Synthetic Undecidability of MSELL via FRACTRAN Mechanised in Coq

Dominique Larchey-Wendling

Synthetic Undecidability of MSELL via FRACTRAN Mechanised in Coq

Dominique Larchey-Wendling
Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France

Abstract

We present an alternate undecidability proof for entailment in (intuitionistic) multiplicative sub-exponential linear logic (MSELL). We contribute the result and its mechanised proof to the Coq library of synthetic undecidability. The result crucially relies on the undecidability of the halting problem for two counters Minsky machines, which we also hand out to the library. As a seed of undecidability, we start from FRACTRAN halting which we (many-one) reduce to Minsky machines termination by implementing Euclidean division using two counters only. We then give an alternate presentation of those two counters machines as sequent rules, where computation is performed by proof-search, and halting reduced to provability. We use this system called non-deterministic two counters Minsky machines to describe and compare both the legacy reduction to linear logic, and the more recent reduction to MSELL. In contrast with that former MSELL undecidability proof, our correctness argument for the reduction uses trivial phase semantics in place of a focused calculus.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Linear logic; Theory of computation → Type theory

Keywords and phrases Undecidability, computability theory, many-one reduction, Minsky machines, Fractran, sub-exponential linear logic, Coq

Digital Object Identifier 10.4230/LIPIcs.FSCD.2021.18

Supplementary Material *Software (Source Code)*:
`https://github.com/uds-psl/coq-library-undecidability/releases/tag/FSCD-2021`
archived at `swh:1:rev:4115398f10c42a41833036f8c4500f24233cc9a7`

Funding *Dominique Larchey-Wendling*: partially supported by the TICAMORE project (ANR grant 16-CE91-0002).

1 Introduction

In the late 80s, Lincoln et al. [17] gave a first proof of the undecidability of propositional linear logic (LL) via a many-one reduction from "and-branching two-counter machines without zero-test," a variant of Minsky machines extended with a *fork* instruction. The ability of LL to simulate the increment and decrement operations characteristic of Petri net operations was spotted very early and lead to paradigmatically characterise LL as a logic for counting resources. Critically, the exponential modality ! can be exploited to allow *unbounded reuse* of some specific resources like (Petri net) transitions or (Minsky machines) instructions.

To establish undecidability, one needed of course to go beyond Petri nets because those have a decidable reachability problem, a major result from the early 80s with a very involved proof still actively revisited nowadays [18, 15, 4, 16, 5]. As opposed to Minsky machines, Petri nets are not able to perform zero tests *combined* with a jump. Hence, the main idea of the reduction was to use forking to separate comparison with zero from jumping. In there, the additive conjunction of LL plays a central role:

$$\frac{\Sigma \vdash \alpha = 0 \qquad \Sigma \vdash \mathtt{jump}}{\Sigma \vdash \alpha = 0 \,\&\, \mathtt{jump}}$$

Indeed this right introduction rule *duplicates* the context Σ in the left and right sub-proofs which allows to delegate checking for emptiness in the left branch, and jumping in the right branch, the requirement of the two premises ensuring the correctness of the combination.

The same idea was then exploited to establish the undecidability of smaller fragments of LL [10, 11, 13]. In our own work [8], we gave the first mechanisation of the undecidability of the elementary fragment of LL in Coq, and hence ILL, based on this forking idea as well.

The multiplicative and exponential fragment (MELL) of linear logic lacks additive connectives, and is thus unable to duplicate the context. Arguably, the question of its decidability is the most important open conjecture (see e.g. [14]) in the context of LL, even with some claimed proof of decidability [1], later refuted [20]. The recent encoding of two counters Minsky machines in a fragment of LL lacking additives opened a new logical perspective on the MELL question [2]. Indeed, at the cost of a more complex modal structure, forking with & can be replaced with a constraint on modalities in the *promotion rule*. This extension of MELL is called multiplicative sub-exponential linear logic (MSELL).

In this paper, we mechanise this reduction from two counters Minsky machines to MSELL, following the encoding of [2]. However, we proceed in the intuitionistic version of the logic (two sided sequents with exactly one conclusion formula) that we call IMSELL. That fragment only involves the linear implication $\multimap$ and the modalities $!^m$ with $m \in \Lambda = \{a, b, \infty\}$, so it is short to describe. It is also convenient for comparing with our previous encoding in (elementary) intuitionistic LL [13, 8]. Schematically, we describe and mechanise the following many-one reduction chain, explained below:

$$\mathsf{FRACTRAN}_{\mathrm{reg}} \preceq \mathsf{MMA0}_2 \preceq \mathsf{MM}_{\mathrm{nd}} \preceq \mathsf{IMSELL}_{\Lambda}$$

Our work is based on and contributes to the Coq library of undecidability proofs; see [9] for a quick overview. As opposed to the legacy LL argument of forking, which can cope with Minsky machines using arbitrary many counters, the MSELL and IMSELL reductions rely on two counters machines in an essential way. Hence, we first had to implement the undecidability of the "halting on the zero state" problem for two counters Minsky machines, that we denote $\mathsf{MMA0}_2$; see Section 3. To establish this, we could follow the legacy reduction from many counters to just two by Minsky [19], that uses a Gödel coding of lists of natural numbers as essential trick. Following [12], we profit from the FRACTRAN language [3] that adequately abstracts away the Gödel coding phase, hence we establish the undecidability of $\mathsf{MMA0}_2$ by reducing from (regular) FRACTRAN halting instead, mainly by mechanising Euclidean division with two counters only.

In Section 4, we provide a sequent calculus style presentation of $\mathsf{MMA0}_2$, i.e. the instance $(\mathcal{M}, x, y)$ of $\mathsf{MMA0}_2$ is viewed as a sequent $\Sigma_{\mathcal{M}} /\!\!/_{\mathrm{n}} x \oplus y \vdash 1$, and the Minsky machine $\mathcal{M}$ starting at PC value 1 with register values (x, y) halts on the zero state if and only if the sequent $\Sigma_{\mathcal{M}} /\!\!/_{\mathrm{n}} x \oplus y \vdash 1$ has a derivation. We call this system and the associated problem non-deterministic two counters Minsky machines, denoted $\mathsf{MM}_{\mathrm{nd}}$. As $\mathsf{MM}_{\mathrm{nd}}$ is essentially a specialised proof theory for Minsky machines, reducing from it to logical entailment problems mainly consists in transformations of derivations. Hence Section 5, targeting IMSELL, can be understood from a proof theoretic perspective only. In there, we gives details of the reduction of two counters halting, explaining how the legacy fork trick for ILL is replaced by the modal constraints in the promotion rule of IMSELL, following [2]. Additionally, our proof of correctness of the reduction differs significantly: the former proof relies on the completeness of focused proof-search; we instead generalise our semantic argument [8], i.e. we prove and use the soundness of trivial phase semantics for IMSELL.

