HILBERT'S TENTH PROBLEM IN COQ [*]

DOMINIQUE LARCHEY-WENDLING AND YANNICK FORSTER

Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France
e-mail address: dominique.larchey-wendling@loria.fr

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
e-mail address: forster@ps.uni-saarland.de

Abstract. We formalise the undecidability of solvability of Diophantine equations, i.e. polynomial equations over natural numbers, in Coq's constructive type theory. To do so, we give the first full mechanisation of the Davis-Putnam-Robinson-Matiyasevich theorem, stating that every recursively enumerable problem – in our case by a Minsky machine – is Diophantine. We obtain an elegant and comprehensible proof by using a synthetic approach to computability and by introducing Conway's FRACTRAN language as intermediate layer. Additionally, we prove the reverse direction and show that every Diophantine relation is recognisable by μ-recursive functions and give a certified compiler from μ-recursive functions to Minsky machines.

1. Introduction

Hilbert's tenth problem (H10) was posed by David Hilbert in 1900 as part of his famous 23 problems [22] and asked for the "determination of the solvability of a Diophantine equation." A Diophantine equation[1] is a polynomial equation over natural numbers (or, equivalently, integers) with constant exponents, e.g. $x^2 + 3z = yz + 2$. When Hilbert asked for "determination," he meant, in modern terms, a decision procedure, but computability theory was yet several decades short of being developed.

The first undecidable problems found by Church, Post and Turing were either native to mathematical logic or dependent on a fixed model of computation. H10, to the contrary, can be stated to every mathematician and its formulation is independent from a model of computation. Emil Post stated in 1944 that H10 "begs for an unsolvability proof" [34]. From a computational perspective, it is clear that H10 is recursively enumerable (or *recognisable*), meaning there is an algorithm that halts on a Diophantine equation iff it is solvable.

Post's student Martin Davis conjectured that even the converse is true, i.e. that every recognisable set is also Diophantine. More precisely, he conjectured that if $A \subseteq \mathbb{N}^k$ is recognisable then $(a_1, \ldots, a_k) \in A \leftrightarrow \exists x_1 \ldots x_n, P(a_1, \ldots, a_k, x_1, \ldots, x_n) = 0$ holds for some

Key words and phrases: Hilbert's tenth problem, Diophantine equations, undecidability, computability theory, reduction, Minsky machines, Fractran, Coq, type theory.

[*] extended version of [25].

[1] Named after the Greek mathematician Diophantus of Alexandria, who started the study of polynomial equations in the third century.

polynomial P in $k+n$ variables. He soon improved on a result by Gödel [20] and gave a proof of his conjecture, however requiring up to one bounded universal quantification [5]: $(a_1, \dots, a_k) \in A \leftrightarrow \exists z, \forall y < z, \exists x_1 \dots x_n, P(a_1, \dots, a_k, x_1, \dots, x_n, y, z) = 0$. Davis and Putnam [7] further improved on this, and showed that, provided a certain number-theoretic assumption holds, every recognisable set is *exponentially* Diophantine, meaning variables are also allowed to appear in exponents. Julia Robinson then in 1961 modified the original proof to circumvent the need for the assumption, resulting in the DPR theorem [8], namely that every recognisable set is exponentially Diophantine. Due to another result from Robinson [35], the gap now only consisted of proving that there is a Diophantine equation exhibiting exponential growth. In 1970, Yuri Matiyasevich showed that the Fibonacci sequence grows exponentially while being Diophantine, closing the gap and finishing the proof of the theorem nowadays called *DPRM theorem*, ultimately establishing that exponentiation is Diophantine itself [27] (known as "Matiyasevich's theorem").

Even the most modern and simpler proofs of the DPRM theorem still require many preliminaries and complicated number-theoretic ideas, for an overview see [30]. We formalise one such proof as part of our ongoing work on a library of undecidable problems [15, 18] in the proof assistant Coq [40]. Since H10 is widely used as a seed [9, 21] for showing the undecidability of problems using *many-one reductions*, this will open further ways of extending the library. Given that our library already contains a formalisation of Minsky machines [16], we follow the approach of Jones and Matijasevič [23], who use register machines, being very well-suited since they already work on numbers. They encode full computations of register machines as Diophantine equations in one single, monolithic step. To make the proof more tractable for both mechanisation and explanation, we factor out an intermediate language, John Conway's FRACTRAN [4], which can simulate Minsky machines.

We first introduce three characterisations of Diophantine equations over natural numbers, namely *Diophantine logic* DIO_FORM (allowing to connect basic Diophantine equations with conjunction, disjunction and existential quantification), *elementary Diophantine constraints* DIO_ELEM (a finite set of constraints on variables, oftentimes used for reductions [9, 21]) and *single Diophantine equations* DIO_SINGLE, including parameters, as described above. H10 then asks about the solvability of single Diophantine equations with no parameters.

Technically, the reduction chain to establish the unsolvability of H10 starts at the halting problem for single-tape Turing machines Halt, reduced to the Post correspondence problem PCP in [10]. In previous work [16] we have reduced PCP to a specialised halting problem for Minsky machines, which we use here in a slightly generalised form as MM. We then reduce Minsky machine halting to FRACTRAN termination. FRACTRAN is very natural to describe using polynomials, and the encoding does not rely on any complicated construction. The technical difficulty then only lies in the Diophantine encoding of the reflexive-transitive closure of a relation which follows from the direct elimination of bounded universal quantification, given that the proof in [28] involves no detour via models of computation. In total, we obtain the following chain of reductions,[2] establishing the undecidability of H10 and it's many-one interreducibility with several decision problems:

$$\mathsf{Halt} \preceq \mathsf{PCP} \preceq \mathsf{MM} \preceq \mathsf{FRACTRAN} \preceq \mathsf{DIO\_FORM} \preceq \mathsf{DIO\_ELEM} \preceq \mathsf{DIO\_SINGLE} \preceq \mathsf{H10} \preceq \mu\text{-}\mathsf{rec} \preceq \mathsf{MM}$$

where Fig. 1 lists high-level descriptions of these problems. Furthermore, we prove that $\mathsf{H10} \preceq \mathsf{H10}_{\mathbb{Z}}$ via Lagrange's theorem. In the present paper, we focus on explaining this

[2] A *many-one reduction* $P \preceq Q$, later defined formally in this section, is a computable function from problem P to problem Q that maps instances of P into instances of Q, pereserving both validity and invalidity.

PCP: Post correspondence problem, see e.g. [16]. (matching)
MM: Given $n : \mathbb{N}$, a Minsky machine $P : \mathbb{L}\,\mathsf{I}_n$ with n registers, and $\vec{v} : \mathbb{N}^n$, does $(1, P)$ terminate from input state $(1, \vec{v})$? (termination)
FRACTRAN: Given a regular FRACTRAN program $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ and an input state s, does Q terminate from input state x? (termination)
DIO_FORM: Given a Diophantine logic formula $A : \mathbb{D}_{\text{form}}$ and a valuation $\nu : \mathsf{V} \to \mathbb{N}$, does $[\![A]\!]_\nu$ hold? (satisfaction)
DIO_ELEM: Given a list $l : \mathbb{L}\,\mathbb{D}_{\text{cstr}}$ of elementary Diophantine constraints and a valuation $\nu : \mathsf{V} \to \mathbb{N}$, does there exist $\varphi : \mathsf{U} \to \mathbb{N}$ such that $\forall c \in l$, $[\![c]\!]_\nu^\varphi$? (simultaneous satisfiability)
DIO_SINGLE: Given a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathbb{N})$ and a valuation $\nu : \mathbb{N} \to \mathbb{N}$, does there exist $\varphi : \mathbb{N} \to \mathbb{N}$ s.t. $[\![p]\!]_\nu^\varphi = [\![q]\!]_\nu^\varphi$? (solvability)
H10: Given a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_n, \mathbb{F}_0)$ (over $\mathbb{N}$ with possibly n variables but no parameters), does it have a solution in $\mathbb{N}$? (solvability)
μ-rec: Given $n : \mathbb{N}$, an n-ary μ-recursive function $f : \mathcal{A}_n$, and $\vec{v} : \mathbb{N}^n$, does $\vec{v}$ belong to the domain of f? (termination)
H10$_\mathbb{Z}$: Given a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_n, \mathbb{F}_0)$ (over $\mathbb{Z}$ with possibly n variables but no parameters), does it have a solution in $\mathbb{Z}$? (solvability)

Figure 1: Summary description of some decision problems.

factorisation of the proof and give some details for the different stages. While we contribute Coq mechanisations of Matiyasevich's theorem and the elimination of bounded universal quantification, we treat them mainly as black-boxes and only elaborate on their challenging formalisation rather than the proofs themselves (see Section 2.3).

To the best of our knowledge, we are the first to give a *full verification* of the DPRM theorem and the undecidability of Hilbert's tenth problem in a proof assistant. We base the notion of recognisability in the DPRM theorem on Minsky machines.

When giving undecidability proofs via many-one reductions, it is critical to show that all reduction functions are actually computable. We could in theory verify the computability of all functions involved using an explicit model of computation. In pen-and-paper proofs, this approach is however almost never used, because implementing high-level mathematical transformations as provably correct low-level programs is a daunting task. Instead, we rely on a synthetic approach [10, 11, 16] based on the computability of all functions definable in Coq's constructive type theory, which is closer to the practice of pen-and-paper proofs. In this approach, a problem P is considered undecidable if there is a reduction from an obviously undecidable problem, e.g. Halt $\preceq P$.

The axiom-free Coq formalisation of all the results in this paper is available online [17] and the main lemmas and theorems in the pdf version of the paper are hyper-linked with the html version of the source code at `https://github.com/uds-psl/coq-library-undecidability/tree/H10-LMCS-v1.1`. Starting from our already existing library which included most of the Minsky machine code [16], the additional code for proving the undecidability of H10 and the DPRM theorem consists of about 8k loc including 3k loc for Matiyasevich's results alone, together with a 4k loc addition to our shared libraries; see Appendix A for more details. The paper itself can be read without in-depth knowledge of Coq or type theory.

1.1. **Contribution.** This paper is an extended journal version of a conference paper [25], which, besides a full mechanisation of the DPRM theorem, contributed a novel refactoring of the proof via FRACTRAN improving the explainability of the DPRM theorem. Compared to the conference version, we contribute mechanised proofs showing that

- H10 reduces to solvability of Diophantine equations over integers, reduction obtained via a low-level implementation of Lagrange's theorem (Section 9);
- Diophantine relations are recognisable by μ-recursive algorithms (Section 10);
- μ-recursively recognisable relations are MM-recognisable (Section 10), thereby proving that all considered problems are in the same many-one reduction class;
- μ-recursive algorithms can be simulated in the weak call-by-value λ-calculus (Section 11), thereby proving that all considered problems are in the same many-one reduction class as most problems in the Coq library of undecidable problems [18].

Apart from the new results, we have simplified the account of Diophantine logic considerably and expanded various explanations of proofs.

1.2. Preliminaries.

For the text, while we cannot completely avoid it, we will try to minimize reliance on type theoretic language and notations. We write $\mathbb{P}$ for the (impredicative) type of propositions and `Type` for the (predicative hierachy of) types of Coq. When X and Y are types, we write $X \to Y$ for functions from X to Y.[3] We write $x \times y$ or $x \cdot y$ for multiplication of natural numbers $x, y : \mathbb{N}$ and we will leave out the symbol where convenient. We write $\mathbb{L}\,X$ for the type of *lists* over X and $l \mathbin{+\!\!+} l'$ for the *concatenation* of two lists. We write X^n for *vectors* $\vec{v}$ over type X with length n, and $\mathbb{F}_n$ for the *finite type* with exactly n elements. For $p : \mathbb{F}_n$, we write $\vec{v}_p$ for the p-th component of $\vec{v} : X^n$. Notations for lists are overloaded for vectors.

We write $\mathbb{O}\,X$ for the type of *options* over X with elements $\lfloor x \rfloor$ where $x : X$ and $\emptyset$. We write $X + Y$ for the type-theoretic sum of types X and Y, consisting of $\mathsf{inl}\ x$ for $x : X$ and $\mathsf{inr}\ y$ for $y : Y$. For a list $l : \mathbb{L}\,X$, $l_n : \mathbb{O}\,X$ denotes the n-th value in l if it exists.

If $P : X \to \mathbb{P}$ is a predicate (on X) and $Q : Y \to \mathbb{P}$ is a predicate, we write $P \preceq Q$ if there is a function $f : X \to Y$ s.t. $\forall x : X,\ P\,x \leftrightarrow Q(f\,x)$, i.e. a *many-one reduction* from P to Q. In the synthetic approach [10, 11, 16], the computability of the reduction f is automatically ensured because f is typeable in Coq without relying on any axiom.

2. Diophantine Relations

Diophantine relations are composed of polynomials over natural numbers. There are several equivalent approaches to characterise these relations and oftentimes, the precise definition is omitted from papers. Basically, one can form equations between polynomial expressions and then combine these with conjunctions, disjunctions, and existential quantification.[4] For instance, these operations are assumed as Diophantine producing operators in e.g. [23, 27, 28, 29]. Sometimes Diophantine relations are restricted to a single polynomial equation. Sometimes the exponentiation function $x, y \mapsto x^y$ is assumed as Diophantine [23]. To complicate the picture, Diophantine relations might equivalently range over $\mathbb{Z}$ (instead of $\mathbb{N}$) but expressions like x^y implicitly assume that y never gets a negative value.

[3] In the case where X and Y are moreover propositions, the type/proposition $X \to Y$ is understood as X implies Y, inhabited with functions mapping proofs of X into proofs of Y; the type theory of Coq fully implements the Curry-Howard isomorphism.

[4] Universal quantification or negation are not accepted as is.

Although seemingly diverging, these approaches are not contradictory because in the end, they characterise the same class of relations on natural numbers. However, mechanisation does not allow for such implicit assumptions. To give some mechanisable structure to some of these approaches, we propose three increasingly restricted characterisations of Diophantine relations: *Diophantine logic*, *elementary Diophantine constraints* and *single Diophantine equations*, between which we provide computable transformations in Sections 3 and 4. In Section 9, we also elaborate on the case of polynomials over $\mathbb{Z}$, i.e. we give an undecidability proof for Hilbert tenth problem over integers. But before turning to formal definitions, we motivate our approach for the automated analysis and recognition of Diophantine shapes.

2.1. **Diophantine Shapes.**

We introduce the meta-level notion of *Diophantine shape*. On purpose, this notion does not have a precise formal definition because it is a *dynamically evolving property* of meta-level expressions that upgrades itself as more and more closure results are proved about those shapes.

Of course, we cannot rely on some blurry notion to formally prove theorems about Diophantine relations. So at some point, we have to choose one (or several) formal representation(s) of Diophantine relations. Irrelevant to the actual syntax we finally pick up for the formal description of Diophantine relations, we call them *object-level representations*.

The problem we face is the following: how can we minimize the work we have to do to actually build an object-level witness corresponding to a given meta-level expression representing a Diophantine relation. Directly manipulating object-level syntax is far from the ideal way to proceed for a lazy Coq programmer,[5] and indeed, this empirical lesson can be learned the painful way.

Let us illustrate this on the following complex example of meta-level expression

$$x = z^{x+k} \land \forall y,\, y < k \to \exists u\, v, u \equiv 2v\,[p] \land p = 2^y \land \mathcal{C}_u^z \equiv x\,[v].$$

After a quick analysis of its structure, it appears that before being able to actually establish that this is a Diophantine relation, we would probably have to show that polynomials, identities, arithmetic congruences, the exponential function, binomial coefficients, conjunction, existential quantification and bounded universal quantification are all Diophantine admissible, hence to give *procedures* to derive object-level representations for all these meta-level constructions. This already amounts to significant work. But once this is done, we want to avoid both the hand-building of the object-level witness for the above expression, and the proofs that it correctly reflects its semantics. We essentially require our framework to be able to automatically combine those procedures and build a provably correct witness for us.

To summarise, we aspire at the formal definition of an object-level representation and at the same time, at avoiding its direct manipulation. This is where the dynamic notion of Diophantine shape plays a central role. At first, there are very few basic Diophantine shapes, typically constants, variables, addition, multiplication, equality. But at some point, we would e.g. have a result stating that if the expressions f and g have a Diophantine shape, then so does the expression f^g, that is the Diophantine admissibility of the exponential function, nowadays called Matiyasevich's theorem. Critically, shapes can be added dynamically as they are proved admissible as opposed to be carved in the stone of a given object-level syntax. We now describe how to do this in a successful way using some of the automation provided by Coq.

[5]being able to defer repetitive work to computers is critical to the successful completion of mechanizations.

2.2. **Diophantine Logic.** We define the type $\mathbb{D}_{\text{form}}$ of Diophantine formulæ for the abstract syntax of Diophantine logic. An atomic Diophantine logic formula is just expressing basic atomic identities between variables like $x_i \doteq x_j \dot{+} x_k$ or $x_i \doteq x_j \dot{\times} x_k$ and we combine those with binary disjunction, binary conjunction, and existential quantification.

$$A, B : \mathbb{D}_{\text{form}} ::= x_i \doteq n \mid x_i \doteq x_j \mid x_i \doteq x_j \dot{+} x_k \mid x_i \doteq x_j \dot{\times} x_k \mid A \dot{\wedge} B \mid A \dot{\vee} B \mid \dot{\exists}\, A$$

The letters A, B range over formulæ and $n : \mathbb{N}$ represents constant ranging over natural numbers. We use standard *De Bruijn syntax* with variables $x_0, x_1, \ldots$ of type $\mathsf{V} := \mathbb{N}$ for better readability. If we have $x_i : \mathsf{V}$, we write x_{1+i} for the next variable in V. As an example, the meta-level formula $\exists y, (y = 0 \wedge \exists z,\, y = z + k)$ would be represented as e.g. $\dot{\exists}(x_0 \doteq 0 \mathbin{\dot{\wedge}} \dot{\exists}(x_1 \doteq x_0 \dot{+} x_2))$, i.e. the variable x_i refers to the i-th binder in the context. Notice that there is no type or syntactic distinction between variables and parameters in Diophantine logic. However some variables are bound in their context and others are free.

We provide a semantics for Diophantine logic. Given a valuation for variables $\nu : \mathsf{V} \to \mathbb{N}$, we define the interpretation $[\![x_i \doteq \ldots]\!]_\nu : \mathbb{P}$ of atomic formulæ by

$$\begin{array}{ll} [\![x_i \doteq n]\!]_\nu \;:= \nu\, x_i = n & [\![x_i \doteq x_j \dot{+} x_k]\!]_\nu := \nu\, x_i = \nu\, x_j + \nu\, x_k \\ [\![x_i \doteq x_j]\!]_\nu := \nu\, x_i = \nu\, x_j & [\![x_i \doteq x_j \dot{\times} x_k]\!]_\nu := \nu\, x_i = \nu\, x_j \times \nu\, x_k \end{array}$$

and $[\![A]\!]_\nu : \mathbb{P}$ for a compound formula $A : \mathbb{D}_{\text{form}}$ by the following recursive equations:

$$[\![A \mathbin{\dot{\wedge}} B]\!]_\nu := [\![A]\!]_\nu \wedge [\![B]\!]_\nu \qquad [\![A \mathbin{\dot{\vee}} B]\!]_\nu := [\![A]\!]_\nu \vee [\![B]\!]_\nu \qquad [\![\dot{\exists}\, A]\!]_\nu := \exists n : \mathbb{N}, [\![A]\!]_{n\cdot\nu}$$

where $n\cdot\nu : \mathsf{V} \to \mathbb{N}$ is the standard De Bruijn extension[6] of a valuation ν by n, defined by $n\cdot\nu\,(x_0) := n$ and $n\cdot\nu\,(x_{1+i}) := \nu\, x_i$.

We give a first object-level representation of Diophantine relations as members of type $(\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ mapping valuations of variables to propositions. Moreover, they must be identical to $\lambda\nu.[\![A]\!]_\nu$ for some Diophantine formula A, up to propositional extensionality. We give an informative content to this sub-type of $(\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ to be able to do some computations with the witness $A : \mathbb{D}_{\text{form}}$ of Diophantineness, typically when moving to another formal representation like elementary Diophantine constraints in Section 3.

Definition 1. *The class of* Diophantine relations $\mathbb{D}_{\text{rel}} : ((\mathsf{V} \to \mathbb{N}) \to \mathbb{P}) \to \mathtt{Type}$ *is the informative sub-type defined for* $R : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ *by*

$$\mathbb{D}_{\text{rel}}\, R := \sum A : \mathbb{D}_{\text{form}}, (\forall \nu, [\![A]\!]_\nu \leftrightarrow R\ \nu).$$

Note that $\sum$ denotes type-theoretic dependent pairs. Hence an inhabitant w of $\mathbb{D}_{\text{rel}}\, R$ is a (dependent) pair (A, H_A) where $A = \pi_1(w)$ is a Diophantine formula and $H_A = \pi_2(w)$ a proof that $[\![A]\!]_{(\cdot)}$ and R are extensionally equivalent[7] The informal notion of Diophantine shape will correspond to the dynamically growing collection of established closure properties of the class $\mathbb{D}_{\text{rel}}$ of Diophantine relations. Definition 1 of the sub-type $\mathbb{D}_{\text{rel}}$ already entails that $\mathbb{D}_{\text{rel}}$ is closed under conjunction, disjunction, existential quantification and renaming.

