Constructive Many-One Reduction from the Halting Problem to Semi-Unification

Andrej Dudenhefner
Saarland University, Saarbrücken, Germany

Abstract

The undecidability of semi-unification (unification combined with matching) has been proven by Kfoury, Tiuryn, and Urzyczyn in the 1990s. The original argument is by Turing reduction from Turing machine immortality (existence of a diverging configuration).

There are several aspects of the existing work which can be improved upon. First, many-one completeness of semi-unification is not established due to the use of Turing reductions. Second, existing mechanizations do not cover a comprehensive reduction from Turing machine halting to semi-unification. Third, reliance on principles such as König's lemma or the fan theorem does not support constructivity of the arguments.

Improving upon the above aspects, the present work gives a constructive many-one reduction from the Turing machine halting problem to semi-unification. This establishes many-one completeness of semi-unification. Computability of the reduction function, constructivity of the argument, and correctness of the argument is witnessed by an axiom-free mechanization in the Coq proof assistant. The mechanization is incorporated into the existing Coq library of undecidability proofs. Notably, the mechanization relies on a technique invented by Hooper in the 1960s for Turing machine immortality.

An immediate consequence of the present work is an alternative approach to the constructive many-one equivalence of System F typability and System F type checking, compared to the argument established in the 1990s by Wells.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases constructive mathematics, undecidability, mechanization, semi-unification

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.18

Supplementary Material *Software (Source Code)*: `https://github.com/uds-psl/coq-library-undecidability/tree/coq-8.12/theories/SemiUnification`

Acknowledgements The author is grateful for encouragement and assistance by Paweł Urzyczyn and the members of the programming systems lab led by Gert Smolka at Saarland University.

1 Introduction

Semi-unification is the combination of first-order unification and first-order matching. That is, given a finite set of pairs of first-order terms, is there a valuation φ such that for each pair (σ, τ) in the set of first-order terms we have $\psi(\varphi(\sigma)) = \varphi(\tau)$ for some valuation ψ? Semi-unification naturally arises in type inference for functional and logic programs [30, 26, 19] which allow for polymorphic recursion [34]. Intuitively, the valuation φ establishes global code invariants, and the individual valuations ψ establish additional local conditions for each polymorphic function application. While both first-order unification and first-order matching are decidable problems, the status of semi-unification remained open throughout the 1980s, until answered negatively by Kfoury, Tiuryn, and Urzyczyn [25, 27]. The undecidability of semi-unification impacted programming language design and analysis with respect to polymorphic recursion [31, 21], loop detection [36], and data flow [12]. Another prominent result based on the undecidability of semi-unification is the undecidability of System F [18, 37] typability and type checking [39]. Of course, the negative result motivated the complementary line of work [31] in search for expressive, decidable fragments of semi-unification. A notable decidable fragment is *acyclic* semi-unification, used for standard ML typability [28].

Due to the importance of semi-unification in functional programming, it is natural to ask for *surveyable* evidence (both locally and globally in the sense of [4]) for its undecidability. The original undecidability proof [25] is quite sophisticated, and it was recently simplified [10] and partially mechanized (in the Coq proof assistant). Unfortunately, there are several aspects that obstruct surveyability of previous work.

First, existing arguments rely on the undecidability of Turing machine immortality, shown by Hooper [20]. The corresponding construction has received more attention [24], however, it was never published in full detail. Hooper remarks:

> *A routine and unimaginative analysis-of-cases proof would point this out more clearly; but it has remained unwritten since, as a rather tedious insult to the alert, qualified reader, it would surely remain unread.*

While the omissions are justified by accessibility, they hinder verification in full detail. The existing mechanization [10] of the undecidability of semi-unification does not improve upon this aspect, as it treats the undecidability of Turing machine immortality axiomatically.

Second, existing arguments use principles such as excluded middle, König's lemma [25], or the fan theorem [10] that do not support constructivity of the arguments. As a result, anti-classical theories, such as synthetic computability theory [5], may be in conflict with such constructions. The question arises, whether non-constructive principles are inherent to semi-unification or could be avoided.

Third, existing arguments use Turing reductions and are insufficient to establish many-one completeness of semi-unification. Hitherto, a many-one reduction from Turing machine halting to semi-unification is not given.

This work improves upon the above aspects as follows. It provides a comprehensive chain of many-one reductions from Turing machine halting to semi-unification, replacing immortality with uniform boundedness. Crucially, each many-one reduction is mechanized in full detail in the Coq proof assistant [8]. The mechanization witnesses correctness and constructivity of the argument. Specifically, the notion of a *constructive* proof is identified with an axiom-free Coq mechanization (cf. calculus of inductive constructions). It neither assumes functional extensionality (cf. homotopy type theory), Markov's principle (cf. Russian constructivism), nor the fan theorem (cf. Brouwer's intuitionism). Finally, the mechanization is integrated into the Coq Library of Undecidability Proofs [17], and contributes a (first of its kind) mechanized variant of Hooper's immortality construction [20].

The described improvements allow for an alternative approach to show many-one equivalence of System F typability and System F type checking, compared to the argument established in the 1990s by Wells [39]. The original argument interreduces type checking and typability directly, which requires a technically sophisticated argument. Having a constructive many-one reduction from Turing machine halting to semi-unification at our disposal (together with recursive enumerability of System F typability and type checking), it suffices (and is simpler) to reduce semi-unification to type checking and typability individually [11].

Synopsis

The reduction from Turing machine halting to semi-unification is divided into several reduction steps. Each reduction step is many-one, constructive, and mechanized as part of the Coq Library of Undecidability Proofs [17]. In particular, a predicate P over the domain X *constructively many-one* reduces to a predicate Q over the domain Y, if there exists a computable function $f : X \to Y$ such that for all $x \in X$ we constructively have $P(x) \iff Q(f(x))$.

Section 2: Turing machine halting is reduced to two-counter machine halting (Lemma 3) using Minsky's argument [33, Section 14.1]. This step simplifies the machine model.

Section 3: Two-counter machine halting is reduced to one-counter machine 1-halting (Lemma 12), adapting Minsky's observation [33, Section 14.2]. This step prepares the machine model for nested simulation via two-stack machines without symbol search.

Section 4.1: One-counter machine 1-halting is reduced to deterministic, length-preserving two-stack machine uniform boundedness (Lemma 26), adapting Hooper's construction for Turing machine immortality [20]. This step transitions the machine problem from halting to uniform boundedness.

Section 4.2: Deterministic, length-preserving two-stack machine uniform boundedness is reduced to confluent, simple two-stack machine uniform boundedness (Lemma 35), simplifying machine structure. This step enables reuse of the existing mechanized reduction from a uniform boundedness problem to semi-unification [10].

Section 5.1: Confluent, simple two-stack machine uniform boundedness is reduced to simple semi-unification (Lemma 46), strengthening previous work [10]. This step transitions to an undecidable fragment of semi-unification.

Section 5.2: Simple semi-unification is reduced to right-uniform, two-inequality semi-unification (Lemma 50), establishing the main result (Theorem 53).

Section 6: Outline of the mechanization of the above reduction steps.

2 Two-counter Machines

The key insight of recent work [10] establishes a direct correspondence between semi-unification and a uniform boundedness problem for a machine model. In order to reduce Turing machine halting to such a problem, in this section we consider *two-counter machines* as a well-understood, mechanized [16], and more convenient intermediate model of computation.

Two-counter machines, pioneered by Minsky [33, Section 14.1], are a restricted form of register machines and constitute a particularly simple, Turing-complete model of computation. A two-counter machine (Definition 1) stores data in two *counters*, each containing a natural number. A program instruction may either increment or decrement a counter value, and modify the current program index. To avoid partiality, we model halting via a trivial cycle. The size of a two-counter machine is the length, denoted $|\cdot|$, of the list of its instructions.

▶ **Definition 1** (Two-counter Machine ($\mathcal{M}$)). *A* two-counter machine $\mathcal{M}$ *is a list of* instructions *of shape either* $\mathtt{inc}_0$, $\mathtt{inc}_1$, $\mathtt{dec}_0\, j$, *or* $\mathtt{dec}_1\, j$, *where* $j \in \mathbb{N}$ *is a* program index.

A configuration *of* $\mathcal{M}$ *is of shape* $(i, (a, b))$, *where* $i \in \mathbb{N}$ *is the current* program index *and* $a, b \in \mathbb{N}$ *are the current* counter values.