Our contributions in this work are the following. First, via a proof theoretic presentation of Minsky machines, a comparison of their encoding in ILL and in IMSELL, explaining precisely how and where forking is replaced with modalities. Then, a novel completeness proof of the IMSELL reduction based on the soundness of trivial phase semantics. On the implementation side, we provide the mechanized proof of the undecidability of two counters Minsky machines (with two different presentations), and of IMSELL. The Coq `8.13` code is available at

`https://github.com/uds-psl/coq-library-undecidability/tree/FSCD-2021`

and (sub-)section titles generally provide hyperlinks to the relevant source code. Our code extends the existing library with about 1800 loc, 1200 of which concern the reductions from FRACTRAN to $\mathsf{MM}_{\mathrm{nd}}$, and 600 more for the $\mathsf{MM}_{\mathrm{nd}}$ to IMSELL reduction.

The paper describes the major steps of the implementation, in the language of type theory, but should be readable with only basic knowledge of it. We denote $\mathbb{P}$ (resp. $\mathbb{B}$ and $\mathbb{N}$) the type of propositions (resp. Booleans and natural numbers). We write $\mathbb{L}\,X$ for the type of *lists* over X, where $[\,]$ represents the empty list, $x :: l$ for the *cons* operation, $l +\!\!+ l'$ for the *concatenation* of two lists, and $|l| : \mathbb{N}$ for the length of l. We write X^n for *vectors* $\vec{v}$ over type X with length $n : \mathbb{N}$, and $\mathbb{F}_n$ for the *finite type* with exactly n elements. Notations for lists are overloaded for vectors. Moreover, for $p : \mathbb{F}_n$ and $x : X$, we write $\vec{v}_p$ for the p-th component of $\vec{v} : X^n$ and $\vec{v}\,\{x/p\}$ when $\vec{v}$ is updated with x at component p. The (non-dependent) sum $A + B$ represents a computable/Boolean choice between an inhabitant of A or an inhabitant of B. In the case where A and B are propositions (i.e. of type $\mathbb{P}$), the sum $A + B$: `Type` is stronger than the disjunction $A \vee B : \mathbb{P}$, because one cannot computably determine which of A or B holds in the later case. We also use the type-theoretic dependent sum $\Sigma_{x:A} B(x)$, denoted $\{x : A \mid B\,x\}$ in Coq[1], inhabited by (Coq computable) values $x : A$ paired with a proof of $B\,x$.

The framework of synthetic computability [7] is based on the notion of many-one reduction. If $P : X \to \mathbb{P}$ is a predicate (on X) and $Q : Y \to \mathbb{P}$ is a predicate, we say that P *many-one reduces to* Q and write $P \preceq Q$ if there is a Coq 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. Because we work in constructive (axiom-free) Coq, all definable functions are computable and thus the requirement of the computability of the reduction function f above can be discarded. If $P \preceq Q$ and P is undecidable then so is Q.

2 The FRACTRAN seed (files `FRACTRAN.v` and `fractran_utils.v`)

The FRACTRAN model of computation is very simple to describe. It was introduced by Conway [3] but its main idea, the Gödel coding of a list $[x_1; x_2; \ldots; x_n]$ of natural numbers as the number $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}$, predates the introduction of FRACTRAN by several decades.

In the FRACTRAN formalism, programs are lists of formal fractions, i.e. terms Q of type $\mathbb{L}\,(\mathbb{N} \times \mathbb{N})$.[2] The state of a program is modelled as a natural number $x : \mathbb{N}$. A fraction p/q is *executable at state* x if $x{\cdot}p/q$ is a natural number (i.e. not a proper fraction) and in that case this is the new state. To allow FRACTRAN to discriminate, and b.t.w. turn it into a Turing complete model of computation, the first executable fraction in the list has to be picked up at each step of computation. The program Q *stops* when no fraction in the list is executable.

[1] or simply $\{x \mid B\,x\}$ when the type of x is guessable.
[2] For the moment, we can ignore the case of degenerate fractions like $p/0$.

Formally this prose translates in a straightforward *inductive definition*, not even involving the algebraic notion of fraction, and characterized by the two inductive rules below:

$$\frac{qy = px}{p/q :: Q \;/\!/_{\mathrm{F}}\; x \succ y} \qquad \frac{q \nmid px \qquad Q \;/\!/_{\mathrm{F}}\; x \succ y}{p/q :: Q \;/\!/_{\mathrm{F}}\; x \succ y}$$

where $u \nmid v$ means u does not divide v, and $Q \;/\!/_{\mathrm{F}}\; x \succ y$ reads as the FRACTRAN program Q *transforms state x into state y* in one step of computation. The computation is *terminated at x*, denoted $Q \;/\!/_{\mathrm{F}}\; x \not\succ \star$, when there is no possibility to perform one step from x, and *termination from x*, denoted by $Q \;/\!/_{\mathrm{F}}\; x\!\downarrow$, means there is exists a sequence of steps starting at state x at leading to the terminated state y. Formally, this gives us:

$$Q \;/\!/_{\mathrm{F}}\; x \not\succ \star := \forall y, \neg(Q \;/\!/_{\mathrm{F}}\; x \succ y) \quad \text{and} \quad Q \;/\!/_{\mathrm{F}}\; x\!\downarrow := \exists y, (Q \;/\!/_{\mathrm{F}}\; x \succ^{*} y \wedge Q \;/\!/_{\mathrm{F}}\; y \not\succ \star)$$

There are some obvious quick remarks to make here: the empty program $Q = [\,]$ is terminated in any state; unless $Q = [\,]$, the state 0 is not terminated. The step relation is strongly decidable in the sense that one can discriminate between non-terminated and terminated states, and in the former case, computationally find a next state, expressed below using (Coq) dependent types:

▶ **Proposition 1.** *For any FRACTRAN program we have* $\forall x,\ \{y \mid Q \;/\!/_{\mathrm{F}}\; x \succ y\} + (Q \;/\!/_{\mathrm{F}}\; x \not\succ \star)$.

Proof. By structural induction on the list Q combined with Euclidean division. ◀

The dependent sum $\{y \mid Q \;/\!/_{\mathrm{F}}\; x \succ y\}$ represents a (computable) state y together with a proof that y is next after x. The proposition $Q \;/\!/_{\mathrm{F}}\; x \not\succ \star$ is for a proof that x is a terminated state. Finally, the outer sum $+$ represents a computable choice between the two alternatives.

Non-regular fractions like $0/0$ can make the computation non-deterministic; and non-proper fractions like $1/1$ or $6/2$ are always executable, implying that programs including such fractions have no terminating state. Non-deterministic step relations involves at least two different notions of termination, weak termination as defined above, and strong termination, when no infinite sequence of steps from x can exist. For our use of FRACTRAN, it does not matter because we only consider *regular FRACTRAN programs* where formal fractions $p/0$ are disallowed. Regular FRACTRAN is a universal model of computation, up to a Gödel encoding of natural numbers [3].[3]

▶ **Definition 2.** *A* $\mathsf{FRACTRAN}_{\mathrm{reg}}$ *instance is a pair composed of a list of regular formal fractions and a natural number, i.e. of type* $\big\{(Q, x) : \mathbb{L}\,(\mathbb{N} \times \mathbb{N}) \times \mathbb{N} \mid \forall p,\ p/0 \notin Q\big\}$*, and the question asked is whether* $Q \;/\!/_{\mathrm{F}}\; x\!\downarrow$ *holds or not.*

Notice the use of a dependent sum in the type of instances where the predicate $\forall p,\ p/0 \notin Q$ acts as a guard against non-regular instances.

