)$ we conclude $q' = q$ and $c = a$, since B and A are strings over Σ. By $\delta(q, \lfloor a \rfloor) = (\hat{q}, \lfloor b \rfloor, R)$, the claim $\langle \hat{\delta} c \rangle = (\!| B^R b \hat{q} A |\!)$ holds. □

Theorem 5.9 *Reachability reduces to string rewriting.*

$$\forall\, M\, c_1\, c_2.\, \mathsf{Reach}\,(M, c_1, c_2) \leftrightarrow \mathsf{SR}\,(f\,(M, c_1, c_2))$$

Proof We use the reduction f and prove $c_1 \vdash_M c_2 \leftrightarrow \langle c_1\rangle \Rightarrow^*_{\Delta_M} \langle c_2\rangle$.

$\rightarrow$ Induction on $\vdash$.

- $c_1 = c_2$ — The claim $\langle c_1\rangle \Rightarrow^*_\Delta \langle c_1\rangle$ holds by definition.
- $\hat{\delta}\, c_1 \vdash c_2\,, \neg H_{c_1}$ — We prove the claim $\langle c_1\rangle \Rightarrow^*_\Delta \langle c_2\rangle$ with $\langle c_1\rangle \Rightarrow_\Delta \langle \hat{\delta}\, c_1\rangle$ by Lemma 5.6 and $\langle \hat{\delta}\, c_1\rangle \Rightarrow^*_\Delta \langle c_2\rangle$ using the inductive hypothesis.

$\leftarrow$ Induction on $y \Rightarrow^*_\Delta \langle c_2\rangle$ with the generalized claim $\forall\, c_1\,,\, y = \langle c_1\rangle \rightarrow c_1 \vdash c_2$.

- $\langle c_1\rangle = \langle c_2\rangle$ — With injectivity of $\langle\cdot\rangle$ (Lemma 5.1) the claim $c_1 \vdash c_1$ holds.
- $\langle c_1\rangle \Rightarrow_\Delta z\,,\ z \Rightarrow^*_\Delta \langle c_2\rangle$ — By Lemma 5.8 we have $z = \langle \hat{\delta}\, c_1\rangle$ and $\neg H_{c_1}$. The claim $c_1 \vdash c_2$ follows using the inductive hypothesis with $c_1 := \hat{\delta}\, c_1$. □

The shortness of the correctness proof for the reduction of reachability to string rewriting originates from the one to one correspondence of $\Rightarrow_\Delta$ and $\vdash$, as one rewrite rule simulates one $\hat{\delta}$ application. Lemma 5.6 demonstrates the fact that one rewrite rule carries enough information to transform one configuration in its successor configuration. In theory this proof is quite short, since one can easily check whether there are suitable rewrite rules for all different tapes and transitions in Table 5.2. When doing this proof in Coq, we have to explicitly state the rewrite rule used for each of the 24 cases (there are three possible moves $\{L, N, R\}$, and two writing options $\bot$ and $\lfloor\cdot\rfloor$ for each tape).

For the proof of Lemma 5.8 it is essential that the two tape delimiters are different symbols. Consider the string $q_1(\!|)\!|$ and the transition $\delta\,(q_1, \bot) = (q_2, \lfloor a\rfloor, R)$. If we have $(\!|\neq|\!)$, the only applicable rewrite rule is $(q_1(\!| \,/\, (\!|aq_2)$ and $(\!|aq_2|\!) = \langle (q_2, \underset{\uparrow}{a})\rangle$. If we had defined only one tape delimiter $\|$, also the rule $(q_1\| \,/\, aq_2\|)$ for the right end of the tape would have been suitable. But definitely, $aq_2\|\| \neq \langle (q_2, \underset{\uparrow}{a})\rangle = \|aq_2\|$. Note that in [3], there is only one tape delimiter but an extra blank symbol to allocate new cells.

With the result of this chapter, one could consider Turing machines as special string-rewriting systems. All strings contain one state and two tape delimiters, and the state might be to the left of the $(\!|$ delimiter. When defining rewrite rules for all non final states in either tape configuration one could also represent non deterministic Turing machines. The halting problem then corresponds to terminating rewrite sequences starting with a string containing the initial state.

Chapter 6

Reducing the Halting Problem to String Rewriting

In this chapter we present the reduction of the halting problem to string rewriting. Why this is different from the reduction in Chapter 5 becomes clear if we consider the equivalence

$$\mathsf{Halt}\,(M, t) \leftrightarrow \mathsf{SR}\,(R, x, y)$$

A reduction of Halt to SR transforms a Turing machine M and an input tape t into a set of rewrite rules R and strings x and y. The class Halt is defined as reachability of a final configuration, starting in (q_0, t). Therefore it seems clear to start rewriting with $x := \langle (q_0, t) \rangle$, but it is less obvious what the target string y might be. Intuitively we would think of a final configuration, though this is not realizable. The problem is that there exist infinitely many configurations containing a halting state, and we do not know which one will be reached. For this reason, we fix y as the empty string ε. The set of rewrite rules contains the transition rules Δ defined in 5.3 to rewrite to a final configuration. Additionally, it contains rules to delete symbols to the left and the right of a final state such that finally a single final state is replaced with ε.

6.1 Definition of Rewrite Rules Deleting Symbols

The transition rules in Δ guarantee that whenever a configuration c_2 is reachable from c_1, $\langle c_1 \rangle \Rightarrow_\Delta^* \langle c_2 \rangle$ holds. Rewrite rules which delete symbols should only be applicable to final configurations. Consequently all of them contain a final state in their left component. The following example illustrates the procedure of deleting symbols.

Example 6.1 *The string* $(\!|abq_fa|\!)$ *represents the final configuration of the accepting run stated in Example 5.2. If we provide the rules* (cq_f/q_f) *and* (q_fc/q_f) *for all* $c \in \{a, b, (\!|, |\!)\}$ *we can rewrite* $(\!|abq_fa|\!)$ *to* q_f*. The rule* (q_f/ε) *then removes the final state:*

$$(\!|abq_fa|\!) \Rightarrow (\!|abq_f|\!) \Rightarrow (\!|abq_f \Rightarrow (\!|aq_f \Rightarrow (\!|q_f \Rightarrow q_f \Rightarrow \varepsilon$$

The rules deleting symbols do not differentiate between tape symbols and tape delimiters $\langle$ and $\rangle$. This allows to abstract the difference between these symbols, leading to shorter proofs. For the following definition and lemmas, we assume a finite type Φ and a set of **final symbols** $\mathbf{F} \subseteq \Phi$. The overall goal is to define a set of **deletion rules** such that for any string x over Φ it holds that

$$\forall f \in F \,.\, f \in x \to x \Rightarrow_D^* \varepsilon$$

Definition 6.2 (Deletion rules) *The set* D *of deletion rules contains three types of rules for all final symbols.*

$$D := \{(af, f), (fa, f), (f, \varepsilon) \mid a : \Phi,\ f \in F\}$$

A deletion rule can only be applied to a string containing a symbol $f \in F$, since all left hand sides contain such a final symbol. Furthermore, a deletion rule either keeps the final symbol f or erases it and therefore is the last applicable rule.

Note the fact that rewriting using rules from D may be indeterministic.

Example 6.3 *Consider the string* $a\mathbf{f}b\,\mathbf{f}'c$ *with* $\mathbf{f}, \mathbf{f}' \in F$ *and* $a, b, c \in \Phi$*. The deletion rules facilitate several rewrite sequences which lead to the empty word.*

$$a\,\mathbf{f}b\,\mathbf{f}'c \Rightarrow_D \mathbf{f}b\,\mathbf{f}'c \Rightarrow_D \mathbf{f}\,\mathbf{f}'c \Rightarrow_D \mathbf{f}c \Rightarrow_D \mathbf{f} \Rightarrow_D \varepsilon$$
$$a\,\mathbf{f}b\,\mathbf{f}'c \Rightarrow_D a\,\mathbf{f}b\,\mathbf{f}' \Rightarrow_D \mathbf{f}b\,\mathbf{f}' \Rightarrow_D \mathbf{f}\,\mathbf{f}' \Rightarrow_D \mathbf{f}' \Rightarrow_D \varepsilon$$

In the following we assume x, y and z being strings over Φ and $f \in F$.