The step relation *of* $\mathcal{M}$ *on configurations, written* $(\longrightarrow_{\mathcal{M}})$, *is given by*

- *if* $|\mathcal{M}| \leq i$, *then* $(i, (a, b)) \longrightarrow_{\mathcal{M}} (i, (a, b))$ *and we say* $(i, (a, b))$ halts
- *if* $\mathtt{inc}_0$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (a, b)) \longrightarrow_{\mathcal{M}} (i + 1, (a + 1, b))$
- *if* $\mathtt{inc}_1$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (a, b)) \longrightarrow_{\mathcal{M}} (i + 1, (a, b + 1))$
- *if* $\mathtt{dec}_0\, j$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (0, b)) \longrightarrow_{\mathcal{M}} (i + 1, (0, b))$ *and* $(i, (a + 1, b)) \longrightarrow_{\mathcal{M}} (j, (a, b))$
- *if* $\mathtt{dec}_1\, j$ *is the* i*-th instruction of* $\mathcal{M}$, *then* $(i, (a, 0)) \longrightarrow_{\mathcal{M}} (i + 1, (a, 0))$ *and* $(i, (a, b + 1)) \longrightarrow_{\mathcal{M}} (j, (a, b))$

The reachability relation *of* $\mathcal{M}$ *on configurations, written* $(\longrightarrow^*_{\mathcal{M}})$, *is the reflexive, transitive closure of* $(\longrightarrow_{\mathcal{M}})$.

A configuration $(i, (a, b))$ *is* terminating *in* $\mathcal{M}$, *if we have* $(i, (a, b)) \longrightarrow^*_{\mathcal{P}} (i', (a', b'))$ *for some halting configuration* $(i', (a', b'))$.

Despite its remarkable simplicity, the halting problem for two-counter machines (Problem 2) is undecidable (Corollary 4).

▶ **Problem 2** (Two-counter Machine Halting)**.** Given a two-counter machine $\mathcal{M}$ and two natural numbers $a, b \in \mathbb{N}$, is the configuration $(0, (a, b))$ terminating in $\mathcal{M}$?

▶ **Lemma 3.** *The Turing machine halting problem many-one reduces to the two-counter machine halting problem (Problem 2).*

Proof Sketch. Minsky describes the simulation of Turing machines by machines with four registers [33, Section 11.2] working on Gödel encodings. Then, machines with four registers are simulated by two-counter machines [33, Theorem 14.1-1]. ◀

▶ **Corollary 4.** *Two-counter machine halting (Problem 2) is undecidable.*

▶ Remark 5. Minsky's original argument is constructive and is sufficient for our technical result. However, for mechanization we rely on existing work by Forster et al. [16], which is part of the Coq Library of Undecidability Proofs [17]. This approach many-one reduces Turing machine halting via the Post correspondence problem [35, 13] and Conway's FRACTRAN halting [7, 29] to two-counter machine halting.

Commonly, two-counter machines are easily simulated by other machine models and therefore serve a key role in undecidability proofs for machine immortality problems [20, 24]. However, they have one drawback with respect to uniform boundedness. That is, there is no natural increasing measure on configurations along the step relation, as it allows for non-trivial cycles (Example 6).

▶ **Example 6.** Consider $\mathcal{M} = [\mathtt{inc}_0, \mathtt{dec}_0\, 0]$. For any counter values $a, b \in \mathbb{N}$ the configuration $(0, (a, b))$ is non-terminating in $\mathcal{M}$ because of the non-trivial cycle $(0, (a, b)) \longrightarrow_{\mathcal{M}} (1, (a + 1, b)) \longrightarrow_{\mathcal{M}} (0, (a, b))$.

One could eliminate non-trivial cycles by introducing a third counter, which increases at every step. While this induces a natural increasing measure (value of the third counter), it incurs additional bookkeeping. A simulation of such a three-counter machine by an acyclic two-counter machine is possible [24], however, it again obscures the underlying measure.

We address this drawback of two-counter machines with respect to uniform boundedness in the following Section 3, building upon Minsky's notion of machines with *one* register.

3 One-counter Machines

As Minsky observed [33, Section 14.2], with multiplication and division by constants the halting problem is undecidable for *one-counter machines*. Specifically, increase (resp. decrease) operations for two values a and b can be simulated by multiplication (resp. division) by 2 and 3 for one value $2^a 3^b$.

In this section, we further develop Minsky's construction (similarly to [40]) of universal machines with one counter in pursuit of two goals. First, machine runs should be easy to simulate in the stack machine model (Remark 16). Second, we need a measure on machine configurations that increases along the step relation, directly connecting non-termination and unboundedness (Lemma 10).

A program instruction of a one-counter machine (Definition 7), besides modifying the program index, conditionally multiplies the current counter value with either $\frac{2}{1}$, $\frac{3}{2}$, $\frac{4}{3}$, or $\frac{5}{4}$. Notably, such a multiplication by $\frac{d+1}{d}$ for $d \in \{1, 2, 3, 4\}$ is both easy to simulate uniformly, and strictly increases a (positive) counter value.

▶ **Definition 7** (One-counter Machine ($\mathcal{P}$)). *A* one-counter machine $\mathcal{P}$ *is a list of* instructions *of shape* (j, d)*, where* $j \in \mathbb{N}$ *is a* program index *and* $d \in \{1, 2, 3, 4\}$ *is a* counter modifier.

A configuration *of* $\mathcal{P}$ *is a pair* (i, c)*, where* $i \in \mathbb{N}$ *is the current* program index *and* $c \in \mathbb{N}$ *such that* $c > 0$ *is the current* counter value.

The step relation *of* $\mathcal{P}$ *on configurations, written* $(\longrightarrow_{\mathcal{P}})$*, is given by*

- *if* $|\mathcal{P}| \leq i$*, then* $(i, c) \longrightarrow_{\mathcal{P}} (i, c)$ *and we say* (i, c) halts
- *if* (j, d) *is the* i*-th instruction of* $\mathcal{P}$ *and* d *divides* c*, then* $(i, c) \longrightarrow_{\mathcal{P}} (j, c \cdot \frac{d+1}{d})$
- *if* (j, d) *is the* i*-th instruction of* $\mathcal{P}$ *and* d *does not divide* c*, then* $(i, c) \longrightarrow_{\mathcal{P}} (i + 1, c)$

The reachability relation *of* $\mathcal{P}$ *on configurations, written* $(\longrightarrow_{\mathcal{P}}^{*})$*, is the reflexive, transitive closure of* $(\longrightarrow_{\mathcal{P}})$*.*

A configuration (i, c) *is* terminating *in* $\mathcal{P}$*, if we have* $(i, c) \longrightarrow_{\mathcal{P}}^{*} (i', c')$ *for some halting configuration* (i', c')*.*

▶ **Example 8.** Consider $\mathcal{P} = [(1, 1), (0, 2)]$. The configuration $(0, 1)$ is not terminating in $\mathcal{P}$ because of the infinite configuration chain

$$(0,1) \longrightarrow_{\mathcal{P}} (1, 1 \cdot \frac{2}{1}) \longrightarrow_{\mathcal{P}} (0, 2 \cdot \frac{3}{2}) \longrightarrow_{\mathcal{P}} (1, 3 \cdot \frac{2}{1}) \longrightarrow_{\mathcal{P}} (0, 6 \cdot \frac{3}{2}) \longrightarrow_{\mathcal{P}} (1, 9 \cdot \frac{2}{1}) \longrightarrow_{\mathcal{P}} \cdots$$

The step relation for one-counter machines is total (Lemma 9.1), functional (Lemma 9.2), and forms increasing chains (Lemma 9.3 and Lemma 9.4) up to a halting configuration.

▶ **Lemma 9** (One-counter Machine Step Relation Properties).

1. ***Totality:***
   For all configurations (i, c) *there is a configuration* (i', c') *such that* $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$*.*
2. ***Functionality:***
   If $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$ *and* $(i, c) \longrightarrow_{\mathcal{P}} (i'', c'')$*, then* $(i', c') = (i'', c'')$*.*
3. ***Increasing Measure:***
   If $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$ *and* (i, c) *is not halting, then* $|\mathcal{P}| \cdot c + i < |\mathcal{P}| \cdot c' + i'$*.*
4. ***Monotone Counter:***
   a. *If* $(i, c) \longrightarrow_{\mathcal{P}} (i', c')$*, then* $c \leq c'$*.*
   b. *If* $(i, c) \longrightarrow_{\mathcal{P}}^{|\mathcal{P}|+1} (i', c')$ *and* (i', c') *is not halting, then* $c < c'$*.*

Proof. Routine case analysis. ◀

The above Lemma 9.3 gives an increasing along $(\longrightarrow_{\mathcal{P}})$ measure $|\mathcal{P}| \cdot c + i$ on non-halting configurations (i, c). Therefore, any configuration cycle is trivial, i.e. the corresponding configuration is halting. Additionally, by Lemma 9.4, the counter value is guaranteed to increase after $|\mathcal{P}| + 1$ steps, unless a halting configuration is reached. This results in a characterization of termination via boundedness of reachable counter values (Lemma 10).

▶ **Lemma 10.** *Let* $\mathcal{P}$ *be a one-counter machine. A configuration* (i, c) *is terminating in* $\mathcal{P}$ *iff there is a* $k \in \mathbb{N}$ *such that for all configurations* (i', c') *with* $(i, c) \longrightarrow_{\mathcal{P}}^{*} (i', c')$ *we have* $c' < k$*.*

Proof. If from (i, c) the machine $\mathcal{P}$ halts after n steps, then $k = 1 + c \cdot 2^n$ bounds the reachable from (i, c) counter values. Conversely, if k bounds the reachable from (i, c) counter values, then after at most $k \cdot (|\mathcal{P}| + 1)$ steps (i, c) reaches a halting configuration by Lemma 9.4. ◀

The halting problem for one-counter machines (Problem 11) starting from the configuration $(0, 1)$ is undecidable by reduction from the halting problem for two-counter machines (Lemma 12).