Proposition 2. *Let* $R, S : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ *be relations,* $T : \mathbb{N} \to (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ *be a relation with a singled out variable, and* $\rho : \mathsf{V} \to \mathsf{V}$ *be a renaming function. We have the maps:*

1. $\mathbb{D}_{\text{rel}}\, R \to \mathbb{D}_{\text{rel}}\, S \to \mathbb{D}_{\text{rel}}(\lambda\nu.R\,\nu \wedge S\,\nu)$;
2. $\mathbb{D}_{\text{rel}}\, R \to \mathbb{D}_{\text{rel}}\, S \to \mathbb{D}_{\text{rel}}(\lambda\nu.R\,\nu \vee S\,\nu)$;
3. $\mathbb{D}_{\text{rel}}\big(\lambda\nu.T\,(\nu\, x_0)\,(\lambda x_i.\nu\, x_{1+i})\big) \to \mathbb{D}_{\text{rel}}(\lambda\nu.\exists u, T\, u\ \nu)$;
4. $(\forall \nu, S\,\nu \leftrightarrow R\,\nu) \to \mathbb{D}_{\text{rel}}\, R \to \mathbb{D}_{\text{rel}}\, S$;
5. $\mathbb{D}_{\text{rel}}\, R \to \mathbb{D}_{\text{rel}}\big(\lambda\nu.R\,(\nu \circ \rho)\big)$.

[6]The notation $n\cdot\nu$ emphasizes that the value n is pushed ahead of the infinite sequence $\nu\, x_0; \nu\, x_1; \nu\, x_2; \ldots$

[7]For the efficiency of computations, we usually hide the purely logical part H_A into an *opaque proof term.*

Understood as Diophantine shapes, maps number 1–3 recognise the logical connectives of conjunction, disjunction and existential quantification as newly allowed shapes. Map number 5 allows renaming (free) variables hence Diophantine shapes are closed under renaming.[8] Map number 4 provides a way to replace the goal $\mathbb{D}_{\text{rel}}\, S$ with $\mathbb{D}_{\text{rel}}\, R$ once a proof that they are logically equivalent is established. Hence, if S cannot be analysed because it does not currently have a Diophantine shape, it can still be replaced by an equivalent relation R, hopefully better behaved; see e.g. the proof of Proposition 4.

Working with Diophantine relations already gives a satisfying implementation of Diophantine shapes but it is sometimes more convenient to manipulate Diophantine functions instead of relations so we define the following specialization.

Definition 3. *The class of* Diophantine functions $\mathbb{D}_{\text{fun}} : ((\mathsf{V} \to \mathbb{N}) \to \mathbb{N}) \to \texttt{Type}$ *is the informative sub-type defined for* $f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ *by* $\mathbb{D}_{\text{fun}}\, f := \mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_0 = f(\lambda x_i.\nu\, x_{1+i})\big)$.

We extend Diophantine shapes with polynomials expressions and equations between them. To illustrate the mechanics behind Diophantine shape recognition, for once we give a detailed account of the steps that are usually performed automatically in this framework.

Proposition 4. *Let* $x_i : \mathsf{V}$, $n : \mathbb{N}$, $\rho : \mathsf{V} \to \mathsf{V}$ *and* $f, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$. *We have the maps:*

1. $\mathbb{D}_{\text{fun}}\,(\lambda\nu.\nu\, x_i)$;
2. $\mathbb{D}_{\text{fun}}\,(\lambda\nu.n)$;
3. $\mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}(\lambda\nu.f(\nu \circ \rho))$;
4. $\mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}\, g \to \mathbb{D}_{\text{fun}}\,(\lambda\nu.f\,\nu + g\,\nu)$;
5. $\mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}\, g \to \mathbb{D}_{\text{fun}}\,(\lambda\nu.f\,\nu \times g\,\nu)$;
6. $\mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}\, g \to \mathbb{D}_{\text{rel}}\,(\lambda\nu.f\,\nu = g\,\nu)$.

Proof. Items 1, 2 and map 3 are for projections, constants and renaming functions respectively. The non-trivial cases are for $+$, $\times$ and $=$. We cover the cases of $+$ and then $=$ in details to illustrate how the representations of Diophantine relations/functions behave in proof scripts. In particular, we prove the results reasoning backwards (as is usually done in Coq), applying established theorems to convert a given proof goal into (hopefully) simpler proof goals.

For the goal $\mathbb{D}_{\text{fun}}\,(\lambda\nu.f\,\nu + g\,\nu)$, unfolding the assumptions $\mathbb{D}_{\text{fun}}\, f$ and $\mathbb{D}_{\text{fun}}\, g$ we have

$$\mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_0 = f\,(\lambda x_i.\nu\, x_{1+i})\big) \qquad \text{and} \qquad \mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_0 = g\,(\lambda x_i.\nu\, x_{1+i})\big) \tag{2.1}$$

and we want to establish $\mathbb{D}_{\text{fun}}\,(\lambda\nu.f\,\nu + g\,\nu)$, i.e.

$$\mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_0 = f\,(\lambda x_i.\nu\, x_{1+i}) + g\,(\lambda x_i.\nu\, x_{1+i})\big). \tag{2.2}$$

By map 4 of Proposition 2, we replace Eq. (2.2) with the (obviously) equivalent goal

$$\mathbb{D}_{\text{rel}}\big(\lambda\nu.\exists a \exists b,\, \nu\, x_0 = a + b \wedge a = f\,(\lambda x_i.\nu\, x_{1+i}) \wedge b = g\,(\lambda x_i.\nu\, x_{1+i})\big) \tag{2.3}$$

and we then apply map 3 of Proposition 2 twice to get the goal

$$\mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_2 = \nu\, x_1 + \nu\, x_0 \wedge \nu\, x_1 = f\,(\lambda x_i.\nu\, x_{3+i}) \wedge \nu\, x_0 = g\,(\lambda x_i.\nu\, x_{3+i})\big).$$

We now apply twice map 1 of Proposition 2 and we get the three following sub-goals:

$$\mathbb{D}_{\text{rel}}(\lambda\nu.\nu\, x_2 = \nu\, x_1 + \nu\, x_0) \quad \mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_1 = f\,(\lambda x_i.\nu\, x_{3+i})\big) \quad \mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_0 = g\,(\lambda x_i.\nu\, x_{3+i})\big)$$

(1) For the first sub-goal, we use the formula $x_2 \doteq x_1 \dot{+} x_0 : \mathbb{D}_{\text{form}}$ as object-level witness;
(2) for the second sub-goal, we consider the renaming function $\rho_1 : \mathsf{V} \to \mathsf{V}$ defined by $\rho_1(x_0) := x_1$ and $\rho_1(x_{1+i}) := x_{3+i}$ and derive $\mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_1 = f\,(\lambda x_i.\nu\, x_{3+i})\big)$ by applying map 5 of Proposition 2 to the hypothesis $\mathbb{D}_{\text{rel}}\big(\lambda\nu.\nu\, x_0 = f\,(\lambda x_i.\nu\, x_{1+i})\big)$ in Eqs. (2.1);

[8] $\nu \circ \rho : \mathsf{V} \to \mathbb{N}$ denotes the composition $\lambda x_i.\nu\, x_{\rho_i}$ of the valuation $\nu : \mathsf{V} \to \mathbb{N}$ with the renaming $\rho : \mathsf{V} \to \mathsf{V}$.

(3) the third and last sub-goal $\mathbb{D}_{\rm rel}(\lambda\nu.\nu\, x_0 = g\,(\lambda x_i.\nu\, x_{3+i}))$ is solved similarly with the renaming function $\rho_0 : \mathsf{V} \to \mathsf{V}$ defined by $\rho_0(x_0) := x_0$ and $\rho_0(x_{1+i}) := x_{3+i}$.

We now deal with map 6, hence with the goal $\mathbb{D}_{\rm rel}\,(\lambda\nu.f\,\nu = g\,\nu)$ under the same previous assumptions $\mathbb{D}_{\rm fun}\,f$ and $\mathbb{D}_{\rm fun}\,g$, i.e. Eqs. (2.1). We proceed in a somewhat less detailed explanation. We replace the goal by $\mathbb{D}_{\rm rel}\,(\lambda\nu.\exists a \exists b,\ a = b \land a = f\,\nu \land b = g\,\nu)$ which is equivalent and then, after applying the maps of Proposition 2, we get three sub-goals $\mathbb{D}_{\rm rel}\,(\lambda\nu.\nu\,x_1 = \nu\,x_0)$, $\mathbb{D}_{\rm rel}(\lambda\nu.\nu\,x_1 = f\,(\lambda x_i.\nu\,x_{2+i}))$ and $\mathbb{D}_{\rm rel}(\lambda\nu.\nu\,x_0 = g\,(\lambda x_i.\nu\,x_{2+i}))$. In turn, the first sub-goal corresponds to the witness $x_1 \doteq x_0 : \mathbb{D}_{\rm form}$, while the second and third sub-goals follow from Eqs. (2.1) respectively using straightforward renaming functions. □

On paper these proofs look somehow complicated by the need to infer the renaming functions but from a *mechanisation point of view,* Coq's unification algorithm automatically solves such goals. Provided we populate the *hint database* with enough admissible shapes, we can automate the analysis of the meta-level syntax to establish Diophatineness and reflect a meta-level expression of Diophantine shape into the corresponding object-level witness of type $\mathbb{D}_{\rm form}$ together with the proof that it is an appropriate witness, hence packed into the types $\mathbb{D}_{\rm rel}\,R$ for relational expressions or $\mathbb{D}_{\rm fun}\,f$ for functional expressions.

With Propositions 2 and 4, we populate the hint database for relations with the shapes conjunction, disjunction, existential quantification, renaming and identity between two functional expressions, and for functions, we add the shapes of projections, constants, addition, multiplication and renaming. In our implementation, the analysis of Diophantine shapes is performed by the automatic `dio auto` tactic. With such an automated approach, the remaining (and sometimes difficult) work occurs when we apply map 4 of Proposition 2, that is, we have to find an equivalent expression of Diophantine shape, like in Eq. (2.3) and to prove it is indeed equivalent to Eq. (2.2), which, unlike that specific example, might be non-trivial; see e.g. the discussion in Section 5.

Proposition 5. *Let $f, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$. We have the maps:*

1. $\mathbb{D}_{\rm rel}\,(\lambda\nu.\texttt{True})$;
2. $\mathbb{D}_{\rm rel}\,(\lambda\nu.\texttt{False})$;
3. $\mathbb{D}_{\rm fun}\,f \to \mathbb{D}_{\rm fun}\,g \to \mathbb{D}_{\rm rel}\,(\lambda\nu.f\,\nu \le g\,\nu)$;
4. $\mathbb{D}_{\rm fun}\,f \to \mathbb{D}_{\rm fun}\,g \to \mathbb{D}_{\rm rel}\,(\lambda\nu.f\,\nu < g\,\nu)$;
5. $\mathbb{D}_{\rm fun}\,f \to \mathbb{D}_{\rm fun}\,g \to \mathbb{D}_{\rm rel}\,(\lambda\nu.f\,\nu \ne g\,\nu)$.

Proof. For e.g. $\mathbb{D}_{\rm rel}\,(\lambda\nu.\,f\,\nu < g\,\nu)$, we shift to the equivalent $\mathbb{D}_{\rm rel}\,(\lambda\nu.\exists a, 1 + a + f\,\nu = g\,\nu)$ using map 4 of Proposition 2 and finish the proof calling `dio auto`.[9] □

Again, we populate the hint database with the new shapes of Proposition 5. We follow up with the slightly more complex example of the "does not divide" relation defined by $u \nmid v := \neg(\exists k, v = k \times u)$. At this point, this expression cannot be recognized as a Diophantine shape because it contains a negation.

Proposition 6. $\forall f\, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N},\ \mathbb{D}_{\rm fun}\,f \to \mathbb{D}_{\rm fun}\,g \to \mathbb{D}_{\rm rel}\,(\lambda\nu.f\,\nu \nmid g\,\nu)$.

Proof. However, using Euclidean division, we (easily) prove the equivalence

$$u \nmid v \leftrightarrow (u = 0 \land v \ne 0 \lor \exists a\, b,\ v = a \times u + b \land 0 < b < u)$$

[9]Notice that the actual implemented proofs might differ slightly because we sometimes optimize the shape of expressions for smaller witnesses, especially for these basic shapes which pop up over and over again.

and this new expression can now be recognised as a Diophantine shape. Using this equivalence in combination with map 4 of Proposition 2, we replace the goal $\mathbb{D}_{\text{rel}}\,(\lambda\nu.f\,\nu \nmid g\,\nu)$ with

$$\mathbb{D}_{\text{rel}}\,(\lambda\nu.f\,\nu = 0 \land g\,\nu \neq 0 \lor \exists a\,b,\, g\,\nu = a \times f\,\nu + b \land 0 < b \land b < f\,\nu)$$

and then let the magic of `dio auto` unfold. □

Again, once established, we can add the map $\mathbb{D}_{\text{fun}}\,f \to \mathbb{D}_{\text{fun}}\,g \to \mathbb{D}_{\text{rel}}\,(\lambda\nu.f\,\nu \nmid g\,\nu)$ in the Diophantine hint database so that later encountered proof goals $\mathbb{D}_{\text{rel}}\,(\lambda\nu.f\,\nu \nmid g\,\nu)$ can be immediately solved by `dio auto`.

In this above described approach, the recovery of the object-level witness A of Definition 1 from meta-level syntax is automatic and hidden by the use of the `dio auto` tactic associated with the ever growing hint database. This allows us to proceed as in e.g. Matiyasevich papers where he usually transforms a relation into an equivalent Diophantine shape, accumulating more and more Diophantine shapes on the way. Instead of having to manipulate object-level witnesses by hand, obfuscating sometimes simple to understand proofs, we use Diophantine shapes as the cornerstone of the faithful implementation of existing pen and paper scripts.

2.3. **Exponentiation and Bounded Universal Quantification.** For now, we introduce the *elimination of the exponential relation* and then of *bounded universal quantification* as black boxes expressed in the framework of Diophantine shapes, i.e. new closure properties of the classes $\mathbb{D}_{\text{fun}}/\mathbb{D}_{\text{rel}}$.

While we do contribute implementations for both of these hard results, on purpose, we choose to avoid the detailed presentation of these mechanised proofs for several reasons:

- first of all, there are already fully detailed pen and paper accounts of these results and we implemented two of these somewhat faithfully;
- then, in our modular approach, the proof of these admissibility results can be ignored without hindering the understanding of the overall structure of the main results, e.g. H10;
- finally, already the above cited pen and paper proofs assume some not so standard results in arithmetic like e.g. Lucas's theorem, and we favoured giving accounts of those assumed theorems instead of simply reproducing the rest of the existing arguments.

Hence, for the moment, we postpone remarks and discussions about the Diophantineness of the exponential function and the Diophantine admissibility of bounded universal quantification to Section 5.

Theorem 7 (Exponential). $\forall f\,g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N},\, \mathbb{D}_{\text{fun}}\,f \to \mathbb{D}_{\text{fun}}\,g \to \mathbb{D}_{\text{fun}}(\lambda\nu.(f\,\nu)^{g\,\nu})$.

To prove it, one needs a meta-level Diophantine shape for the exponential relation, *the proof of which is nothing short of extraordinary.* This landmark result is due to Matiyasevich [27], but we have implemented the shorter and more up-to-date proof of [29].

Theorem 8 (Bounded Universal Quantification). *For* $f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$ *and* $T : \mathbb{N} \to (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$*, we have* $\mathbb{D}_{\text{fun}}\,f \to \mathbb{D}_{\text{rel}}(\lambda\nu.T\,(\nu\,x_0)\;(\lambda x_i.\nu\,x_{1+i})) \to \mathbb{D}_{\text{rel}}(\lambda\nu.\forall u, u < f\,\nu \to T\,u\;\nu)$.

This map can be compared with map 3 of Proposition 2 and allows to recognise bounded universal quantification as a legitimate Diophantine shape. We have implemented the direct proof of Matiyasevich [28] which does not involve a detour through a model of computation. Notice that the bound $f\,\nu$ in $\forall u,\, u < f\,\nu \to \ldots$ is not assumed constant otherwise the elimination of the quantifier would proceed as a simple reduction to a finitary conjunction.

2.4. **Reflexive-Transitive Closure is Diophantine.** With these tools – elimination of the exponential relation and of bounded universal quantification – we can show that the reflexive and transitive closure of a Diophantine binary relation is itself Diophantine. We assume a binary relation $R : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$ over natural numbers. The Diophantineness of R can be formalised by assuming that e.g. $\lambda\nu.R\ (\nu\, x_1)\ (\nu\, x_0)$ is a Diophantine relation. We show that the i-th iterate of R is Diophantine (where i is non-constant).

Lemma 9. *For any binary relation $R : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$ and $f, g, h : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}$, we have*

$$\mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}\, g \to \mathbb{D}_{\text{fun}}\, h \to \mathbb{D}_{\text{rel}}\big(\lambda\nu.R\ (\nu\, x_1)\ (\nu\, x_0)\big) \to \mathbb{D}_{\text{rel}}\big(\lambda\nu.R^{h\,\nu}\ (f\,\nu)\ (g\,\nu)\big).$$

Proof. Using Euclidean division, we define the `is_digit` $c\ q\ n\ d$ predicate stating that d is the n-th digit of the base q development of number c, as a Diophantine sentence:

$$\texttt{is\_digit}\ c\ q\ n\ d := d < q \land \exists a\, b,\ c = (aq + d)q^n + b \land b < q^n.$$

The Diophantineness of this follows from previous Diophantine shapes, including the exponential (Theorem 7). Then we define the `is_seq` $R\ c\ q\ i$ predicate stating that the first $i+1$ digits of c in base q form an R-chain, again with a Diophantine shape, established using hypothesis $\mathbb{D}_{\text{rel}}(\lambda\nu.R\ (\nu\, x_1)\ (\nu\, x_0))$ and the Diophantine admissibility of bounded universal quantification (Theorem 8):

$$\texttt{is\_seq}\ R\ c\ q\ i := \forall n,\ n < i \to \exists u\, v,\ \texttt{is\_digit}\ c\ q\ n\ u \land \texttt{is\_digit}\ c\ q\ (1+n)\ v \land R\ u\ v$$

Then we encode $R^i\ u\ v$ by stating that there exists a (large enough) q and a number c such that the first $i+1$ digits of c in base q form an R-chain starting at u and ending at v:

$$R^i\ u\ v \ \leftrightarrow\ \exists q\, c,\ \texttt{is\_seq}\ R\ c\ q\ i \land \texttt{is\_digit}\ c\ q\ 0\ u \land \texttt{is\_digit}\ c\ q\ i\ v$$

and this expression is accepted as a Diophantine shape by `dio auto`. Then assuming Diophantineness of f, g and h, we easily derive that $\lambda\nu.R^{h\,\nu}\ (f\,\nu)\ (g\,\nu)$ is Diophantine. □

We fill in Lemma 9 in the Diophantine hint database and we derive the Diophantineness of the reflexive-transitive closure as a direct consequence of the equivalence $R^*\ u\ v \leftrightarrow \exists i, R^i\ u\ v$.

Theorem 10 (RT-closure). *For any binary relation $R : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$, we have the map*

$$\forall f\ g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N},\ \mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}\, g \to \mathbb{D}_{\text{rel}}\big(\lambda\nu.R\ (\nu\, x_1)\ (\nu\, x_0)\big) \to \mathbb{D}_{\text{rel}}\big(\lambda\nu.R^*\ (f\,\nu)\ (g\,\nu)\big).$$

3. Elementary Diophantine Constraints

We now shift to another, seemingly less expressive, object-level representation of Diophantine relations. Elementary Diophantine constraints are very simple equations where only one instance of either $\dot{+}$ or $\dot{\times}$ is allowed. Schematically, starting from Diophantine logic, we remove disjunction and existential quantification and encode conjunctions into the structure of a list. We give a direct proof that any Diophantine logic formula is semantically equivalent to the simultaneous satisfiability of a list of elementary Diophantine constraints.

Starting from two copies of $\mathbb{N}$, one called U with u, v, w ranging over U for existentially quantified variables, and another one $\mathsf{V} = \{x_0, x_1, \ldots\}$ for parameters, we define the type of elementary Diophantine constraints by:[10]

$$c : \mathbb{D}_{\text{cstr}} ::= u \doteq n \mid u \doteq v \mid u \doteq x_i \mid u \doteq v \,\dot{+}\, w \mid u \doteq v \,\dot{\times}\, w \qquad \text{where } n : \mathbb{N}$$

[10]The equation $u \doteq v$ is redundant because it could be replaced with $z \doteq 0 \mathrel{\dot{\land}} u \doteq z \,\dot{+}\, v$, for some fresh z. However we keep $u \doteq v$ in the syntax because this simplifies arguments when parameters x_i are mapped to existential variables v in the proof of Lemma 13, the type $\mathbb{D}_{\text{cstr}}$ being thus closed under this transformation.

Notice that these constraints do not have a "real" inductive structure, they are flat and of size (number of symbols) either 3 or 5. Given two interpretations, $\varphi : \mathsf{U} \to \mathbb{N}$ for variables and $\nu : \mathsf{V} \to \mathbb{N}$ for parameters, it is trivial to define the semantics $[\![c]\!]^{\varphi}_{\nu} : \mathbb{P}$ of a single constraint c of type $\mathbb{D}_{\text{cstr}}$:

$$[\![u \doteq n]\!]^{\varphi}_{\nu} := \varphi\,u = n \qquad [\![u \doteq v]\!]^{\varphi}_{\nu} := \varphi\,u = \varphi\,v \qquad [\![u \doteq v \dotplus w]\!]^{\varphi}_{\nu} := \varphi\,u = \varphi\,v + \varphi\,w$$
$$[\![u \doteq x_i]\!]^{\varphi}_{\nu} := \varphi\,u = \nu\,x_i \qquad [\![u \doteq v \mathbin{\dot\times} w]\!]^{\varphi}_{\nu} := \varphi\,u = \varphi\,v \times \varphi\,w$$

Given a list $l : \mathbb{L}\,\mathbb{D}_{\text{cstr}}$ of constraints, we write $[\![l]\!]^{\varphi}_{\nu}$ when all the constraints in l are simultaneously satisfied, i.e. $[\![l]\!]^{\varphi}_{\nu} := \forall c,\ c \in l \to [\![c]\!]^{\varphi}_{\nu}$. We show the following result:

Theorem 11. *For any Diophantine formula $A : \mathbb{D}_{\text{form}}$ one can compute a list of elementary Diophantine constraints $l : \mathbb{L}\,\mathbb{D}_{\text{cstr}}$ such that $\forall \nu : \mathsf{V} \to \mathbb{N},\ [\![A]\!]_{\nu} \leftrightarrow \exists \varphi : \mathsf{U} \to \mathbb{N},\ [\![l]\!]^{\varphi}_{\nu}$.*

Put in other terms, for any given interpretation ν of parameters, $[\![A]\!]_{\nu}$ holds if and only if the constraints in l are simultaneously satisfiable. Any Diophantine logic formula is equivalent to the satisfiability of the conjunction of finitely many elementary Diophantine constraints. The proof of Theorem 11 spans the rest of this section. We will strengthen the result a bit to be able to get an easy argument by induction on A.