Fact 6.4

a) $xfy \Rightarrow_D^* xf$

b) $xfy \Rightarrow_D^* fy$

c) $f \Rightarrow_D \varepsilon$

Lemma 6.5 *If* z *contains a final symbol* f*, then* $z \Rightarrow_D^* \varepsilon$*.*

Proof Let $z = xfy$ for some strings x and y. By transitivity of $\Rightarrow^*$ and Fact 6.4 (a) the claim is $xf \Rightarrow_D^* \varepsilon$. From (b) with $y := [\,]$ we obtain $xf \Rightarrow_D^* f$ and $f \Rightarrow_D^* \varepsilon$ follows from (c). □

Lemma 6.6 *If* $x \Rightarrow_D y$*, then* x *contains a final symbol.*

Proof Let $u/v \in D$ and $x = x_1 u x_2$. As all rules in D have a final symbol f in their first component, it follows that $f \in u$ and therefore $f \in x$. □

To state and verify the reduction of Turing machines to string-rewriting systems, we assume a Turing machine $M = (Q, \delta, q_0, H)$ over the finite alphabet Σ. The final states Q_H of M serve as the set of final symbols F for the definition of the deletion rules D. Remember the type Γ for the string-rewriting system:

$$\Gamma := q : Q \mid a : \Sigma \mid (\!|\,|\!)$$

Definition 6.7 (Reduction) *The reduction f yields a string-rewriting system combining the transition rules from Table 5.2 and deletion rules for all final states. The initial string is the encoding of the initial configuration and the target string is ε.*

$$\begin{aligned} f &: (\mathsf{TM} \times \mathsf{tape}) \to (\mathbf{L}\,(\Gamma^* \times \Gamma^*) \times \Gamma^* \times \Gamma^*) \\ f\,(M, t) &:= (\Delta_M \cup D_{Q_H}, \langle (q_0, t) \rangle, \varepsilon) \end{aligned}$$

6.2 Correctness Proof

Lemmas 6.5 and 6.6 result in the following properties for final configurations.

Lemma 6.8 *If c is a final configuration, then $\langle c \rangle \Rightarrow_D^* \varepsilon$.*

Lemma 6.9 *If $\langle c \rangle \Rightarrow_D x$ for some x, then c is a final configuration.*

Now we rewrite an arbitrary configuration $\langle c \rangle$ to ε with rules in $\Delta \cup D$, and prove a final configuration reachable from $\langle c \rangle$ using the results from Chapter 5.

Lemma 6.10 *If $\langle c \rangle \Rightarrow_{\Delta \cup D}^* \varepsilon$, then $c \vdash c_f$ for a final configuration c_f.*

Proof We generalize the claim to $\forall\, c, y = \langle c \rangle \to \exists\, c_f,\, c \vdash c_f \wedge H_{c_f}$ and continue by induction on $y \Rightarrow_{\Delta \cup D}^* \varepsilon$.

$\langle c \rangle = \varepsilon$ Contradiction because every string representation of any configuration contains at least a state and the two tape delimiters.

$\langle c \rangle \Rightarrow_{\Delta \cup D} z$ $z \Rightarrow_{\Delta \cup D}^* \varepsilon$ We have $\langle c \rangle = xux'$ and $z = xvx'$ and proceed by case analysis on the membership of u/v:

- $u/v \in \Delta$ By Lemma 5.8, $z = \langle \hat{\delta}\, c \rangle$ and $\neg H_c$. The inductive hypothesis yields a final configuration c_f and $\hat{\delta}\, c \vdash c_f$. The claim $c \vdash c_f$ follows from the definition of $\vdash$.
- $u/v \in D$ We prove $c \vdash c$ and H_c by Lemma 6.9. □

Theorem 6.11 *The halting problem reduces to string rewriting.*

$$\forall\, M\, t.\, \mathsf{Halt}\,(M, t) \leftrightarrow \mathsf{SR}\,(f\,(M, t))$$

Proof Let M be a Turing machine with initial state q_0. We show the equivalence $(\exists\, c_f,\, (q_0, t) \vdash c_f \wedge H_{c_f}) \leftrightarrow \langle (q_0, t) \rangle \Rightarrow^*_{\Delta \cup D} \varepsilon$. Let $\langle (q_0, t) \rangle$ be abbreviated by c_0.

$\rightarrow$ By Theorem 5.9 we have $c_0 \Rightarrow^*_\Delta c_f$. The claim $c_0 \Rightarrow^*_{\Delta \cup D} \varepsilon$ follows from transitivity and $c_f \Rightarrow^*_D \varepsilon$ using Lemma 6.8.

$\leftarrow$ Lemma 6.10 provides a final configuration c_f such that $c_0 \vdash c_f$. □

The main effort of this reduction lies in the simulation of the Turing computation, the reduction of Reach to SR. Nonetheless, it is crucial to define a fixed target string which is derivable from all final configurations. Of course it does not need to be the empty string ε. Davis et al. [3] define two new states, one for deleting the symbols to the right and the other to delete all symbols to the left. Their deletion process ends with a string containing one deletion state in between two tape delimiters. This makes rewriting with deletion rules deterministic but does not seem to shorten the verification of the reduction.

Chapter 7

Reducing the Halting Problem to MPCP

Hopcroft et al. [10] prove the undecidability of MPCP with a direct reduction of the halting problem. Remember that we already defined reductions from Halt to SR and from SR to MPCP. In this chapter we verify the direct approach, which combines the concepts of the reductions involving string rewriting: An MPCP match simulates a halting computation of a Turing machine. One $\hat{\delta}$ application on configurations is simulated by one transition domino, containing the current state and neighboring symbols at the top and the successor state with surrounding symbols at the bottom. In contrast to string rewriting, we have to construct the whole configuration round the transition domino and append symbols which have not changed using copy dominoes. In partial matches the bottom row is always one configuration ahead the upper row until a final configuration is reached. Then we use deletion dominoes to remove one symbol after another to both sides of a final state until the match can be completed with a last domino containing a single final state.

Following the proof of Hopcroft et al. [10], we need to adjust the transition dominoes to our model of Turing machines. Their machines are different in terms of the movements of the tape head. It moves either to the left or the right, but never moves left from its initial head position.

We illustrate the process of constructing an MPCP match using a Turing machine $M := (\{q_s, q_0, q_1, q_f\}, \delta, q_s, \{q_f\})$ over the alphabet $\Sigma := \{a, b\}$. It accepts all strings containing an even number of a's and replaces them with b. The transition function δ is defined in Table 7.1. Note that the initial position of the tape head should be to the left of the tape content.

Example 7.1 (Accepting) *M halts on the input tape $\underset{\uparrow}{}aba$ with the following configuration sequence:*

$$\begin{array}{ccccccccccc} & aba & \vdash & aba & \vdash & bba & \vdash & bba & \vdash & bbb & \vdash & bbb \\ \uparrow & & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\ q_s & & & q_0 & & q_1 & & q_1 & & q_0 & & q_f \end{array}$$

q_i	$\delta(q_i, \lfloor a \rfloor)$	$\delta(q_i, \lfloor b \rfloor)$	$\delta(q_i, \bot)$
q_s	$(q_s, \bot, N)$	$(q_s, \bot, N)$	$(q_0, \bot, R)$
q_0	(q_1, b, R)	$(q_0, \bot, R)$	$(q_f, \bot, L)$
q_1	(q_0, b, R)	$(q_1, \bot, R)$	$(q_1, \bot, L)$

Table 7.1: Transition function δ for TM M.

Constructing an MPCP match requires an encoding similar to $\langle\cdot\rangle$ from Table 5.1. This time, we use the delimiters $(\!|$ and $|\!)$ exclusively to separate two configurations and introduce a blank symbol $\sqcup$. This symbol occurs in configurations where the tape is empty or the head is at the left end of the tape.

tape	$\emptyset$	leftof	midtape	rightof
c	$(q, \emptyset)$	$(q, \underset{\uparrow}{a}A)$	$(q, B\underset{\uparrow}{a}A)$	$(q, Ba\underset{\uparrow}{\ })$
$\langle c\rangle$	$(\!\|q\sqcup|\!)$	$(\!\|q \sqcup aA|\!)$	$(\!\|BqaA|\!)$	$(\!\|Baq|\!)$

Table 7.2: Encoding of configurations using a blank symbol.