▶ **Theorem 3** (mechanized in [12])**.** *There is a many-one reduction from the Halting problem for single tape Turing machines to termination of regular FRACTRAN programs, i.e.* $\mathsf{Halt} \preceq \mathsf{FRACTRAN}_{\mathrm{reg}}$*, and thus* $\mathsf{FRACTRAN}_{\mathrm{reg}}$ *is undecidable.*

As a consequence, we can safely use $\mathsf{FRACTRAN}_{\mathrm{reg}}$ as our seed of undecidability for the chain of many-one reductions described in this paper.

[3] However, e.g. the function $n \mapsto 0$ cannot be *directly* represented by a FRACTRAN program where n would be the starting state leading, after finitely many steps of computation, to the 0 terminated state.

3 From FRACTRAN to two registers alternate Minsky machines

3.1 Alternate Minsky machines (files `MM.v` and `mma_defs.v`)

We describe alternate n counters (or registers) Minsky machines, where states are described as $(i, \vec{v}) : \mathbb{N} \times \mathbb{N}^n$. The number $i : \mathbb{N}$ is the current program counter (PC) value and the vector $\vec{v} : \mathbb{N}^n$ describes the n current values of the registers. When convenient, we also denote states as st, st_1... Instructions consist of either incrementing $\mathtt{INC_a}\, x$ a register by one, or decrementing $\mathtt{DEC_a}\, x\, j$ a register by one. Notice that when the register values 0, it is not possible to decrement it. So a conditional jump at j helps at discriminating between the zero and non-zero cases. Unless there is a conditional jump, the default behaviour after the register is updated is to jump to the next instruction at PC $+ 1$. In contrast with [8, 12] where the $\mathtt{DEC}\, x\, j$ instruction jumps at j when $\vec{v}_x$ is empty, here in $\mathtt{DEC_a}\, x\, j$, the jump occurs when decrementing is possible, and this is the reason we call these machines alternate and suffix instructions with an "a" just as a reminder for this alternate semantics. Hence a single (atomic) step of computation is described by the following relation

$$\begin{array}{ll}
\mathtt{INC_a}\, x \mathbin{/\!\!/_a} (i, \vec{v}) \succ \big(1{+}i, \vec{v}\{(1{+}u)/x\}\big) & \text{when } \vec{v}_x = u \\
\mathtt{DEC_a}\, x\, j \mathbin{/\!\!/_a} (i, \vec{v}) \succ \big(j, \vec{v}\{u/x\}\big) & \text{when } \vec{v}_x = 1{+}u \\
\mathtt{DEC_a}\, x\, j \mathbin{/\!\!/_a} (i, \vec{v}) \succ (1{+}i, \vec{v}) & \text{when } \vec{v}_x = 0
\end{array}$$

where $\sigma \mathbin{/\!\!/_a} (i_1, \vec{v}_1) \succ (i_2, \vec{v}_2)$ reads as the MMA_n instruction σ at PC value i_1 transforms the state $(i_1, \vec{v}_1)$ into the state $(i_2, \vec{v}_2)$. Notice that this alternate semantics allows to implement a universal jump without needing an empty register, which will be critical when we will need to limit the number of registers to $n = 2$.

▶ **Proposition 4.** *The step relation for alternate Minsky machines is deterministic and total:*

1. *for any states st, st_1 and st_2, if $\sigma \mathbin{/\!\!/_a} st \succ st_1$ and $\sigma \mathbin{/\!\!/_a} st \succ st_2$ then $st_1 = st_2$;*
2. *for any state $(i_1, \vec{v}_1)$, one can compute a state $(i_2, \vec{v}_2)$ such that $\sigma \mathbin{/\!\!/_a} (i_1, \vec{v}_1) \succ (i_2, \vec{v}_2)$.*

This means that starting from state $(i_1, \vec{v}_1)$, the instruction σ at PC value i_1 (provided there is one) changes the state in exactly one possible way, and the new state $(i_2, \vec{v}_2)$ is Coq-computable from the initial state $(i_1, \vec{v}_1)$. So the only way for such programs to terminate is to jump to a PC value which holds no instruction.

A program is pair $(i, P) : \mathbb{N} \times \mathbb{L}\, \mathsf{MMA}_n$ composed of the PC value of its first instruction and the sequence P of consecutive instructions of which is it composed. Informally, the program $(i, [\sigma_0; \ldots; \sigma_{m-1}])$ would be read as e.g. $i : \sigma_0;\ 1{+}i : \sigma_1;\ \ldots;\ m{-}1{+}i : \sigma_{m-1}$ using labelled instructions. We define the k-steps relation for a program (i, P) inductively with

$$\frac{}{(i, P) \mathbin{/\!\!/_a} st \succ^0 st} \qquad \frac{i_1 = |L| + i \quad P = L +\!\!+ \sigma :: R \quad \sigma \mathbin{/\!\!/_a} (i_1, \vec{v}_1) \succ st_2 \quad (i, P) \mathbin{/\!\!/_a} st_2 \succ^k st_3}{(i, P) \mathbin{/\!\!/_a} (i_1, \vec{v}_1) \succ^{1+k} st_3}$$

where the constraints $i_1 = |L| + i$ and $P = L +\!\!+ \sigma :: R$ impose that the instruction at PC value i_1 of (i, P) is σ. From its structure as lists of instructions, there is at most one instruction at a given PC value and thus, the k-steps relation is also deterministic. However, it can be non-total if a jump outside of the interval $[i, |P| - 1 + i]$ occurs, the lack of an instruction blocking the computation. We write $\mathtt{out}\ j\ (i, P) := j < i \vee |P| + i \leq j$ when there is no instruction at j in (i, P), and because of Proposition 4 (totality), blocked states are exactly those outside of the code, i.e. $\mathtt{out}\ i_1\ (i, P) \leftrightarrow \forall st_2, \neg (i, P) \mathbin{/\!\!/_a} (i_1, \vec{v}_1) \succ^1 st_2$.

We define the predicates of *computation*, of *progress*, of *output* and of *termination* as:

$$\begin{aligned}
(i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \succ^{*} st_2 &:= \exists k, (i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \succ^{k} st_2 && \text{(computation)}\\
(i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \succ^{+} st_2 &:= \exists k > 0, (i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \succ^{k} st_2 && \text{(progress)}\\
(i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \rightsquigarrow (i_2,\vec{v}_2) &:= (i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \succ^{*} (i_2,\vec{v}_2) \wedge \mathtt{out}\ i_2\ (i,P) && \text{(output)}\\
(i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \downarrow &:= \exists st_2, (i,P) \mathbin{/\!\!/_{\mathrm{a}}} st_1 \rightsquigarrow st_2 && \text{(termination)}
\end{aligned}$$

output meaning that we have computed until we reach a state blocking the computation.