▶ **Problem 11** (One-counter Machine 1-Halting). Given a one-counter machine $\mathcal{P}$, is the configuration $(0, 1)$ terminating in $\mathcal{P}$?

▶ **Lemma 12.** *Two-counter machine halting (Problem 2) many-one reduces to one-counter machine* 1*-halting (Problem 11).*

Proof. Let $\mathcal{M}$ be a two-counter machine and let $a_0, b_0 \in \mathbb{N}$ be two values. We represent a pair (a, b) of counter values of $\mathcal{M}$ by the family of counter values $2^a 3^b 5^m$ where $m \in \mathbb{N}$.

We simulate instructions of $\mathcal{M}$ on two counters (a, b) by instructions of $\mathcal{P}$ on one counter $c = 2^a 3^b 5^m$ as follows. Increase a is simulated by $2^a 3^b 5^m \cdot \frac{2}{1} = 2^{a+1} 3^b 5^m$. Increase b is simulated by $2^a 3^b 5^m \cdot \frac{2}{1} \cdot \frac{3}{2} = 2^a 3^{b+1} 5^m$. Decrease a is simulated by $2^{a+1} 3^b 5^m \cdot \frac{3}{2} \cdot \frac{4}{3} \cdot \frac{5}{4} = 2^a 3^b 5^{m+1}$. Decrease b is simulated by $2^a 3^{b+1} 5^m \cdot \frac{4}{3} \cdot \frac{5}{4} = 2^a 3^b 5^{m+1}$. Failed decrease instructions may rely on a simulation of an unconditional jump instruction via $2^a 3^b 5^m \cdot \frac{2}{1} \cdot \frac{2}{1} \cdot \frac{5}{4} = 2^a 3^b 5^{m+1}$. Initialization of counter values (a_0, b_0) is simulated via $2^0 3^0 5^0 \cdot (\frac{2}{1})^{a_0+b_0} \cdot (\frac{3}{2})^{b_0} = 2^{a_0} 3^{b_0} 5^0$. Overall, the configuration $(0, (a_0, b_0))$ is terminating in the two-counter machine $\mathcal{M}$ iff the configuration $(0, 1)$ is terminating in the one-counter machine $\mathcal{P}$. ◀

▶ **Corollary 13.** *One-counter machine* 1*-halting (Problem 11) is undecidable.*

The following Example 14 illustrates the construction in the proof of Lemma 12, simulating a looping two-counter machine from Example 6.

▶ **Example 14.** Consider $\mathcal{M} = [\mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 6 with the initial counter values $(a_0, b_0) = (1, 1)$. Following the proof of Lemma 12, we construct the one-counter machine

$$\mathcal{P} = [(1,1), (2,1), (3,2), (4,1), (5,2), (6,3), (3,4)]$$

Starting from the configuration $(0, 1) = (0, 2^0 3^0 5^0)$ a run of $\mathcal{P}$ starts with the initialization

$$(0, 2^0 3^0 5^0) \xrightarrow{(1,1)}_{\mathcal{P}} (1, 2^1 3^0 5^0) \xrightarrow{(2,1)}_{\mathcal{P}} (2, 2^2 3^0 5^0) \xrightarrow{(3,2)}_{\mathcal{P}} (3, 2^1 3^1 5^0) = (3, 2^{a_0} 3^{b_0} 5^0)$$

Next, the infinite loop of $\mathcal{M}$ is simulated, returning to the program index 3

$$(3, 2^1 3^1 5^0) \xrightarrow{(4,1)}_{\mathcal{P}} (4, 2^2 3^1 5^0) \xrightarrow{(5,2)}_{\mathcal{P}} (5, 2^1 3^2 5^0) \xrightarrow{(6,3)}_{\mathcal{P}} (6, 2^3 3^1 5^0) \xrightarrow{(3,4)}_{\mathcal{P}} (3, 2^1 3^1 5^1)$$

Overall, the configuration $(0, 1) = (0, 2^0 3^0 5^0)$ is non-terminating in $\mathcal{P}$, simulating non-termination of the configuration $(0, (a_0, b_0))$ in $\mathcal{M}$ as follows

$$(0, 2^0 3^0 5^0) \longrightarrow^3_{\mathcal{P}} (3, 2^1 3^1 5^0) \longrightarrow^4_{\mathcal{P}} (3, 2^1 3^1 5^1) \longrightarrow^4_{\mathcal{P}} (3, 2^1 3^1 5^2) \longrightarrow^4_{\mathcal{P}} (3, 2^1 3^1 5^3) \longrightarrow^4_{\mathcal{P}} \dots$$

Indeed, there is no upper bound on the counter value (cf. Lemma 10).

▶ Remark 15. One-counter machines can be understood as a variant of Conway's FRACTRAN language [7] with a relaxed program index transition rule, and restricted to the instruction set $(\frac{2}{1}, \frac{3}{2}, \frac{4}{3}, \frac{5}{4})$.

▶ Remark 16. The deliberate choice of counter multiplication by $\frac{d+1}{d}$ for $d \in \{1, 2, 3, 4\}$ has several benefits. First, instructions are of uniform shape. Therefore, simulation of and reasoning about such instructions requires less case analysis, also impacting the underlying mechanization. Second, counter modification is non-decreasing by definition. Third, for a counter value $c = k \cdot d$ multiplication by $\frac{d+1}{d}$ results in $c \cdot \frac{d+1}{d} = c + k$. Therefore, it can be simulated by rewriting a binary word $0^c 1 0^k$ to $0^{c+k} 1$. This can be performed iteratively (and uniformly) by shifting the symbol 1 to the right for every consecutive occurrence of 0^d in 0^c (cf. proof of Lemma 26).

4 Two-stack Machines

In this section, our goal is the simulation of one-counter machines in a *stack machine* model of computation without unbounded *symbol search*. Specifically, given a one-counter machine $\mathcal{P}$, we construct a two-stack machine $\mathcal{S}$ such that the configuration $(0, 1)$ is terminating in $\mathcal{P}$ iff there is a uniform bound on the number of configurations reachable in $\mathcal{S}$ from any configuration.

The main difficulty, similarly to the undecidability proof for Turing machine immortality [20], is to simulate symbol search (traverse data, searching for a particular symbol) using a uniformly bounded machine. Most problematic are unsuccessful searches that may traverse an arbitrary, i.e. not uniformly bounded, amount of data. The key idea [20, Part IV] (see also [24, 22]) is to implement unbounded symbol search by nested bounded symbol search.

In the present work, we supplement a high level explanation of the construction (cf. proof sketch of Lemma 26) with a comprehensive case analysis as a mechanized proof (in the Coq proof assistant). This approach, arguably worth striving for in general, has three advantages over existing work. First, the proof idea is not cluttered with mundane technical details, while the mechanized proof is highly precise. Second, a mechanized proof leaves little doubt regarding proof correctness and is open to scrutiny, as there is nothing left to imagination. Third, the Coq proof assistant tracks any non-constructive assumptions which may hide beneath technical details.

Let us specify the two-stack machine (Definition 17) model of computation, which we use to simulate one-counter machines. An instruction of such a machine may modify the current machine state, pop from, and push onto two stacks of binary symbols.

▶ **Definition 17** (Two-stack Machine ($\mathcal{S}$)). *A* two-stack machine $\mathcal{S}$ *is a list of* instructions *of shape* $A\vert p\vert B \to A'\vert q\vert B'$*, where* $A, B, A', B' \in \{0,1\}^*$ *are binary* words *and* $p, q \in \mathbb{S}$ *are* states*, where* $\mathbb{S}$ *is countably infinite.*

A configuration *of* $\mathcal{S}$ *is of shape* $A\vert p\vert B$ *where* $p \in \mathbb{S}$ *is the* current state*,* $A \in \{0,1\}^*$ *is the content of the* left stack *and* $B \in \{0,1\}^*$ *is the content of the* right stack.

The step relation *of* $\mathcal{S}$ *on configurations, written* $(\longrightarrow_{\mathcal{S}})$*, is given by*

- *if* $(A\vert p\vert B \to A'\vert q\vert B') \in \mathcal{S}$*, then for* $C, D \in \{0,1\}^*$ *we have* $CA\vert p\vert BD \longrightarrow_{\mathcal{S}} CA'\vert q\vert B'D$

The reachability relation *of* $\mathcal{S}$ *on configurations, written* $(\longrightarrow_{\mathcal{S}}^*)$*, is the reflexive, transitive closure of* $(\longrightarrow_{\mathcal{S}})$*.*

▶ Remark 18. A two-stack machine can be understood as a restricted semi-Thue system on the alphabet $\{0,1\} \cup \mathbb{S}$ in which each word contains exactly one symbol from $\mathbb{S}$. Such rewriting systems are employed in the setting of synchronous distributivity [1].