We illustrate how to construct a match for the input tape $\underset{\uparrow}{a}ba$ which is accepted by M reaching the final state q_f. The match starts with a domino containing the initial configuration at the bottom. The second domino represents the first transition $\delta(q_s, \bot) = (q_0, \bot, R)$ and the next three copy dominoes complete the first configuration at the top and the second at the bottom.

$$\boxed{\frac{\$}{\$(\!|q_s \sqcup aba|\!)}}\ \boxed{\frac{(\!|q_s \sqcup a}{(\!|q_0 a}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{|\!)}{|\!)}}$$

Dominoes simulating a transition are defined for all non halting states $q \in \{q_s, q_0, q_1\}$ such that the partial match can be expanded until the final state q_f is reached.

$$\boxed{\frac{\$}{\$(\!|q_s \sqcup aba|\!)}}\ \boxed{\frac{(\!|q_s \sqcup a}{(\!|q_0 a}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{q_0 a}{bq_1}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{q_1 b}{bq_1}}\ \boxed{\frac{a}{a}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{q_1 a}{bq_0}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{bq_0|\!)}{q_f b|\!)}}$$

A domino sequence ending with a final configuration is a partial match with the top row being a substring of the bottom row. Since it represents a halting computation, we need to complement it to a match. Similar to the reduction of Halt to SR, deletion dominoes are used to remove symbols next to the final state until a domino $\boxed{\frac{(\!|q_f|\!)\$}{\$}}$ makes the top and bottom row equal.

$$\ldots\boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{q_f b}{q_f}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{b}{b}}\ \boxed{\frac{bq_f}{q_f}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|}{(\!|}}\ \boxed{\frac{bq_f}{q_f}}\ \boxed{\frac{|\!)}{|\!)}}\ \boxed{\frac{(\!|q_f|\!)\$}{\$}}$$

Put together, these dominoes concatenate to a match for the halting computation of M on the input tape $\underset{\uparrow}{\mathtt{aba}}$:

$$\underset{q_s}{\overset{\uparrow}{\mathtt{aba}}} \vdash \underset{q_0}{\overset{\uparrow}{\mathtt{aba}}} \vdash \underset{q_1}{\overset{\uparrow}{\mathtt{bba}}} \vdash \underset{q_1}{\overset{\uparrow}{\mathtt{bba}}} \vdash \underset{q_0}{\overset{\uparrow}{\mathtt{bbb}}} \vdash \underset{q_f}{\overset{\uparrow}{\mathtt{bbb}}}$$

$$\frac{\$(\!|q_s \sqcup aba|\!)\,(\!|q_0 aba|\!)\,(\!|bq_1 ba|\!)\,(\!|bbq_1 a|\!)\,(\!|bbbq_0|\!)\,(\!|bbq_f b|\!)\,(\!|bbq_f|\!)\,(\!|bq_f|\!)\,(\!|q_f|\!)\$}{\$(\!|q_s \sqcup aba|\!)\,(\!|q_0 aba|\!)\,(\!|bq_1 ba|\!)\,(\!|bbq_1 a|\!)\,(\!|bbbq_0|\!)\,(\!|bbq_f b|\!)\,(\!|bbq_f|\!)\,(\!|bq_f|\!)\,(\!|q_f|\!)\$}$$

7.1 Definition of MPCP Dominoes

The definition of the dominoes and the verification of the reduction take place in the context of a Turing machine $M = (Q, \delta, q_0, H)$ over a finite alphabet Σ and a given initial tape t_0. The constructed dominoes will be of type

$$\Gamma := q : Q \mid s : \Sigma \mid \sqcup \mid |\!) \mid (\!| \mid \$$$

and the type of the reduction is $f : (\mathsf{TM} \times \mathsf{tape}) \to \mathsf{mpcp}_\Gamma$.

Definition 7.2 (Initial domino) *The first domino of the MPCP match contains the initial configuration $\langle (q_0, t_0) \rangle$ for the tape t_0.*

$$D_{\mathsf{init}} := \boxed{\frac{\$}{\$\,\langle (q_0, t_0) \rangle}}$$

Definition 7.3 (Copy dominoes) *For Σ symbols, and the tape delimiters $(\!|$ and $|\!)$ we provide copy dominoes.*

$$D_{\mathsf{copy}} := \left\{ \boxed{\frac{(\!|}{(\!|}}, \boxed{\frac{|\!)}{|\!)}}, \boxed{\frac{a}{a}} \;\middle|\; a : \Sigma \right\}$$

To transform the transition function δ into dominoes, we determine the result of $\delta(q_1, \bot)$ and $\delta(q_1, \lfloor a \rfloor)$ for all non-halting states q_1 and symbols $a : \Sigma$. Table 7.3 contains these dominoes, which are similar (up to the blank symbol) to the rewrite rules defined in Table 5.2.

Definition 7.4 (Transition dominoes) *We abbreviate the process of choosing the dominoes belonging to the result of a transition $\delta(q_1, \cdot)$ by $\mathsf{trans}(\delta(q_1, \cdot))$. The set D_{trans} of transition dominoes is defined as*

$$D_{\mathsf{trans}} := \bigcup_{\substack{q_1 \in Q \setminus Q_H \\ a : \Sigma}} (\mathsf{trans}(\delta(q_1, \bot)) \cup \mathsf{trans}(\delta(q_1, \lfloor a \rfloor)))$$

$\delta(q_1, \bot)$	$(q_2, \bot, L)$	$(q_2, \bot, N)$	$(q_2, \bot, R)$
	$\boxed{\frac{(q_1 \sqcup}{(q_2 \sqcup}}$ $\boxed{\frac{c q_1)}{q_2 c)}}$	$\boxed{\frac{(q_1 \sqcup}{(q_2 \sqcup}}$ $\boxed{\frac{q_1)}{q_2)}}$	$\boxed{\frac{(q_1 \sqcup)}{(q_2 \sqcup)}}$ $\boxed{\frac{q_1)}{q_2)}}$ $\boxed{\frac{(q_1 \sqcup c}{(q_1 c}}$
$\delta(q_1, \bot)$	$(q_2, \lfloor b \rfloor, L)$	$(q_2, \lfloor b \rfloor, N)$	$(q_2, \lfloor b \rfloor, R)$
	$\boxed{\frac{(q_1 \sqcup}{(q_2 \sqcup b}}$ $\boxed{\frac{c q_1)}{q_2 c b)}}$	$\boxed{\frac{(q_1 \sqcup}{(q_2 b}}$ $\boxed{\frac{q_1)}{q_2 b)}}$	$\boxed{\frac{(q_1 \sqcup}{(b q_2}}$ $\boxed{\frac{q_1)}{b q_2)}}$
$\delta(q_1, \lfloor a \rfloor)$	$(q_2, \bot, L)$	$(q_2, \bot, N)$	$(q_2, \bot, R)$
	$\boxed{\frac{(q_1 a}{(q_2 \sqcup a}}$ $\boxed{\frac{c q_1 a}{q_2 c a}}$	$\boxed{\frac{q_1 a}{q_2 a}}$	$\boxed{\frac{q_1 a}{a q_2}}$
$\delta(q_1, \lfloor a \rfloor)$	$(q_2, \lfloor b \rfloor, L)$	$(q_2, \lfloor b \rfloor, N)$	$(q_2, \lfloor b \rfloor, R)$
	$\boxed{\frac{(q_1 a}{(q_2 \sqcup b}}$ $\boxed{\frac{c q_1 a}{q_2 c b}}$	$\boxed{\frac{q_1 a}{q_2 b}}$	$\boxed{\frac{q_1 a}{b q_2}}$

Table 7.3: The dominoes in the two upper rows simulate a transition where the machine does not read a symbol and the head is at the left or the right end of the tape. The dominoes below represent transitions where the head points to a symbol a. The rules are defined for all symbols $c : \Sigma$.

Definition 7.5 (Deletion dominoes) *The set of deletion dominoes is defined for all halting states $q_f \in Q_H$. One domino removes a Σ symbol or $\sqcup$ to the left or the right of a final state.*

$$D_{\mathsf{del}} := \left\{ \boxed{\frac{q_f a}{q_f}}, \boxed{\frac{a q_f}{q_f}}, \boxed{\frac{q_f \sqcup}{q_f}} \;\middle|\; q_f \in Q_H,\, a : \Sigma \right\}$$

Definition 7.6 (Final dominoes) *For all halting states there is one domino which can complete a partial match to a match.*

$$D_{\mathsf{fin}} := \left\{ \boxed{\frac{(q_f)\$}{\$}} \;\middle|\; q_f \in Q_H \right\}$$

Definition 7.7 (Reduction) *The composed reduction f is defined as*

$$f(M, t) := \left(D_{\mathsf{init}},\, D_{\mathsf{copy}} \cup D_{\mathsf{trans}} \cup D_{\mathsf{del}} \cup D_{\mathsf{fin}}\right)$$

We use $\mathcal{D}$ to refer to the union of all dominoes $D_{\mathsf{init}} \cup D_{\mathsf{copy}} \cup D_{\mathsf{trans}} \cup D_{\mathsf{del}} \cup D_{\mathsf{fin}}$.

7.2 Correctness Proof

When proving f correct, it will be convenient to assume dominoes which copy more than one symbol. We we use symbols $a, b : \Sigma$ and strings A, B, C over Σ. If $A = a_1 a_2 \dots a_n$, we abbreviate the list of dominoes $\left[\frac{a_1}{a_1}\right]\left[\frac{a_2}{a_2}\right]\dots\left[\frac{a_n}{a_n}\right]$ as $\left[\frac{A}{A}\right]$. Of course, the top string and the bottom string of such a domino represents the list itself, $C_1\left[\frac{A}{A}\right] = A$ and $C_2\left[\frac{A}{A}\right] = A$.

The correctness proof is based on the fact that each pair of subsequent configurations, $\langle c\rangle$ and $\langle\hat{\delta}c\rangle$ where c is not final, have a unique representation with dominoes from $\mathcal{D}$. This leads to the definition of a function next, which yields, given a configuration c, a domino sequence where the top components concatenate to $\langle c\rangle$ and the bottom components to $\langle\hat{\delta}c\rangle$.