▶ **Definition 5.** *The problems* MMA_2 *and* $\mathsf{MMA0}_2$ *have the same instances: a pair* $(P,\vec{v})$ *where* P *is a list of MMA*$_2$ *instructions (starting at PC value 1) and the vector* $\vec{v} : \mathbb{N}^2$ *represents the initial values of the two registers.* MMA_2 *asks for termination, i.e.* $(1,P) \mathbin{/\!\!/_{\mathrm{a}}} (1,\vec{v})\downarrow$. $\mathsf{MMA0}_2$ *asks for termination on the zero state, i.e.* $(1,P) \mathbin{/\!\!/_{\mathrm{a}}} (1,\vec{v}) \rightsquigarrow (0,[0;0])$.

We mention that there is substantial machinery for (alternate) Minsky machines, and more generally PC based state machines, in the Coq library of undecidable problem initially described in [8]. These tools enable *modular reasoning* in those models of computation.

3.2 A basic MMA_n library up to Euclidean division (file `mma_utils.v`)

We specify, implement and verify a small library to compute some basic operations with MMA_n. For this section, $n : \mathbb{N}$ is a fixed number of registers but all the below sub-programs involve at most two registers. In the coming statements, the vector $\vec{v} : \mathbb{N}^n$ is implicitly universally quantified over. The names $i,j,p,q : \mathbb{N}$ range over PC values, $k : \mathbb{N}$ over natural number constants, and the names $x,t,s,d : \mathbb{F}_n$ over registers indices.

Let us start with the easy simulations of an unconditional jump, the nullification of register x and the operation that adds k units to register x.

▶ **Proposition 6.** *For* $i,j : \mathbb{N}$ *and* $x : \mathbb{F}_n$ *we have* $(i,\mathtt{JUMP}_{\mathrm{a}}\, j\, x) \mathbin{/\!\!/_{\mathrm{a}}} (i,\vec{v}) \succ^{+} (j,\vec{v})$ *where* $\mathtt{JUMP}_{\mathrm{a}}\, j\, x := [\mathtt{INC}_{\mathrm{a}}\, x; \mathtt{DEC}_{\mathrm{a}}\, x\, j]$.

▶ **Proposition 7.** *For* $x : \mathbb{F}_n$ *and* $i : \mathbb{N}$*, we have* $(i,\mathtt{NULL}_{\mathrm{a}}\, x\, i) \mathbin{/\!\!/_{\mathrm{a}}} (i,\vec{v}) \succ^{+} (1+i,\vec{v}\{0/x\})$ *where* $\mathtt{NULL}_{\mathrm{a}}\, x\, i := [\mathtt{DEC}_{\mathrm{a}}\, x\, i]$.

▶ **Proposition 8.** *For* $i,k : \mathbb{N}$*,* $x : \mathbb{F}_n$*, we have* $(i,\mathtt{INCS}_{\mathrm{a}}\, x\, k) \mathbin{/\!\!/_{\mathrm{a}}} (i,\vec{v}) \succ^{*} (k+i,\vec{v}\{(k+\vec{v}_x)/x\})$ *where* $\mathtt{INCS}_{\mathrm{a}}\, x\, k := [\mathtt{INC}_{\mathrm{a}}\, x; \ldots; \mathtt{INC}_{\mathrm{a}}\, x]$ *is of length* $|\mathtt{INCS}_{\mathrm{a}}\, x\, k| = k$.

Then we simulate test for emptiness of register x, jumping to PC value p when x is empty, or else to the end of the sub-program otherwise. Registers are restored to their initial values when the sub-program is finished (assuming p points outside of its code).

▶ **Proposition 9.** *For* $x : \mathbb{F}_n$ *and* $p,i : \mathbb{N}$ *we have* $(i,\mathtt{EMPTY}_{\mathrm{a}}\, x\, p\, i) \mathbin{/\!\!/_{\mathrm{a}}} (i,\vec{v}) \succ^{+} (j,\vec{v})$ *where* $\mathtt{EMPTY}_{\mathrm{a}}\, x\, p\, i := [\mathtt{DEC}_{\mathrm{a}}\, x\, (3+i); \mathtt{JUMP}_{\mathrm{a}}\, p\, x; \mathtt{INC}_{\mathrm{a}}\, x]$*, and* $j := p$ *in case* $\vec{v}_x = 0$*, or else* $j := 4+i$ *in case* $\vec{v}_x \neq 0$.

Notice that this sub-program is of length $|\mathtt{EMPTY}_{\mathrm{a}}\, x\, p\, i| = 4$ (despite looking 3), because we abuse the list notation $[\ldots;\ldots;\ldots]$ by allowing dots to be not only single instructions but also lists of instructions such as $\mathtt{JUMP}_{\mathrm{a}}\, p\, x$. Hence, $\mathtt{EMPTY}_{\mathrm{a}}\, x\, p\, i$ is formally defined as $\mathtt{DEC}_{\mathrm{a}}\, x\, (3+i) :: \mathtt{JUMP}_{\mathrm{a}}\, p\, x +\!\!+ \mathtt{INC}_{\mathrm{a}}\, x :: [\,]$ but we choose the friendly display for readability.

We now simulate the transfer of the contents of register s (for source) to d (for destination).

▶ **Proposition 10.** *For* $s \neq d : \mathbb{F}_n$ *and* $i : \mathbb{N}$ *we have* $(i,\mathtt{TRANSFER}_{\mathrm{a}}\, s\, d\, i) \mathbin{/\!\!/_{\mathrm{a}}} (i,\vec{v}) \succ^{+} (3+i,\vec{w})$ *where* $\mathtt{TRANSFER}_{\mathrm{a}}\, s\, d\, i := [\mathtt{INC}_{\mathrm{a}}\, d; \mathtt{DEC}_{\mathrm{a}}\, s\, i; \mathtt{DEC}_{\mathrm{a}}\, d\, (3+i)]$ *and* $\vec{w} := \vec{v}\{0/s\}\{(\vec{v}_s+\vec{v}_d)/d\}$.

We simulate multiplication of a register by a constant. The idea is similar to transfer but instead of transferring one for one, when one unit is removed from s, k units are added to d.

▶ **Proposition 11.** *For $s \neq d : \mathbb{F}_n$, $k, i : \mathbb{N}$ we have $(i, \texttt{MULT\_CST}_\text{a}\, s\, d\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (j, \vec{w})$ where $\texttt{MULT\_CST}_\text{a}\, s\, d\, k\, i := [\texttt{DEC}_\text{a}\, s\, (3+i); \texttt{JUMP}_\text{a}\, (5+k+i)\, s; \texttt{INCS}_\text{a}\, d\, k; \texttt{JUMP}_\text{a}\, i\, s]$, $j := 5 + k + i$, $\vec{w} := \vec{v}\{0/s\}\{(k\vec{v}_s + \vec{v}_d)/d\}$, and $|\texttt{MULT\_CST}_\text{a}\, s\, d\, k\, i| = 5 + k$.*

We simulate the minus k operation (with overflow management), jumping to PC value p when k units can be removed from register x, or else to PC value q when register x contains less than k units (overflow).