Definition 12. *Given a relation $R : (\mathsf{V} \to \mathbb{N}) \to \mathbb{P}$ and an interval $[u_a, u_{a+n}[\ \subseteq \mathsf{U}$, an* elementary representation of R in $[u_a, u_{a+n}[$ *is given by:*

(1) *a list $\mathcal{E} : \mathbb{L}\,\mathbb{D}_{\text{cstr}}$ of constraints and a reference variable $\mathfrak{r} : \mathsf{U}$;*
(2) *proofs that $\mathfrak{r}$ and the (existentially quantified) variables of $\mathcal{E}$ belong to $[u_a, u_{a+n}[$;*
(3) *a proof that the constraints in $\mathcal{E}$ are always (simultaneously) satisfiable, i.e. $\forall \nu \exists \varphi\, [\![\mathcal{E}]\!]^{\varphi}_{\nu}$;*
(4) *a proof that the list $(\mathfrak{r} \doteq 0) :: \mathcal{E}$ is equivalent to R, i.e. $\forall \nu,\ R\ \nu \leftrightarrow (\exists \varphi,\ \varphi\,\mathfrak{r} = 0 \wedge [\![\mathcal{E}]\!]^{\varphi}_{\nu})$.*

It is obvious that an elementary representation of $\lambda\nu.[\![A]\!]_{\nu}$ in any interval $[u_a, u_{a+n}[$ is enough to prove Theorem 11 because of item 4 of Definition 12. But actually, computing such a representation is simpler than proving Theorem 11 directly.[11] Below, the size of a Diophantine formula A, denoted $|A|$, is defined as the number of nodes of its syntactic tree.

Lemma 13. *For any $a : \mathbb{N}$ and any $A : \mathbb{D}_{\text{form}}$, one can compute $n \leq 8|A|$ and an elementary representation of the relation $\lambda\nu.[\![A]\!]_{\nu}$ in $[u_a, u_{a+n}[$.*

Proof. We show the result by structural induction on A.

- If A is e.g. $x_i \doteq x_j \mathbin{\dot\times} x_k$, we get the representation with $n := 8$ and the pair $(\mathcal{E}, \mathfrak{r})$ with

$$\mathcal{E} := \begin{bmatrix} u_{a+7} \doteq u_a \dotplus u_{a+1}\ ;\ u_{a+6} \doteq u_a \dotplus u_{a+2}\ ;\ u_{a+6} \doteq u_{a+1} \dotplus u_{a+5}\ ; \\ u_{a+5} \doteq u_{a+3} \mathbin{\dot\times} u_{a+4}\ ;\ u_{a+4} \doteq x_k\ ;\ u_{a+3} \doteq x_j\ ;\ u_{a+2} \doteq x_i \end{bmatrix} \qquad \mathfrak{r} := u_{a+7}$$

  Property (2) is obviously satisfied. Property (3), i.e. the satisfiability of $\mathcal{E}$ whatever the values of x_i, x_j and x_k, is simple to establish: indeed, the values of $u_{a+2} = x_i$, $u_{a+3} = x_j$, $u_{a+4} = x_k$ and $u_{a+5} = x_j x_k$ are uniquely determined. Pick $u_a := x_j x_k$ and $u_{a+1} := x_i$ and then again $u_{a+6} = u_{a+7} = x_i + x_j x_k$ are both uniquely determined. This assignment of variables satisfies all the constraints in $\mathcal{E}$.

  For property (4), let us now add the extra constraint $u_{a+7} \doteq 0$ and consider a valuation satisfying the constraints in $(u_{a+7} \doteq 0) :: \mathcal{E}$. We must have $u_{a+7} = u_a + u_{a+1} = 0$, hence $u_a = u_{a+1} = 0$ which entails $u_{a+6} = u_{a+2} = u_{a+5}$. But $u_{a+2} = x_i$ and $u_{a+5} = u_{a+3}u_{a+4} = x_j x_k$. Hence $x_i \doteq x_j \mathbin{\dot\times} x_k$ is satisfied.

[11] Proving Theorem 11 directly involves renamings of existential variables and might produce exponential blow-up in the number of constraints when handled naively.

Conversely, parameter values satisfying $x_i \doteq x_j \dot\times x_k$ can be extended to a valuation of variables satisfying $(u_{a+7} \doteq 0) :: \mathcal{E}$ in a unique way with $u_{a+7}, u_a, u_{a+1} := 0$, $u_{a+6}, u_{a+5}, u_{a+2} := x_i$, $u_{a+3} := x_j$ and $u_{a+4} := x_k$;

- We proceed similarly for the other atomic cases $x_i \doteq n$, $x_i \doteq x_j$ and $x_i \doteq x_j \dot+ x_k$;
- When A is $B \mathbin{\dot\wedge} C$, we get a representation in $[u_a, u_{a+n_A}[$ by induction. Hence, let $(\mathcal{E}_B, \mathfrak{r}_B)$ be the representation of B in $[u_a, u_{a+n_B}[$. Then, inductively again, let $(\mathcal{E}_C, \mathfrak{r}_C)$ be a representation of C in $[u_{a+n_B}, u_{a+n_B+n_C}[$. We define $\mathfrak{r}_A := u_{a+n_A+n_B}$ and $\mathcal{E}_A := (\mathfrak{r}_A \doteq \mathfrak{r}_B \dot+ \mathfrak{r}_C) :: \mathcal{E}_B \mathbin{+\!\!+} \mathcal{E}_C$ and then $(\mathcal{E}_A, \mathfrak{r}_A)$ represents $A = B \mathbin{\dot\wedge} C$ in $[u_a, u_{a+1+n_B+n_C}[$;[12]
- The case of $B \mathbin{\dot\vee} C$ is similar: simply replace $\mathfrak{r}_A \doteq \mathfrak{r}_B \dot+ \mathfrak{r}_C$ with $\mathfrak{r}_A \doteq \mathfrak{r}_B \dot\times \mathfrak{r}_C$;
- We finish with the case when A is $\dot\exists\, B$. Let $(\mathcal{E}_B, \mathfrak{r}_B)$ be a representation of B in $[u_a, u_{a+n_B}[$. Let σ be the substitution mapping parameters in V and defined by $\sigma(x_0) := u_{a+n_B}$ and $\sigma(x_{1+i}) := x_i$; existential variables in U are left unmodified by this substitution. Then $(\sigma(\mathcal{E}_B), \mathfrak{r}_B)$ is a representation of $A = \dot\exists\, B$ in $[u_a, u_{a+1+n_B}[$.

This concludes the recursive construction of a representation of $\lambda\nu.[\![A]\!]_\nu$. □

4. Single Diophantine Equations

We now give our last and most naive object-level representation of Diophantine relations as a single polynomial equation. In this section, we show how a list of elementary Diophantine constraints can be simulated by a single identity between two Diophantine polynomials. We use the following well known convexity identity to achieve the reduction.

Proposition 14. *Let $(p_1, q_1), \ldots, (p_n, q_n)$ be a sequence of pairs in $\mathbb{N} \times \mathbb{N}$. Then*

$$\sum_{i=1}^{n} 2p_iq_i = \sum_{i=1}^{n} p_i^2 + q_i^2 \;\leftrightarrow\; p_1 = q_1 \wedge \cdots \wedge p_n = q_n.$$

There are many possible justifications for this equivalence which, if ranging over $\mathbb{Z}$, would more conventionally be written as $\sum_i (p_i - q_i)^2 = 0 \leftrightarrow \forall i,\, p_i = q_i$. This form does not hold directly for $\mathbb{N}$ however, hence the reformulation without using the minus/$-$ operator.[13] See Appendix D for an elementary justification of the result.

Similarly to elementary Diophantine constraints, we define Diophantine polynomials distinguishing the types of U of bound variables and V of parameters (or free variables) but the types U and V are not fixed copies of $\mathbb{N}$ anymore, but type parameters of arbitrary value.

Definition 15. *The type of* Diophantine polynomials $\mathbb{D}_{\text{poly}}(\mathsf{U}, \mathsf{V})$ *and the type of* single Diophantine equations $\mathbb{D}_{\text{single}}(\mathsf{U}, \mathsf{V})$ *are defined by:*

$$p, q : \mathbb{D}_{\text{poly}}(\mathsf{U}, \mathsf{V}) ::= u : \mathsf{U} \mid x : \mathsf{V} \mid n : \mathbb{N} \mid p \dot+ q \mid p \dot\times q \qquad E : \mathbb{D}_{\text{single}}(\mathsf{U}, \mathsf{V}) ::= p \doteq q.$$

For $\varphi : \mathsf{U} \to \mathbb{N}$ and $\nu : \mathsf{V} \to \mathbb{N}$ we define the semantic interpretations of polynomials $[\![p]\!]_\nu^\varphi : \mathbb{N}$ and single Diophantine equations $[\![p \doteq q]\!]_\nu^\varphi : \mathbb{P}$ in the obvious way.

Theorem 16. *For any list $l : \mathbb{L}\,\mathbb{D}_{\text{cstr}}$ of elementary Diophantine constraints, one can compute a single Diophantine equation $E : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathbb{N})$ such that $\forall \nu \forall \varphi,\, [\![E]\!]_\nu^\varphi \leftrightarrow [\![l]\!]_\nu^\varphi$.*

[12] Since the intervals $[u_a, u_{a+n_B}[$ and $[u_{a+n_B}, u_{a+n_B+n_C}[$ are built disjoint, there is no difficulty in merging valuations whereas this usually involves renamings when existential variables are not carefully chosen.

[13] Remember that $0 - 1 = 0$ holds in $\mathbb{N}$.

Proof. We write $l = [p_1 \doteq q_1; \dots ; p_n \doteq q_n]$ and then use Proposition 14. In the code, we moreover show that the size of E is linear in the length of l. If needed, one could also show that the degree of the polynomial is less than 4. $\square$

Corollary 17. *Let $R : (\mathbb{N} \to \mathbb{N}) \to \mathbb{P}$. Assuming $\mathbb{D}_{\text{rel}}\, R$, one can compute a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathbb{N})$ such that $\forall \nu,\ R\ \nu \leftrightarrow \exists \varphi,\ [\![p]\!]^{\varphi}_{\nu} = [\![q]\!]^{\varphi}_{\nu}$.*

Proof. Direct combination of Definition 1 and Theorems 11 and 16. In the formalisation, we also show that the size of the obtained single Diophantine equation is linearly bounded by the size of the witness formula contained in the proof of $\mathbb{D}_{\text{rel}}\, R$. $\square$

We have shown that the automation we designed to recognise relations of Diophantine shape entail that these relations are also definable by satisfiability of a single equation between Diophantine polynomials, so these tools are sound w.r.t. a formally restrictive characterisation of Diophantineness. One could argue that the above existential quantifier $\exists \varphi$ encodes infinitely many existential quantifiers but it can easily be replaced by finitely many existential quantifiers over the bound variables that actually occur in p or q.

Proposition 18. *For any single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathsf{V})$, one can compute $n : \mathbb{N}$ and a new single Diophantine equation $p' \doteq q' : \mathbb{D}_{\text{single}}(\mathbb{F}_n, \mathsf{V})$ such that for any $\nu : \mathsf{V} \to \mathbb{N}$, $(\exists \varphi : \mathbb{N} \to \mathbb{N},\ [\![p]\!]^{\varphi}_{\nu} = [\![q]\!]^{\varphi}_{\nu}) \leftrightarrow (\exists \varphi : \mathbb{F}_n \to \mathbb{N},\ [\![p']\!]^{\varphi}_{\nu} = [\![q']\!]^{\varphi}_{\nu})$.*

Proof. We pick n greater that the number of bound variables which occur in either p or q. This subset of $\mathbb{N}$ can be faithfully embedded into the finite type $\mathbb{F}_n$ and we use such a renaming to compute (p', q'). Remark that the size of (p', q') is the same as that of (p, q). $\square$

By Corollary 17 and Proposition 18, we see that a Diophantine logic formula $A : \mathbb{D}_{\text{form}}$ potentially containing inner existential quantifiers and representing the Diophantine relation $\lambda\nu.[\![A]\!]_{\nu}$ can effectively be reduced to a single Diophantine equation $p' \doteq q' : \mathbb{D}_{\text{single}}(\mathbb{F}_n, \mathsf{V})$ such that $[\![A]\!]_{\nu} \leftrightarrow \exists \varphi : \mathbb{F}_n \to \mathbb{N},\ [\![p']\!]^{\varphi}_{\nu} = [\![q']\!]^{\varphi}_{\nu}$. Because $\mathbb{F}_n$ is the finite type of n elements, the (higher order) existential quantifier $\exists \varphi$ simply encodes n successive (first order) existential quantifiers. The existential quantifiers that occur deep inside A are *not erased* by the reduction, they are *moved at the outer level* and to be ultimately understood as *solvability* for some polynomial equation of which the parameters match the free variables of A.

5. Remarks on the Implementation of Matiyasevich's Theorems

Matiyasevich's theorem stating that there is a Diophantine description of the exponential relation $x = y^z$ is a masterpiece which concluded the line of work by Davis, Putnam and Robinson, starting at Davis's conjecture in 1953. Already in 1952, Julia Robinson discovered that in order to show the exponential relation Diophantine, it suffices to find a single binary Diophantine relation exhibiting exponential growth [35], a so-called Robinson predicate, i.e. a predicate $J(u, v)$ in two variables s.t. $J(u, v)$ implies $v < u^u$ and for every k there are u, v with $J(u, v)$ and $v > u^k$. Robinson's insight meant the only thing missing to prove what is nowadays called the DPRM theorem, was a *single* polynomial equation capturing *any* freely chosen Robinson predicate. Similar to other famous hard problems of mathematics, the question is easy to state, but from the start of the study of Diophantine equations to the late 60s, no such relation was known, rendering the problem one of the most baffling questions for mathematicians and computer scientists alike.

In 1970, Yuri Matiyasevich [27] discovered that $v = \mathrm{fib}_{2u}$ is both a Robinson predicate and Diophantine. Here $(\mathrm{fib}_n)_{n\in\mathbb{N}}$ is the well known Fibonacci sequence defined by the second order recurrence relation $\mathrm{fib}_0 = 0$, $\mathrm{fib}_1 = 1$ and $\mathrm{fib}_{n+2} = \mathrm{fib}_{n+1} + \mathrm{fib}_n$. Combined with previous results, this concluded the multi-decades effort to establish the Diophantineness of all recursively enumerable predicates, implying a negative solution to Hilbert's tenth problem. That proof which included the original proof of Matiyasevich [27] was later simplified. For instance, exploiting similar ideas but in the easier context of the solutions of another second order equation – namely Pell's equations $x^2 - (a^2-1)y^2 = 1$ with parameter $a > 1$, – Martin Davis [6] gave a standalone proof of the DPRM-theorem where recursively enumerable predicates are characterised by a variant of μ-recursive functions. In that paper, Davis also provided a proof of the admissibility of bounded universal quantification using the Chinese remainder theorem to encode finite sequences of numbers. There exists more recent and simpler proofs of this admissibility result as well, see e.g. [28].

Before we discuss the mechanisation of the Diophantineness of both the exponential relation and of bounded universal quantification, we want to remark on the difficulty of mechanising the former proof. Both on its own and as a stepping stone towards the negative solution to Hilbert's tenth problem, it is clear that Matiyasevich's theorem was an extremely difficult question which required superior intellectual resources to be solved. The mechanisation of a modernised form of the proof, although not trivial, cannot be compared to the difficulty of finding a solution. In particular, the modern proof relies on very mature background theories, lowering the number of possible design choices for the mechanisation. Moreover, very detailed pen and paper accounts of the proof are available, which can be followed closely.

An aspect that is more challenging in mechanisation than on paper are proofs regarding the computability of certain functions. Since paper proofs oftentimes rely on a vague notion of algorithm, most of the reasoning about these algorithms is hand-waved away by computer scientists, relying on the implicit understanding of what is an algorithm. By using a synthetic approach to computability [10, 11, 16], we make the notion of an algorithm precise and thus enable mechanisation, at the same time circumventing the verification of low-level programs.

5.1. **Exponential is Diophantine (Theorem 7).** For our mechanised proof, we rely on a more recent account of Matiyasevich's theorem from [29], which, among the many options we considered, seemed the shortest. The proof employs the equation $x^2 - bxy + y^2 = 1$ for $b \geq 2$, also called Pell's equation in [6]. We use the second order recurrence relation $\alpha_b(-1) = -1$, $\alpha_b(0) = 0$ and $\alpha_b(n+2) = b\alpha_b(n+1) - \alpha_b(n)$ to describe the set of solutions of Pell's equation by $\{(\alpha_b(n), \alpha_b(n+1)) \mid n \in \mathbb{N}\}$. The recurrence can be characterised by the following square 2×2 matrix equation:

$$A_b(n) = (B_b)^n \quad \text{with } A_b(n) := \begin{pmatrix} \alpha_b(n+1) & -\alpha_b(n) \\ \alpha_b(n) & -\alpha_b(n-1) \end{pmatrix} \text{ and } B_b := \begin{pmatrix} b & -1 \\ 1 & 0 \end{pmatrix}$$

Then, studying the properties of the sequence $n \mapsto \alpha_b(n)$ in $\mathbb{N}$ or $\mathbb{Z}$, one can show that $\alpha_2(n) = n$ for all n and $n \mapsto \alpha_b(n)$ grows exponentially for $b \geq 3$. Studying the properties of the same sequence in $\mathbb{Z}/p\mathbb{Z}$ (for varying values of the modulus p), one can for instance show that $n = \alpha_2(n) \equiv \alpha_b(n)\,[b-2]$, which relates n and $\alpha_b(n)$ modulo $(b-2)$. With various intricate but elementary results;[14] such as e.g. $\alpha_b(k) \mid \alpha_b(m) \leftrightarrow k \mid m$ and $\alpha_b^2(k) \mid \alpha_b(m) \leftrightarrow k\alpha_b(k) \mid m$ (both for $b \geq 2$ and any $k, m \in \mathbb{N}$), one can show that $a, b, c \mapsto 3 < b \land a = \alpha_b(c)$ has a Diophantine representation. In our formalisation, we get a Diophantine logic formula of size 1445 as a witness (see `dio_rel_alpha_size`).

Once $\alpha_b(n)$ is proven Diophantine, one can recover the exponential relation $x, y, z \mapsto x = y^z$ using the eigenvalue λ of the matrix B_b which satisfies $\lambda^2 - b\lambda - 1 = 0$. By wisely choosing $m := bq - q^2 - 1$, one gets $\lambda \equiv q\,[m]$ and thus, using the corresponding eigenvector, one derives $q\alpha_b(n) - \alpha_b(n-1) \equiv q^n\,[m]$. For a large enough value of m, hence a large enough value[15] of b, this gives a Diophantine representation of q^n. In our code, we get a Diophantine logic formula of size 4903 as a witness (see `dio_fun_expo_example_size`).

The main libraries which are needed to solve Pell's equation and characterise its solutions are linear algebra (or at least square 2×2 matrices) over commutative rings such as $\mathbb{Z}$ and $\mathbb{Z}/p\mathbb{Z}$, a good library for modular algebra ($\mathbb{Z}/p\mathbb{Z}$), and the binomial theorem over rings. Without the help of the Coq `ring` tactic, such a development would be extremely painful. These libraries are then used again to derive the Diophantine encoding of the exponential.

5.2. Admissibility of Bounded Universal Quantification (Theorem 8).

As hinted earlier, we provide an implementation of the algorithm for the elimination of bounded universal quantification described in [28]. It does not involve the use of a model of computation, hence does not create a chicken-and-egg problem when used for the proof of the DPRM theorem. The technique of [28] uses the exponential function and thus Theorem 7 (a lot), and a combination of arithmetic and bitwise operations over $\mathbb{N}$ through base 2 and base 2^q representations of natural numbers.

The Diophantine admissibility of bitwise operations over $\mathbb{N}$ is based on the relation stating that every bit of a is lower or equal than the corresponding bit in b and denoted $a \preccurlyeq b$. The equation $a \preccurlyeq b \leftrightarrow \mathcal{C}_b^a$ is odd[16] gives a Diophantine representation for $a \preccurlyeq b$ and then bitwise operators are derived from $\preccurlyeq$ in combination with regular addition $+$, in particular, the *digit by digit AND* operation called "projection." To obtain that $a \preccurlyeq b$ holds if and only if $\mathcal{C}_b^a \equiv 1\,[2]$, we prove Lucas's theorem [26] which allows for the computation of the binomial coefficient in base p. It states that $\mathcal{C}_b^a \equiv \mathcal{C}_{b_n}^{a_n} \times \cdots \times \mathcal{C}_{b_0}^{a_0}\,[p]$ holds when p is prime and $a = a_n p^n + \cdots + a_0$ and $b = b_n p^n + \cdots + b_0$ are the respective base p representations of a and b; see Appendix B for an elementary combinatorial proof of Lucas's theorem.