Definition 7.8 (Next configuration) *The function*

$$\mathsf{next} : \mathsf{conf} \to \mathsf{pcp}_\Gamma$$

yields a list of dominoes such that $C_1(\mathsf{next}\, c) = \langle c\rangle$ *and* $C_2(\mathsf{next}\, c) = \langle\hat{\delta}c\rangle$.

Since the complete definition of next is a straightforward case analysis on the tape of c and the outcome of δ but quite long, we explain the intuition using two examples.

Example 7.9 *Assume a configuration* $(q_1, A\underset{\uparrow}{a})$ *where* A *is a possibly empty string over* Σ. *Let* $\delta(q_1, \bot) = (q_2, \lfloor b\rfloor, L)$.

$$\mathsf{next}(q_1, A\underset{\uparrow}{a}) = \left[\frac{\langle\!|}{\langle\!|}\right]\left[\frac{A}{A}\right]\left[\frac{a q_1 |\!\rangle}{q_2 a b |\!\rangle}\right]$$

The dominoes first copy the tape delimiter $\langle\!|$ *and all symbols in* A *before an appropriate transition domino completes both configurations.*

Example 7.10 *Assume a configuration* $(q_2, A\underset{\uparrow}{a}B)$ *where* A *and* B *are possibly empty strings over* Σ. *Let* $\delta(q_2, \lfloor a\rfloor) = (q_3, \bot, L)$. *This situation needs special attention because we defined two dominoes which might fit here,* $\left[\frac{\langle\!| q_2 a}{\langle\!| q_3 \sqcup a}\right]$ *and* $\left[\frac{c q_2 a}{q_3 c a}\right]$ *for some* $c : \Sigma$. *To choose the right one, we do a case analysis on* A *which yields*

$$\mathsf{next}(q_2, \underset{\uparrow}{a}B) = \left[\frac{\langle\!| q_2 a}{\langle\!| q_3 \sqcup a}\right]\left[\frac{B}{B}\right]\left[\frac{|\!\rangle}{|\!\rangle}\right]$$

$$\mathsf{next}(q_2, Ac\underset{\uparrow}{a}B) = \left[\frac{\langle\!|}{\langle\!|}\right]\left[\frac{c q_2 a}{q_3 c a}\right]\left[\frac{B}{B}\right]\left[\frac{|\!\rangle}{|\!\rangle}\right]$$

Arranging the two results of Example 7.9 and 7.10 successively, demonstrates that next expands a partial match by one configuration.

$$\mathsf{next}\,(q_1, \underset{\uparrow}{a}) +\!\!+ \mathsf{next}\,(q_2, \underset{\uparrow}{a}b) = \left[\frac{(\!|}{(\!|}\right]\left[\frac{aq_1)\!|}{q_2ab)\!|}\right]\left[\frac{(\!|q_2a}{(\!|q_3 \sqcup a}\right]\left[\frac{b}{b}\right]\left[\frac{)\!|}{)\!|}\right]$$

$$\frac{(\!|aq_1)\!|\,(\!|q_2ab)\!|}{(\!|q_2ab)\!|\,(\!|q_3 \sqcup ab)\!|}$$

Lemma 7.11 *Let* c *be a configuration. Then* $C_1\,(\mathsf{next}\,c) = \langle c \rangle$.

Proof Let $c = (q, t)$. A case analysis on t and the outcome of $\delta\,(q, \cdot)$ which is the move and the new symbol $\Sigma_\perp$ prove the equivalence. □

Lemma 7.12 *Let* c *be a configuration. Then* $C_2\,(\mathsf{next}\,c) = \langle \hat{\delta} c \rangle$.

Proof Let $c_1 = (q, t)$. By case analysis on t, the move m, and the new symbol $o : \Sigma_\perp$ in $\delta\,(q, \perp) = (q', o, m)$, both sides evaluate to equal domino lists. If $t = A\underset{\uparrow}{a}B$, a case analysis on $\delta\,(q, \lfloor a \rfloor)$ and A proves the claim. □

Lemma 7.13 *If* q *is not a halting state, then* $\mathsf{next}\,(q, t) \subseteq \mathcal{D}$ *for all tapes* t.

Proof The dominoes of $\mathsf{next}\,(q, t)$ are either copy dominoes or transition dominoes. Since we assume q to be a non halting state, all transition dominoes used are contained in D_{trans}. Let us consider the result of the two previous examples. The dominoes $\left[\frac{(\!|}{(\!|}\right]\left[\frac{A}{A}\right]\left[\frac{aq_1)\!|}{q_2ab)\!|}\right]$ from Example 7.9 are copy dominoes and one transition domino which can be found in the second row and the leftmost column of Table 7.3. Just as well in Example 7.10, where $\left[\frac{(\!|q_2a}{(\!|q_3 \sqcup a}\right]$ and $\left[\frac{cq_2a}{q_3ca}\right]$ are transition dominoes defined in the third row. □

The function next is used in both directions of the correctness proof which is split into two sections.

7.2.1 Halting Computations to MPCP Matches

In this section we prove that the MPCP instance $f\,(M, t_0)$ is solvable, if M halts on input t_0.

$$\mathsf{Halt}\,(M, t_0) \to \mathsf{MPCP}\,(f\,(M, t_0))$$

To construct the MPCP match, the function next will be used to reach the final configuration. Afterwards it is not more applicable since transition dominoes are not defined for halting states. Instead we use deletion dominoes and a final domino to complete the match. The following lemmas state that strings to the left and the right of a final state can be removed using deletion dominoes. They are equivalent to the statements of Fact 6.4 from the reduction of Halt to string rewriting, but their proofs are longer since we additionally have to deal with copy dominoes.

Lemma 7.14 *For all halting states* q_f *and strings* B *and* C *over* Σ*, there is some list of dominoes* $A \subseteq \mathcal{D}$ *with*

$$(C_1\ A) +\!\!+ (\!|Bq_fC|\!) = (\!|Bq_f|\!) +\!\!+ (C_2\ A)$$

Proof By induction on C.

$C = [\,]$ Take $A := [\,]$

$C = a :: C$ With the inductive hypothesis we obtain A' with $(C_1\ A') +\!\!+ (\!|Bq_f|\!) = (\!|Bq_fC|\!) +\!\!+ (C_2\ A')$. We define $A := \left[\frac{(\!|}{(\!|}\right]\left[\frac{B}{B}\right]\left[\frac{q_fa}{q_f}\right]\left[\frac{C}{C}\right]\left[\frac{|\!)}{|\!)}\right] +\!\!+ A'$. $A \subseteq \mathcal{D}$ is routine, since the dominoes not in A' are either in D_{copy} or D_{del}, because q_f is a halting state. The equivalence for A' proves the claim $(\!|Bq_faC|\!) +\!\!+ (C_1\ A') +\!\!+ (\!|Bq_f|\!) = (\!|Bq_faC|\!) +\!\!+ (\!|Bq_fC|\!) +\!\!+ (C_2\ A)$ □

Remember the notation A^R to reverse a list. This is needed when dealing with leftof configurations.

Lemma 7.15 *For all halting states* q_f *and strings* B *over* Σ*, there is some list of dominoes* $A \subseteq \mathcal{D}$ *with*

$$(C_1\ A) +\!\!+ (\!|q_f|\!) = (\!|B^Rq_f|\!) +\!\!+ (C_2\ A)$$

Proof By induction on B.

$B = [\,]$ Take $A := [\,]$

$B = b :: B$ With the inductive hypothesis we obtain A' with $(C_1\ A') +\!\!+ (\!|q_f|\!) = (\!|B^Rq_f|\!) +\!\!+ (C_2\ A')$. We define $A := \left[\frac{(\!|}{(\!|}\right]\left[\frac{B^R}{B^R}\right]\left[\frac{bq_f}{q_f}\right]\left[\frac{|\!)}{|\!)}\right] +\!\!+ A'$. $A \subseteq \mathcal{D}$ is routine, since the new dominoes not in A' are either in D_{copy} or D_{del}, because q_f is a halting state. The equivalence for A' proves the claim $(\!|B^Rbq_f|\!) +\!\!+ (C_1\ A') +\!\!+ (\!|q_f|\!) = (\!|B^Rbq_f|\!) +\!\!+ (\!|B^Rq_f|\!) +\!\!+ (C_2\ A)$ □

The two lemmas lead to the statement that for all final configurations there is some list of deletion dominoes which is a match if the configuration is added at the bottom and the final state at the top.

Lemma 7.16 *For all final configurations* (q_f, t)*, there is a list of dominoes* $A \subseteq \mathcal{D}$ *with*

$$(C_1\ A) +\!\!+ (\!|q_f|\!) = \langle (q_f, t) \rangle +\!\!+ (C_2\ A)$$

Proof By case analysis on t.