▶ **Proposition 12.** *For $p, q, k, i : \mathbb{N}$ and $x : \mathbb{F}_n$, we define*

$$\texttt{DECS}_\text{a}\, x\, p\, q\, k\, i := [\texttt{DEC}_\text{a}\, x\, (3+i); \texttt{JUMP}_\text{a}\, q\, x; \ldots; \texttt{DEC}_\text{a}\, x\, (3k+i); \texttt{JUMP}_\text{a}\, q\, x; \texttt{JUMP}_\text{a}\, p\, x]$$

where the pattern $\texttt{DEC}_\text{a}\, x\, (3u+i); \texttt{JUMP}_\text{a}\, q\, x$ is repeated for $u = 1, \ldots, k$.

Depending on the comparison between $\vec{v}_x$ and k, we have the following:

- *if $\vec{v}_x < k$ then $(i, \texttt{DECS}_\text{a}\, x\, p\, q\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (q, \vec{v}\{0/x\})$;*
- *if $\vec{v}_x \geq k$ then $(i, \texttt{DECS}_\text{a}\, x\, p\, q\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (p, \vec{v}\{\vec{v}_x - k/x\})$.*

Using an extra temporary register t, we implement a non-destructive minus k operation (with overflow management).

▶ **Proposition 13.** *For $x \neq t : \mathbb{F}_n$ and $p, q, k, i : \mathbb{N}$, we define*

$$\texttt{DECS\_COPY}_\text{a}\, x\, t\, p\, q\, k\, i := \begin{bmatrix} \texttt{DEC}_\text{a}\, x\, (4-1+i); \texttt{JUMP}_\text{a}\, q\, x; \texttt{INC}_\text{a}\, t; \\ \ldots \\ \texttt{DEC}_\text{a}\, x\, (4k-1+i); \texttt{JUMP}_\text{a}\, q\, x; \texttt{INC}_\text{a}\, t; \\ \texttt{JUMP}_\text{a}\, p\, x \end{bmatrix}$$

where the pattern $\texttt{DEC}_\text{a}\, x\, (4u-1+i); \texttt{JUMP}_\text{a}\, q\, x; \texttt{INC}_\text{a}\, t$ is repeated for $u = 1, \ldots, k$.

Depending on the comparison between $\vec{v}_x$ and k, we have the following:

- *if $\vec{v}_x < k$ then $(i, \texttt{DECS\_COPY}_\text{a}\, x\, t\, p\, q\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (q, \vec{v}\{0/x\}\{(\vec{v}_x + \vec{v}_t)/t\})$;*
- *if $\vec{v}_x \geq k$ then $(i, \texttt{DECS\_COPY}_\text{a}\, x\, t\, p\, q\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (p, \vec{v}\{(\vec{v}_x - k)/x\}\{(k + \vec{v}_t)/t\})$.*

The length is $|\texttt{DECS\_COPY}_\text{a}\, x\, t\, p\, q\, k\, i| = 2 + 4k$.

Notice that the initial value of t has to be known if one wants to recover the initial value of x, e.g. if the initial value of t is 0 and x contains less than k units, then once the computation is finished, t contains a copy of the initial value of x.

We implement a non-destructive computation of a divisibility test of register x by a constant $k > 0$, using a spare register t to preserve the initial value of x.

▶ **Proposition 14.** *For $x, t : \mathbb{F}_n$ and $p, q, k, i : \mathbb{N}$, we define*

$$\texttt{MOD\_CST}_\text{a}\, x\, t\, p\, q\, k\, i := [\texttt{EMPTY}_\text{a}\, x\, p\, i; \texttt{DECS\_COPY}_\text{a}\, x\, t\, i\, q\, k\, (4+i)]$$

and we check the identity $|\texttt{MOD\_CST}_\text{a}\, x\, t\, p\, q\, k\, i| = 6 + 4k$. Assuming $x \neq t$ and $k > 0$, we have $(i, \texttt{MOD\_CST}_\text{a}\, x\, t\, p\, q\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (j, \vec{v}\{0/s\}\{(\vec{v}_x + \vec{v}_t)/t\})$ where $j := p$ when k divides $\vec{v}_x$, and $j := q$ otherwise.

We now implement division by a constant $k > 0$. It will only work when the contents of the input register s is a multiple of k and the quotient is then stored in d.

▶ **Proposition 15.** *For $s, d : \mathbb{F}_n$ and $k, i : \mathbb{N}$, we define*

$$\texttt{DIV\_CST}_\text{a}\, s\, d\, k\, i := [\texttt{DECS}_\text{a}\, s\, (2+3k+i)\, (5+3k+i)\, k\, i; \texttt{INC}_\text{a}\, d; \texttt{JUMP}_\text{a}\, i\, s]$$

and we check the identity $|\texttt{DIV\_CST}_\text{a}\, s\, d\, k\, i| = 5 + 3k$. Assuming $s \neq d$, $k > 0$ and $\vec{v}_s = ak$, we have $(i, \texttt{DIV\_CST}_\text{a}\, s\, d\, k\, i) \mathrel{/\!\!/_\text{a}} (i, \vec{v}) \succ^+ (5 + 3k + i, \vec{v}\{0/s\}\{(a + \vec{v}_d)/d\})$.

3.3 Compiling regular FRACTRAN programs (file `fractran_mma.v`)

We are now in position to compile regular FRACTRAN programs (with no $p/0$ fractions). We start with a sub-program for simulating the FRACTRAN step relation for one regular fraction p/q then we will chain those sub-programs.

We fix $n := 2$, and $s := 0 : \mathbb{F}_2$ and $d := 1 : \mathbb{F}_2$ are the two available registers for two counters alternate Minsky machines. Let use assume a regular fraction, i.e. $p, q : \mathbb{N}$ with $q \neq 0$, and $i, j : \mathbb{N}$ where i the starting PC value of the sub-program.