A Diophantine representation of the binomial coefficient can be obtained via e.g. the binomial theorem: $\mathcal{C}_n^k$ is the k-th digit of the development of $(1+q)^n = \sum_{i=0}^n \mathcal{C}_n^i q^i$ in base $q = 2^{n+1}$. This gives a Diophantine representation using the `is_digit` relation of Lemma 9.

The rest of the admissibility proof for bounded universal quantification $\forall i,\ i < n \to A$ is a very nice encoding of vectors of natural numbers of type $\mathbb{N}^n$ into natural numbers $\mathbb{N}$ such that regular addition $+$ (resp. multiplication $\times$) somehow performs parallel/simultaneous additions (resp. multiplications) on the encoded vectors. More precisely, a vector $(a_1, \ldots, a_n) \in [0, 2^q - 1]^n$ of natural numbers is encoded as the "cipher" $a_1 r^2 + a_2 r^4 + a_3 r^8 + \cdots + a_n r^{2^n}$

[14]by elementary we certainly do not mean either simple or obvious, but we mean that they only involve standard tools from modular and linear algebra.

[15]the largeness of which is secured using α itself again, but with other input values. But this works only in the case where $n > 0$ and $q > 0$. The cases where $n = 0$ (and hence $q^n = 1$) or $q = 0$ and $n > 0$ (and hence $q^n = 0$) are trivial and treated separately.

[16]where $\mathcal{C}_b^a$ denotes the binomial coefficient with the usual convention that $\mathcal{C}_b^a = 0$ when $a > b$.

with $r = 2^{4q}$. In these *sparse ciphers,* only the digits occurring at r^{2^i} are non-zero. We remark that none of the parameters, including n or q, are constant in the encoding.

Besides the low-level inductive proof of Lucas's theorem presented in Appendix B, the essential library for the removal of bounded universal quantification consists of tools to manipulate the type $\mathbb{N}$ simultaneously and smoothly both as (a) usual natural numbers and (b) sparse base $r = 2^{4q}$ encodings of vectors of natural numbers in $[0, 2^q - 1]$. Notice that r is defined as $r = 2^{2q}$ in [28] but we favour the alternative choice $r = 2^{4q}$ which allows for an easier soundness proof for vector multiplication because there is no need to manage for digit overflows (see Appendix C).

A significant step in the Diophantine encoding of $+$ and $\times$ on $\mathbb{N}^n$ is the Diophantine encoding of $u = \sum_{i=1}^{n} r^{2^i}$ and $u' = \sum_{i=2}^{n+1} r^{2^i}$ as the ciphers of the constant vectors $[1; \ldots; 1] \in \mathbb{N}^n$ and $[0; 1; \ldots; 1] \in \mathbb{N}^{n+1}$ respectively, obtained by masking u^2 with $w = \sum_{i=0}^{2^{n+1}} r^i$ and $2w$.

Finally, it should be noted that prior to the elimination of the quantifier in $\forall i,\ i < n \to A$, the Diophantine formula A is first normalised into a conjunction of elementary constraints using Theorem 11, and then the elimination is performed on that list of elementary constraints, encoding e.g. $v_0 \doteq v_1 \dot{+} v_2$ and $v_0 \doteq v_1 \dot{\times} v_2$ with their respective sparse cipher counterparts.

6. Minsky Machines Reduce to FRACTRAN

In previous work, we have reduced the halting problem for Turing machines to PCP [10] and on to a specialised halting problem for Minsky machines [16] in Coq. The specialised halting problem asked whether a machine on a given input halts in a configuration with all registers containing zeros. In order to define Minsky machine recognisability, we consider a general halting problem which allows *any* final configuration, final meaning that computation cannot further proceed. The adaptation of the formal proofs reducing PCP via binary stack machines to Minsky machines is quite straightforward and reuses the certified compiler for low-level languages defined in [16].

We first show that one can remove self loops from Minsky machines, i.e. instructions which jump to their own location, using the compositional reasoning techniques developed in [16]. We then formalise the FRACTRAN language [4] and show how the halting problem for Minsky machines can be encoded into the halting problem for FRACTRAN programs. While the verification of Minsky machines can be complex and needs preliminary thoughts on compositional reasoning, the translation from Minsky machines to FRACTRAN is elementary and needs no heavy machinery.

6.1. Minsky Machines.

We employ Minsky machines [31] with instructions $\iota : \mathsf{I}_n ::= \texttt{INC}\ (\alpha : \mathbb{F}_n) \mid \texttt{DEC}\ (\alpha : \mathbb{F}_n)\ (p : \mathbb{N})$. A Minsky machine with n registers is a sequence of consecutively indexed instructions $s : \iota_0;\ \ldots\ s + k : \iota_k$; represented as a pair $(s : \mathbb{N}, [\iota_0; \ldots; \iota_k] : \mathbb{L}\,\mathsf{I}_n)$. Its state $(i, \vec{v})$ is a program counter (PC) value $i : \mathbb{N}$ and a vector of values for registers $\vec{v} : \mathbb{N}^n$. $\texttt{INC}\ \alpha$ increases the value of register α and the PC by one. $\texttt{DEC}\ \alpha\ p$ decreases the value of register α by one if that is possible and increases the PC, or, if the register is already 0, jumps to PC value p. Given a Minsky machine (s, P), we write $(s, P) \mathbin{/\!/_M} (i_1, \vec{v}_1) \succ^k (i_2, \vec{v}_2)$ when (s, P) transforms state $(i_1, \vec{v}_1)$ into $(i_2, \vec{v}_2)$ in k steps of computation. For (s, P) to do a step in state $(i, \vec{v})$ the instruction at label i in (s, P) is considered. We define $\texttt{out}\ i\ (s, P) := i < s \lor |P| + s \leq i$ to characterize when label i is outside of the code of (s, P). In that case (and only that case), no computation step can occur. We define the halting problem for Minsky Machines as

$$\mathsf{MM}(n : \mathbb{N}, P : \mathbb{L}\,\mathsf{I}_n, \vec{v} : \mathbb{N}^n) := (1, P) \;/\!\!/_M\; (1, \vec{v}) \downarrow$$

$$\text{where } \begin{cases} (s,P) \;/\!\!/_M\; (i,\vec{v}) \rightsquigarrow (j,\vec{w}) := \exists k,\ (s,P) \;/\!\!/_M\; (i,\vec{v}) \succ^k (j,\vec{w}) \wedge \texttt{out}\ j\ (s,P) \\ (s,P) \;/\!\!/_M\; (i,\vec{v}) \downarrow := \exists j\,\vec{w},\ (s,P) \;/\!\!/_M\; (i,\vec{v}) \rightsquigarrow (j,\vec{w}) \end{cases}$$

meaning that the machine $(1, P)$ has a terminating computation starting at state $(1, \vec{v})$, the value of the final state being irrelevant. Notice that since $\mathsf{MM}(n, P, \vec{v})$ considers only Minsky machines starting at PC value 1, the PC value 0 is always outside of their code (i.e. $\texttt{out}\ 0\ (1, P)$ holds), hence computations can be halted by jumping there. We refer to [16] for a more in-depth formal description of those counter machines. Note that the halting problem defined there as $\mathsf{MM}_0(n : \mathbb{N}, P : \mathbb{L}\,\mathsf{I}_n, \vec{v} : \mathbb{N}^n) := (1, P) \;/\!\!/_M\; (1, \vec{v}) \rightsquigarrow (0, \vec{0})$ is more specific than the problem MM above defined but both are proved undecidable in our library.

We say that a machine *has a self loop* if it contains an instruction of the form $i : \texttt{DEC}\ \alpha\ i$, i.e. jumps to itself in case the register α has value 0, leading necessarily to non-termination (in case the PC reaches value i). For every machine P with self loops, we can construct an equivalent machine Q using one additional register α_0 with constant value 0, which has the same behaviour but no self loops. Since the effect of a self loop $i : \texttt{DEC}\ \alpha\ i$ is either decrement and move to the next instruction at $i + 1$ if $\alpha > 0$ or else enter in a forever loop at i, it is easily simulated by a jump to a length-2 cycle, i.e. replacing $i : \texttt{DEC}\ \alpha\ i$ with $i : \texttt{DEC}\ \alpha\ j$ and adding $j : \texttt{DEC}\ \alpha_0\ (j + 1); j + 1 : \texttt{DEC}\ \alpha_0\ j$ somewhere near the end of the program.

Theorem 19. *Given a Minsky machine P with n registers one can compute a machine Q with $1 + n$ registers and no self loops s.t. for any $\vec{v}$,*

$$(1, P) \;/\!\!/_M\; (1, \vec{v}) \downarrow \;\leftrightarrow\; (1, Q) \;/\!\!/_M\; (1, 0 :: \vec{v}) \downarrow.$$

Proof. We explain how any Minsky machine $(1, P)$ with n registers can be transformed into an equivalent one that uses an extra 0 valued spare register $\alpha_0 = 0 \in \mathbb{F}_{1+n}$ and avoids self loops. Let k be the length of P and let P' be the Minsky machine with $1 + n$ registers defined by performing a 1-1 replacement of instructions of $(1, P)$:

- instructions of the form $i : \texttt{INC}\ \alpha$ are replaced by $i : \texttt{INC}\ (1 + \alpha)$;
- self loops $i : \texttt{DEC}\ \alpha\ i$ are replaced by $i : \texttt{DEC}\ (1 + \alpha)\ (2 + k)$;
- proper inside jumps $i : \texttt{DEC}\ \alpha\ j$ for $i \neq j$ and $1 \leq j \leq k$ are replaced by $i : \texttt{DEC}\ (1 + \alpha)\ j$;
- and outside jumps $i : \texttt{DEC}\ \alpha\ j$ for $j = 0 \vee k < j$ are replaced by $i : \texttt{DEC}\ (1 + \alpha)\ 0$.

Then we define $Q := P' +\!\!+ [\texttt{DEC}\ \alpha_0\ 0; \texttt{DEC}\ \alpha_0\ (3 + k); \texttt{DEC}\ \alpha_0\ (2 + k)]$. Notice that P' is immediately followed $\texttt{DEC}\ \alpha_0\ 0$, i.e. by an unconditional jump to 0 (because α_0 has value 0), and that $(1, Q)$ ends with the length-2 cycle composed of $2 + k : \texttt{DEC}\ \alpha_0\ (3 + k); 3 + k : \texttt{DEC}\ \alpha_0\ (2 + k)$. We show that $(1, Q)$ is a program without self loops (obvious) that satisfies the required simulation equivalence. Indeed, self loops are replaced by jumps to the length-2 cycle that uses the unmodified register α_0 to loop forever. One should just be careful that the outside jumps of $(1, P)$ do not accidentally fall into that cycle and this is why we redirect them all to PC value 0, which halts the computation because $\texttt{out}\ 0\ (1, Q)$ holds. □

A predicate $R : \mathbb{N}^n \to \mathbb{P}$ *is MM-recognisable* if there exist $m : \mathbb{N}$ and a Minsky machine $P : \mathbb{L}\,\mathsf{I}_{n+m}$ of $(n+m)$ registers such that for any $\vec{v} : \mathbb{N}^n$ we have $R\,\vec{v} \leftrightarrow (1, P) \;/\!\!/_M\; (1, \vec{v} +\!\!+ \vec{0}) \downarrow$. The last m registers serve as spare registers during the computation. Notice that not allowing for spare registers would make e.g. the empty predicate un-recognisable.[17] It is possible to limit the number of (spare) registers but that question is not essential in our development.

6.2. **The FRACTRAN language.** We formalise the language FRACTRAN, introduced as a universal programming language for arithmetic by Conway [4]. A FRACTRAN program Q consists of a list of positive fractions $[p_1/q_1; \ldots; p_n/q_n]$. The current state of a FRACTRAN program is just a natural number s. The first fraction p_i/q_i in Q such that $s \cdot (p_i/q_i)$ is still integral determines the successor state, which then is $s \cdot (p_i/q_i)$. If there is no such fraction in Q, the program terminates.

We make this precise inductively for Q being a list of fractions $p/q : \mathbb{N} \times \mathbb{N}$:

$$\frac{q \cdot y = p \cdot x}{(p/q :: Q) \;/\!/_F\; x \succ y} \qquad \frac{q \nmid p \cdot x \qquad Q \;/\!/_F\; x \succ y}{(p/q :: Q) \;/\!/_F\; x \succ y}$$

i.e. at state x the first fraction p/q in Q where q divides $p \cdot x$ is used, and x is multiplied by p and divided by q. For instance, the FRACTRAN program $[5/7; 2/1]$ runs forever when starting from state 7, producing the sequence $5 = 7 \cdot (5/7)$, $10 = 5 \cdot (2/1)$, $20 = 10 \cdot (2/1)$ ...[18]

We say that a FRACTRAN program $Q = [p_1/q_1; \ldots; p_n/q_n]$ is *regular* if none of its denominators is 0, i.e. if $q_1 \neq 0, \ldots, q_n \neq 0$. For a FRACTRAN program $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ and $s : \mathbb{N}$, we define the decision problem as the question "does Q halt when starting from s":

$$\mathsf{FRACTRAN}(Q, s) := Q \;/\!/_F\; s\downarrow \quad \text{with } Q \;/\!/_F\; s\downarrow := \exists x,\, Q \;/\!/_F\; s \succ^* x \wedge \forall y,\, \neg\, Q \;/\!/_F\; x \succ y$$

Following [4], we now show how (regular) FRACTRAN halting can be used to simulate Minsky machines halting. The idea is to use a simple Gödel encoding of the states of a Minsky machine. We first fix two infinite sequences of prime numbers $\mathfrak{p}_0, \mathfrak{p}_1, \ldots$ and $\mathfrak{q}_0, \mathfrak{q}_1, \ldots$ all distinct from each other. We define the encoding of n-register Minsky machine states as $\overline{(i, \vec{v})} := \mathfrak{p}_i \mathfrak{q}_0^{x_0} \cdots \mathfrak{q}_{n-1}^{x_{n-1}}$ where $\vec{v} = [x_0, \ldots, x_{n-1}]$:

- To simulate the step semantics of Minsky machines for $i : \texttt{INC}\ \alpha$, we divide the encoded state by $\mathfrak{p}_i$ and multiply by $\mathfrak{p}_{i+1}$ for the change in PC value, and increment the register α by multiplying with $\mathfrak{q}_\alpha$, hence we add the fraction $\mathfrak{p}_{i+1}\mathfrak{q}_\alpha/\mathfrak{p}_i$;
- To simulate $i : \texttt{DEC}\ \alpha\ j$ when $\vec{v}_\alpha = 1 + n$ we divide by $\mathfrak{p}_i$, multiply by $\mathfrak{p}_{i+1}$ and decrease register α by dividing by $\mathfrak{q}_\alpha$, hence we add the fraction $\mathfrak{p}_{i+1}/\mathfrak{p}_i\mathfrak{q}_\alpha$;
- To simulate $i : \texttt{DEC}\ \alpha\ j$ when $\vec{v}_\alpha = 0$ we divide by $\mathfrak{p}_i$ and multiply by $\mathfrak{p}_j$. To make sure that this is only executed when the previous rule does not apply, we add the fraction $\mathfrak{p}_j/\mathfrak{p}_i$ *after* the fraction $\mathfrak{p}_{i+1}/\mathfrak{p}_i\mathfrak{q}_\alpha$.

In short, we define the encoding of labelled instructions and then programs as

$$\begin{aligned}\overline{(i, \texttt{INC}\ \alpha)} &:= [\mathfrak{p}_{i+1}\mathfrak{q}_\alpha/\mathfrak{p}_i] \\ \overline{(i, \texttt{DEC}\ \alpha\ j)} &:= [\mathfrak{p}_{i+1}/\mathfrak{p}_i\mathfrak{q}_\alpha; \mathfrak{p}_j/\mathfrak{p}_i]\end{aligned} \qquad \overline{(i, [\iota_0; \ldots; \iota_k])} := \overline{(i, \iota_0)} +\!\!+ \cdots +\!\!+ \overline{(i+k, \iota_k)}.$$

Notice that we only produce regular programs and that a self loop like $i : \texttt{DEC}\ \alpha\ i$, jumping on itself when $\vec{v}_\alpha = 0$, will generate the fraction $\mathfrak{p}_i/\mathfrak{p}_i$ potentially capturing any state $\overline{(j, \vec{v})}$ even when $j \neq i$. So this encoding does not work on Minsky machines containing self loops because the corresponding FRACTRAN program would never terminate, even when the PC never reaches self loops.

[17]For any Minsky machine $(1, P)$, if it starts on large enough register values, for instance if they are all greater than the length of P, then no jump can occur and the machine terminates after its last instruction executes. Such unfortunate behavior can be circumvented with a 0-valued spare register.

[18]No FRACTRAN program can ever stop when it contains a fraction having an integer value like 2/1.

Lemma 20. *If* $(1, P)$ *has no self loops then* $(1, P) \mathbin{/\!\!/_M} (1, \vec{v}) \downarrow \leftrightarrow \overline{(1, P)} \mathbin{/\!\!/_F} \overline{(1, \vec{v})} \downarrow$.

Proof. Let (i, P) be a Minsky machine with no self loops. We show that the simulation of (i, P) by $\overline{(i, P)}$ is 1-1, i.e. each step is simulated by one step. We first show the forward simulation, i.e. that $(i, P) \mathbin{/\!\!/_M} (i_1, \vec{v}_1) \succ (i_2, \vec{v}_2)$ entails $\overline{(i, P)} \mathbin{/\!\!/_F} \overline{(i_1, \vec{v}_1)} \succ \overline{(i_2, \vec{v}_2)}$, by case analysis. Conversely we show that if $\overline{(i, P)} \mathbin{/\!\!/_F} \overline{(i_1, \vec{v}_1)} \succ st$ holds then $st = \overline{(i_2, \vec{v}_2)}$ for some $(i_2, \vec{v}_2)$ such that $(i, P) \mathbin{/\!\!/_M} (i_1, \vec{v}_1) \succ (i_2, \vec{v}_2)$. Backward simulation involves the totality of MM one step semantics and the determinism of regular FRACTRAN one step semantics combined with the forward simulation.

Using these two simulation results, the desired equivalence follows by induction on the length of terminating computations. □

Theorem 21. *For any n-register Minsky machine P one can compute a regular FRACTRAN program Q s.t.* $(1, P) \mathbin{/\!\!/_M} (1, [x_1; \dots; x_n]) \downarrow \leftrightarrow Q \mathbin{/\!\!/_F} \mathfrak{p}_1 \mathfrak{q}_1^{x_1} \cdots \mathfrak{q}_n^{x_n} \downarrow$ *holds for any* $x_1, \dots, x_n$.

Proof. Using Theorem 19, we first compute a Minsky machine $(1, P_1)$ equivalent to $(1, P)$ but with one extra 0-valued spare register and no self loops. Then we apply Lemma 20 to $(1, P_1)$ and let $Q := \overline{(1, P_1)}$. The program Q is obviously regular and given $\vec{v} = [x_1; \dots; x_n]$, the encoding of the starting state $(1, 0 :: \vec{v})$ for $(1, P_1)$ is $\mathfrak{p}_1 \mathfrak{q}_0^0 \mathfrak{q}_1^{x_1} \cdots \mathfrak{q}_n^{x_n}$ hence the result. □

This gives us a formal constructive proof that (regular) FRACTRAN is Turing complete as a model of computation and is consequently undecidable.

Corollary 22. *Halt reduces to FRACTRAN.*

Proof. Theorem 21 gives us a reduction from MM to FRACTRAN which can be combined with the reduction of Halt to PCP from [10] and a slight modification of PCP to MM from [16]. □

7. Diophantine Encoding of FRACTRAN

We show that a single step of FRACTRAN computation is a Diophantine relation.

Lemma 23. *For any FRACTRAN program* $Q : \mathbb{L}(\mathbb{N} \times \mathbb{N})$, *one can compute a map*

$$\forall f\, g : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N},\ \mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{fun}}\, g \to \mathbb{D}_{\text{rel}}\, (\lambda \nu.\, Q \mathbin{/\!\!/_F} f\,\nu \succ g\,\nu).$$

Proof. The map is built by induction on Q. If $Q = [\,]$, then we show $[\,] \mathbin{/\!\!/_F} f\,\nu \succ g\,\nu \leftrightarrow \texttt{False}$, and thus $\mathbb{D}_{\text{rel}}\, (\lambda \nu. Q \mathbin{/\!\!/_F} f\,\nu \succ g\,\nu)$ by map 4 of Proposition 2 followed by `dio auto`. If Q is a composed list $Q = p/q :: Q'$, then we show the equivalence

$$(p/q :: Q') \mathbin{/\!\!/_F} f\,\nu \succ g\,\nu \ \leftrightarrow\ \big(q \cdot (g\,\nu) = p \cdot (f\,\nu)\big) \lor \big(q \nmid (p \cdot (f\,\nu))\big) \land \big(Q' \mathbin{/\!\!/_F} f\,\nu \succ g\,\nu\big)$$

and we derive $\mathbb{D}_{\text{rel}}\, (\lambda \nu. Q \mathbin{/\!\!/_F} f\,\nu \succ g\,\nu)$ by map 4 of Proposition 2 followed by `dio auto`, the induction hypothesis being used locally as a hint for the tactic. □

In addition, the "Q has terminated at x" predicate is Diophantine for any FRACTRAN program Q. The proof is similar to the previous one:

Lemma 24. *For any FRACTRAN program* $Q : \mathbb{L}(\mathbb{N} \times \mathbb{N})$, *one can compute a map*

$$\forall f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N},\ \mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{rel}}\, (\lambda \nu. \forall y, \neg Q \mathbin{/\!\!/_F} f\,\nu \succ y).$$

Proof. The map $\forall f, \mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{rel}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!\!/_F} f\,\nu \succ y)$ is built by induction on Q. If $Q = [\,]$, then we show $(\forall y, \neg\,[\,] \mathbin{/\!\!/_F} f\,\nu \succ y) \leftrightarrow \texttt{True}$, and thus $\mathbb{D}_{\text{rel}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!\!/_F} f\,\nu \succ y)$ by map 4 of Proposition 2 followed by `dio auto`. If $Q = p/q :: Q'$, then we show the equivalence

$$\forall y, \neg\, Q \mathbin{/\!\!/_F} f\,\nu \succ y \;\leftrightarrow\; \big(q \nmid (p \cdot (f\,\nu))\big) \wedge \big(\forall y, \neg\, Q' \mathbin{/\!\!/_F} f\,\nu \succ y\big)$$

and we get $\mathbb{D}_{\text{rel}}\,(\lambda\nu.\forall y, \neg\, Q \mathbin{/\!\!/_F} f\,\nu \succ y)$ by map 4 of Proposition 2 followed by `dio auto`, the induction hypothesis being used as a hint again. □

We can now deduce a core result of the paper which states that FRACTRAN programs have Diophantine termination predicates.