▶ Remark 19. To accommodate for arbitrary large machines, the state space $\mathbb{S}$ is not finite. However, the *effective* state space of any two-stack machine $\mathcal{S}$ is bounded by the finitely many states occurring in the instructions of $\mathcal{S}$.

The key undecidable property of two-stack machines, used in [10], is whether the number of distinct, reachable configurations from any configuration is *uniformly bounded* (Definition 20).

▶ **Definition 20** (Uniformly Bounded). *A two-stack machine* $\mathcal{S}$ *is* uniformly bounded *if there exists an* $n \in \mathbb{N}$ *such that for any configuration* $A\vert p\vert B$ *we have*

$$|\{A'\vert p'\vert B' \mid A\vert p\vert B \longrightarrow_{\mathcal{S}}^* A'\vert p'\vert B'\}| \leq n$$

Notably, uniform boundedness and *uniform termination* [32] (is every configuration chain finite?) are orthogonal notions, illustrated by the following Examples 21–22.

▶ **Example 21.** Consider $\mathcal{S} = [0|p|\epsilon \to \epsilon|p|1]$. From the configuration $0^m|p|\epsilon$, where $m \in \mathbb{N}$, reachable in $\mathcal{S}$ configurations are exactly $0^{m-i}|p|1^i$ for $i = 0 \dots m$. Therefore, there is no *uniform* bound on the number of reachable configurations. However, the length of every configuration chain in $\mathcal{S}$ is finite (bounded by one plus the length of the left stack). Overall, $\mathcal{S}$ is uniformly terminating but not uniformly bounded.

▶ **Example 22.** Consider $\mathcal{S} = [(0|p|\epsilon \to \epsilon|q|1), (\epsilon|q|1 \to 0|p|\epsilon)]$. The number of distinct, reachable in $\mathcal{S}$ configurations from any configuration is uniformly bounded by $n = 2$. However, there is an infinite configuration chain $0|p|\epsilon \longrightarrow_{\mathcal{S}} \epsilon|q|1 \longrightarrow_{\mathcal{S}} 0|p|\epsilon \longrightarrow_{\mathcal{S}} \epsilon|q|1 \longrightarrow_{\mathcal{S}} \dots$ Overall, $\mathcal{S}$ is uniformly bounded, but not uniformly terminating.

In literature [20, 24], counter machine termination is simulated using uniformly bounded Turing machines directly rather than by two-stack machines. This is reasonable when omitting technical details regarding the exact Turing machine construction. However, for verification in full detail, Turing machines are quite unwieldy, compared to two-stack machines. Unfortunately, we cannot rely on existing mechanized Turing machine programming techniques [15], as they establish functional properties, but are incapable to establish uniform boundedness.

4.1 Deterministic, Length-preserving Two-stack Machines

There are several properties of two-stack machines that are of importance in our construction in order to reuse existing work [10].

For *length-preserving* two-stack machines (Definition 23) the sum of lengths of the two stacks is invariant wrt. reachability. For each configuration, length-preservation bounds (albeit, not *uniformly*) the number of distinct, reachable configurations. Therefore, reachability is decidable for length-preserving two-stack machines.

▶ **Definition 23** (Length-preserving). *A two-stack machine $\mathcal{S}$ is* length-preserving *if for all instructions $(A|p|B \to A'|q|B') \in \mathcal{S}$ we have $0 < |A| + |B| = |A'| + |B'|$.*

▶ **Definition 24** (Deterministic). *A two-stack machine $\mathcal{S}$ is* deterministic *if for all configurations $A|p|B$, $A'|p'|B'$, and $A''|p''|B''$ such that $A|p|B \longrightarrow_{\mathcal{S}} A'|p'|B'$ and $A|p|B \longrightarrow_{\mathcal{S}} A''|p''|B''$ we have $A'|p'|B' = A''|p''|B''$.*

For example, two-stack machines in Examples 21–22 are deterministic and length-preserving.

The key undecidable problem, that in our argument assumes the role of Turing machine immortality of previous approaches [27, 10], is uniform boundedness of deterministic, length-preserving two-stack machines (Problem 25). A central insight of the present work is that using this problem as an intermediate step we neither require Turing reductions, König's lemma (cf. [27]), nor the fan theorem (cf. [10]). Additionally, this problem strikes a balance between ease to reduce to (from counter machine halting) and ease to reduce from (to semi-unification). This balance is essential for a mechanization of manageable size.

▶ **Problem 25** (Deterministic, Length-preserving Two-stack Machine Uniform Boundedness). Given a deterministic, length-preserving two-stack machine $\mathcal{S}$, is $\mathcal{S}$ uniformly bounded?

The original undecidability proof of semi-unification contains a hint [27, Proof of Corollary 6] that Turing machine immortality may be avoided in a comprehensive reduction. Accordingly, the following Lemma 26 captures the decisive step that avoids Turing machine immortality.

▶ **Lemma 26.** *One-counter machine* 1*-halting (Problem 11) many-one reduces to deterministic, length-preserving two-stack machine uniform boundedness (Problem 25).*

Proof Sketch. Let $\mathcal{P}$ be a one-counter machine. Similarly to [24, Theorem 7], we adapt Hooper's argument [20] for symbol search.

A naive simulation of $\mathcal{P}$ by a deterministic, length-preserving two-stack machine $\mathcal{S}$ is easy. A $\mathcal{P}$-configuration (i, c) is represented by the $\mathcal{S}$-configuration $1\imath i\imath 0^c 10^m$ for $m \in \mathbb{N}$, where 0^m is a large enough supply of zeroes. A $\mathcal{P}$-instruction (j, d) is simulated as follows. First, the $\mathcal{S}$-instruction $\epsilon\imath i_?\imath 0^d \longrightarrow_{\mathcal{S}} 0^d\imath i_?\imath\epsilon$ tests for divisibility by d, moving consecutive blocks of d zeroes from the right to the left stack. The divisibility test fails if the $\mathcal{S}$-instruction $\epsilon\imath i_?\imath 0^k 1 \longrightarrow_{\mathcal{S}} \epsilon\imath i_\#\imath 0^k 1$ where $0 < k < d$ can be applied. In this case, we move the zeroes back to the right stack, and via the $\mathcal{S}$-instruction $1\imath i_\#\imath\epsilon \longrightarrow_{\mathcal{S}} 1\imath i+1\imath\epsilon$ we reach the $\mathcal{S}$-configuration $1\imath i+1\imath 0^c 10^m$, representing the $\mathcal{P}$-configuration $(i+1, c)$. The divisibility test succeeds if the $\mathcal{S}$-instruction $\epsilon\imath i_?\imath 1 \longrightarrow_{\mathcal{S}} \epsilon\imath i_!\imath 1$ can be applied. In this case, multiplication by $\frac{d+1}{d}$ is simulated by shifting to the right the symbol 1 on the right stack for each block of consecutive d zeroes on the left stack (Remark 16). Finally, via the $\mathcal{S}$-instruction $1\imath i_!\imath\epsilon \longrightarrow_{\mathcal{S}} 1\imath j\imath\epsilon$, we arrive at the $\mathcal{S}$-configuration $1\imath j\imath 0^{\frac{c(d+1)}{d}} 10^{m-\frac{c}{d}}$, representing the $\mathcal{P}$-configuration $(j, c \cdot \frac{d+1}{d})$.

Inductively, we obtain $(i, c) \longrightarrow^*_{\mathcal{P}} (i', c')$ iff $A1\imath i\imath 0^c 10^{c'-c} B \longrightarrow^*_{\mathcal{S}} A1\imath i'\imath 0^{c'} 1B$ for all $A, B \in \{0,1\}^*$. Therefore, the configuration $(0, 1)$ is terminating in $\mathcal{P}$ iff configurations $A1\imath 0\imath 01B$ are uniformly bounded in $\mathcal{S}$ (the uniform bound is derived from the halting counter value).

Unfortunately, the naive construction fails in general. For example, termination of $(0, 1)$ in $\mathcal{P}$ does not necessarily uniformly bound the configurations $1\imath 0\imath 0^m$ where $m \in \mathbb{N}$ is arbitrary large. At fault is a failed symbol search (for the symbol 1 on the right stack) that needs to traverse an arbitrary amount of data. Observe that in the naive construction any search for the symbol 1 is expected to succeed. The ingenious idea by Hooper [20, Part IV] is to uniformly bound symbol search via nested simulation in a sufficiently large uniformly bounded configuration space (in our case, the space of configurations $A1\imath 0\imath 01B$ for $A, B \in \{0,1\}^*$). That is, to search for the symbol 1 on the right stack, start a nested simulation from the $\mathcal{P}$-configuration $(0, 1)$ inside the space of consecutive zeros on the right stack. Specifically, use $A\imath p\imath 0^{k+3} B \longrightarrow_{\mathcal{S}} AC1\imath 0\imath 01B$ to reset the program index p to 0 and retain the binary encoding of p in C of fixed length k. Let c be the counter value (arbitrarily large by Lemma 9.4) of the halting configuration in $\mathcal{P}$ from the $\mathcal{P}$-configuration $(0, 1)$. For the configuration $AC1\imath 0\imath 01B$ which represents the $\mathcal{P}$-configuration $(0, 1)$, there are three cases.