$\emptyset$ ↑ We use $A := \boxed{\frac{(\!|}{(\!|}}\boxed{\frac{q_f\sqcup}{q_f}}\boxed{\frac{|\!)}{|\!)}}$ to prove the claim $(C_1\ A) +\!\!+ (\!|q_f|\!) = (\!|q_f\sqcup|\!) +\!\!+ (C_2\ A)$.

↑ aC By Lemma 7.14 with $B := [\,]$, we have A' with $(C_1\ A') +\!\!+ (\!|q_f|\!) = (\!|q_f aC|\!) +\!\!+ (C_2\ A')$. By prefixing A' with $\boxed{\frac{(\!|}{(\!|}}\boxed{\frac{q_f\sqcup}{q_f}}\boxed{\frac{a}{a}}\boxed{\frac{C}{C}}\boxed{\frac{|\!)}{|\!)}}$, it is obvious that $(\!|q_f \sqcup aC|\!) +\!\!+ (C_1\ A') +\!\!+ (\!|q_f|\!) = (\!|q_f \sqcup aC|\!) +\!\!+ (C_2\ A')$ holds.

$B^R\ b$ ↑ A list of deletion dominoes A and the claim $(C_1\ A) +\!\!+ (\!|q_f|\!) = (\!|B^R b q_f|\!) +\!\!+ (C_2\ A)$ follow directly from Lemma 7.15.

$B^R\ a\ C$ ↑ By Lemma 7.14 with $B :=_f B^R$ and $C := a :: C$, we have A' with $(C_1\ A') +\!\!+ (\!|B^R q_f|\!) = (\!|B^R q_f aC|\!) +\!\!+ (C_2\ A')$. By Lemma 7.15 with $B := B$ we have $\hat{A}$ with $(C_1\ \hat{A}) +\!\!+ (\!|q_f|\!) = (\!|B^R q_f|\!) +\!\!+ (C_2\ \hat{A})$. Using $A := A' +\!\!+ \hat{A}$, $A \subseteq \mathcal{D}$ trivially holds. It remains to show that $(C_1\ A') +\!\!+ (C_1\ \hat{A}) +\!\!+ (\!|q_f|\!) = (\!|B^R q_f aC|\!) +\!\!+ (C_2\ A') +\!\!+ (C_2\ \hat{A})$. This follows using the characterizing equations for $\hat{A}$ and A' stated above. □

The following lemma use next to compute the list of dominoes containing the successor configuration and the Lemma 7.16 to complete a partial match with a final configuration.

Lemma 7.17 *Let c be a configuration and c_f a final configuration. If $c \vdash c_f$ then there is a list of dominoes $S \subseteq \mathcal{D}$ with $\boxed{\frac{\$}{\$\langle c\rangle}} :: S$ being a match.*

Proof By induction on the proof of $c \vdash c_f$. Let $c_f = (q_f, t)$.

$c = c_f$ Lemma 7.16 provides a list A with $(C_1\ A) +\!\!+ (\!|q_f|\!) = \langle(q_f, t)\rangle +\!\!+ (C_2\ A)$. We define $S := A' +\!\!+ \boxed{\frac{(\!|q_f|\!)\$}{\$}}$, using a final domino from D_{fin} to complete the match. With the equation from above it follows that $\$\ (C_1\ A) +\!\!+ (\!|q_f|\!)\$ = \$\langle(q_f, t_f)\rangle +\!\!+ (C_2\ A)\ \$$.

$\hat{\delta}\ c \vdash c_f$ We have $\neg H_c$ and a list $A \subseteq \mathcal{D}$ with $\boxed{\frac{\$}{\$\langle\delta c\rangle}} :: A$ being a match. With $(\mathsf{next}\ c)$ we obtain dominoes leading from $\langle c\rangle$ to $\langle\hat{\delta} c\rangle$. Hence $S := (\mathsf{next}\ c) +\!\!+ A$. The claim $S \subseteq \mathcal{D}$ follows from Lemma 7.13. $\$(C_1\ (\mathsf{next}\ c)) +\!\!+ (C_1\ A) = \$\langle c\rangle +\!\!+ (C_2\ (\mathsf{next}\ c)) +\!\!+ (C_2\ A)$ follows from Lemmas 7.11 and 7.12. □

Lemma 7.18 *Assume M halts on input tape t_0. Then there is a match for the* MPCP *instance $f(M, t_0)$.*

Proof By Lemma 7.17 we know that $\boxed{\frac{\$}{\$\langle(q_0, t_0)\rangle}} :: S$ is a match. □

7.2.2 MPCP Matches to Halting Computations

It remains to show that the existence of an MPCP match for $f(M, t_0)$ implies that M halts on the input tape t_0.

$$\mathsf{MPCP}(f(M, t_0)) \to \mathsf{Halt}(M, t_0)$$

Initially, we only know about a list $S \subseteq \mathcal{D}$ with $\boxed{\frac{\$}{\$\langle(q_0, t)\rangle}} +\!\!+ S$ being a match. The further structure of S is still uncertain, but it is important to identify this pattern. We aim to prove for any list S prefixed with a configuration $\langle c\rangle$ at the bottom, that either the first dominoes describe the step from $\langle c\rangle$ to $\langle\hat{\delta}c\rangle$, or c is already a final configuration. We start with two lemmas, stating that we use copy dominoes to complete a configuration, and to reach the position where a transition domino can be used.

Lemma 7.19 *Let B and $S \subseteq \mathcal{D}$ be lists of dominoes and A a string over Σ. If $C_1\, S = A +\!\!+ \langle\!| B$ we have $S = \boxed{\frac{A}{A}} +\!\!+ \boxed{\frac{\rangle}{\rangle}} S'$ for some S'.*

Proof By induction on A with the generalized claim for all S. In both cases we prove S to be of the form $d :: S$ and do a case analysis on $d \in \mathcal{D}$.

$A = [\,]$ All dominoes except from $d = \boxed{\frac{\rangle}{\rangle}}$ are contradictory. The claim $\boxed{\frac{\rangle}{\rangle}} S = \boxed{\frac{\rangle}{\rangle}} S'$ follows with with $S' := S$.

$A = a :: A$ The copy domino $\boxed{\frac{a}{a}}$ is the only domino fitting. Then we have $a :: (C_1\, S) = a\, A +\!\!+ \langle\!| B$ and show $\boxed{\frac{a}{a}} +\!\!+ S = \boxed{\frac{a}{a}}\boxed{\frac{A}{A}} +\!\!+ \boxed{\frac{\rangle}{\rangle}} S'$ using the inductive hypothesis. □

Lemma 7.20 *Let B and $S \subseteq \mathcal{D}$ be lists of dominoes, $b : \Sigma$, and A a string over Σ. If $C_1\, S = A +\!\!+ [b] +\!\!+ B$ we have $S = \boxed{\frac{A}{A}} +\!\!+ S'$ for some S'.*

Proof By induction on A with the generalized claim for all S. In both cases we prove S to be of the form $d :: S$ and do a case analysis on $d \in \mathcal{D}$.

$A = [\,]$ Trivial.

$A = a :: A$ Because no transition domino has two subsequent symbols ab on the top component, we have $d = \boxed{\frac{a}{a}}$. We use the inductive hypothesis to get S' with $S = \boxed{\frac{A}{A}} +\!\!+ S'$ and show $\boxed{\frac{a}{a}} :: S = \boxed{\frac{a}{a}}\boxed{\frac{A}{A}} +\!\!+ S'$. □

Lemma 7.21 *Let* c *be a non final configuration and* $S \subseteq \mathcal{D}$ *with* $C_1\ S = \langle c \rangle +\!\!+ (C_2\ S)$*. We can identify the structure of* S *as* $S = (\mathsf{next}\ c) +\!\!+ S'$ *for some list* S'*.*

Proof Let $c = (q, t)$. We start by case analysis on tape t and can show that $S \neq [\,]$. In the sequel we will do case analysis on the type of the dominoes in $d :: S$. Since we have five types of dominoes and numerous transition dominoes which all need to be considered, we leave out some cases of the formal proof. Let $d = (d_1, d_2)$.