To help the readability of the following code, we decorate it with relevant labels (PC values), although those are not formally present in the mechanisation:

$$(i, \texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j) := \begin{bmatrix} i_0\colon\ \texttt{MULT\_CST}_\text{a}\, s\, d\, p\, i_0; \\ i_1\colon\ \texttt{MOD\_CST}_\text{a}\, d\, s\, i_2\, i_5\, q\, i_1; \\ i_2\colon\ \texttt{DIV\_CST}_\text{a}\, s\, d\, q\, i_2; \\ i_3\colon\ \texttt{TRANSFER}_\text{a}\, d\, s\, i_3; \\ i_4\colon\ \texttt{JUMP}_\text{a}\, j\, d; \\ i_5\colon\ \texttt{DIV\_CST}_\text{a}\, s\, d\, p\, i_5; \\ i_6\colon\ \texttt{TRANSFER}_\text{a}\, d\, s\, i_6 \\ i_7\colon \end{bmatrix} \quad \text{where} \quad \begin{cases} i_0 := i, \\ i_1 := 5 + p + i_0, \\ i_2 := 6 + 4q + i_1, \\ i_3 := 5 + 3q + i_2, \\ i_4 := 3 + i_3, \\ i_5 := 2 + i_4, \\ i_6 := 5 + 3p + i_5, \\ i_7 := 3 + i_6 \end{cases}$$

▶ **Proposition 16.** *$|\texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j| = 29 + 4p + 7q$ and $i_7 = |\texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j| + i$.*

▶ **Proposition 17.** *If $qy = px$ then $(i, \texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j) \mathbin{/\!\!/_\text{a}} (i, [x; 0]) \succ^+ (j, [y; 0])$.*

▶ **Proposition 18.** *If $q \nmid px$ then $(i, \texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j) \mathbin{/\!\!/_\text{a}} (i, [x; 0]) \succ^+ (i_7, [x; 0])$.*

Proof. The proof of Proposition 17 (resp. 18) is sketched in Appendix A (resp. B). ◀

Hence $(i, \texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j)$ performs the multiplication of x by p/q if the result is a natural number, transferring the control to PC value j, or else, would the result be a proper fraction, the registers are globally unmodified and the PC is transferred at i_7, the end of this sub-program. Notice that the register d is assumed to be initially empty.

We now chain those sub-programs to simulate one step of a regular FRACTRAN program, encoding a list Q of fractions by structural recursion on Q:

$$\texttt{FRAC\_STEP}_\text{a}\ j\ [\,]\ i := [\,] \qquad \texttt{FRAC\_STEP}_\text{a}\ j\ (p/q :: Q)\ i := P +\!\!+ \texttt{FRAC\_STEP}_\text{a}\ j\ Q\ (|P| + i) \quad \text{where } P := \texttt{FRAC\_ONE}_\text{a}\, p\, q\, i\, j$$

▶ **Lemma 19.** *For any regular FRACTRAN program $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ and any $i, j, x, y : \mathbb{N}$, if $Q \mathbin{/\!\!/_\text{F}} x \succ y$ then $(i, \texttt{FRAC\_STEP}_\text{a}\ j\ Q\ i) \mathbin{/\!\!/_\text{a}} (i, [x; 0]) \succ^+ (j, [y; 0])$.*

Proof. By induction on the predicate $Q \mathbin{/\!\!/_\text{F}} x \succ y$ using Propositions 17 and 18. ◀

▶ **Lemma 20.** *For any regular FRACTRAN program $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ and any $i, j, x : \mathbb{N}$, if $Q \mathbin{/\!\!/_\text{F}} x \not\succ \star$ then $(i, \texttt{FRAC\_STEP}_\text{a}\ j\ Q\ i) \mathbin{/\!\!/_\text{a}} (i, [x; 0]) \succ^* (|\texttt{FRAC\_STEP}_\text{a}\ j\ Q\ i| + i, [x; 0])$.*

Proof. By induction on Q using Proposition 18. ◀

The instance $\texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1$ starts at $i = 1$ and loops on itself ($j = 1$) until no fraction can be executed. In addition, we finish by nullifying s and then jump to PC value 0:

$$\texttt{FRAC\_MMA}_\text{a}\, Q := \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1 +\!\!+ \texttt{NULL}_\text{a}\, s\, \big(|\texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1| + 1\big) +\!\!+ \texttt{JUMP}_\text{a}\, 0\, s$$

$$\frac{}{\Sigma /\!\!/_{\mathrm{n}} 0 \oplus 0 \vdash p}\ \mathtt{STOP}_{\mathrm{n}}\, p \in \Sigma$$

$$\frac{\Sigma /\!\!/_{\mathrm{n}} 1{+}a \oplus b \vdash q}{\Sigma /\!\!/_{\mathrm{n}} a \oplus b \vdash p}\ \mathtt{INC}_{\mathrm{n}}\, \alpha\, p\, q \in \Sigma \qquad \frac{\Sigma /\!\!/_{\mathrm{n}} a \oplus b \vdash q}{\Sigma /\!\!/_{\mathrm{n}} 1{+}a \oplus b \vdash p}\ \mathtt{DEC}_{\mathrm{n}}\, \alpha\, p\, q \in \Sigma \qquad \frac{\Sigma /\!\!/_{\mathrm{n}} 0 \oplus b \vdash q}{\Sigma /\!\!/_{\mathrm{n}} 0 \oplus b \vdash p}\ \mathtt{ZERO}_{\mathrm{n}}\, \alpha\, p\, q \in \Sigma$$

$$\frac{\Sigma /\!\!/_{\mathrm{n}} a \oplus 1{+}b \vdash q}{\Sigma /\!\!/_{\mathrm{n}} a \oplus b \vdash p}\ \mathtt{INC}_{\mathrm{n}}\, \beta\, p\, q \in \Sigma \qquad \frac{\Sigma /\!\!/_{\mathrm{n}} a \oplus b \vdash q}{\Sigma /\!\!/_{\mathrm{n}} a \oplus 1{+}b \vdash p}\ \mathtt{DEC}_{\mathrm{n}}\, \beta\, p\, q \in \Sigma \qquad \frac{\Sigma /\!\!/_{\mathrm{n}} a \oplus 0 \vdash q}{\Sigma /\!\!/_{\mathrm{n}} a \oplus 0 \vdash p}\ \mathtt{ZERO}_{\mathrm{n}}\, \beta\, p\, q \in \Sigma$$

Figure 1 The S-$\mathsf{MM}_{\mathrm{nd}}$ sequent style calculus for non-deterministic two counters Minsky machines.

▶ **Theorem 21.** *For any regular FRACTRAN program* $Q : \mathbb{L}\,(\mathbb{N} \times \mathbb{N})$ *and any* $x : \mathbb{N}$*, the three following properties are equivalent:*

1. $Q /\!\!/_{\mathrm{F}} x{\downarrow}$*;*
2. $(1, \mathtt{FRAC\_MMA}_{\mathrm{a}}\, Q) /\!\!/_{\mathrm{a}} (1, [x; 0]) \rightsquigarrow (0, [0; 0])$*;*
3. $(1, \mathtt{FRAC\_MMA}_{\mathrm{a}}\, Q) /\!\!/_{\mathrm{a}} (1, [x; 0]){\downarrow}$.

Proof. A sketch of the proof can be found in Appendix C. ◀

▶ **Corollary 22.** $\mathsf{FRACTRAN}_{\mathrm{reg}} \preceq \mathsf{MMA}_2$ *and* $\mathsf{FRACTRAN}_{\mathrm{reg}} \preceq \mathsf{MMA0}_2$.

4 Minsky machine termination as provability

While the (heavy) alternate Minsky machines framework was useful to simulate FRACTRAN programs with two counter machines, using it as a seed for other reductions is not recommended. First, explaining the semantics and termination predicates requires many definitions, not necessarily obvious at first. Also, manipulating them without the tools for modular reasoning is quite difficult.