Theorem 25. *If $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ is a FRACTRAN program then one can compute a map*

$$\forall f : (\mathsf{V} \to \mathbb{N}) \to \mathbb{N}, \mathbb{D}_{\text{fun}}\, f \to \mathbb{D}_{\text{rel}}\,(\lambda\nu.\, Q \mathbin{/\!\!/_F} f\,\nu \downarrow).$$

Proof. By definition we have $Q \mathbin{/\!\!/_F} f\,\nu \downarrow \leftrightarrow \exists x\,(Q \mathbin{/\!\!/_F} f\,\nu \succ^* x \wedge \forall y, \neg\, Q \mathbin{/\!\!/_F} x \succ y)$ and hence we obtain the claim using Theorem 10 together with Lemma 23 and Lemma 24. □

We conclude with the undecidability of Hilbert's tenth problem by a reduction chain starting from the Halting problem for single tape Turing machines:

Theorem 26 (Hilbert's tenth problem). *We have the following reduction chain*

$$\mathsf{Halt} \preceq \mathsf{PCP} \preceq \mathsf{MM} \preceq \mathsf{FRACTRAN} \preceq \mathsf{DIO\_FORM} \preceq \mathsf{DIO\_ELEM} \preceq \mathsf{DIO\_SINGLE} \preceq \mathsf{H10}$$

and as a consequence, H10 is undecidable.

Proof. The proof combines the previous results like Theorems 21 and 25 and Corollary 17. □

8. The Davis-Putnam-Robinson-Matiyasevich Theorem

We give a proof of an instance of the DPRM theorem stating that recursively enumerable predicates are Diophantine.[19] Here we assume that the informal notion of "recursive enumerability" (justified by Church's thesis) can be characterised by Minsky machines recognisability as defined in Section 6.1.

Proposition 27. *The Gödel encoding is Diophantine, i.e. we have $\mathbb{D}_{\text{fun}}\,(\lambda\nu.\mathfrak{q}_1^{\nu\,x_0} \cdots \mathfrak{q}_n^{\nu\,x_{n-1}})$.*

Proof. By induction on $n : \mathbb{N}$ using Proposition 4 and Theorem 7. Notice that the $\mathfrak{q}_i$'s are hard-coded in the Diophantine representation, which means we do not need to encode the algorithm that actually computes them, which would otherwise be very painful. □

Lemma 28. *For any FRACTRAN program Q we have $\mathbb{D}_{\text{rel}}\,(\lambda\nu.\, Q \mathbin{/\!\!/_F} \mathfrak{p}_1\mathfrak{q}_1^{\nu\,x_0} \cdots \mathfrak{q}_n^{\nu\,x_{n-1}} \downarrow)$.*

Proof. By Theorem 25, we only have to show $\mathbb{D}_{\text{fun}}\, f$ for $f\,\nu := \mathfrak{p}_1\mathfrak{q}_1^{\nu\,x_0} \cdots \mathfrak{q}_n^{\nu\,x_{n-1}}$. This follows from Propositions 4 and 27. □

To simplify the notation $[\![p]\!]_{\vec{v}}^{\vec{w}}$ below, we abusively identify the vector $\vec{v} : \mathbb{N}^n$ (resp. $\vec{w} : \mathbb{N}^m$) with the valuation $\lambda(i : \mathbb{F}_n).\vec{v}_i$ (resp. $\lambda(j : \mathbb{F}_m).\vec{w}_j$) that accesses the components of the vector $\vec{v}$ (resp. $\vec{w}$).

[19]By instance, we mean that the DPRM is an open theorem bound to be extended for any newly proposed Turing complete model of computation.

Theorem 29 (DPRM). *Any MM-recognisable relation $R : \mathbb{N}^n \to \mathbb{P}$ is Diophantine: one can compute a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$ with n parameters and m variables s.t. $\forall \vec{v} : \mathbb{N}^n,\ R\ \vec{v} \leftrightarrow \exists \vec{w} : \mathbb{N}^m,\ [\![p]\!]_{\vec{v}}^{\vec{w}} = [\![q]\!]_{\vec{v}}^{\vec{w}}$.*

Proof. By definition, $R : \mathbb{N}^n \to \mathbb{P}$ is recognised by some Minsky machine P with $(n+m)$ registers, i.e. $R\ \vec{v} \leftrightarrow (1, P) \mathbin{/\!\!/_M} (1, \vec{v} \mathbin{+\!\!+} \vec{0}) \downarrow$. By Theorem 21, we compute a $\mathsf{FRACTRAN}$ program Q s.t. $(1, P) \mathbin{/\!\!/_M} (1, [v_1; \dots; v_n; w_1; \dots; w_m]) \downarrow \leftrightarrow Q \mathbin{/\!\!/_F} \mathfrak{p}_1 \mathfrak{q}_1^{v_1} \cdots \mathfrak{q}_n^{v_n} \mathfrak{q}_{n+1}^{w_1} \cdots \mathfrak{q}_{n+m}^{w_m} \downarrow$.

Hence we deduce $R\ [v_1; \dots; v_n] \leftrightarrow Q \mathbin{/\!\!/_F} \mathfrak{p}_1 \mathfrak{q}_1^{v_1} \cdots \mathfrak{q}_n^{v_n} \downarrow$. As a consequence, the relation $\lambda\nu.R\ [\nu\, x_0; \dots; \nu\, x_{n-1}]$ is Diophantine by Lemma 28. By Corollary 17, there is a Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{N}, \mathsf{V})$ such that $R\ [\nu\, x_0; \dots; \nu\, x_{n-1}] \leftrightarrow \exists \varphi,\ [\![p]\!]_\nu^\varphi = [\![q]\!]_\nu^\varphi$. Notice that the value $\nu\, x_i$ of any parameter of $p \doteq q$ greater than x_n does not influence its solvability.

Now let m be an upper bound of the number of (existentially quantified) variables in $p \doteq q$. We injectively map those variables in $\mathbb{F}_m$ and we project the parameters of $p \doteq q$ onto $\mathbb{F}_n$ by replacing every parameter greater than x_n with the 0 constant. We get a Diophantine equation $p' \doteq q' : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$ of which the solvability at $\vec{v}$ is equivalent to $R\ \vec{v}$. □

9. Hilbert's Tenth Problem Over Integers

In our formalisation, polynomials are defined over natural numbers, that is both constants and solutions come from $\mathbb{N}$. The standard way to extend the undecidability of $\mathsf{H10}$ to a formalisation based on integers is via Lagrange's theorem, stating that an integer is positive if and only if it is the sum of four squares.

Similar to Definition 15, we define define polynomials over integers:

Definition 30. *The type of* Diophantine polynomials $\mathbb{D}_{\text{poly}}^{\mathbb{Z}}(\mathsf{U}, \mathsf{V})$ *over $\mathbb{Z}$ is defined by:*

$$p, q : \mathbb{D}_{\text{poly}}^{\mathbb{Z}}(\mathsf{U}, \mathsf{V}) ::= u : \mathsf{U} \mid x : \mathsf{V} \mid z : \mathbb{Z} \mid p \dotplus q \mid p \dot{\times} q.$$

The interpretation of a polynomial $p : \mathbb{D}_{\text{poly}}^{\mathbb{Z}}(\mathsf{U}, \mathsf{V})$ in $\mathbb{Z}$ given $\varphi : \mathsf{U} \to \mathbb{Z}$, $\nu : \mathsf{V} \to \mathbb{Z}$ and denoted $[\![p]\!]_\nu^\varphi$, is defined in the obvious way. Again, if $p : \mathbb{D}_{\text{poly}}^{\mathbb{Z}}(\mathbb{F}_m, \mathbb{F}_n)$ we abusively write $[\![p]\!]_{\vec{v}}^{\vec{w}}$ when $\vec{v} : \mathbb{N}^n$ and $\vec{w} : \mathbb{N}^m$. We can then define

$$\mathsf{H10}_{\mathbb{Z}}(n, p : \mathbb{D}_{\text{poly}}^{\mathbb{Z}}(\mathbb{F}_n, \mathbb{F}_0)) := \exists \vec{w} : \mathbb{N}^n,\ [\![p]\!]_{[]}^{\vec{w}} = 0$$

that is "does the polynomial equation $p(x_1, \dots, x_n) = 0$ in (at most) n variables have a solution in $\mathbb{Z}$?" We first outline a proof of Lagrange's theorem and then reduce $\mathsf{H10}$ to $\mathsf{H10}_{\mathbb{Z}}$.

9.1. Lagrange's theorem.

The proof we have implemented roughly follows the "classical proof" in Wikipedia's account of the theorem. Their use of the "classical" qualifier should be understood as typical/standard, and certainly not as opposed to constructive/intuitionistic. The below proof perfectly fits in our constructive setting.

Proposition 31 (Euler, 1748). *Let us assume the two equations $n = a_1^2 + b_1^2 + c_1^2 + d_1^2$ and $m = a_2^2 + b_2^2 + c_2^2 + d_2^2$ hold in $\mathbb{Z}$. Let us define the four relative intergers:*

$$\begin{aligned} a &:= a_1a_2 + b_1b_2 + c_1c_2 + d_1d_2 & b &:= a_1b_2 - b_1a_2 + d_1c_2 - c_1d_2 \\ c &:= a_1c_2 - c_1a_2 + b_1d_2 - d_1b_2 & d &:= a_1d_2 - d_1a_2 + c_1b_2 - b_1c_2. \end{aligned}$$

Then the identity $nm = a^2 + b^2 + c^2 + d^2$ holds.

Proof. This holds in any commutative ring and the proof just calls the `ring` tactic. □

Because of Proposition 31, "being the sum of four squares" is a multiplicative property. Hence to show that it holds for every natural number, it is enough to establish it for primes.

Theorem 32 (Prime induction). *Let $P : \mathbb{N} \to \mathbb{P}$ be a predicate. To establish $\forall n : \mathbb{N}, P\, n$, it is enough to prove these four induction steps:*

$$P\,0 \qquad P\,1 \qquad \forall a\, b : \mathbb{N}, P\, a \to P\, b \to P(ab) \qquad \forall p : \mathbb{N}, \texttt{prime}\, p \to P\, p.$$

Proof. This is one possible form of the *fundamental theorem of arithmetic*. For the proof, first show by strong induction on $n : \mathbb{N}$ that one can discriminate whether $n < 2$ or compute a prime factor of n, including the possibility that n itself is prime. Then, to prove the prime induction principle, proceed by strong induction again. □

Lemma 33. *If $p : \mathbb{N}$ is prime then $\exists n\, a\, b : \mathbb{N}, np = 1 + a^2 + b^2 \wedge 0 < n < p$.*

Proof. Let us first rule out the case $p = 2$ which has an obvious solution. So let us write $p = 2m + 1$ because all the other primes are odd. In the field $\mathbb{Z}/p\mathbb{Z}$, let us study the modular equation $a^2 \equiv -(1 + b^2)\,[p]$. Because $\mathbb{Z}/p\mathbb{Z}$ has no zero divisor, the map $a \mapsto a^2$ from $[0, m] \to \mathbb{Z}/p\mathbb{Z}$ is injective. As a consequence, so is the map $b \mapsto -(1+b^2)$ from $[0, m] \to \mathbb{Z}/p\mathbb{Z}$. Hence none of the two lists $[a^2 : \mathbb{Z}/p\mathbb{Z} \mid a \in [0, m]]$ and $[-(1 + b^2) : \mathbb{Z}/p\mathbb{Z} \mid b \in [0, m]]$ contain a duplicate. Since they have a combined length of $2(m + 1) > 2m + 1$ and their concatenation is contained in a list of length $p = 2m + 1$ enumerating $\mathbb{Z}/p\mathbb{Z}$, by the pigeon hole principle, they must intersect and this gives $a, b \in [0, m]$ such that $a^2 \equiv -(1 + b^2)\,[p]$. This in turn gives $n : \mathbb{N}$ such that $np = 1 + a^2 + b^2$. Given that $2a < p$ and $2b < p$, we deduce $0 < n < p$. □

Lagrange's theorem gives a Diophantine characterisation of those relative integers which are positive as the sum of four squares.

Theorem 34 (Lagrange, 1770). *For any relative integer $z : \mathbb{Z}$, z is positive if and only if there exists $a, b, c, d : \mathbb{Z}$ such that $z = a^2 + b^2 + c^2 + d^2$.*

Proof. It is enough to show that any natural number $n : \mathbb{N}$ is the sum of four squares, i.e. there exists $a, b, c, d : \mathbb{Z}$ such that $n = a^2 + b^2 + c^2 + d^2$. By Euler's four-square identity (see Proposition 31) and the principle of prime induction (see Theorem 32), we only need to show the property for prime numbers.

We fix a prime number p and define the predicate "mp is the sum of four squares" as

$$P(m : \mathbb{N}) := \exists a\, b\, c\, d : \mathbb{Z}, mp = a^2 + b^2 + c^2 + d^2.$$

We want to show that $P\,1$ holds. By Lemma 33, we know that $P\, n$ holds for some $0 < n < p$. We are going to decrease this value of n until it reaches 1, i.e. to establish $P\,1$, it is enough to give a proof of

$$\forall m : \mathbb{N}, 1 < m < p \to P\, m \to \exists r : \mathbb{N}, 1 \leq r < m \wedge P\, r$$

and then finish the argument by strong induction (implementing "infinite descent" here).

So let us assume $1 < m < p$ and $mp = x_1^2 + x_2^2 + x_3^2 + x_4^2$. For each $i = 1, \ldots, 4$ we compute a "small" representative of x_i in $\mathbb{Z}/m\mathbb{Z}$, i.e. $y_i : \mathbb{Z}$ such that $x_i \equiv y_i\,[m]$ and $4y_i^2 \leq m^2$. Then we have $mp = x_1^2 + x_2^2 + x_3^2 + x_4^2 \equiv y_1^2 + y_2^2 + y_3^2 + y_4^2\,[m]$ hence $y_1^2 + y_2^2 + y_3^2 + y_4^2 \equiv 0\,[m]$ and we get $r : \mathbb{N}$ such that $y_1^2 + y_2^2 + y_3^2 + y_4^2 = rm$. Because $y_1^2 + y_2^2 + y_3^2 + y_4^2 \leq m^2$, we deduce $r \leq m$. Now we rule out the cases $r = 0$ and $r = m$:

- if $r = 0$ then $y_1 = \cdots = y_4 = 0$ and thus $x_i^2 \equiv 0\,[m^2]$ for $i = 1, \ldots, 4$. As a consequence, $mp = x_1^2 + x_2^2 + x_3^2 + x_4^2 \equiv 0\,[m^2]$ and thus m divides p, contradicting the primality of p;

- if $r = m$ then $y_1^2 + y_2^2 + y_3^2 + y_4^2 = m^2$ and since $4y_i^2 \leq m^2$ holds for $i = 1, \ldots, 4$, we deduce $4y_1^2 = \cdots = 4y_4^2 = m^2$ and thus $m = 2q$ and $y_i = \pm q$ for $i = 1, \ldots, 4$. As a consequence $x_i^2 \equiv q^2\,[m^2]$ and thus $x_1^2 + x_2^2 + x_3^2 + x_4^2 \equiv 4q^2\,[m^2]$ hence $mp \equiv 0\,[m^2]$, in contradiction with the primality of p again.

So we have $1 \leq r < m$, $rm = y_1^2 + y_2^2 + y_3^2 + y_4^2$ and $x_i \equiv y_i\,[m]$ for $i = 1, \ldots, 4$. We also have $mp = x_1^2 + x_2^2 + x_3^2 + x_4^2$ and using Euler's four squares identity we get $a, b, c, d : \mathbb{Z}$ such that $(mp)(rm) = a^2 + b^2 + c^2 + d^2$. Using $x_i \equiv y_i\,[m]$ and the values of a, b, c and d as defined in Proposition 31, we show that $a \equiv 0\,[m]$, $b \equiv 0\,[m]$, $c \equiv 0\,[m]$ and $d \equiv 0\,[m]$, i.e. m divides a, b, c and d. Hence $rp = (a/m)^2 + (b/m)^2 + (c/m)^2 + (d/m)^2$ which establishes $P\,r$ for $1 \leq r < m$, as required. □

9.2. **H10 reduces to H10$_\mathbb{Z}$.**

Let $f : \mathbb{D}_{\text{poly}}(\mathbb{F}_n, \mathbb{F}_0) \to \mathbb{D}^{\mathbb{Z}}_{\text{poly}}(\mathbb{F}_{4n}, \mathbb{F}_0)$ be the function replacing every variable $i : \mathbb{F}_n$ in a polynomial by the polynomial expression

$$(4i) \dot{\times} (4i) \dot{+} (4i+1) \dot{\times} (4i+1) \dot{+} (4i+2) \dot{\times} (4i+2) \dot{+} (4i+3) \dot{\times} (4i+3).$$

The function f can now be used to define the reduction from H10 to H10$_\mathbb{Z}$:

Corollary 35. *H10 $\preceq$ H10$_\mathbb{Z}$.*

Proof. Given polynomials $p, q : \mathbb{D}_{\text{poly}}(\mathbb{F}_n, \mathbb{F}_0)$ the reduction function returns the polynomial $f(p) \dot{+} (-1) \dot{\times} f(q) : \mathbb{D}^{\mathbb{Z}}_{\text{poly}}(\mathbb{F}_{4n}, \mathbb{F}_0)$.

If H10(n, p, q) holds, i.e. if $[\![p]\!]^{\vec{w}}_{[]} = [\![q]\!]^{\vec{w}}_{[]}$ for $\vec{w} = [w_1; \ldots; w_n]$ we know via Lagrange's Theorem 34 that there exist $a_1, b_1, c_1, d_1, \ldots, a_n, b_n, c_n, d_n$ s.t. $w_i = a_i^2 + b_i^2 + c_i^2 + d_i^2$ and thus that $[\![f(p)]\!]^{\vec{v}}_{[]} = [\![p]\!]^{\vec{w}}_{[]}$ for $\vec{v} : \mathbb{Z}^{4n} := [a_1; b_1; c_1; d_1; \ldots; a_n; b_n; c_n; d_n]$ as well as $[\![f(q)]\!]^{\vec{v}}_{[]} = [\![q]\!]^{\vec{w}}_{[]}$. Thus, $[\![f(p) \dot{+} (-1) \dot{\times} f(q)]\!]^{\vec{v}}_{[]} = 0$ and H10$_\mathbb{Z}(4n, f(p) \dot{+} (-1) \dot{\times} f(q))$.