$t = \underset{\uparrow}{\emptyset}$ We have $d_1 +\!\!+ C_1\ S = (\!|q\sqcup|\!) +\!\!+ d_2 +\!\!+ (C_2\ S)$. By case analysis on $d \in \mathcal{D}$ we can exclude most types of dominoes

- $d = \boxed{\frac{\$}{\$\langle(q_0, t_0)\rangle}}$ Contradiction because $\$ \neq (\!|$.
- $d \in D_{\mathsf{copy}}$ The only copy domino which fits is $d = \boxed{\frac{(\!|}{(\!|}}$ which leads to the assumption $(\!|::(C_1\ S) = (\!|q\sqcup|\!) +\!\!+ (\!|::(C_2\ S)$. We do another case analysis on S. $S = [\,]$ is contradictory, so we continue with $S = e :: S'$. It follows another case analysis on $e \in \mathcal{D}$, but neither domino will fit, since all transition dominoes having a blank symbol at the second position begin with $(\!|$.
- $d \in D_{\mathsf{trans}}$ All transition dominoes from Table 7.3 with $(\!|q_1\sqcup$ or $(\!|q_1\sqcup|\!)$ in the top component are successful candidates for d. Let $d = \boxed{\frac{(\!|q_1\sqcup|\!)}{(\!|q_2\sqcup|\!)}}$. We have $(\!|q_1\sqcup|\!) +\!\!+ C_1\ S = (\!|q\sqcup|\!) +\!\!+ (\!|q_2\sqcup|\!) +\!\!+ (C_2\ S)$ and conclude $q_1 = q$. Because the domino d has been defined as a transition domino using the step function δ, we have $\delta\,(q, \bot) = (q_2, \bot, R)$. With this information, $\mathsf{next}\ c$ computes to exact this domino, proving $d::S = (\mathsf{next}\ c) +\!\!+ S'$ with $S' := S$. For all other suitable transition dominoes, where $|\!)$ is not included, we need do prove additionally that $\boxed{\frac{|\!)}{|\!)}}$ comes next in S.
- $d \in D_{\mathsf{del}}$ Contradiction because none of the two deletion dominoes $\boxed{\frac{q_f a}{q_f}}$ and $\boxed{\frac{a q_f}{q_f}}$ agree with the first symbol $(\!|$ and q is not a halting state.
- $d \in D_{\mathsf{fin}}$ Contradiction because $(\!|q_f|\!) \neq (\!|q\sqcup|\!)$ for all q_f and q is not a halting state.

$t = \underset{\uparrow}{a\,A}$ This case is similar to the one above, using Lemma 7.19 when the state is covered by a transition domino and only Σ symbols are left.

$t = A^R\,\underset{\uparrow}{a}$ We have $d_1 +\!\!+ (C_1\ S) = (\!|A^R a q|\!) +\!\!+ d_2 +\!\!+ (C_2\ S)$. All dominoes except from

$d = \boxed{\frac{(\!|}{(\!|}}$ are contradictory. Then the assumption is $(\!|::(C_1\ S) = (\!|A^R a q|\!) +\!\!+ (\!|::(C_2\ S)$ and we split S into $\boxed{\frac{A^R}{A^R}} +\!\!+ S'$ by Lemma 7.20. With $S' = a q|\!) +\!\!+ (\!|::(C_2\ S)$, the next dominoes in S' are either a transition domino covering the symbol a, or a copy domino $\boxed{\frac{a}{a}}$ and a transition $\boxed{\frac{q|\!)}{\hat{q}|\!)}}$.

$t = A^R\ \underset{\uparrow}{a}\ B$ With the assumption $d_1 +\!\!+ (C_1\ S) = (\!|A^R q a B|\!) +\!\!+ d_2 +\!\!+ (C_2\ B)$ we combine the techniques used for leftof and rightof tapes using both Lemmas, 7.19 and 7.20. □

Lemma 7.22 *Let c be a configuration and $S \subseteq \mathcal{D}$. If $C_1\ S = \langle c \rangle +\!\!+ (C_2\ S)$, then a final configuration is reachable from c.*

Proof By size induction on the length of S with the generalized claim for all c. Let $c = (q, t)$. We do case analysis whether q is a halting state.

$H\ q = \mathsf{true}$ The claim $(q, t) \vdash (q, t)$ holds by definition.

$H\ q = \mathsf{false}$ By Lemma 7.21 with $C_1\ S = \langle (q, t) \rangle +\!\!+ (C_2\ S)$ we have S' and $S = \mathsf{next}\ (q, t) +\!\!+ S'$. Since the result of next is not empty we have $|S'| \leqslant |S|$. To use the inductive hypothesis with $c := \hat{\delta} c$, we need to prove $C_1\ S' = (C_2\ \langle \hat{\delta} c \rangle) +\!\!+ (C_2\ S')$ which follows from $(C_1\ \mathsf{next}\ c) +\!\!+ (C_1\ S') = (C_2\ \langle c \rangle) +\!\!+ (C_2\ \mathsf{next}\ \hat{\delta} c) +\!\!+ (C_2\ S')$ and Lemmas 7.11, 7.12. The result is a final configuration c_f and $\hat{\delta} c \vdash c_f$. Since q is not a halting state we have $c \vdash c_f$. □

Lemma 7.23 *If the MPCP instance $f\ (M, t_0)$ has a match, then* $\mathsf{Halt}\ (M, t_0)$.

Proof By definition of MPCP we have a list S with $\boxed{\frac{\$}{\$\ \langle (q_0, t_0) \rangle}} :: S$ being a match. The claim follows from lemma 7.22.

The general reduction is defined for all Turing machines M and input tapes t.

Theorem 7.24 *The halting problem reduces to MPCP.*

$$\forall\, M\ t.\ \mathsf{Halt}\ (M, t) \leftrightarrow \mathsf{MPCP}\ (f\ (M, t))$$

Proof The equivalence $(\exists c_f.\ (q_0, t) \vdash c_f \wedge H_{c_f}) \leftrightarrow \exists S.\ \boxed{\frac{\$}{\$ \langle (q_0, t) \rangle}} :: S$ is a match for $f\ (M, t)$ can be shown with Lemmas 7.18 and 7.23. □

The correctness proof of this reduction shows that we do not gain a more elegant or shorter proof by skipping the intermediate reduction to string rewriting. It rather combines both ideas and turns out to have similar length than the reductions from Halt to SR and SR to MPCP together. The notion of string rewriting is hidden in the transition dominoes which simulate the rewrite process. Additionally we always have to construct complete configurations using copy dominoes next to transition dominoes. While it is simple to construct an MPCP match following a Turing computation or a string-rewriting sequence, it is particularly taxing to reveal the structure of a given MPCP match in Coq.

Consider a list S of dominoes from $\mathcal{D}$ such that $C_1\ S = (\!|abcq|\!) :: (C_2\ S)$ and q is not a halting state. For all possible results of $\delta(q, \bot)$ we aim to show that S begins with the dominoes we would expect to match. If the transition function yields $\delta(q, \bot) = (\hat{q}, \lfloor a \rfloor, L)$ one would have no doubts that $S = \left[\frac{(\!|}{(\!|}\middle|\frac{a}{a}\middle|\frac{b}{b}\middle|\frac{cq|\!)}{qca|\!)}\right] :: S'$ for some S'. To fix just one of these four dominoes, we have to show that all other dominoes in $\mathcal{D}$ are not suitable. These include the three types of deletion dominoes, copy dominoes, the initial domino, the final domino, and 21 different transition dominoes which all have to be considered.

The decision how we define dominoes plays a major role when we aim to prove a domino being the head of a list S contradictory. This starts with the initial domino $\left[\frac{\$}{\$\,(\!|q_0, t_0|\!)}\right]$ where the top and bottom row is preceded by the \$ symbol. Hopcroft et al. [10] omit this symbol, but for formal proofs it turns out to be beneficial since it does not occur inside a match. Furthermore, we encode configurations to be surrounded by the tape delimiters $(\!|$ and $|\!)$. In this proof the delimiters could be defined as the same symbol. We just adopted them from the previous encoding. If we had used the encoding from Table 5.1 without an additional blank symbol, the transition dominoes for the left end of the tape would have started with a state, for example $\left[\frac{q_1(\!|}{q_2(\!|a}\right]$. Here only the second symbol $(\!|$ ensures that this domino can exclusively be used at the left of a configuration. In our proofs it is convenient to use $(\!|$ right at the beginning, such as in $\left[\frac{(\!|q_1 \sqcup}{(\!|q_2 \sqcup a}\right]$.

Chapter 8

Reducing String Rewriting to PCP

Davis et al. [3] present a reduction of string rewriting to PCP without mentioning MPCP. However, we can easily modify their reduction to serve as a reduction of SR to MPCP, which is presented in Chapter 4. Reducing directly to PCP has the same underlying concept of simulating a rewrite sequence with dominoes. The transformation of a rewriting system R and strings x and y into a PCP instance defines a starting domino containing x, a final domino containing y, copy dominoes for all symbols and a string separator $\star$, and dominoes representing the rewrite rules in R. To make sure that the initial domino starts the match, the copy dominoes and rewrite dominoes must be modified. For each of them, two versions are defined where either the symbols at the top row or the symbols at the bottom row change to a tagged version. Tagged dominoes like $\left[\frac{\tilde{a}}{a}\right]$ and $\left[\frac{a}{\tilde{a}}\right]$ for instance, cannot be used at the beginning of a match. If the symbols at the top are all different from the symbols at the bottom, there must be an offset between these symbols since they do not match. In a partial match the bottom row is one string ahead the top row such that the top and the bottom component of a domino are separated. All strings are either represented completely by tagged or untagged dominoes and the subsequent string is represented by the contrary version.