4.1 Non-deterministic two counters Minsky machines (file `ndMM2.v`)

For our reductions to linear logic, we replace MMA_2 with an equivalent model, much easier to describe and work with, where computations are performed by proof-search and termination matches the provability/derivability predicate. Reductions to entailment in logical systems will thus mainly consist in encoding derivations from one system to another. We call this model non-deterministic two counters Minsky machines and denote $\mathsf{MM}_{\mathrm{nd}}$.

We comment this logical presentation, sequent style, of Minsky machines. *MM_{nd} instructions* are of the form $\mathtt{STOP}_{\mathrm{n}}\, p \mid \mathtt{INC}_{\mathrm{n}}\, x\, p\, q \mid \mathtt{DEC}_{\mathrm{n}}\, x\, p\, q \mid \mathtt{ZERO}_{\mathrm{n}}\, x\, p\, q$ where $x \in \{\alpha, \beta\}$ is a register index, either the first α or the second β,[4] and $p, q : \mathbb{N}$ are labels, here in type $\mathbb{N}$, but the definitions in this section are completely parametric in the type of labels. A *sequent of MM_{nd}* is of the form $\Sigma /\!\!/_{\mathrm{n}} a \oplus b \vdash p$ where Σ is a list of $\mathsf{MM}_{\mathrm{nd}}$ instructions viewed as a finite set, a and b of type $\mathbb{N}$ represent the values of the counters α and β respectively and p is the current label.

We define provability/derivability inductively by the rules the calculus S-$\mathsf{MM}_{\mathrm{nd}}$ in Fig. 1. Notice that since computation is simulated by proof-search, the initial state is the conclusion of a rule and it is transformed into the premise, when there is one. For example, the $\mathtt{INC}_{\mathrm{n}}\, \_\, p\, q$ rule contains both the initial label and the jump-to label, hence it can only execute at label

[4] hence formally a Boolean value of type $\mathbb{B}$.

p. However, nothing prevents the simultaneous occurrence of another instruction $\mathtt{INC}_{\mathrm{n}}\ \_\ p\ q'$ in Σ, and this could render proof-search non-deterministic, hence our choice of terminology. However, non-determinism is not relevant to the undecidability of the $\mathsf{MM}_{\mathrm{nd}}$.

Notice that it is common practice to represent the sequent and the derivability predicate of the sequent by the same denotation $\Sigma \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash p$ which could lead to confusion. Usually, we qualify the notation with the "sequent" word to make it explicit. Unqualified or followed with "is derivable" means that the notation represents the $\mathsf{S}\text{-}\mathsf{MM}_{\mathrm{nd}}$ derivability predicate.

▶ **Definition 23.** *A* $\mathsf{MM}_{\mathrm{nd}}$ *problem instance is the data of a sequent* $\Sigma \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash p$*, and the question is whether this sequent is derivable or not using the rules of* $\mathsf{S}\text{-}\mathsf{MM}_{\mathrm{nd}}$ *(Fig. 1).*

Notice that $\mathtt{ZERO}_{\mathrm{n}}\, x\, p\, q$ performs both a zero-test on x and if zero, a jump from p to q without changing registers. If we remove the $\mathtt{ZERO}_{\mathrm{n}}\ \_\ p\, q$ rules, we get Petri nets reachability, more specifically VASS with states, which have a decidable reachability problem with non-elementary complexity [4], even non-primitive recursive according to [16, 5].

4.2 From $\mathsf{MMA0}_2$ to $\mathsf{MM}_{\mathrm{nd}}$ (file `MMA2_to_ndMM2_ACCEPT.v`)

We give an alternate presentation of termination on zero for two registers Minsky machines, using the $\mathsf{S}\text{-}\mathsf{MM}_{\mathrm{nd}}$ calculus of Section 4.1. Let us consider alternate Minsky machines MMA_2 with two counters, $s := 0 : \mathbb{F}_2$ and $d := 1 : \mathbb{F}_2$. We denote by $\alpha, \beta : \mathbb{B}$ the two registers of $\mathsf{MM}_{\mathrm{nd}}$ instructions. We define the following encodings of single instructions and programs:

$$\begin{array}{lll}
\overline{(\cdot)} : \mathbb{F}_2 \to \mathbb{B} & \langle\cdot,\cdot\rangle : \mathbb{N} \to \mathsf{MMA}_2 \to \mathbb{L}\,\mathsf{MM}_{\mathrm{nd}} & \langle\!\langle\cdot,\cdot\rangle\!\rangle : \mathbb{N} \to \mathbb{L}\,\mathsf{MMA}_2 \to \mathbb{L}\,\mathsf{MM}_{\mathrm{nd}} \\
\overline{0} := \alpha & \langle i, \mathtt{INC}_{\mathrm{a}}\, x\rangle := [\mathtt{INC}_{\mathrm{n}}\, \overline{x}\, i\, (1{+}i)] & \langle\!\langle i, [\,]\rangle\!\rangle := [\,] \\
\overline{1} := \beta & \langle i, \mathtt{DEC}_{\mathrm{a}}\, x\, j\rangle := [\mathtt{DEC}_{\mathrm{n}}\, \overline{x}\, i\, j; \mathtt{ZERO}_{\mathrm{n}}\, \overline{x}\, i\, (1{+}i)] & \langle\!\langle i, \sigma :: P\rangle\!\rangle := \langle i, \sigma\rangle +\!\!+ \langle\!\langle 1{+}i, P\rangle\!\rangle
\end{array}$$

▶ **Proposition 24.** *The encodings* $\langle\cdot,\cdot\rangle$ *and* $\langle\!\langle\cdot,\cdot\rangle\!\rangle$ *are sound:*

1. *assuming the inclusion* $\langle i, \sigma\rangle \subseteq \Sigma$*, if* $\sigma \mathbin{/\!\!/_{\mathrm{a}}} (i, [a; b]) \succ (j, [a'; b'])$ *and* $\Sigma \mathbin{/\!\!/_{\mathrm{n}}} a' \oplus b' \vdash j$ *is derivable then so is* $\Sigma \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash i$*;*
2. *assuming* $\langle\!\langle 1, P\rangle\!\rangle \subseteq \Sigma$*, if* $(1, P) \mathbin{/\!\!/_{\mathrm{a}}} (i, [a; b]) \succ^1 (j, [a'; b'])$ *and* $\Sigma \mathbin{/\!\!/_{\mathrm{n}}} a' \oplus b' \vdash j$ *is derivable then so is* $\Sigma \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash i$*.*

Proof. Item 1 is by case analysis on σ and item 2 follows from item 1. ◀

Let us now define $\Sigma_P := \mathtt{STOP}_{\mathrm{n}}\, 0 :: \langle\!\langle 1, P\rangle\!\rangle$ which constitutes the encoding of MMA_2 programs into $\mathsf{MM}_{\mathrm{nd}}$ sequents. We establish its soundness.