For the other direction, let $\vec{v} : \mathbb{Z}^{4n}$ be given s.t. $[\![f(p) \dot{+} (-1) \dot{\times} f(q)]\!]^{\vec{v}}_{[]} = 0$. We thus know that $[\![f(p)]\!]^{\vec{v}}_{[]} = [\![f(q)]\!]^{\vec{v}}_{[]}$. Because $\vec{v} : \mathbb{Z}^{4n}$, it has the form $\vec{v} = [a_1; b_1; c_1; d_1; \ldots; a_n; b_n; c_n; d_n]$. Then all elements of $\vec{w} := [a_1^2 + b_1^2 + c_1^2 + d_1^2; \ldots; a_n^2 + b_n^2 + c_n^2 + d_n^2]$ are natural numbers and the equations $[\![f(p)]\!]^{\vec{v}}_{[]} = [\![p]\!]^{\vec{w}}_{[]}$ and $[\![f(q)]\!]^{\vec{v}}_{[]} = [\![q]\!]^{\vec{w}}_{[]}$ hold. Thus $[\![p]\!]^{\vec{w}}_{[]} = [\![q]\!]^{\vec{w}}_{[]}$, i.e. H10$(n, p, q)$. □

10. μ-Recursive Algorithms

In order to show that MM, FRACTRAN, and H10 are in the same many-one reduction class (i.e. interreducible via many-one reductions), we introduce μ-recursive algorithms as intermediate layer. Programming in this well-known model of computation resembles functional programming (in a first-order language) and we will use it first for a reduction H10 $\preceq$ μ-rec. Afterwards, we explain a compiler to Minsky machines, yielding a reduction from μ-rec to MM. The next section will then connect μ-recursive algorithms to the weak call-by-value λ-calculus.

$$
\begin{array}{rclrcl}
\llbracket \mathtt{cst}_n \rrbracket\ \vec v\ x & \iff & n = x & \llbracket \mathtt{zero} \rrbracket\ \vec v\ x & \iff & 0 = x \\
\llbracket \mathtt{succ} \rrbracket\ \vec v\ x & \iff & 1 + \vec v_0 = x & \llbracket \mathtt{prj}_p \rrbracket\ \vec v\ x & \iff & \vec v_p = x
\end{array}
$$

$$
\begin{array}{rcl}
\llbracket \mathtt{comp}\, f\, \vec g \rrbracket\ \vec v\ x & \iff & \exists \vec w,\ \llbracket f \rrbracket\ \vec w\ x \land \forall p,\ \llbracket \vec g_p \rrbracket\ \vec v\ \vec w_p \\
\llbracket \mathtt{rec}\, f\, g \rrbracket\ (0 :: \vec v)\ x & \iff & \llbracket f \rrbracket\ \vec v\ x \\
\llbracket \mathtt{rec}\, f\, g \rrbracket\ (1 + n :: \vec v)\ x & \iff & \exists y, \llbracket \mathtt{rec}\, f\, g \rrbracket\ (n :: \vec v)\ y \land \llbracket g \rrbracket\ (n :: y :: \vec v)\ x \\
\llbracket \mu f \rrbracket\ \vec v\ x & \iff & \exists \vec w : \mathbb{N}^x, \llbracket f \rrbracket\ (x :: \vec v)\ 0 \land \forall y : \mathbb{F}_x,\ \llbracket f \rrbracket\ (\overline{y} :: \vec v)\ (1 + \vec w_y)
\end{array}
$$

Figure 2: Relational semantics for μ-recursive algorithms.

$$
\frac{}{[\mathtt{cst}_n; \vec v] \dashv\!\![1 + c\rangle\ n}
\qquad
\frac{}{[\mathtt{zero}; \vec v] \dashv\!\![1 + c\rangle\ 0}
\qquad
\frac{}{[\mathtt{succ}; x :: \vec v] \dashv\!\![1 + c\rangle\ 1 + x}
$$

$$
\frac{}{[\mathtt{prj}_p; \vec v] \dashv\!\![1 + c\rangle\ \vec v_p}
\qquad
\frac{\forall p : \mathbb{F}_k,\ [\vec g_p; \vec v] \dashv\!\![c - p\rangle\ \vec w_p \qquad [f; \vec w] \dashv\!\![1 + c\rangle\ x}{[\mathtt{comp}\, f\, \vec g; \vec v] \dashv\!\![2 + c\rangle\ x}
$$

$$
\frac{[f; \vec v] \dashv\!\![c\rangle\ x}{[\mathtt{rec}\, f\, g; 0 :: \vec v] \dashv\!\![1 + c\rangle\ x}
\qquad
\frac{[\mathtt{rec}\, f\, g; n :: \vec v] \dashv\!\![c\rangle\ y \qquad [g; n :: y :: \vec v] \dashv\!\![c\rangle\ x}{[\mathtt{rec}\, f\, g; (1 + n) :: \vec v] \dashv\!\![1 + c\rangle\ x}
$$

$$
\frac{[f; x :: \vec v] \dashv\!\![c - x\rangle\ 0 \qquad \forall p : \mathbb{F}_x,\ [f; p :: \vec v] \dashv\!\![c - p\rangle\ 1 + \vec w_p}{[\mu f; \vec v] \dashv\!\![1 + c\rangle\ x}
$$

Figure 3: Cost aware big-step semantics for μ-recursive algorithms.

10.1. **μ-Recursive Recognisability.** We define a type $\mathcal{A}_k$ of μ-recursive algorithms representing μ-recursive partial functions in $\mathbb{N}^k \longrightarrow \mathbb{N}$.

$$
\frac{n : \mathbb{N}}{\mathtt{cst}_n : \mathcal{A}_0}
\qquad
\frac{}{\mathtt{zero} : \mathcal{A}_1}
\qquad
\frac{}{\mathtt{succ} : \mathcal{A}_1}
\qquad
\frac{p : \mathbb{F}_k}{\mathtt{prj}_p : \mathcal{A}_k}
$$

$$
\frac{f : \mathcal{A}_k \qquad g : \mathcal{A}_i^k}{\mathtt{comp}\, f\, g : \mathcal{A}_i}
\qquad
\frac{f : \mathcal{A}_k \qquad g : \mathcal{A}_{2+k}}{\mathtt{rec}\, f\, g : \mathcal{A}_{1+k}}
\qquad
\frac{f : \mathcal{A}_{1+k}}{\mu f : \mathcal{A}_k}
$$

We represent these μ-recursive partial functions using the standard relational semantics $\llbracket f \rrbracket : \mathbb{N}^k \to \mathbb{N} \to \mathbb{P}$ of the μ-recursive algorithm $f : \mathcal{A}_k$, formalised in Figure 2 as a fixpoint definition. Intuitively, $\mathtt{cst}_n$ represents the constant n of arity 0, $\mathtt{zero}$ is the constant 0-function of arity 1, $\mathtt{succ}$ the successor function of arity 1, projection $\mathtt{prj}_p$ returns the p-th argument, $\mathtt{comp}\, f\, g$ where g is a k-vector of functions of arity i first applies each element of g to the i inputs and then f to the resulting k numbers. $\mathtt{rec}\, f\, g$ is performing primitive recursion on the first argument. If the argument is 0, f is used. If the argument is $1 + n$, g is applied to n, the recursive call and the rest of the arguments. Finally, minimisation μf performs unbounded search returning the smallest number x s.t. f on x returns 0 and f terminates on a non-zero value for every $y < x$.

Following [24], in Fig. 3 we also formalise a cost aware big-step evaluation predicate $[f; v] \dashv\!\![c\rangle\ x$, where $f : \mathcal{A}_k$, $v : \mathbb{N}^k$, $x : \mathbb{N}$, and where $c : \mathbb{N}$ denotes the cost of a computation, the cost being tailored toward the naive step-indexed evaluator to be defined in the next section.[20] We can directly relate the relational semantics with the big-step semantics:

Lemma 36. $\forall\, k\, (f : \mathcal{A}_k)\, (\vec{v} : \mathbb{N}^k)\, (x : \mathbb{N}),\ [\![f]\!]\ \vec{v}\ x \leftrightarrow \exists c,\ [f;\vec{v}] \dashv\!\![c\rangle\ x.$

With these characterizations of the semantics of μ-recursive algorithms, we can define μ-recursive halting as follows:

$$\mu\text{-rec}(k : \mathbb{N}, f : \mathcal{A}_k, \vec{v} : \mathbb{N}^k) := \exists x,\ [\![f]\!]\ \vec{v}\ x$$

A function $f : \mathcal{A}_k$ is called *total* if it terminates on any input, i.e. $\forall \vec{v} : \mathbb{N}^k\, \exists x,\ [\![f]\!]\ \vec{v}\ x$.

Definition 37. *We call a relation k-ary relation R over natural numbers* μ-recursively recognisable *if there is a μ-recursive algorithm $f : \mathcal{A}_k$ s.t. for any $\vec{v} : \mathbb{N}^k$ the equivalence $R\ \vec{v} \leftrightarrow \exists x,\ [\![f]\!]\ \vec{v}\ x$ holds.*

10.2. **Compiling μ-recursive algorithms into Minsky machines.** We describe how to compile a μ-recursive algorithm $f : \mathcal{A}_k$ into a Minsky machine. To avoid solving complicated inequality constraints over the bounded numbers in $\mathbb{F}_m$,[21] we here work with Minsky machines where registers are indexed with $\mathbb{N}$ instead of $\mathbb{F}_m$. Hence the number of registers is not bounded a priori but of course, for a given Minsky machine, the number of register that actually occur in the code is bounded. And indeed, in the end of the process in Theorem 40, we project to Minsky machines with registers in $\mathbb{F}_m$ (for some m) after the μ-recursive algorithm has been fully compiled. We will not enter into the details on how particular μ-recursive operators are implemented but instead focus on the global invariant of the compiler.

For a μ-recursive algorithm $f : \mathcal{A}_k$ with k inputs and one output, we produce a Minsky machine $P : \mathsf{I}_{\mathbb{N}}$ with the following structure for registers, described by the four extra parameters i, p, o and m, all of type $\mathbb{N}$:

***i*:** is the PC-index of the first instruction of P;
***p*:** the k inputs for f are to be read in the k registers $\{p, \dots, p+k-1\}$ of P;
***o*:** the output of f is to be written in the register o of P;
***m*:** P can use spare registers above m and assume their initial value is 0.

Moreover, we require that all registers except the output register o are returned to their initial value when the computation is terminated. In particular, spare registers return to their initial value 0.

Definition 38. *We say that P* properly compiles *f under the constraints i, p, o, m and we write* `ra_compiled` $k\ (f : \mathcal{A}_k)\ (i\ p\ o\ m : \mathbb{N})\ (P : \mathsf{I}_{\mathbb{N}})$ *if for any $\vec{v} : \mathbb{N}^k$ and $\rho : \mathbb{N} \to \mathbb{N}$ such that (1) $\forall u \geq m,\ \rho_u = 0$ and (2) $\forall q : \mathbb{F}_k,\ \rho_{q+p} = \vec{v}_q$, we have*

soundness: $\forall x : \mathbb{N},\ [\![f]\!]\ \vec{v}\ x \to (i, P)\ /\!\!/_M\ (i, \rho) \succ^* (|P| + i, \rho\{o \leftarrow x\})$*;*
completeness: $(i, P)\ /\!\!/_M\ (i, \rho)\downarrow \to \exists x,\ [\![f]\!]\ \vec{v}\ x.$

[20]In the Coq code, the predicate has one additional auxiliary argument to ease the correctness proof for the step-indexed interpreter which is only internal to the proof and thus omitted here.

[21]Unlike $\mathbb{N}$ or $\mathbb{Z}$, the inductive type $\mathbb{F}_m$ representing the numbers bounded by m is not equipped with powerful linear constraints solving tactics such as `lia` or `omega`. Working with bounded numbers here would require from us to manipulate embeddings of e.g. the disjoint sum $\mathbb{F}_i + \mathbb{F}_j$ into $\mathbb{F}_k$ (for some large enough k), which would also impact types that depend on i, j and k. For those who experienced it, this should instantly trigger the bad memory of *setoid hell*. Using the above mentioned tools, it is much simpler to work with intervals of $\mathbb{N}$ instead.

In these $\mathbb{N}$-indexed register machines, the state of a machine cannot be described by a finite vector $\vec{v} : \mathbb{N}^k$ but is instead represented by an environment $\rho : \mathbb{N} \to \mathbb{N}$ mapping register indices (in $\mathbb{N}$) to the values they contain (in $\mathbb{N}$ also, but with a different meaning). The two conditions (1) and (2) state that ρ is null above register m and ρ contains $\vec{v}$ starting at register p. And under these conditions, (i, P) should simulate a terminating computation of f on $\vec{v}$ outputting x (soundness), while conversely, whenever (i, P) terminates starting from (i, ρ), this entails that f terminates on $\vec{v}$ (completeness).

We can now construct a certified compiler from μ-recursive algorithms to $\mathbb{N}$-indexed Minsky machines. Literally, Theorem 39 below states that whenever the output register o does not belong to the input registers $\{p, \ldots, p+k-1\}$ or the spare registers $\{m, m+1, \ldots\}$ and that spare registers have indices above input registers, then one can properly compile f at PC value i.

Theorem 39. *Given a μ-recursive algorithm $f : \mathcal{A}_k$, we can build a term:*

$$\forall\, i\, p\, o\, m,\ o < m \to \neg(p \le o < k+p) \to k+p \le m \to \textstyle\sum P,\ \texttt{ra\_compiled}\ k\ f\ i\ p\ o\ m\ P.$$

The proof proceeds by structural induction on $f : \mathcal{A}_k$. The details are quite involved and not exposed in here: we just wanted to state the above invariant, which is closed under each μ-recursive constructor. The sub-Minsky machines are built and composed using the compositional techniques already presented in [16]. Notice that the statement of `ra_compiled` takes the output value x in $[\![f]\!]\ \vec{v}\ x$ into account while in below Theorem 40, we only care about termination. Because the termination of e.g. the μ-recursive composition `comp` $f\,\vec{g}$ depends not just on the termination of $\vec{g}_p$ (for all $p : \mathbb{F}_k$) but on the actual output values $\vec{w}_p$ of $[\![\vec{g}_p]\!]\ \vec{v}\ \vec{w}_p$, it is necessary to be more precise in the stated invariant used in the inductive construction of the compiler.

Theorem 40. *Given a μ-recursive algorithm $f : \mathcal{A}_k$, one can compute $n : \mathbb{N}$ and a list of Minsky machine instructions $P : \mathsf{I}_{\mathbb{F}_{k+1+n}}$ such that for any $\vec{v} : \mathbb{N}^k$,*

$$(\exists x,\ [\![f]\!]\ \vec{v}\ x) \leftrightarrow (1, P)\ /\!\!/_M\ (1, \vec{v} +\!\!+ \vec{0})\downarrow.$$

Proof. By Theorem 39, we compile f into $P' : \mathsf{I}_\mathbb{N}$ under the constraints $i := 1$, $p := 0$, $o := k$ and $m := 1 + k$. Then we compute n such that $k + 1 + n$ is a strict upper bound of all the register indices that occur in P'. Now we can map $P' : \mathsf{I}_\mathbb{N}$ to $P : \mathsf{I}_{\mathbb{F}_{k+1+n}}$ while preserving its semantics. Notice that contrary to Theorem 39, the resulting output value of f or $(1, P)$ can now be disregarded since we only care about termination. □

Corollary 41. *μ-rec $\preceq$ MM.*

10.3. **Diophantine relations are μ-recursively recognisable.** We encode n-ary Diophantine relations using μ-recursive algorithms. Following the DPRM Theorem 29, we consider a n-ary relation $R : \mathbb{N}^n \to \mathbb{P}$ to be Diophantine if there is a single Diophantine equation $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$ with n parameters and m (existential) variables such that

$$\forall \vec{v} : \mathbb{N}^n,\ R\ \vec{v} \leftrightarrow \exists \vec{w} : \mathbb{N}^m,\ [\![p]\!]^{\vec{w}}_{\vec{v}} = [\![q]\!]^{\vec{w}}_{\vec{v}}.$$

Again we abusively confuse vectors with maps using the equivalence $X^n \simeq \mathbb{F}_n \to X$. We will prove that every Diophantine relation in this sense is μ-recursively recognisable (see Definition 37). The proof idea is relatively straightforward.

First, we implement `eval` $: \mathbb{D}_{\text{poly}}(\mathbb{F}_m, \mathbb{F}_n) \to \mathcal{A}_{m+n}$ evaluating any polynomial:

Lemma 42. *Given any Diophantine polynomial $p : \mathbb{D}_{\text{poly}}(\mathbb{F}_m, \mathbb{F}_n)$, one can compute a μ-recursive algorithm $\texttt{eval}_p : \mathcal{A}_{m+n}$ s.t. $[\![\texttt{eval}_p]\!]\ (\vec{w} +\!\!+ \vec{v})\ [\![p]\!]^{\vec{w}}_{\vec{v}}$.*

Proof. The implementation relies on implementations of addition and multiplication on natural numbers, which are relatively straightforward for μ-recursive algorithms. □

Secondly, we implement a bijection between $\mathbb{N}$ and $\mathbb{N}^m$ for any m:

Lemma 43. *There are functions $\texttt{pr}_m : \mathbb{N} \to \mathbb{N}^m$ and $\texttt{inj}_m : \mathbb{N}^m \to \mathbb{N}$ s.t. $\texttt{pr}_m(\texttt{inj}_m\,\vec{v}) = \vec{v}$. Furthermore, given $i : \mathbb{F}_m$, one can compute a μ-recursive algorithm $\texttt{project}_i : \mathcal{A}_1$ s.t. $[\![\texttt{project}_i]\!]\ (x :: [\,])\ (\texttt{pr}_m\,x)_i$, i.e. all the m components of $\texttt{pr}_m$ are μ-recursive.*

Thirdly, given an equation $p \doteq q$, we compute a test algorithm $\texttt{test}_{p,q} : \mathcal{A}_{1+n}$, always terminating and returning 0 iff $p \doteq q$ is satisfied when decoding the first argument as a vector $\vec{w}$ using $\texttt{pr}_m$:

Lemma 44. *Given any $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$, one can compute a μ-recursive algorithm $\texttt{test}_{p,q} : \mathcal{A}_{1+n}$ s.t. $[\![\texttt{test}_{p,q}]\!]\ (x :: \vec{v})\ 0 \leftrightarrow [\![p]\!]^{\vec{w}}_{\vec{v}} = [\![q]\!]^{\vec{w}}_{\vec{v}}$ where $\vec{w} := \texttt{pr}_m(x)$. Furthermore, $\texttt{test}_{p,q}$ is primitive recursive hence total.*

To finish, given a vector $\vec{v} : \mathbb{N}^n$ describing the values of parameters, we use minimisation μ to search for a number x such that $\vec{w} := \texttt{pr}_m(x)$ is a solution for $p \doteq q$:

Theorem 45. *Given $p \doteq q : \mathbb{D}_{\text{single}}(\mathbb{F}_m, \mathbb{F}_n)$, one can compute a μ-recursive algorithm $\texttt{find}_{p,q} : \mathcal{A}_n$ s.t. for any $\vec{v} : \mathbb{N}^n$, we have the equivalence*

$$(\exists x : \mathbb{N},\ [\![\texttt{find}_{p,q}]\!]\ \vec{v}\ x)\ \leftrightarrow\ \exists \vec{w} : \mathbb{N}^m,\ [\![p]\!]^{\vec{w}}_{\vec{v}} = [\![q]\!]^{\vec{w}}_{\vec{v}}.$$

Notice that while $\texttt{find}_{p,q}$ cannot be primitive recursive (it does not terminate when the equation $p \doteq q$ is unsolvable), it is however implemented as a single unbounded minimization applied to an otherwise primitive recursive algorithm.

Corollary 46. *H10 $\preceq$ μ-rec.*

Proof. Direct application of Theorem 45 with no parameters, i.e. $n := 0$. □

11. The Weak Call-by-Value λ-Calculus L

Until now, we have shown that the problems MM, FRACTRAN, H10, and μ-rec are all interreducible w.r.t. many-one reductions. We contribute our proofs to the Coq library of undecidability proofs [18], which contains several other well-known reduction proofs. Amongst them is a chain of reductions published in related work and discussed in Section 12, establishing that the halting problem of Turing machines reduces to MM and thus to all other problems we consider.

It is also possible to prove that all considered problems are in fact interreducible to the halting problem of Turing machines. We demonstrate one technique to do so, based on the weak call-by-value λ-calculus L, which is already shown interreducible with the halting problem of Turing machines in [13].

We briefly introduce L in this section and then reduce μ-rec to halting in L. Syntactically, L is just the untyped λ-calculus, De Bruijn style:

$$s, t, u : \mathsf{L} ::= n \mid st \mid \lambda s \qquad \text{where } n : \mathbb{N}$$

We define a weak call-by-value evaluation predicate $s \triangleright t$, coinciding with other definitions of reduction for closed terms, where $s[t/0]$ means "replace De Bruijn variable 0 in s by t".

$$\frac{}{\lambda s \triangleright \lambda s} \qquad\qquad \frac{s \triangleright \lambda s' \qquad t \triangleright t' \qquad s'[t'/0] \triangleright u}{st \triangleright u}$$

The halting problem for L can then be defined as $\mathsf{WCBV}(s : \mathsf{L}) := \exists t,\ s \triangleright t$.

It is possible to encode natural numbers, vectors over natural numbers, and other data types into L using Scott's encoding. We will denote with $\overline{\cdot}$ such encoding functions. For further details on encodings and L we refer to [19], since they do not actually matter to understand the reduction.

In general, a predicate reduces to the L-halting problem WCBV if it is L-recognisable:

Definition 47. *We say that a function $f : X_1 \to \cdots \to X_n \to Y$ is* L-computable *if there exists a closed term t s.t. $\forall x_1 : X_1 \ldots \forall x_n : X_n.\ t\ \overline{x_1} \ldots \overline{x_n} \triangleright \overline{f x_1 \ldots x_n}$.*

Given a type X encodable in L and a predicate P over X we say that P is L-recognisable *if there is an L-computable function $f : X \to \mathbb{N} \to \mathbb{B}$ s.t. $P\,x \leftrightarrow \exists n, f\,x\,n = \mathsf{true}$.*

Theorem 48. *Let X be encodable in L. For a predicate P over X we have that $P \preceq \mathsf{WCBV}$ if and only if P is L-recognisable.*

To instantiate the theorem to μ-rec one would like to give a step-indexed evaluation function for μ-recursive algorithms, i.e. for $f : \mathcal{A}_k$ a function $[\![f]\!] : \mathbb{N} \to \mathbb{N}^k \to \mathbb{O}\,\mathbb{N}$ s.t. $[f;\vec{v}] \dashv\!\![c\rangle\ x \leftrightarrow [\![f]\!]c\ \vec{v} = \lfloor x \rfloor$ and then show that it is computable in L. The second step, i.e. proving that this function is L-computable, is in principle fully automatic using the certifying extraction framework from Forster and Kunze [12], which is implemented using tools from the MetaCoq projet [36]. The framework supports extracting non-dependent, non-mutual, non-nested, simply-typed Coq functions to L automatically, and also generates a proof of correctness of the extract. However, recall that the type $\mathcal{A}_k$ of μ-recursive functions is both heavily dependent and used nested constructions: It has a type index in k, makes use of finite types $\mathbb{F}_k$, and has a nested use of the dependent vector type, i.e. contains subterms of type $(\mathcal{A}_k)^i$. Thus, a direct step-indexed interpreter will not be extractable. Instead, we use a general technique and implement a step-indexed interpreter working on the *syntactic skeleton* of μ-recursive functions.