First, in case $B = 0^m 1D$, where $m < c - 1$ and $D \in \{0,1\}^*$, the number m of zeroes on the right stack is too small to accommodate for c. Eventually, the nested simulation is unable to simulate counter increase. In this situation, the initial search for the symbol 1 succeeds, and control is returned to the previous level.

Second, in case $B = 0^m$, where $m < c - 1$, the size of the right stack is too small to accommodate for c. Eventually, the nested simulation is unable to apply any instruction, and halts. In this situation, the initial search for the symbol 1 fails, respecting the uniform bound derived from c.

Third, in case $B = 0^{c-1} D$, where $D \in \{0,1\}^*$, there is enough space for the nested simulation to reach a halting state in $\mathcal{P}$ from the initial configuration $(0, 1)$, respecting the uniform bound derived from c. This renders the initial search for the symbol 1 immaterial, because the simulation already achieved its ultimate purpose.

In each case (using Lemma 9), a uniform bound on the number of configurations for the nested simulation can be derived from c. Each case may require further nested computation. However, the nesting depth is at most c because each nesting level uses space inside the consecutive zeroes that represent the counter value (bounded by c) on the previous level. ◀

▶ **Corollary 27.** *Deterministic, length-preserving two-stack machine uniform boundedness (Problem 25) is undecidable.*

▶ Remark 28. The exact analysis of the nested simulation in the proof of Lemma 26 requires a tremendous inductive proof with many corner cases. Arguably, it is unreasonable for a human without mechanical assistance to write it down in full detail (cf. Hooper's remark in Section 1). Additionally, it would require a comparable amount of effort for others to verify such a massive construction. This is why, in order to guarantee its correctness, a mechanized proof of Lemma 26 is adequate (cf. Section 6) to establish the result.

4.2 Confluent, Simple Two-stack Machines

In order to build upon existing work [10], we consider two-stack machines with instructions of *simple* (Definition 29, cf. [10, Definition 16]) shape. To further streamline the construction, we relax determinism of two-stack machines to confluence (Definition 32). Overall, confluent, simple two-stack machine uniform boundedness (Problem 34) is well-suited for reduction to a fragment of semi-unification (cf. Section 5.1).

▶ **Definition 29** (Simple). *A two-stack machine $\mathcal{S}$ is* simple *if for all instructions $(A\vert p\vert B \to A'\vert q\vert B') \in \mathcal{S}$ we have $1 = |A| + |B| = |A'| + |B'| = |A| + |A'| = |B| + |B'|$.*

▶ Remark 30. A deterministic, simple two-stack machine is just another way to present a deterministic Turing machine. The left and right stacks contain the respective tape content to the left and to the right from the current head. Reading and writing at the head while moving the head position is easily presented as simple instructions (cf. [10, Remark 19]).

▶ Remark 31. Turing machine immortality is reducible to uniform boundedness of deterministic, simple two-stack machines by a bounded Turing reduction [10, Theorem 2]. However, the argument uses the fan theorem, therefore, it is crucial for the present argument not to rely on this particular reduction.

▶ **Definition 32** (Confluent). *A two-stack machine $\mathcal{S}$ is* confluent *if for all configurations $A\vert p\vert B$, $A'\vert p'\vert B'$, and $A''\vert p''\vert B''$ such that $A\vert p\vert B \longrightarrow_{\mathcal{S}}^{*} A'\vert p'\vert B'$ and $A\vert p\vert B \longrightarrow_{\mathcal{S}}^{*} A''\vert p''\vert B''$ there exists a configuration $C\vert q\vert D$ such that $A'\vert p'\vert B' \longrightarrow_{\mathcal{S}}^{*} C\vert q\vert D$ and $A''\vert p''\vert B'' \longrightarrow_{\mathcal{S}}^{*} C\vert q\vert D$.*

Clearly, any deterministic two-stack machine is confluent (but not necessarily vice versa).

▶ **Lemma 33.** *If a two-stack machine $\mathcal{S}$ is deterministic, then $\mathcal{S}$ is confluent.*

Compared to deterministic machines, confluent machines are quite practical. For example, a confluent machine may (without additional bookkeeping) "try out" different configuration chains before choosing the preferable one (cf. proof sketch of Lemma 35).

▶ **Problem 34** (Confluent, Simple Two-stack Machine Uniform Boundedness). Given a confluent, simple two-stack machine $\mathcal{S}$, is $\mathcal{S}$ uniformly bounded?

By Lemma 33, the above Problem 34 subsumes deterministic, simple two-stack machine uniform boundedness [10, Problem 26]. This, in combination with the following Lemma 35, allows for adaptation of previous work[1] in Section 5.1.

[1] It is possible to carry out the construction in the original, deterministic scenario without adaptation. However, this is technically more challenging and provides no tangible benefit.

▶ **Lemma 35.** *Deterministic, length-preserving two-stack machine uniform boundedness (Problem 25) many-one reduces to confluent, simple two-stack machine uniform boundedness (Problem 34).*

Proof Sketch. Our objective is to shorten length-preserving instructions, while maintaining confluence. This is routine, storing local stack information in additional fresh states. For example, an instruction $00 \mid p \mid \epsilon \longrightarrow 11 \mid q \mid \epsilon$ can be replaced by the simple instructions $0 \mid p \mid \epsilon \longrightarrow \epsilon \mid p_1 \mid 0$, $0 \mid p_1 \mid \epsilon \longrightarrow \epsilon \mid p_2 \mid 0$, $\epsilon \mid p_2 \mid 0 \longrightarrow 1 \mid p_3 \mid \epsilon$, and $\epsilon \mid p_3 \mid 0 \longrightarrow 1 \mid q \mid \epsilon$, where p_1, p_2, p_3 are fresh states. This results in the configuration chain $00 \mid p \mid \epsilon \longrightarrow 0 \mid p_1 \mid 0 \longrightarrow \epsilon \mid p_2 \mid 00 \longrightarrow 1 \mid p_3 \mid 0 \longrightarrow 11 \mid q \mid \epsilon$ which simulates the instruction $00 \mid p \mid \epsilon \longrightarrow 11 \mid q \mid \epsilon$.

Notably, in the exemplified transformation it is difficult to maintain determinism. However, in order to maintain confluence it suffices to add reverse instructions from fresh states, i.e. $\epsilon \mid p_1 \mid 0 \longrightarrow 0 \mid p \mid \epsilon$, $\epsilon \mid p_2 \mid 0 \longrightarrow 0 \mid p_1 \mid \epsilon$, and $1 \mid p_3 \mid \epsilon \longrightarrow \epsilon \mid p_2 \mid 0$. Therefore, any failed attempt to read local stack information is reversible and computation is confluent. ◀

5 Semi-unification

Semi-unification (Problem 38) can be understood as combination of first-order unification (cf. valuation φ) and first-order matching (cf. valuations ψ). For the undecidability of semi-unification [26, Theorem 12], it suffices to restrict the syntax of the underlying terms (Definition 36) to variables together with a binary constructor ($\to$).

In this section, we recapitulate necessary definitions and properties of semi-unification from existing work [27, 10], in order to complete a constructive many-one reduction from Turing machine halting to semi-unification (Theorem 53).

▶ **Definition 36** (Terms ($\mathbb{T}$))**.** *Let α, β, γ range over a countably infinite set $\mathbb{V}$ of* variables*. The set of* terms $\mathbb{T}$*, ranged over by σ, τ, is given by the grammar $\sigma, \tau \in \mathbb{T} ::= \alpha \mid \sigma \to \tau$.*

▶ **Definition 37** (Valuation $(\varphi), (\psi)$)**.** *A* valuation $\varphi : \mathbb{V} \to \mathbb{T}$ *assigns terms to variables, and is tacitly lifted to terms.*

▶ **Problem 38** (Semi-unification [27, SUP], [10, Problem 3])**.**
Given a set $\mathcal{I} = \{\sigma_1 \leq \tau_1, \ldots, \sigma_n \leq \tau_n\}$ of *inequalities*, is there a valuation φ such that for each inequality $(\sigma \leq \tau) \in \mathcal{I}$ there exists a valuation $\psi : \mathbb{V} \to \mathbb{T}$ such that $\psi(\varphi(\sigma)) = \varphi(\tau)$?

▶ Remark 39. As given by Definition 37, the set of valuations is not countable. However, in any semi-unification instance $\mathcal{I}$ the number of inequalities (consisting of first-order terms) is finite. Therefore, restricting valuations to be finite maps (from the relevant variables) does not change the expressive power of semi-unification. As a result, semi-unification is recursively enumerable.

The following Examples 40 (resp. Example 41) illustrates a positive (resp. negative) instance of semi-unification.