The example below recalls the reduction of SR to MPCP defined in 4.2 and demonstrates why it does not suffice as reduction to PCP.

Example 8.1 *Consider the set of rewrite rules* $R := \{ab/ba, aa/ab\}$ *and the question whether* $(bbb) \Rightarrow_R^* (bba)$. *Since the predicate does not hold, the corresponding PCP instance should be unsolvable. Treating the special first domino as all other dominoes, the reduction* f *provides:*

$$\left\{ \left[\frac{\$}{\$bbb\star}\right], \left[\frac{bba\star\$}{\$}\right], \left[\frac{a}{a}\right]\left[\frac{b}{b}\right], \left[\frac{\star}{\star}\right], \left[\frac{ab}{ba}\right], \left[\frac{aa}{ab}\right] \right\}$$

Clearly, this is not a correct reduction to PCP: Since we are not forced to start the match with a specific domino, $\left[\frac{\star}{\star}\right]$ *and* $\left[\frac{a}{a}\right]$ *are both matches for this instance.*

8.1 Definition of PCP Dominoes

To make sure that only the domino containing the initial string starts a match, we tag the symbols from the bottom or the top row with $\sim$. Assuming a word problem (R, x, y) over alphabet Σ, the new alphabet of the PCP dominoes is defined as

$$\Gamma := \$ \mid \star \mid \tilde{\star} \mid a \mid \tilde{a} \qquad (a : \Sigma)$$

Definition 8.2 (Reduction) *The reduction* f *provides a domino to start a solution including the string* x *and a domino to end a solution with* y *at the top. Besides, we need two variants of all other types of dominoes.*

$$f : \mathbf{L}\,(\Sigma^* \times \Sigma^*) \times \Sigma^* \times \Sigma^* \to \mathsf{mpcp}_\Gamma$$

$$f\,(R, x, y) := \left\{ \left[\frac{\$}{\$x\star}\right], \left[\frac{y\star\$}{\$}\right], \left[\frac{\star}{\tilde{\star}}\right], \left[\frac{\tilde{\star}}{\star}\right] \right\} \cup \left\{ \left[\frac{a}{\tilde{a}}\right], \left[\frac{\tilde{a}}{a}\right] \,\middle|\, a : \Sigma \right\} \cup \left\{ \left[\frac{u}{\tilde{v}}\right], \left[\frac{\tilde{u}}{v}\right] \,\middle|\, u/v \in R \right\}$$

The next example shows how to solve a PCP using tagged dominoes.

Example 8.3 *Let* $R := \{aa/ab, ab/ba\}$, $x := baa$ *and* $y := bab$ *with* $baa \Rightarrow_R^* bab$. *We rewrite only once to obtain the target string* y. *Intuitively we would start a solution with*

$$\left[\frac{\$}{\$baa\star}\right]\left[\frac{b}{\tilde{b}}\right]\left[\frac{aa}{\tilde{a}\tilde{b}}\right]\left[\frac{\star}{\tilde{\star}}\right]\cdots$$

The final domino $\left[\frac{bab\star\$}{\$}\right]$ *comes with an untagged version of* bab, *so we need to copy the final string to complete the match.*

$$\cdots\left[\frac{\tilde{b}}{b}\right]\left[\frac{\tilde{a}}{a}\right]\left[\frac{\tilde{b}}{b}\right]\left[\frac{\tilde{\star}}{\star}\right]\left[\frac{bab\star\$}{\$}\right]$$

The correctness claim for the reduction f states

$$x \Rightarrow_R^* y \leftrightarrow \mathsf{PCP}\,(f\,(R, x, y))$$

With our general definition of string-rewriting systems this is not provable. The problem are empty rules that can be put together forming a match, although a derivation of y from x is not possible.

Example 8.4 *Let* $R := \{b/\varepsilon, \varepsilon/b\}$*, the initial string* $x := a$*, and the target* $y := b$*. The reduction* f *yields:*

$$f(R, a, b) = \left\{ \boxed{\frac{\$}{\$a\star}}, \boxed{\frac{b\star\$}{\$}}, \boxed{\frac{\star}{\tilde{\star}}}, \boxed{\frac{\tilde{\star}}{\star}}, \boxed{\frac{a}{\tilde{a}}}, \boxed{\frac{\tilde{a}}{a}}, \boxed{\frac{b}{\tilde{b}}}, \boxed{\frac{\tilde{b}}{b}}, \boxed{\frac{b}{-}}, \boxed{\frac{\tilde{b}}{-}}, \boxed{\frac{-}{\tilde{b}}}, \boxed{\frac{-}{b}} \right\}$$

Obviously, $\boxed{\frac{b}{-}}\boxed{\frac{-}{b}}$ *serves as solution for this PCP instance but* $a \not\Rightarrow_R^* b$.

8.2 Reducing String Rewriting to Restricted String Rewriting

To avoid the problem with empty rules, citetdavis assume their string-rewriting systems to have only non empty strings in both components. They justify this decision by the reduction of Halt to SR, which yields only rewrite rules that are non empty. We give a full reduction of SR to RSR. Transforming a set of rules R over alphabet Σ, and strings x and y into R', x', and y' such that all rules in R' are nonempty and $x \Rightarrow_R^* y \leftrightarrow x' \Rightarrow_{R'}^* y'$ is an easy exercise. We use the function $\#_R$ defined in 3.5 and interleave all strings with # symbols. Further we add an additional # at the beginning of each string to ensure that they are nonempty. The alphabet of the restricted string-rewriting system is defined as

$$\Sigma^{\#} := \# \mid (a : \Sigma).$$

Definition 8.5 *The function* h *interleaves all rules of a set* R *with # symbols.*

$$h\,R := \{(\# :: (u\#_R) \,/\, \# :: (v\#_R)) \mid (u/v) \in R\}$$

Lemma 8.6 $\forall\, u\, v.\, (u/v) \in h\,R \to u \neq [\,] \wedge v \neq [\,]$.

Definition 8.7 (Reduction) *When reducing SR to RSR we also modify the initial string* x *and the target string* y.

$$g(R, x, y) := (\{h\,R \mid \forall\, u\, v.\, (u/v) \in g\,R \to u \neq [\,] \wedge v \neq [\,]\}, \# :: (x\#_R), \# :: (y\#_R)).$$

Theorem 8.8 *The class SR reduces to RSR.*

$$\forall\, R\, x\, y.\, \mathsf{SR}(R, x, y) \leftrightarrow \mathsf{RSR}(g(R, x, y))$$

Proof Rewriting is not affected by inserting # symbols, but it ensures that for all $(u/v) \in R$, $u \neq [\,]$ and $v \neq [\,]$. The correctness proof of this reduction can be found online. □

8.3 Correctness Proof

Actually we do not reduce SR to PCP, but put the reduction of SR to RSR in front. This solves the problem with empty rules from Example 8.4 and leaves us with the claim $x \Rightarrow_R^* y \leftrightarrow \mathsf{PCP}\ (f\,(R, x, y))$ having the assumption $\forall\, u\, v.\, (u/v) \in R \to u \neq [\,] \wedge v \neq [\,]$.

The verification of the reduction f requires the same techniques as the correctness proof in Chapter 4. The construction of a PCP match is straightforward but the other direction needs considerable work. After identifying the first domino of the match, $\boxed{\frac{\$}{\$x\star}}$, the inner structure is still unknown. The proof requires all lemmas in two versions, one where a tagged string exceeds the bottom row of the partial match and another one with the untagged string. A presentation of the formal proof would be quite repetitive, such that we only state the theorem and refer the interested reader to the online proof.

Theorem 8.9 *RSR reduces to PCP.*

$$\forall\, (H : \{R \,|\, \forall\, u\, v.\, (u/v) \in R \to u \neq [\,] \wedge v \neq [\,]\})\, x\, y.\, \mathsf{RSR}\,(H, x, y) \leftrightarrow \mathsf{PCP}\,(f\,(R, x, y))$$

The reduction presented in this chapter seems to be unnecessarily complicated in contrast to the two independent reductions involving MPCP. However, the idea of tagged dominoes is interesting since it cannot be used to reduce MPCP to PCP. The reason for this is that an MPCP match might have symbols from one domino matching at the top and the bottom row. If the domino is tagged in one row, the symbols do not match. A second property of the matches for string rewrite sequences is that one string is tagged or untagged uniformly, because we complete the string using copy dominoes of the same variant than the rewrite domino. Consequently we never face the situation that either version of a rewrite rule does not fit because a string is tagged heterogeneously. The idea of tagging dominoes could be used to give a direct reduction of Halt to PCP with cannot be found in the literature. However, a formal proof of the combination of three reductions would be quite involved.