A syntactic skeleton for a type I mirrors the constructors of I, but without any dependent types. If I furthermore has nested applications of types $N_1, \ldots, N_n$, the syntactic skeleton also contains the constructors of $N_1, \ldots, N_n$. That is, if I has i constructors and $N_1, \ldots, N_n$ have $m_1, \ldots, m_n$ constructors respectively, the syntactic skeleton of I has $i + m_1 + \cdots + m_n$ constructors. The syntactic skeleton $\mathcal{A}'$ of the type $\mathcal{A}$ is defined as follows:

$$\frac{n : \mathbb{N}}{\mathtt{cst}_n : \mathcal{A}'} \qquad \frac{}{\mathtt{zero} : \mathcal{A}'} \qquad \frac{}{\mathtt{succ} : \mathcal{A}'} \qquad \frac{j : \mathbb{N}}{\mathtt{prj}_j : \mathcal{A}'} \qquad \frac{}{\mathtt{nil} : \mathcal{A}'}$$

$$\frac{f : \mathcal{A}' \qquad g : \mathcal{A}'}{\mathtt{comp}\ f\ g : \mathcal{A}'} \qquad \frac{f : \mathcal{A}' \qquad g : \mathcal{A}'}{\mathtt{rec}\ f\ g : \mathcal{A}'} \qquad \frac{f : \mathcal{A}'}{\mu f : \mathcal{A}'} \qquad \frac{f : \mathcal{A}' \qquad g : \mathcal{A}'}{\mathtt{cons}\ \ f\ \ g : \mathcal{A}'}$$

Note that we re-use constructor names. In the syntactic skeleton we have $j : \mathbb{N}$ instead of $j : \mathbb{F}_k$ for $\mathtt{prj}_j$, $g : \mathcal{A}'$ instead of $G : (\mathcal{A}_k)^i$ for $\mathtt{comp}\ f\ g$, and the constructors $\mathtt{cons}$ and $\mathtt{nil}$ as the constructors of the vector type are added.

It is straightforward to implement mutually recursive functions $\mathtt{erase} : \mathcal{A}_k \to \mathcal{A}'$ and $\mathtt{erase}' : (\mathcal{A}_k)^i \to \mathcal{A}'$ which are essentially the identity. We call a skeleton $s : \mathcal{A}'$ *valid* if it corresponds to a μ-recursive function or a vector, i.e. if $\exists k : \mathbb{N},\ (\exists f : \mathcal{A}_k,\ s = \mathtt{erase} f) \vee (\exists i : \mathbb{N}, \exists l : (\mathcal{A}_k)^i,\ s = \mathtt{erase}' l)$.

For $\mathcal{A}'$ we can now define a step-indexed evaluation function

$$[\![\cdot]\!] : \mathcal{A}' \to \mathbb{N} \to \mathbb{N} \to \mathbb{L}\,\mathbb{N} \to \mathbb{O}\,(\mathbb{N} + \mathbb{L}\,\mathbb{N}).$$

A call $[\![f]\!]_c^m l$ uses c as step-index, m as auxiliary counter to implement unbounded search, and l as input. If $[\![f]\!]_c^m l = \lfloor \mathsf{inl}\ v \rfloor$, then v is the value of the evaluation. If $[\![f]\!]_c^m l = \lfloor \mathsf{inr}\ l' \rfloor$, then f encoded a list of functions via `cons` and `nil` which pointwise evaluated to $l' : \mathbb{L}\,\mathbb{N}$. If $[\![f]\!]_c^m l = \emptyset$ either if usual the step-index c was not big enough, the function f does not terminate on input l, or f is not a valid skeleton.

$$\begin{array}{rll}
[\![\mathtt{cst}_n]\!]_{1+c}^m\ l := & \lfloor \mathsf{inl}\ n \rfloor & \\
[\![\mathtt{zero}]\!]_{1+c}^m\ l := & \lfloor \mathsf{inl}\ 0 \rfloor & \\
[\![\mathtt{succ}]\!]_{1+c}^m\ (x :: l) := & \lfloor \mathsf{inl}\ (1+x) \rfloor & \\
[\![\mathtt{prj}_p]\!]_{1+c}^m\ l := & \lfloor \mathsf{inl}\ x \rfloor & \text{for } l_p = \lfloor x \rfloor \\
[\![\mathtt{comp}\, f\, g]\!]_{1+c}^m\ l := & \lfloor \mathsf{inl}\ x \rfloor & \text{for } [\![g]\!]_c^m\ l = \lfloor \mathsf{inr}\ l' \rfloor \text{ and } [\![f]\!]_c^m\ l' = \lfloor \mathsf{inl}\ x \rfloor \\
[\![\mathtt{rec}\, f\, g]\!]_{1+c}^m\ (0 :: l) := & \lfloor \mathsf{inl}\ x \rfloor & \text{for } [\![f]\!]_c^m\ l = \lfloor \mathsf{inl}\ x \rfloor \\
[\![\mathtt{rec}\, f\, g]\!]_{1+c}^m\ \big((1+n) :: l\big) := & \lfloor \mathsf{inl}\ x \rfloor & \text{for } [\![\mathtt{rec}\, f\, g]\!]_c^m\ (n :: l) = \lfloor \mathsf{inl}\ y \rfloor \text{ and } [\![g]\!]_c^m\ (n :: y :: l) = \lfloor \mathsf{inl}\ x \rfloor \\
[\![\mathtt{nil}]\!]_{1+c}^m\ l := & \lfloor \mathsf{inr}\ [\,] \rfloor & \\
[\![\mathtt{cons}\ f\ g]\!]_{1+c}^m\ l := & \lfloor \mathsf{inr}\ (x :: l') \rfloor & \text{for } [\![f]\!]_c^m\ l = \lfloor \mathsf{inl}\ x \rfloor \text{ and } [\![g]\!]_c^m\ l = \lfloor \mathsf{inr}\ l' \rfloor \\
[\![\mu f]\!]_{1+c}^m\ l := & \lfloor \mathsf{inl}\ m \rfloor & \text{for } [\![f]\!]_c^0\ (m :: l) = \lfloor \mathsf{inl}\ 0 \rfloor \\
[\![\mu f]\!]_{1+c}^m\ l := & \lfloor \mathsf{inl}\ x \rfloor & \text{for } [\![f]\!]_c^0\ (m :: l) = \lfloor \mathsf{inl}\ (1+y) \rfloor \text{ and } [\![\mu f]\!]_c^{1+m}\ l = \lfloor \mathsf{inl}\ x \rfloor \\
[\![f]\!]_c^m\ l := & \emptyset & \text{in all other cases}
\end{array}$$

Lemma 49. *For all $f : \mathcal{A}_k$, $\vec{v} : \mathbb{N}^k$, $c : \mathbb{N}$, and $n : \mathbb{N}$ we have*

$$[f; \vec{v}] \dashv\!\! c\rangle\ n \leftrightarrow [\![f]\!]_c^0(\mathtt{v2l}\ \vec{v}) = \lfloor \mathsf{inl}\ n \rfloor$$

Note that on the right hand side of this equivalence, the vector $\vec{v}$ is just mapped to its underlying list $\mathtt{v2l}\ \vec{v}$.

Theorem 50. *μ-rec $\preceq$ WCBV.*

Proof. We use Theorem 48 and have to provide a function F s.t. $\mu\text{-}\mathsf{rec}(k : \mathbb{N}, f : \mathcal{A}_k, \vec{v} : \mathbb{N}^k) \leftrightarrow \exists c,\ F c(k, f, \vec{v}) = \mathsf{true}$.

We define $F\,(k : \mathbb{N}, f : \mathcal{A}_k, \vec{v} : \mathbb{N}^k)\, c := \mathsf{true}$ if $[\![f]\!]_c^0(\mathtt{v2l}\ \vec{v}) = \lfloor \mathsf{inl}\ n \rfloor$ for some n and $F(k, f, v)c := \mathsf{false}$ otherwise.

Now F is part of the Coq fragment supported by the extraction framework in [12], since the step-indexed evaluator $[\![\cdot]\!]$ does not use any dependent, nested, mutual, or non-simple types. Thus F is L-computable.

Furthermore we have as wanted that

$$\begin{aligned}
\mu\text{-}\mathsf{rec}(k, f, \vec{v}) &\leftrightarrow \exists n,\ [\![f]\!]\vec{v}n \\
&\leftrightarrow \exists n\, c,\ [f; \vec{v}] \dashv\!\! c\rangle\ n \\
&\leftrightarrow \exists n\, c,\ [\![f]\!]_c^0\,(\mathtt{v2l}\ \vec{v}) = \lfloor \mathsf{inl}\ n \rfloor \\
&\leftrightarrow \exists c,\ F\,(k, f, \vec{v})\, c = \mathsf{true}
\end{aligned}$$

$\square$

12. Related Work

Regarding formalisations of Hilbert's tenth problem, there are various unfinished and preliminary results in different proof assistants: Carneiro [2] formalises Matiyasevich's theorem (Diophantineness of exponentiation) in Lean, but does not consider computational models or the DPRM theorem. Pak formalises results regarding Pell's equation [32] and proves that Diophantine sets are closed under union and intersection [33], both as parts of the Mizar Mathematical Library. Stock et al. [39, 1] report on an unfinished formalisation of the DPRM theorem in Isabelle based on [29]. They cover some parts of the proof, but acknowledge for important missing results like Lucas's or "Kummer's theorem" and a "formalisation of a register machine." Moreover, none of the cited reports considers the computability of the reductions involved or the verification of a universal machine in the chosen model of computation yet, one of them being a necessary proof goal for an actual undecidability result in the classical meta-theories of Isabelle/HOL and Mizar.

Regarding undecidability proofs in type theory, Forster, Heiter, and Smolka [10] reduce the halting problem of Turing machines to PCP. Forster and Larchey-Wendling [16] reduce PCP to provability in linear logic via the halting problem of Minsky machines, which we build on. Forster, Kirst and Smolka develop the notion of synthetic undecidability in Coq and prove the undecidability of various notions in first-order logic [11]. Spies and Forster mechanise the undecidability proof of second-order unification by reduction from H10 [38] originally shown by Goldfarb [21]. Forster, Kunze, and Wuttke reduce the halting problem of multi-tape Turing machines to single-tape Turing machines [14]. Dudenhefner and Rehof [9] mechanise a recently simplified undecidability proof for System F inhabitation.

Regarding formalisations of μ-recursive functions, Larchey-Wendling [24] shows that every total μ-recursive function can directly be computed in Coq and Carneiro [3] mechanises standard computability theory based on μ-recursive functions. Xu, Zhang, and Urban [42] mechanise μ-recursive functions and Turing machines in Isabelle and prove their computational equivalence. Their proof uses Abacus machines as intermediate layer, which are similar to our Minsky machines.

Acknowledgements

We would like to thank Gert Smolka, Dominik Kirst, and Simon Spies for helpful discussion regarding the presentation. The first author was partially supported by the TICAMORE project (ANR grant 16-CE91-0002).

References

[1] Jonas Bayer, Marco David, Abhik Pal, Benedikt Stock, and Dierk Schleicher. The DPRM Theorem in Isabelle (Short Paper). In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:7, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
[2] Mario Carneiro. A Lean formalization of Matiyasevič's theorem, 2018.
[3] Mario Carneiro. Formalizing Computability Theory via Partial Recursive Functions. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International*

Conference on Interactive Theorem Proving (ITP 2019), volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[4] John H. Conway. *FRACTRAN: A Simple Universal Programming Language for Arithmetic*, pages 4–26. Springer New York, New York, NY, 1987.

[5] Martin Davis. Arithmetical problems and recursively enumerable predicates 1. *The Journal of Symbolic Logic*, 18(1):33–41, 1953.

[6] Martin Davis. Hilbert's Tenth Problem is Unsolvable. *The American Mathematical Monthly*, 80(3):233–269, 1973.

[7] Martin Davis and Hilary Putnam. *A computational proof procedure; Axioms for number theory; Research on Hilbert's Tenth Problem.* Air Force Office of Scientific Research, Air Research and Development, 1959.

[8] Martin Davis, Hilary Putnam, and Julia Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, pages 425–436, 1961.

[9] Andrej Dudenhefner and Jakob Rehof. A Simpler Undecidability Proof for System F Inhabitation. *TYPES 2018*, 2018.

[10] Yannick Forster, Edith Heiter, and Gert Smolka. Verification of PCP-Related Computational Reductions in Coq. In *ITP 2018*, pages 253–269. Springer, 2018.

[11] Yannick Forster, Dominik Kirst, and Gert Smolka. On Synthetic Undecidability in Coq, with an Application to the Entscheidungsproblem. In *CPP 2019*, pages 38–51, 2019.

[12] 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)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[13] Yannick Forster, Fabian Kunze, Gert Smolka, and Maximilian Wuttke. A mechanised proof of the time invariance thesis for the weak call-by-value λ-calculus. 193(29), 2021.

[14] Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified Programming of Turing Machines in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs. ACM*, 2020.

[15] Yannick Forster and Dominique Larchey-Wendling. Towards a library of formalised undecidable problems in Coq: The undecidability of intuitionistic linear logic. *Workshop on Syntax and Semantics of Low-level Languages, Oxford*, 2018.

[16] Yannick Forster and Dominique Larchey-Wendling. Certified Undecidability of Intuitionistic Linear Logic via Binary Stack Machines and Minsky Machines. In *CPP 2019*, pages 104–117. ACM, 2019.

[17] Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Lennard Gäher, Edith Heiter, Marc Hermes, Dominik Kirst, Fabian Kunze, Maximilian Wuttke, Gert Smolka, Simon Spies, and Dominik Wehr. The Coq Library of Undecidability Proofs: Hilbert's Tenth problem in Coq (LMCS) v1.1, May 2021.

[18] 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.

[19] Yannick Forster and Gert Smolka. Weak Call-By-Value Lambda Calculus as a Model of Computation in Coq. In *ITP 2018*, pages 189–206. Springer, 2017.

[20] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.

[21] Warren D. Goldfarb. The undecidability of the secondorder unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[22] David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

[23] J. P. Jones and Y. V. Matijasevič. Register Machine Proof of the Theorem on Exponential Diophantine Representation of Enumerable Sets. *J. Symb. Log.*, 49(3):818–829, 1984.

[24] Dominique Larchey-Wendling. Typing Total Recursive Functions in Coq. In *ITP 2017*, pages 371–388. Springer, 2017.

[25] 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)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[26] Edouard Lucas. Théorie des Fonctions Numériques Simplement Périodiques. [Continued]. *American Journal of Mathematics*, 1(3):197–240, 1878.

[27] Yuri V. Matijasevič. Enumerable sets are Diophantine. In *Soviet Mathematics: Doklady*, volume 11, pages 354–357, 1970.

[28] Yuri V. Matiyasevich. A new technique for obtaining Diophantine representations via elimination of bounded universal quantifiers. *J. Math. Sci.*, 87(1):3228–3233, 1997.

[29] Yuri V. Matiyasevich. On Hilbert's Tenth Problem. Expository Lectures 1, Pacific Institute for the Mathematical Sciences, University of Calgary, February 2000.

[30] Yuri V. Matiyasevich. Martin Davis and Hilbert's Tenth Problem. In *Martin Davis on Computability, Computational Logic, and Mathematical Foundations*. Springer, 2016.

[31] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[32] Karol Pak. The Matiyasevich Theorem. Preliminaries. *Formalized Mathematics*, 25(4):315–322, 2017.

[33] Karol Pak. Diophantine sets. Preliminaries. *Formalized Mathematics*, 26(1):81–90, 2018.

[34] Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *bulletin of the American Mathematical Society*, 50(5):284–316, 1944.

[35] Julia Robinson. Existential definability in arithmetic. *Transactions of the American Mathematical Society*, 72(3):437–449, 1952.

[36] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The metacoq project. *J. Autom. Reason.*, 64(5):947–999, 2020.

[37] Matthieu Sozeau and Cyprien Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in coq. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019.

[38] Simon Spies and Yannick Forster. Undecidability of higher-order unification formalised in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 143–157, 2020.

[39] Benedikt Stock et al. Hilbert meets Isabelle: Formalisation of the DPRM theorem in Isabelle. *Isabelle Workshop 2018*, 2018.

[40] The Coq Development Team. The Coq Proof Assistant, January 2021.

[41] The Mathematical Components team. Mathematical Components.

[42] Jian Xu, Xingyuan Zhang, and Christian Urban. Mechanising Turing machines and computability theory in Isabelle/HOL. In *International Conference on Interactive Theorem Proving*, pages 147–162. Springer, 2013.

Appendix A. Some Remarks about the Coq Code Contents

The file names below are hyper-linked to the corresponding files in the following specific release of the Coq library of undecidability proofs:

`https://github.com/uds-psl/coq-library-undecidability/tree/H10-LMCS-v1.1`

To help at understanding the H10 code from a high-level point of view, we provide two additional Coq source code files:

- the file summary.v which gives specific pointers to the main problems contained in the reduction chain described in this paper;
- the file standalone.v which gives a (combined) reduction from the binary BPCP to H10 but where the statements of those two problems have been rewritten to minimize dependencies.

Additionally, we here give a detailed overview of the structure of the code corresponding to the results presented in this paper, and which was contributed to our Coq library of undecidability proofs. The following *lines of code (loc)* measurements combine both definitions and proof scripts but do not account for comments. Notice that there are more files in the whole library than those needed to actually cover H10, but here, we only present the latter. In total, we contribute 16k loc to our undecidability project, 4k being additions to its shared libraries as extensions of the Coq standard library.

Concerning the multi-purpose shared libraries in Shared/Libs/DLW/Utils:

- we implemented finitary sums/products (over monoids) up to the binomial theorem (Newton) over non-commutative rings in sums.v and binomial.v for a total of 550 loc;
- we implemented bitwise operations over $\mathbb{N}$, both a lists of bits in bool_list.v and Peano `nat` in bool_nat.v for a total of 1700 loc;
- we implemented many results about Euclidean division and Bézout's identity in gcd.v, prime numbers and their unboundedness in prime.v, and base p representations in power_decomp.v for a total of 1200 loc;
- we implemented miscellaneous libraries for the reification of bounded_quantification.v (120 loc), the Pigeon Hole Principle in php.v (350 loc) and iterations of binary relations in rel_iter.v (230 loc).

Concerning the libraries for Minsky machines and FRACTRAN programs:

- by a slight update to the existing code [16], we proved in mm_comp.v that MM-termination (on any state) is undecidable (10 loc). Both the pre-existing result (undecidability of MM-termination on the zero state) and the new result derive from the correctness of the compiler of binary stack machines into Minsky machines;
- we implemented the removal of self loops in Minsky machines in mm_no_self.v (340 loc);
- we construct two infinite sequences of primes $\mathfrak{p}_i$ and $\mathfrak{q}_i$ in prime_seq.v (240 loc);
- FRACTRAN definitions and basic results occur in fractran_utils.v (310 loc) and the verified compiler from Minsky machines to FRACTRAN occurs in MM_FRACTRAN.v (300 loc);

Concerning the libraries for proving Matiyasevich's theorems:

- we implemented a library for modular arithmetic ($\mathbb{Z}/p\mathbb{Z}$) in Zp.v (920 loc);

- we implemented a library for 2×2-matrix computation including exponentiation and determinants in matrix.v (210 loc);
- we implemented an elementary proof of Lucas's theorem in luca.v (330 loc);
- we implemented the "classical proof" of Lagrange's theorem in lagrange.v (520 loc);
- the solution $\alpha_b(n)$ of Pell's equation and its (modular) arithmetic properties up to a proof of its Diophantineness are in alpha.v (1150 loc);
- from $\alpha_b(n)$, we implement the meta-level Diophantine encoding of the exponential in expo_diophantine.v (150 loc);
- we implement the sparse ciphers used in the Diophantine elimination of bounded universal quantification in cipher.v (1450 loc).

Concerning the object-level Diophantine libraries:

- the definition of Diophantine logic and basic results is in dio_logic.v (540 loc);
- the definition of elementary Diophantine constraints and the reduction from Diophantine logic is in dio_elem.v (440 loc);
- the definition of single Diophantine equations and the reduction from elementary Diophantine constraints is in dio_single.v (350 loc);
- we implement the object-level Diophantine encoding of the exponential relation in dio_expo.v (60 loc); but all the work is done in the previously mentioned libraries;
- the object-level Diophantine encoding of bounded universal quantification spans over dio_binary.v, dio_cipher.v and dio_bounded.v (460 loc);
- we derive the object-level Diophantine encoding of the reflexive-transitive closure in dio_rt_closure.v (40 loc);
- we implement the object-level Diophantine encoding of the FRACTRAN termination predicate in fractran_dio.v (80 loc).

Concerning μ-recursive algorithms, reducing H10 and reduced to Minsky machines:

- building on the pre-existing developments corresponding to [24] (1000 loc);
- the μ-recursive algorithm that searches for a solution to a single Diophantine equation in ra_utils.v, ra_dio_poly.v, recomp.v and ra_recomp.v (1370 loc);
- extensions of the Minsky machines library for $\mathbb{N}$-indexed registers in env.v, mme_defs.v and mme_utils.v (600 loc);
- the certified compiler from μ-recursive algorithms to Minsky machines in ra_mm.v and ra_mm_env.v (1250 loc).

Concerning the reduction from μ-recursive algorithms to L:

- the step-indexed evaluator and the reduction are in MuRec.v (450 loc)
- the framework used to extract the step-indexed evaluator is in the directory L and consists of about 2500 lines of code. More details can be found in [12].

To finish, the main undecidability results and the DPRM:

- the undecidability of Minsky machines is in MM_undec.v (20 loc);
- the reduction from MM to FRACTRAN is in MM_FRACTRAN.v (90 loc);
- the Diophantine encoding of FRACTRAN termination is in FRACTRAN_DIO.v (70 loc);
- the whole reduction chain leading to the undecidability of H10 is in H10_undec.v (60 loc);
- the reduction from H10 to $\mathsf{H10}_{\mathbb{Z}}$ is in H10Z.v (200 loc)
- and the DPRM theorem is in DPRM.v (170 loc).

Appendix B. Lucas's theorem

Lucas's theorem allows for the computation of the binomial coefficient $\mathcal{C}_m^n$ modulo a prime number p using the base p expansions of m and n. There are various proofs of this theorem but most of them involve high-level concepts like *generating functions* or *group action* and we choose instead to implement a low-level combinatorial proof of the theorem. Such a proof provides specific combinatorial insights into Lucas's results.