▶ **Example 40.** Consider $\mathcal{I} = \{\alpha \leq \alpha \to \alpha,\ \alpha \leq \alpha \to \alpha \to \alpha\}$. The semi-unification instance $\mathcal{I}$ is solved by the valuation φ such that $\varphi(\alpha) = \alpha$.
For the inequality $\alpha \leq \alpha \to \alpha$ there exists a valuation ψ such that $\psi(\alpha) = \alpha \to \alpha$, and therefore $\psi(\varphi(\alpha)) = \varphi(\alpha \to \alpha)$. For the inequality $\alpha \leq \alpha \to \alpha \to \alpha$ there exists a valuation ψ such that $\psi(\alpha) = \alpha \to \alpha \to \alpha$, and therefore $\psi(\varphi(\alpha)) = \varphi(\alpha \to \alpha \to \alpha)$.

▶ **Example 41.** Consider $\mathcal{I} = \{\alpha \to \alpha \leq \alpha\}$. The semi-unification instance $\mathcal{I}$ is not solvable. Assume that there exist valuations φ and ψ such that $\psi(\varphi(\alpha \to \alpha)) = \varphi(\alpha)$. Therefore, the size of the syntax tree of $\varphi(\alpha)$ is twice the size of the syntax tree $\psi(\varphi(\alpha))$ which is not possible for (non-empty, finite) terms.

Unfortunately, semi-unification does not admit a decision procedure based on an occurs-check, which is a common approach to both first-order unification and first-order matching [3] (see also the redex contraction procedure [27, Section 2]). However, it is challenging to construct an unsolvable example of manageable size, for which the occurs-check fails.

Originally [27, Theorem 12], semi-unification is proven undecidable by Turing reduction from Turing machine immortality [20]. As intermediate problems, the argument relies on symmetric intercell Turing machine boundedness, path equation derivability, and termination of a redex contraction procedure that is custom-tailored for semi-unification. Additionally, the argument uses König's lemma and it is not obvious whether it can be presented constructively.

A modern approach [10, Theorem 4] simplifies the traditional argument. It still relies on a Turing reduction from Turing machine immortality, but uses only deterministic, simple two-stack machine uniform boundedness to show undecidability of a fragment of semi-unification. Additionally, it relies on the fan theorem, which is strictly weaker than König's lemma and is valid in Brouwer's intuitionism. The argument is partially mechanized (treating Turing machine immortality axiomatically) in Coq.

In the remainder of this section we briefly recapitulate and reuse the modern approach [10] in the more general case of confluent, simple two-stack machines. This allows us to avoid Turing machine immortality, Turing reductions, and the fan theorem in the overall argument.

5.1 Simple Semi-unification

In this section, we recapitulate the intermediate problem of *simple semi-unification* (Problem 44) [10, Problem 15], which connects stack machine computation and semi-unification. Intuitively, term variables represent machine states, simple constraints (Definition 42) represent local stack transformations, and the model relation (Definition 43) captures machine reachability via valuations.

▶ **Definition 42** (Simple Constraint [10, Definition 6]). *A* simple constraint *has the shape* $a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$*, where* $a, b \in \{0, 1\}$ *are symbols and* $\alpha, \beta \in \mathbb{V}$ *are variables.*

▶ **Definition 43** (Model Relation [10, Definition 9]). *A valuation triple* $(\varphi, \psi_0, \psi_1)$ models *a simple constraint* $a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$*, written* $(\varphi, \psi_0, \psi_1) \models a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$*, if one of the following conditions holds*

- $b = 0$ *and* $\psi_a(\varphi(\alpha)) \to \tau = \varphi(\beta)$ *for some term* $\tau \in \mathbb{T}$
- $b = 1$ *and* $\sigma \to \psi_a(\varphi(\alpha)) = \varphi(\beta)$ *for some term* $\sigma \in \mathbb{T}$

▶ **Problem 44** (Simple Semi-unification [10, Problem 15]). Given a finite set $\mathcal{C}$ of simple constraints, do there exist valuations $\varphi, \psi_0, \psi_1 : \mathbb{V} \to \mathbb{T}$ such that for each simple constraint $(a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b) \in \mathcal{C}$ we have $(\varphi, \psi_0, \psi_1) \models a \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid b$?

The following Example 45 illustrates a model of a set of simple constraints, i.e. a solvable instance of simple semi-unification.

▶ **Example 45.** Consider $\mathcal{C} = \{0 \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid 1, 1 \shortmid \alpha \shortmid \epsilon \doteq \epsilon \shortmid \beta \shortmid 0\}$. A possible valuation triple $(\varphi, \psi_0, \psi_1)$ which models each simple constraint in $\mathcal{C}$ is such that $\varphi(\alpha) = \alpha$, $\varphi(\beta) = \beta_1 \to \beta_2$, $\psi_0(\alpha) = \beta_2$, $\psi_1(\alpha) = \beta_1$. Indeed, we have $\beta_1 \to \psi_0(\varphi(\alpha)) = \beta_1 \to \beta_2 = \varphi(\beta)$ and $\psi_1(\varphi(\alpha)) \to \beta_2 = \beta_1 \to \beta_2 = \varphi(\beta)$.

The pivotal step in previous work [10] is a many-one reduction from deterministic, simple two-stack machine uniform boundedness to simple semi-unification. This result readily generalizes to the confluent case (Lemma 46).

▶ **Lemma 46.** *Confluent, simple two-stack machine uniform boundedness (Problem 34) many-one reduces to simple semi-unification (Problem 44).*

Proof Sketch. We follow exactly the argument structure of [10, Section 4]. Tacitly inject machine states into variables, i.e. let $\mathbb{S} \subseteq \mathbb{V}$. Given a confluent, simple two-stack machine $\mathcal{S}$, construct the set of simple constraints

$$\mathcal{C} = \{a \shortmid p \shortmid \epsilon \doteq \epsilon \shortmid q \shortmid b \mid (a \shortmid p \shortmid \epsilon \to \epsilon \shortmid q \shortmid b) \in \mathcal{S} \text{ or } (\epsilon \shortmid q \shortmid b \to a \shortmid p \shortmid \epsilon) \in \mathcal{S}\}$$

If $\mathcal{S}$ is uniformly bounded, then the exact construction of $(\varphi, \psi_0, \psi_1)$ [10, Definition 42] yields a model of each simple constraint in $\mathcal{C}$.

Conversely, if $(\varphi, \psi_0, \psi_1)$ models each constraint in $\mathcal{C}$, then by the exact argument of [10, Lemma 48] the maximal depth of the syntax trees in the range of φ induces a uniform bound on the number of configurations reachable from any configuration in $\mathcal{S}$. ◀

▶ Remark 47. Although the original construction [10, Lemma 45 and Lemma 48] requires determinism, only confluence is used in the actual proofs [10, Lemma 30, Lemma 39]. While tedious to verify by hand, the available mechanization allows for simple replacement of determinism by confluence, while the proof assistant guarantees correctness of any related details. This highlights the effectiveness of proof assistants to accommodate for changes in a complex argument, reevaluating its overall correctness.

5.2 Right-uniform, Two-inequality Semi-unification

In this section, we consider a restriction of semi-unification to only two inequalities with identical right-hand sides (Problem 48). Such a restriction is a convenient byproduct of the reduction from simple semi-unification, and may simplify existing undecidability proofs that rely on the undecidability of semi-unification.

▶ **Problem 48** (Right-uniform, Two-inequality Semi-unification)**.** Given two inequalities $\sigma_0 \leq \tau$ and $\sigma_1 \leq \tau$ with identical right-hand sides, do there exist valuations φ, ψ_0, ψ_1 such that $\psi_0(\varphi(\sigma_0)) = \varphi(\tau)$ and $\psi_1(\varphi(\sigma_1)) = \varphi(\tau)$?

▶ Remark 49. Simply put, the above Problem 48 can be stated as follows: Given three terms σ_0, σ_1, τ, are there valuations φ, ψ_0, ψ_1 such that $\psi_0(\varphi(\sigma_0)) = \varphi(\tau) = \psi_1(\varphi(\sigma_1))$?

It is an easy exercise to reduce simple semi-unification to (non right-uniform) two-inequality semi-unification [10, Theorem 1]. We slightly adjust the existing construction to produce right-uniform inequalities.

▶ **Lemma 50.** *Simple semi-unification (Problem 44) many-one reduces to right-uniform, two-inequality semi-unification (Problem 48).*

Proof. Given constraints $\mathcal{C} = \{a_i \shortmid \alpha_i \shortmid \epsilon \doteq \epsilon \shortmid \beta_i \shortmid b_i \mid i = 1 \dots n\}$, define $\tau = \beta_1 \to \dots \to \beta_n$. Let γ_i be fresh variables for $i = 1 \dots n$ and define $\sigma^j = \sigma^j_1 \to \dots \to \sigma^j_n$ for $j \in \{0, 1\}$ where

$$\sigma^j_i = \alpha_i \to \gamma_i \text{ if } a_i = j \text{ and } b_i = 0 \qquad \sigma^j_i = \gamma_i \to \alpha_i \text{ if } a_i = j \text{ and } b_i = 1 \qquad \sigma^j_i = \gamma_i \text{ else}$$

We show that $\mathcal{C}$ is solvable iff the right-uniform inequalities $\sigma^0 \leq \tau$ and $\sigma^1 \leq \tau$ are solvable.