In addition to nonempty rules, Davis et al. [3] assume the underlying alphabet to consist of exactly two symbols. Therefore, they give a reduction of string-rewriting systems without empty rules to string-rewriting systems with only two symbols in their alphabet beforehand. We do not need this extra reduction in our proof, because the definition of the dominoes can easily be adapted to alphabets containing more than two symbols and this does not introduce any significant overhead in the formal proof.

Chapter 9

Conclusion

We conclude the thesis by a brief discussion of the undecidability result and outline directions for future work.

9.1 Undecidability of PCP

The results of the previous chapters lead to the undecidability of the Post correspondence problem, which is obtained by a reduction of the halting problem.

Theorem 9.1 *Halt reduces to PCP.*

Proof The claim follows from transitivity of reductions (Fact 2.3) using different intermediate classes:

SR, MPCP By Theorems 3.17, 4.9, and 6.11.

MPCP By Theorems 3.17 and 7.24.

SR, RSR By Theorems 6.11, 8.8, and 8.9. □

Recall that a class is undecidable if Halt reduces to it. This leads to the final theorem stating the undecidability of PCP.

Theorem 9.2 *The class PCP is undecidable.*

Proof By Theorem 9.1. □

We formalized, verified, and compared different approaches to prove the Post correspondence problem undecidable. First we defined a formal notion of reductions and undecidability as well as the classes PCP, MPCP, string rewriting, and the halting problem for Turing machines. We presented reductions from MPCP to PCP and from Halt to MPCP following Hopcroft et al. [10]. By reducing Halt to SR, which

includes a reduction of Reach to SR, and SR to MPCP, we disclosed the principle of string rewriting which is hidden in the direct reduction to MPCP. Additionally we verified reductions from SR to RSR, and from RSR to PCP following Davis et al. [3].

9.2 Future Work

The formal undecidability result of PCP leads to several options for future work. One direction is the field of context-free grammars and related undecidable problems. The undecidability of deciding inclusion for two context-free grammars or the question whether the intersection of two context-free grammars is nonempty can both be obtained by a reduction of PCP. The proof ideas are described in [10] and [3] and could be verified formally in Coq.

Furthermore, PCP can be used as a stepping stone to the formalized undecidability of other formalisms. One could formalize undecidability proofs for the validity problem in first-order logic [15] or the secrecy problem for security protocols [19] amongst others.

Lastly, one could prove that PCP is not decidable with respect to a concrete model of computation. Forster and Smolka [6] use a weak call-by-value λ-calculus they call L to develop basic computability theory in Coq. In order to to show λ undecidability, all functions used in this thesis need to be implemented in L. Furthermore, one would have to formalize the computational equivalence of L and Turing machines proven in [11]. This would prove λ undecidability, and, by the equivalence result, Turing undecidability of PCP.

Appendix A

Realization in Coq

All reductions presented in this thesis have been defined and verified in the proof assistant Coq [17] without using additional axioms. The development is available online at `https://www.ps.uni-saarland.de/~heiter/bachelor.php`. The files have been compiled using version 8.6 of Coq and are structured as follows:

Preliminaries	The file *Prelim.v* contains definitions and tactics adapted from the ICL base library [1] and a formalization of finite types adopted from [12].
Definitions	*Reductions.v*, *String_rewriting.v*, *PCP.v*, and *Single_TM.v* provide the definitions stated in Chapter 2.
Reductions	All reductions are defined and verified in separate files, except the reduction of Reach to SR, which is included in the file containing the reduction of Halt to SR.
	Chapter 3: MPCP to PCP is formalized in *MPCP_PCP.v*.
	Chapter 4: SR to MPCP is formalized in *SR_MPCP.v*.
	Chapter 5: Reach to SR and
	Chapter 6: Halt to SR are formalized in *Halt_SR.v*.
	Chapter 7: Halt to MPCP is formalized in *Halt_MPCP.v*.
	Chapter 8: SR to RSR is formalized in *SR_RSR.v*.
	RSR to PCP is formalized in *RSR_PCP.v*.
Undecidability	*PCP_undecidability.v* uses all reduction files to state the undecidability result of PCP.

[1] Gert Smolka. *Base library for ICL lecture*. Saarland University, 2016. URL `http://www.ps.uni-saarland.de/courses/cl-ss16/LectureNotes/html/LectureNotes.Base.html`

Overall, the development has about 2800 lines of code, split into 1100 lines of specification and 1700 lines of proofs. The distribution between the different reductions is stated in Table A.1. The preliminary definitions are included in the first row.

File	Spec	Proof	Σ
Definitions	292	121	413
MPCP_PCP.v	75	145	220
SR_MPCP.v	50	127	177
Halt_SR.v	209	349	558
Halt_MPCP.v	306	517	823
SR_RSR.v	37	71	108
RSR_PCP.v	118	328	446
PCP_undecidability.v	9	12	21
	1096	1670	2766

Table A.1: Lines of code of the formalized reductions.

The independent reduction of Reach to SR make up 70% of the reduction of Halt to SR. Note that the three different approaches to reduce Halt to PCP are almost equally long: The lines of code add up to 955 in the reduction via SR and MPCP, to 1043 via MPCP, and to 1112 via SR.

The only proof techniques needed to verify the reductions is induction, size induction, and case analysis. When dealing with nested case analyses it is advantageously to formalize the proofs in Coq, since keeping track of all assumptions and unsolved cases by hand would be difficult. This applies primarily to the reductions that transform an instance of Halt to SR or MPCP, including proofs which require large case analyses on the outcome of transitions. To shorten the cumbersome process of solving all cases one by one, we define tactics which automate simple and repetitive parts of these proofs. In *Halt_MPCP.v*, the split_element tactic mainly applies lemmas related to list membership. In *Halt_SR.v* we additionally automate the process of extracting information hidden in the equivalence of two lists. If we have, for instance, $A ++ (a :: q_1 :: B) = c :: (C ++ (q_2 :: D))$ and $c :: C$ and D do not include a state, it is obvious that $A ++ [a] = c :: C$, $B = D$, and $q_1 = q_2$ but it requires several tactics to get the desired result in Coq. In Lemma 5.8 we have to analyze such list equivalences for all 21 transition rules. The definition and verification of reductions of the halting problem could be shortened by using a simpler model of Turing machines. If we had not allowed the tape head to remain steady, or had defined a semi-infinite tape, there would have been less transition rules and transition dominoes to consider.

Bibliography

[1] Andrea Asperti and Wilmer Ricciotti. Formalizing Turing machines. In *Logic, Language, Information and Computation*, pages 1–25. Springer, 2012.

[2] Andrea Asperti and Wilmer Ricciotti. A formalization of multi-tape Turing machines. *Theoretical Computer Science*, 603:23–42, 2015.

[3] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, 1994.

[4] Andrzej Ehrenfeucht, Juhani Karhumäki, and Grzegorz Rozenberg. The (generalized) Post correspondence problem with lists consisting of two words is decidable. *Theoretical Computer Science*, 21:119 – 144, 1982.

[5] Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *CONCUR 2016*, pages 13:1–13:14, 2016.

[6] Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in Coq. In *ITP 2017*, Apr 2017. To appear.

[7] Vesa Halava. Another proof of undecidability for the correspondence decision problem - had I been Emil Post. *CoRR*, abs/1411.5197, 2014.

[8] Vesa Halava, Tero Harju, and Mika Hirvensalo. Binary (generalized) Post correspondence problem. *Theoretical Computer Science*, 276:183–204, April 2002.

[9] Wim H. Hesselink. Post's correspondence problem and the undecidability of context-free intersection. Manuscript, July 2015. URL `http://wimhesselink.nl/pub/whh513.pdf`.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2006.

[11] Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theoretical Computer Science*, 398(1-3):32–50, 2008.

[12] Jan Menz. A Coq library for finite types, 2016. URL `https://www.ps.uni-saarland.de/~menz/bachelor.php`. Bachelor's Thesis.

[13] Turlough Neary. Undecidability in binary tag systems and the Post correspondence problem for four pairs of words. *CoRR*, abs/1312.6700, 2013.

[14] Emil L Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

[15] Uwe Schöning. *Logic for Computer Scientists*. Birkhäuser Boston, 2009.

[16] Fu Song and Zhilin Wu. Extending temporal logics with data variable quantifications. In *FSTTCS 2014*, pages 253–265, 2014.

[17] The Coq Development Team. Reference manual, Version 8.6, 2016. `https://coq.inria.fr/distrib/current/refman/`.

[18] Axel Thue. *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln*. J. Dybwad, 1914.

[19] Ferucio Laurentiu Tiplea, Constantin Enea, and Catalin V. Birjoveanu. Decidability and complexity results for security protocols. In *NATO 2005*, pages 185–211, 2005.

[20] Jian Xu, Xingyuan Zhang, and Christian Urban. Mechanising Turing machines and computability theory in Isabelle/HOL. In *ITP 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2013.

[21] Ling Zhao. Tackling Post's correspondence problem. In *Computers and Games*, pages 326–344, July 2002.