While a high-level proof could rightfully be considered as more beautiful, it would also likely require a lot of additional library code. We could of course rely on an external library such as MathComp [41] but this would generate a new external dependency, additional to the already involved Equations [37] and MetaCoq [36] libraries, increasing the likelihood of synchronization issues between all these developments. An additional benefit of a low-level proof is that it could reasonably be imported in alternate developments (e.g. [1]) in other proof assistants (e.g. Isabelle/HOL) without the need to convert a substantial helper library.

Before we enter this low-level proof of Lucas's theorem, we must give a light-weight, working and formal definition of binomial coefficients. For this, we use Pascal's identity as a ground for a fixpoint definition:

$$\mathcal{C}_m^0 := 1 \qquad \text{and} \qquad \mathcal{C}_0^{1+n} := 0 \qquad \text{and} \qquad \mathcal{C}_{1+m}^{1+n} := \mathcal{C}_m^n + \mathcal{C}_m^{1+n}$$

where we use the compact notation $\mathcal{C}_m^n$ for a more compact typesetting of the upcoming equations. Starting from Pascal's definition, one can derive the following identities

$$n!\,(m-n)!\,\mathcal{C}_m^n = m! \quad \text{for } n \le m \qquad \text{and} \qquad \mathcal{C}_m^n = 0 \quad \text{for } m < n$$

and we call the leftmost one as *the binomial identity*.[22]

Lemma 51. *Let p be a prime number and let us consider $M = mp + m_0$ and $N = np + n_0$ with $m_0, n_0 < p$. Then the identity $\mathcal{C}_M^N \equiv \mathcal{C}_m^n\,\mathcal{C}_{m_0}^{n_0}\,[p]$ holds.*

Proof. Let us first notice that the identity is trivial (in $\mathbb{N}$ already) when $n > m$ or $n = m \wedge n_0 > m_0$ because both sides evaluate to 0. So below, we only consider the cases $n \le m \wedge n_0 \le m_0$ or $n < m \wedge m_0 < n_0$ which cover the remaining part of the domain.

Before we split those two cases, let us define $\phi : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ and $\psi : \mathbb{N} \to \mathbb{N}$ by

$$\phi_r^i := (ip+1)\cdots(ip+r) \qquad \text{and} \qquad \psi_i := \phi_{p-1}^0 \cdots \phi_{p-1}^{i-1}.$$

Notice that ϕ_r^i generalize the factorial function as $r! = \phi_r^0$ but we will always use ϕ_r^i with $r < p$. Projected on $\mathbb{Z}/p\mathbb{Z}$, both ϕ and ψ simplify to factorial as it is easy to show[23]

$$\phi_r^i \equiv r!\,[p] \qquad \text{and} \qquad \psi_i \equiv (p-1)!^i\,[p]. \tag{B.1}$$

Moreover, ϕ_r^i is invertible in $\mathbb{Z}/p\mathbb{Z}$ for any $r < p$, and in particular $(p-1)! = \phi_{p-1}^0$ and ψ_i are also invertible in $\mathbb{Z}/p\mathbb{Z}$. Using ϕ, ψ and the semiring structure of $\mathbb{N}$, we establish the identity

$$(ip+r)! = \phi_{p-1}^0 \cdot (1p+0) \cdots \phi_{p-1}^{i-1} \cdot (ip+0) \cdot \phi_r^i = i!\,p^i\,\phi_r^i\,\psi_i \tag{B.2}$$

In the case $n \le m$ and $n_0 \le m_0$, we have $M - N = (m-n)p + (m_0 - n_0)$ and we rewrite the binomial equation $N!\,(M-N)!\,\mathcal{C}_M^N = M!$ in $\mathbb{N}$ as

$$(np+n_0)!\,((m-n)p + (m_0-n_0))!\,\mathcal{C}_M^N = (mp+m_0)!$$

[22] We do not enter the details of these inductive proofs which are standard exercises in basic arithmetic.

[23] Notice that Wilson's theorem establishes $(p-1)! \equiv -1\,[p]$ but this equation is not needed in this proof.

and then rewrite again with Eq. (B.2) and the binomial equation $m! = n!\,(m-n)!\,\mathcal{C}_m^n$ as

$$n!\,p^n\,\phi_{n_0}^n\,\psi_n \cdot (m-n)!\,p^{m-n}\,\phi_{m_0-n_0}^{m-n}\,\psi_{m-n} \cdot \mathcal{C}_M^N = n!\,(m-n)!\,\mathcal{C}_m^n \cdot p^m\,\phi_{m_0}^m\,\psi_m$$

In $\mathbb{N}$, we simplify by $p^n p^{m-n} = p^m$ and $n!(m-n)!$ and get

$$\phi_{n_0}^n\,\psi_n\,\phi_{m_0-n_0}^{m-n}\,\psi_{m-n}\,\mathcal{C}_M^N = \mathcal{C}_m^n\,\phi_{m_0}^m\,\psi_m$$

Now switching to $\mathbb{Z}/p\mathbb{Z}$ using Eqs. (B.1), we derive:

$$n_0!\,(p-1)!^n\,(m_0-n_0)!\,(p-1)!^{m-n}\,\mathcal{C}_M^N \equiv \mathcal{C}_m^n\,m_0!\,(p-1)!^m\,[p]$$

Because $(p-1)!$ is invertible in $\mathbb{Z}/p\mathbb{Z}$, we can simplify by $(p-1)!^n(p-1)!^{m-n} = (p-1)!^m$. Then we rewrite using the binomial equation $m_0! = n_0!\,(m_0-n_0)!\,\mathcal{C}_{m_0}^{n_0}$ and deduce

$$n_0!\,(m_0-n_0)!\,\mathcal{C}_M^N \equiv \mathcal{C}_m^n\,m_0! \equiv \mathcal{C}_m^n\,n_0!\,(m_0-n_0)!\,\mathcal{C}_{m_0}^{n_0}\,[p]$$

Finally, as both $n_0 < p$ and $m_0 - n_0 < p$ hold, then $n_0!\,(m_0-n_0)!$ is invertible in $\mathbb{Z}/p\mathbb{Z}$ and, simplifying, we get $\mathcal{C}_M^N \equiv \mathcal{C}_m^n\mathcal{C}_{m_0}^{n_0}\,[p]$ as required.

Then we consider the alternative case where $n < m$ and $m_0 < n_0 < r$. In this case we have $M - N = (m-(n+1))p + (p-(n_0-m_0))$ and again we develop the binomial equation $N!\,(M-N)!\,\mathcal{C}_M^N = M!$ in $\mathbb{N}$ as

$$(np+n_0)!\,((m-(n+1))p + (p-(n_0-m_0)))!\,\mathcal{C}_M^N = (mp+m_0)!$$

and thus, using Eq. (B.2) and the binomial equation $m! = n!\,(m-n)!\,\mathcal{C}_m^n$, rewriting it into

$$n!\,p^n\,\phi_{n_0}^n\,\psi_n(m-(n+1))!\,p^{m-(n+1)}\,\phi_{p-(n_0-m_0)}^{m-(n+1)}\,\psi_{m-(n+1)}\mathcal{C}_M^N = n!\,(m-n)!\,\mathcal{C}_m^n\,p^m\,\phi_{m_0}^m\,\psi_m$$

that we then simplify by $n!\,(m-(n+1))!$ and $p^n\,p^{m-(n+1)} = p^{m-1}$ to get

$$\phi_{n_0}^n\,\psi_n\,\phi_{p-(n_0-m_0)}^{m-(n+1)}\,\psi_{m-(n+1)}\,\mathcal{C}_M^N = (m-n)\,\mathcal{C}_m^n\,p\,\phi_{m_0}^m\,\psi_m$$

Notice that p appears (at least once) on the right-hand side, thus switching to $\mathbb{Z}/p\mathbb{Z}$, we get:

$$n_0!\,(p-1)!^n\,(p-(n_0-m_0))!\,(p-1)!^{m-(n+1)}\,\mathcal{C}_M^N \equiv 0\,[p]$$

Now, as $n_0 < p$ and $p - (n_0 - m_0) < p$, the factorials $n_0!$ and $(p-(n_0-m_0))!$ are both invertible in $\mathbb{Z}/p\mathbb{Z}$. This is also the case of any power of $(p-1)!$. We deduce that $\mathcal{C}_M^N \equiv 0\,[p]$ which leads us to conclude $\mathcal{C}_M^N \equiv \mathcal{C}_m^n\mathcal{C}_{m_0}^{n_0}\,[p]$ as $\mathcal{C}_{m_0}^{n_0} = 0$ trivially follows from $m_0 < n_0$. □

Theorem 52 (Lucas [26]). *Whenever $u_0, \dots u_n, v_1, \dots, v_n < p$, the following identity holds:*

$$\binom{u_n p^n + \cdots + u_0}{v_n p^n + \cdots + v_0} \equiv \binom{u_n}{v_n} \cdots \binom{u_0}{v_0}\,[p].$$

Proof. By induction on the number n of coefficients using Lemma 51. □

Definition 53. *Let $a = a_n 2^n + \cdots + a_0$ and $b = b_n 2^n + \cdots + b_0$ representations in base 2. We denote $a \preccurlyeq b$ if $a_i \leq b_i$ holds for any $i \leq n$.*

Notice that the definition of $a \preccurlyeq b$ is irrelevant to which base 2 representations of a and b are picked up, i.e. it is not influenced by trailing zeros that might appear in front of (the representations of) a or b.

Corollary 54. *For any $a, b : \mathbb{N}$ we have $a \preccurlyeq b \iff \mathcal{C}_b^a \equiv 1\,[2]$.*

Appendix C. Avoiding Overflows in the Proof of Theorem 8

The section explains why we slightly modified the original proof of the elimination of bounded universal quantification [28] to avoid overflows when dealing with parallel multiplications, in search for a Diophantine encoding of the simultaneous equations $a_1 = b_1c_1, \dots, a_n = b_nc_n$, which can also be viewed as the identity between the vectors $\vec{a} = [a_1; \dots; a_n]$ and the scalar/dot product $\vec{b} \cdot \vec{c} = [b_1c_1; \dots; b_nc_n]$.

Following Matiyasevich [28], the component of those vectors are encoded sparsely within the digits in base r of the corresponding so called *ciphers* (see below for examples). Using the natural product of ciphers, he can somehow recover the identity between the components of $\vec{a}$ and those of $\vec{b} \cdot \vec{c}$. The basis r for the encoding is chosen large enough w.r.t. the component of $\vec{a}$, $\vec{b}$ and $\vec{c}$ to avoid digit overflow during the natural product of ciphers. However, this product generates artefacts on in-between digits that need to be masked out and, with Matiyasevich's choice for r as $r := 2^{2q}$, some overflow could occur within these artefacts, a subtlety which we speculate he might have missed. This overflow pointlessly complicates the correctness proof of masking. In our Coq code, we propose to avoid any digit overflow by increasing r to $r := 2^{4q}$, but, as we realized later on, any power of 2 greater than 2^{2q+1} would work as well.

Let us switch to the technical details explaining how the overflow occurs and how it can be avoided, with little impact on the rest of the argument. For the moment, we describe what Matiyasevich does in [28] so we stick with his definition of r as $r := 2^{2q}$ and consider the encoding of vectors of D_q^n as ciphers, written in base r, where $q \geq 1$ and $D_q := \{0, \dots, 2^q - 1\}$ is the set of allowed digits for vectors. He considers five vectors and their respective ciphers:

vector/length	cipher
$\vec{u} = [1; \dots; 1] \in D_q^n$	$u = 1.r^{2^1} + \cdots + 1.r^{2^n}$
$\vec{u}\,' = [0; 1; \dots; 1] \in D_q^{n+1}$	$u' = 0.r^{2^1} + 1.r^{2^2} + \cdots + 1.r^{2^{n+1}}$
$\vec{a} = [a_1; \dots; a_n] \in D_q^n$	$a = a_1r^{2^1} + \cdots + a_nr^{2^n}$
$\vec{b} = [b_1; \dots; b_n] \in D_q^n$	$b = b_1r^{2^1} + \cdots + b_nr^{2^n}$
$\vec{c} = [c_1; \dots; c_n] \in D_q^n$	$c = c_1r^{2^1} + \cdots + c_nr^{2^n}$.

Each summand is an actual digit of the cipher in base r. Notice that q is to be chosen large enough so that the digits in D_q cover the components of the vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$. Also, the ciphers have a sparse representation in base r, i.e. only the digits which correspond to powers r^{2^i} are possibly non-zero. Finally, most of the digits of base r are unused, i.e. only 2^q of them out of $r = 2^{2q}$ many, to avoid overflows when adding or multiplying digits. But remark that which overflows are avoided (or not) is the purpose of the whole discussion here.

The problem solved in [28] is to find, depending on q/r and n, Diophantine representations for (the ciphers of) $\vec{u}$ and $\vec{u}\,'$, and most importantly, for the identities $\vec{a} = \vec{b} + \vec{c}$ (the sum) and $\vec{a} = \vec{b} \cdot \vec{c}$ (the scalar product), implementing some kind of parallel addition and multiplication. We skip over the Diophantine encodings of $\vec{u}$ and $\vec{u}\,'$, of which the description spans about one fourth of [28], to directly consider the parallel sum and multiplication. For $\vec{a} = \vec{b} + \vec{c}$, the encoding is as straightforward as the addition of ciphers $a = b + c$, because the basis r is large enough to avoid overflows of digit additions, i.e. $2(2^q - 1) < r = 2^{2q}$, and no artefacts appear at powers different from r^{2^i}.

The case of parallel multiplication is more complicated. To recover the identity between $\vec{a}$ and the scalar product $\vec{b} \cdot \vec{c} = [b_1c_1; \dots; b_nc_n]$, Matiyasevich also uses the natural product of the ciphers but it generates artefacts on in-between digits that need to be masked out. Indeed, considering the identities

$$au = \sum_{i=1}^{n} a_i r^{2^i} \times \sum_{i=1}^{n} r^{2^i} = \sum_{i=1}^{n} a_i r^{2^{i+1}} + \sum_{1\le i<j\le n} (a_i + a_j) r^{2^i+2^j} \tag{C.1}$$

and

$$bc = \sum_{i=1}^{n} b_i r^{2^i} \times \sum_{i=1}^{n} c_i r^{2^i} = \sum_{i=1}^{n} b_i c_i r^{2^{i+1}} + \sum_{1\le i<j\le n} (b_i c_j + b_j c_i) r^{2^i+2^j} \tag{C.2}$$

by filtering out the two parts $\sum_{1\le i<j\le n}\cdots$ on the right hand side of the + sign, and then identifying the two values, we may recover the identity $\vec{a} = \vec{b}\cdot\vec{c}$.

For this, he uses $(r-1)u' = (r-1)r^{2^2} + \cdots + (r-1)r^{2^{n+1}}$ as a mask. This is the meaning of Equation (40) of page 3232 in [28], (slightly) reformulated here as

$$au \,\&\, (r-1)u' = bc \,\&\, (r-1)u' \tag{C.3}$$

where & is the digit by digit AND operator.[24] However, to show the equivalence between Equation (C.3) and the identity $\vec{a} = \vec{b}\cdot\vec{c}$, we have to make sure that the masking operator $x \mapsto x \,\&\, (r-1)u'$ effectively erases the two rightmost sums $\sum_{1\le i<j\le n}\cdots$ in Equations (C.1) and (C.2), while keeping the left sums intact. Proving this property is not even considered worthy of an explanation in [28] but it turned out to be not that obvious.

We first remark that *the digit by digit AND operator* & *does not commute with* + (in general), hence computing the mask can be tricky. However, & commutes with the digit by digit OR operator. Moreover OR and + coincide whenever the binary digits of the added numbers do not overlap (there are never two 1 on the same binary digit power in each number). As r is a power of 2, this generalises from base 2 to base r, and provided the below summations are base r representations, one can commute the sum/+ with & and apply masking component by component, e.g. one can show the identity

$$\left(\sum_{i\in I} x_i r^{\varphi(i)}\right) \,\&\, \left(\sum_{i\in I} m_i r^{\varphi(i)}\right) = \sum_{i\in I} (x_i \,\&\, m_i) r^{\varphi(i)}$$

as soon as $x_i < r$ and $m_i < r$ holds for any $i \in I$, and $\varphi : I \to \mathbb{N}$ injectively maps the finite type I. But this implies that we have to check that Equations (C.1) and (C.2) contain such genuine base r representations.

In that spirit, if $i < j$ holds then the powers r^{2^k} and $r^{2^i+2^j}$ are never the same power of r. So these represent two distinct digit positions in base r and hence, in Equations (C.1) and (C.2), those digits from the left of +, and those from the right of + do not overlap/interfere. This also motivates the use of the mask $(r-1)u' = (r-1)r^{2^2} + \cdots + (r-1)r^{2^{n+1}}$ which indeed filters out digits at any $r^{2^i+2^j}$ position while leaving those at r^{2^i} intact.

The way Equations (C.1) and (C.2) are written unfortunately suggests that these are base r representations of both au and bc. However, formally, we have to verify that the two rightmost sums $\sum_{1\le i<j\le n}\cdots$ are proper base r representations. This is not the case for (C.2) and we here give a counter example. On the one hand, it is true that $2^i + 2^j = 2^{i'} + 2^{j'}$ implies $(i,j) = (i',j')$ when $i < j$ and $i' < j'$, ensuring that digits at those powers of r do not accidentally overlap. On the other hand, it possible that the value $b_i c_j + b_j c_i$ overflows the digit range of base r, even though considerable room was initially reserved to avoid that

[24]The Diophantine encoding of the AND operator is itself derived from Lucas's theorem.

situation. For instance, when $b_i = c_j = b_j = c_i = 2^q - 1$ (the maximum allowed digit in D_q), then we get $b_i c_j + b_j c_i = 2^{2q+1} - 2^{q+2} + 2 \geq r + 2$ as soon as $q > 1$. So this component at digit position $r^{2^i+2^j}$ overflows and we would have to consider the spill out at digit position $r^{2^i+2^j+1}$ and show that it does not overlap with the other digits or other spill outs. This is a property which we think holds true but the proof of it would add significant formal complexity.

On the other hand, with our proposed alternative choice of $r := 2^{4q}$, keeping the same set $D_q = \{0, \ldots, 2^q - 1\}$ for the allowed digits in vectors, the overflow does not occur any more, i.e. no spill out, and Equations (C.1) and (C.2) become genuine base r representations, allowing smooth component by component masking.

Because there is an obvious way out of a tricky overflow management problem, we think that it is possible that Matiyasevich simply did not notice the eventuality of an overflow in the to be masked parts of Equation (C.2). We speculate this because this overflow does not occur in the parts which are left intact by the mask, at the left of the + sign. With this remark, we do not imply that Equation (40) of [28] is improper in any way. However, the formal proof of its equivalence with the identity $\vec{a} = \vec{b} \cdot \vec{c}$ is really more complicated when overflows occur and we did not try to prove it in the case where $r := 2^{2q}$, this situation being straightforward to avoid with a change (of otherwise low impact) in the value of r.

Appendix D. Proof of Proposition 14

Let $(p_1, q_1), \ldots, (p_n, q_n)$ be a sequence of pairs in $\mathbb{N} \times \mathbb{N}$. We establish the equivalence:

$$\sum_{i=1}^{n} 2p_i q_i = \sum_{i=1}^{n} p_i^2 + q_i^2 \;\leftrightarrow\; p_1 = q_1 \wedge \cdots \wedge p_n = q_n.$$

Proof. We give an elementary arithmetic justification of the result, proof which involves none of the high-level tools of mathematical analysis. We first show the two following statements

$$2ab \leq a^2 + b^2 \quad \text{and} \quad 2ab = a^2 + b^2 \leftrightarrow a = b \qquad \text{for any } a, b : \mathbb{N} \tag{D.1}$$

Assuming w.l.o.g. that $a \leq b$, we can write $b = a + \delta$ with $\delta \in \mathbb{N}$ and then, for $\bowtie \in \{\leq, =\}$ we have $2ab \bowtie a^2 + b^2 \leftrightarrow 2a^2 + 2a\delta \bowtie a^2 + a^2 + 2a\delta + \delta^2 \leftrightarrow 0 \bowtie \delta^2$ hence the desired result.

From the left inequality (D.1), we easily generalize by induction on n and obtain the following inequality:

$$\textstyle\sum_{i=1}^{n} 2p_i q_i \leq \sum_{i=1}^{n} p_i^2 + q_i^2 \tag{D.2}$$

Now we can proceed with the proof of the main stated equivalence. The *if case* is obvious so we only describe the *only if case*. Hence we show that $\sum_{i=1}^{n} 2p_i q_i = \sum_{i=1}^{n} p_i^2 + q_i^2$ entails $p_i = q_i$ for all $i \in [1, n]$. We proceed by induction on n again. The base case $n = 0$ is trivial. For the inductive step $1 + n$, let us assume

$$\textstyle\sum_{i=1}^{n} 2p_i q_i + 2p_{n+1} q_{n+1} = \sum_{i=1}^{n} (p_i^2 + q_i^2) + p_{n+1}^2 + q_{n+1}^2. \tag{D.3}$$

By the left inequality of (D.1) and inequality (D.2), we have both $2p_{n+1} q_{n+1} \leq p_{n+1}^2 + q_{n+1}^2$ and $\sum_{i=1}^{n} 2p_i q_i \leq \sum_{i=1}^{n} p_i^2 + q_i^2$. The only possibility for the identity in hypothesis (D.3) to hold is that both inequalities are in fact identities, hence both $2p_{n+1} q_{n+1} = p_{n+1}^2 + q_{n+1}^2$ and $\sum_{i=1}^{n} 2p_i q_i = \sum_{i=1}^{n} p_i^2 + q_i^2$ hold. From this we derive $p_{n+1} = q_{n+1}$ by the equivalence on the right of (D.1) and $p_1 = q_1, \ldots, p_n = q_n$ by the induction hypothesis. $\square$