First, assume that the valuation triple $(\varphi, \psi_0, \psi_1)$ models each simple constraint in $\mathcal{C}$. Wlog. $\varphi(\gamma_i) = \psi_0(\gamma_i) = \psi_1(\gamma_i) = \gamma_i$ for $i = 1 \dots n$. By routine case analysis, we may adjust $\psi_0(\gamma_i)$ and $\psi_1(\gamma_i)$ for $i = 1 \dots n$ to obtain valuations ψ'_0 and ψ'_1 such that $\psi'_0(\varphi(\sigma^0)) = \varphi(\tau) = \psi'_1(\varphi(\sigma^1))$.

Second, any solution φ, ψ_0, ψ_1 of $\sigma^0 \leq \tau$ and $\sigma^1 \leq \tau$ also models each constraint in $\mathcal{C}$. ◀

▶ **Corollary 51.** *Simple semi-unification (Problem 44) many-one reduces to semi-unification (Problem 38).*

The following Example 52 illustrates the proof of Lemma 50 on the basis of Example 45.

▶ **Example 52.** Consider $\mathcal{C} = \{0 \mid \alpha \mid \epsilon \doteq \epsilon \mid \beta \mid 1, 1 \mid \alpha \mid \epsilon \doteq \epsilon \mid \beta \mid 0\}$ from Example 45. Define the terms $\sigma^0 = (\gamma_1 \to \alpha) \to \gamma_2$, $\sigma^1 = \gamma_1 \to (\alpha \to \gamma_2)$, and $\tau = \beta \to \beta$. The valuation triple $(\varphi, \psi_0, \psi_1)$ from Example 45 is extended to γ_1, γ_2 to obtain a solution $\varphi, \psi_0', \psi_1'$ of the right-uniform inequalities $\sigma^0 \leq \tau$ and $\sigma^1 \leq \tau$ as follows: $\psi_0'(\gamma_1) = \beta_1$, $\psi_0'(\gamma_2) = \beta_1 \to \beta_2$, $\psi_1'(\gamma_1) = \beta_1 \to \beta_2$, and $\psi_1'(\gamma_2) = \beta_2$. We have $(\beta_1 \to \beta_2) \to (\beta_1 \to \beta_2) = \varphi(\tau) = \psi_0'(\varphi(\sigma^0)) = (\psi_1'(\varphi(\sigma^1))$.

5.3 Main Result

Finally, we compose the previously described reductions into a comprehensive, constructive many-one reduction from Turing machine halting to semi-unification (Theorem 53). This constitutes the main result of the present work.

▶ **Theorem 53.** *Turing machine halting constructively many-one reduces to semi-unification (Problem 38).*

Proof. By composition of Lemmas 3, 12, 26, 35, 46, and Corollary 51. Constructivity of the argument is witnessed by an axiom-free mechanization (cf. Section 6) using the Coq proof assistant. ◀

Since semi-unification is recursively enumerable (Remark 39), it is many-one complete (in the sense of [38, Chapter 7.2]).

▶ **Corollary 54.** *Semi-unification (Problem 38) is many-one complete.*

6 Mechanization

This section provides an overview over the constructive mechanization, using the Coq proof assistant [8], of the many-one reduction from Turing machine halting to semi-unification. The mechanization relies on, and is integrated into the growing Coq Library of Undecidability Proofs [17]. The reduction is axiom-free and spans approximately 20000 lines of code, of which 3500 is contributed by the present work.

At the core of the library is the following mechanized notion of many-one reducibility[2]

```
Definition reduction {X Y} (f: X -> Y) (P: X -> Prop) (Q: Y -> Prop) :=
  forall x, P x <-> Q (f x).

Definition reduces {X Y} (P: X -> Prop) (Q: Y -> Prop) :=
  exists f: X -> Y, reduction f P Q.

Notation "P ⪯ Q" := (reduces P Q).
```

In the above, a predicate `P` over the domain `X` many-one reduces to a predicate `Q` over the domain `Y`, denoted `P ⪯ Q`, if there exists a function `f: X -> Y` such that for all `x` in the domain `X` we have `P x` iff `Q (f x)`. In axiom-free Coq any such function `f: X -> Y` is computable, a necessity oftentimes handled with less rigor in traditional (non-mechanized) proofs. Additionally, in axiom-free Coq, a proof of `P x <-> Q (f x)` cannot rely on principles such as functional extensionality, the fan theorem, or the law of excluded middle. Notably, our main Theorem 53 is mechanized in this setting.

[2] cf. `theories/Synthetic/Definitions.v`

The key contribution of the present work is consolidated as part of the following conjunction of many-one reductions[3]

```
Theorem HaltTM_1_chain_SemiU :
  HaltTM 1 ⪯ iPCPb /\
  iPCPb ⪯ BSM_HALTING /\
  BSM_HALTING ⪯ MM2_HALTING /\
  MM2_HALTING ⪯ CM1_HALT /\
  CM1_HALT ⪯ SMNdl_UB /\
  SMNdl_UB ⪯ CSSM_UB /\
  CSSM_UB ⪯ SSemiU /\
  SSemiU ⪯ RU2SemiU /\
  RU2SemiU ⪯ SemiU.
```

The individual problems in the above chain of many-one reductions are as follows.

- `HaltTM 1` is one-tape Turing machine halting, and native to the library as the initial undecidable problem, building upon prior work [15, 2] in computability theory
- `iPCPb` is indexed, binary Post correspondence problem, mechanized in [13]
- `BSM_HALTING` is binary stack machine halting, mechanized in [16]
- `MM2_HALTING` is two-counter machine halting (Problem 2), mechanized in [16]
- `CM1_HALT` is one-counter machine 1-halting (Problem 11)
- `SMNdl_UB` is uniform boundedness of deterministic, length-preserving two-stack machines (Problem 25)
- `CSSM_UB` is uniform boundedness of confluent, simple stack machines (Problem 34)
- `SSemiU` is simple semi-unification (Problem 44), mechanized in [10]
- `RU2SemiU` is right-uniform, two-inequality semi-unification (Problem 48)
- `SemiU` is semi-unification (Problem 38), mechanized in [10]

Correctness of the argument is witnessed by the verification of `Theorem HaltTM_1_chain_SemiU` in axiom-free Coq, for which constructivity is certified using the `Print Assumptions` command [9].

By transitivity of many-one reducibility we obtain `Theorem reduction : HaltTM 1 ⪯ SemiU`[4].

As a result, the statement `HaltTM 1 ⪯ SemiU` faithfully mechanizes our overall formal goal of a constructive many-one reduction from Turing machine halting to semi-unification. In fact, the particular many-one reduction function could be extracted from the proof of `HaltTM 1 ⪯ SemiU` as a λ-term (in the call-by-value λ-calculus model of computation) using existing techniques [14].

The mechanization of `CM1_HALT`, `SMNdl_UB`, and `CSSM_UB` together with corresponding many-one reductions are contributed to the library as part of the present work. The proof of `CSSM_UB ⪯ SSemiU` is an almost verbatim copy of the corresponding `DSSM_UB ⪯ SSemiU` previous result [10, Section 5], in which determinism is replaced by confluence.

Notably, `CM1_HALT ⪯ SMNdl_UB` relies on a variant of Hooper's argument [20]. The particular mechanization details span approximately 3500 lines of code, two thirds of which verify the construction in the proof of Lemma 26. Since the proof structure for uniform bound verification is mostly by extensive case analysis and basic arithmetic, the mechanization benefits greatly from proof automation, i.e. Coq's `lia`, `nia`, and `eauto` tactics [9]. To the best of the author's knowledge, the provided mechanization is the first that implements (a variant of) Hooper's argument.

[3] cf. `theories/SemiUnification/Reductions/HaltTM_1_chain_SemiU.v`
[4] cf. `theories/SemiUnification/Reductions/HaltTM_1_to_SemiU.v`

7 Conclusion

This work gives a constructive many-one reduction from Turing machine halting to semi-unification (Theorem 53). It improves upon existing work [27, 10] in the following aspects.

First, previous approaches use Turing reductions to establish undecidability. Therefore, such arguments are unable to establish many-one completeness of semi-unification shown in the present work (Corollary 54).

Second, previous work relies on the undecidability of Turing machine immortality, which is not recursively enumerable, and obscures the overall picture. In the present work, we avoid Turing machine immortality by adapting Hooper's ingenious construction [20] (also adapted in [24]) to uniform boundedness (Lemma 26). Notably, Hooper's construction is such that the resulting machine is either both mortal and uniformly bounded, or neither [27, Proof of Corollary 6].

Third, correctness of the reduction function is proven constructively (in the sense of axiom-free Coq), whereas previous work uses the principle of excluded middle, König's lemma [27], or the fan theorem [10]. As a result, anti-classical theories, such as synthetic computability theory [5], may accommodate the presented results.

Fourth, computability of the many-one reduction function from Turing machines to semi-unification instances is established rigorously by its mechanization in the Coq proof assistant. Traditionally, this aspect is treated less formally.

Fifth, the reduction is mechanized as part of the Coq Library of Undecidability Proofs [17], building upon existing infrastructure. Arguably, a comprehensive mechanization is the *only* feasible approach to verify a reduction from Turing machine halting to semi-unification with high confidence in full detail. The provided mechanization integrates existing work [10] into the Coq Library of Undecidability Proofs, and contributes a (first of its kind) mechanized variant of Hooper's construction to avoid symbol search.

While this document provides a high-level overview over the overall argument, surveyability (both local and global in the sense of [4]) is established mechanically. Local surveyability is supported by the modular nature of the Coq Library of Undecidability Proofs. That is, the mechanization of each reduction step can be understood and verified independently. Global surveyability is supported by `Theorem HaltTM_1_chain_SemiU` and the statement `HaltTM 1` $\preceq$ `SemiU` (cf. Section 6). That is, the individually mechanized reduction steps do compose transitively.

As ultimate proof of feasibility, the provided mechanization shows the maturity of the Coq proof assistant for mechanical verification of technically challenging proofs. Admittedly, neither Hooper's exact immortality construction [20] nor the exact semi-unification construction by Kfoury, Tiuryn, and Urzyczyn [27] was mechanized. Rather, the overall argument was revised to be mechanization-friendly. For example, the simplicity and uniformity of one-counter machines as an intermediate model of computation serves exactly this purpose.

Already, building upon the present work, there is a novel mechanization showing the undecidability of System F typability and type checking [11]. In addition, we envision further mechanized results. For one, the undecidability of synchronous distributivity [1] relies on uniform boundedness of semi-Thue systems that can be described as the presented (and mechanized) two-stack machines. Further, since the underlying construction is already implemented, it is reasonable to mechanize a many-one reduction from Turing machine halting to Turing machine immortality. This would pave the way for further mechanized results. For example, the undecidability of the finite variant property [6, Section 7] as well as several tiling problems [23] rely on (variants of) Turing machine immortality.

References

1 Siva Anantharaman, Serdar Erbatur, Christopher Lynch, Paliath Narendran, and Michaël Rusinowitch. Unification modulo synchronous distributivity. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 14–29. Springer, 2012. doi:10.1007/978-3-642-31365-3_4.

2 Andrea Asperti and Wilmer Ricciotti. A formalization of multi-tape Turing machines. *Theor. Comput. Sci.*, 603:23–42, 2015. doi:10.1016/j.tcs.2015.07.013.

3 Franz Baader, Wayne Snyder, Paliath Narendran, Manfred Schmidt-Schauß, and Klaus U. Schulz. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001. doi:10.1016/b978-044450813-3/50010-2.

4 O. Bradley Bassler. The surveyability of mathematical proof: A historical perspective. *Synth.*, 148(1):99–133, 2006. doi:10.1007/s11229-004-6221-7.

5 Andrej Bauer. First steps in synthetic computability theory. In Martín Hötzel Escardó, Achim Jung, and Michael W. Mislove, editors, *Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2005, Birmingham, UK, May 18-21, 2005*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 5–31. Elsevier, 2005. doi:10.1016/j.entcs.2005.11.049.

6 Christopher Bouchard, Kimberly A. Gero, Christopher Lynch, and Paliath Narendran. On forward closure and the finite variant property. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems – 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2013. doi:10.1007/978-3-642-40885-4_23.

7 John H. Conway. Fractran: A simple universal programming language for arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.

8 The Coq Proof Assistant. https://coq.inria.fr/. Accessed: 2020-10-08.

9 The Coq Proof Assistant Reference Manual. https://coq.inria.fr/distrib/current/refman/. Accessed: 2020-07-30.

10 Andrej Dudenhefner. Undecidability of semi-unification on a napkin. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.9.

11 Andrej Dudenhefner. The undecidability of system F typability and type checking for reductionists. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–10. IEEE, 2021. doi:10.1109/LICS52264.2021.9470520.

12 Manuel Fähndrich, Jakob Rehof, and Manuvir Das. Scalable context-sensitive flow analysis using instantiation constraints. In Monica S. Lam, editor, *Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Vancouver, Britith Columbia, Canada, June 18-21, 2000*, pages 253–263. ACM, 2000. doi:10.1145/349299.349332.

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

14 Yannick Forster and Fabian Kunze. A certifying extraction with time bounds from Coq to call-by-value lambda calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITP.2019.17.

15 Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of Turing machines in Coq. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 114–128. ACM, 2020. doi:10.1145/3372885.3373816.

16 Yannick Forster and Dominique Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and Minsky machines. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 104–117. ACM, 2019. doi:10.1145/3293880.3294096.

17 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq library of undecidable problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*, 2020. URL: https://github.com/uds-psl/coq-library-undecidability.

18 Jean-Yves Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse d'État, Éditeur inconnu, 1972.

19 Fritz Henglein. Type inference with polymorphic recursion. *ACM Trans. Program. Lang. Syst.*, 15(2):253–289, 1993. doi:10.1145/169701.169692.

20 Philip K. Hooper. The undecidability of the Turing machine immortality problem. *J. Symb. Log.*, 31(2):219–234, 1966. doi:10.2307/2269811.

21 Said Jahama and Assaf J. Kfoury. A general theory of semi-unification. Technical report, Boston University Computer Science Department, 1993.

22 Emmanuel Jeandel. On immortal configurations in Turing machines. In S. Barry Cooper, Anuj Dawar, and Benedikt Löwe, editors, *How the World Computes – Turing Centenary Conference and 8th Conference on Computability in Europe, CiE 2012, Cambridge, UK, June 18-23, 2012. Proceedings*, volume 7318 of *Lecture Notes in Computer Science*, pages 334–343. Springer, 2012. doi:10.1007/978-3-642-30870-3_34.

23 Jarkko Kari. The tiling problem revisited (extended abstract). In Jérôme Olivier Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*, pages 72–79. Springer, 2007. doi:10.1007/978-3-540-74593-8_6.

24 Jarkko Kari and Nicolas Ollinger. Periodicity and immortality in reversible computing. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2008. doi:10.1007/978-3-540-85238-4_34.

25 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The undecidability of the semi-unification problem (preliminary report). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 468–476. ACM, 1990. doi:10.1145/100216.100279.

26 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. Type reconstruction in the presence of polymorphic recursion. *ACM Trans. Program. Lang. Syst.*, 15(2):290–311, 1993. doi:10.1145/169701.169687.

27 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The undecidability of the semi-unification problem. *Inf. Comput.*, 102(1):83–101, 1993. doi:10.1006/inco.1993.1003.

28 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *J. ACM*, 41(2):368–398, 1994. doi:10.1145/174652.174659.

29 Dominique Larchey-Wendling and Yannick Forster. Hilbert's tenth problem in Coq. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.27.

30 Hans Leiß. Polymorphic recursion and semi-unification. In Egon Börger, Hans Kleine Büning, and Michael M. Richter, editors, *CSL '89, 3rd Workshop on Computer Science Logic, Kaiserslautern, Germany, October 2-6, 1989, Proceedings*, volume 440 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 1989. doi:10.1007/3-540-52753-2_41.

31 Hans Leiß and Fritz Henglein. A decidable case of the semi-unification problem. In Andrzej Tarlecki, editor, *Mathematical Foundations of Computer Science 1991, 16th International Symposium, MFCS'91, Kazimierz Dolny, Poland, September 9-13, 1991, Proceedings*, volume 520 of *Lecture Notes in Computer Science*, pages 318–327. Springer, 1991. doi:10.1007/3-540-54345-7_75.

32 Yuri V. Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 523–531. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561469.

33 M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

34 Alan Mycroft. Polymorphic type schemes and recursive definitions. In Manfred Paul and Bernard Robinet, editors, *International Symposium on Programming, 6th Colloquium, Toulouse, France, April 17-19, 1984, Proceedings*, volume 167 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 1984. doi:10.1007/3-540-12925-1_41.

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

36 Paul Walton Purdom. Detecting looping simplifications. In Pierre Lescanne, editor, *Rewriting Techniques and Applications, 2nd International Conference, RTA-87, Bordeaux, France, May 25-27, 1987, Proceedings*, volume 256 of *Lecture Notes in Computer Science*, pages 54–61. Springer, 1987. doi:10.1007/3-540-17220-3_5.

37 John C. Reynolds. Towards a theory of type structure. In Bernard Robinet, editor, *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974. doi:10.1007/3-540-06859-7_148.

38 Hartley Rogers. *Theory of Recursive Functions and Effective Computability (Reprint from 1967)*. MIT Press, 1987. URL: http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=3182.

39 Joe B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Log.*, 98(1-3):111–156, 1999. doi:10.1016/S0168-0072(98)00047-5.

40 Arkadiusz Wojna. Counter machines. *Inf. Process. Lett.*, 71(5-6):193–197, 1999. doi:10.1016/S0020-0190(99)00116-7.