▶ **Lemma 25.** *If* $(1, P) \mathbin{/\!\!/_{\mathrm{a}}} (i, [a, b]) \succ^* (0, [0; 0])$ *then* $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash i$ *is derivable.*

Proof. We have $\langle\!\langle 1, P\rangle\!\rangle \subseteq \Sigma_P$ by definition of Σ_P. Iterating Proposition 24 (item 2), we thus get $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} 0 \oplus 0 \vdash 0 \ \to\ \Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash i$. The derivability of $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} 0 \oplus 0 \vdash 0$ follows from $\mathtt{STOP}_{\mathrm{n}}\, 0 \in \Sigma_P$ and the $\mathtt{STOP}_{\mathrm{n}}\, 0$ rule of $\mathsf{S}\text{-}\mathsf{MM}_{\mathrm{nd}}$. ◀

▶ **Lemma 26.** *If* $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash i$ *is derivable then* $(1, P) \mathbin{/\!\!/_{\mathrm{a}}} (i, [a, b]) \succ^* (0, [0; 0])$*.*

Proof. The argument proceeds by structural induction on the derivation of $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash i$, i.e. by analysing the structure of $\mathsf{S}\text{-}\mathsf{MM}_{\mathrm{nd}}$ derivations. The following result is an essential ingredient in this case analysis: $c \in \langle\!\langle i, P\rangle\!\rangle \ \to\ \exists L\, \sigma\, R,\ P = L +\!\!+ \sigma :: R \ \wedge\ c \in \langle |L| + i, \sigma\rangle$. It allows to recover the MMA_2 instructions from which $\mathsf{MM}_{\mathrm{nd}}$ instructions originate. ◀

▶ **Corollary 27.** $\mathsf{MMA0}_2 \preceq \mathsf{MM}_{\mathrm{nd}}$.

Proof. The reduction maps an instance $(P, [a; b])$ of $\mathsf{MMA0}_2$ to the sequent $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash 1$. Lemmas 25 and 26 provide the equivalence between $(1, P) \mathbin{/\!\!/_{\mathrm{a}}} (1, [a, b]) \rightsquigarrow (0, [0; 0])$ and the derivability of $\Sigma_P \mathbin{/\!\!/_{\mathrm{n}}} a \oplus b \vdash 1$, which ensures the correctness of the reduction. ◀

5 Undecidability of Sub-Exponential Linear Logic

Having established the undecidability of $\mathsf{MM}_{\mathrm{nd}}$ via $\mathsf{FRACTRAN}_{\mathrm{reg}}$ and $\mathsf{MMA0}_2$, we can now switch to undecidability in some fragments of linear logic and give a comparison between two different reductions. We introduce the intuitionistic version of sub-exponential linear logic [2] (IMSELL) and mechanise a many-one reduction from $\mathsf{MM}_{\mathrm{nd}}$ to entailment in IMSELL. Even if the former reduction [2] applies to classical sub-exponential linear logic with one sided sequents, our own reduction function is inspired from it. However, the completeness proof that we have mechanised largely differs since we avoid focused proofs (used to recover computations) and instead, adapt the trivial phase semantics argument [13, 8]. Additionally we precisely compare the reduction to ILL with the reduction to IMSELL by starting from the same $\mathsf{MM}_{\mathrm{nd}}$ seed, detailing what set of logical rules are used to simulate those machines.

5.1 The ILL and IMSELL fragments (files `ILL.v` and `IMSELL.v`)

We introduce two fragments/extensions of intuitionistic linear logic (ILL) that allow for a reduction from non-deterministic two counters Minsky machines.

The first fragment of the ILL logic we consider is composed of propositional formulæ build from two binary connectives, the *linear implication* $\multimap$ and *additive conjunction* &, and one modality, *exponentiation* !. Logical variables come from (a copy of) the $\mathbb{N}$ type. Formally, the formulæ of ILL are of the form $A, B ::= X \mid A \multimap B \mid A \,\&\, B \mid !A$ where $X : \mathbb{N}$. To simplify, *we abusively call this fragment ILL*. By cut-elimination, the reduction discussed below also works for larger fragments containing more connectives like $\otimes$, $\oplus$, etc.

The sequents of ILL are intuitionistic, i.e. a pair (Γ, A) written $\Gamma \vdash A$ where Γ is a multiset of formulæ and A is a single formula. Multisets are just lists identified up-to permutations. If it is more convenient to work with lists, as we do in the Coq mechanization, then an *explicit permutation rule* is added to the sequent rules of the S-ILL calculus in Fig. 2.

The three leftmost rules are the identity (or axiom) rule stating that the sequent $A \vdash A$ has a trivial proof, and then the left- and right-introduction rules for the linear implication $\multimap$. The three rules middle-left are two left- and one right-introduction rules for the additive conjunction &. The two middle-right rules are modal rules, on top, the *promotion* rule, and at bottom, the *dereliction* rule. Finally, on the right-hand-side are the *structural rules* for the ! modality, i.e. *weakening* on top and *contraction* at the bottom. Notice that specifically, linear logic does not allow for general weakening or contraction rules.

On the other hand, IMSELL is a purely multiplicative fragment but with several modalities, among them exponentials. The logic is parameterized with a fixed type Λ of *modalities* and a fixed sub-type $\mathcal{U} : \Lambda \to \mathbb{P}$ of *unbounded* modalities, also called exponentials. We follow the set theoretic syntax and write $u \in \mathcal{U}$ (instead of $\mathcal{U}\, u$) when u is unbounded. The formulæ of IMSELL_Λ are of the form $A, B ::= X \mid A \multimap B \mid !^m A$ where $X : \mathbb{N}$ and $m : \Lambda$. So compared to ILL, the additive & is missing whereas the modality ! becomes indexed as $!^m$ with m spanning over Λ. IMSELL_Λ sequents have the same structure $\Gamma \vdash A$ as those of ILL except that they are composed of IMSELL_Λ formulæ instead.

Before we describe the associated sequent calculus $\mathsf{S\text{-}IMSELL}_\Lambda$, we introduce supplementary structures on modalities: a pre-order $\preccurlyeq : \Lambda \to \Lambda \to \mathbb{P}$, i.e. a *reflexive* and *transitive* binary relation, such that $\mathcal{U}$ *is upward-closed for* $\preccurlyeq$, i.e. $u \preccurlyeq m$ and $u \in \mathcal{U}$ entail $m \in \mathcal{U}$ for any $m, u : \Lambda$. In the sequel, we will somehow abuse the notation and denote Λ both for the base type and the modal structure $(\Lambda, \mathcal{U}, \preccurlyeq)$ moreover assuming the pre-order and upward-closure properties.

$$\frac{}{A \vdash A} \quad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B} \quad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B} \quad \frac{A, \Gamma \vdash C}{A \,\&\, B, \Gamma \vdash C} \quad \frac{B, \Gamma \vdash C}{A \,\&\, B, \Gamma \vdash C} \qquad \frac{!\Gamma \vdash B}{!\Gamma \vdash !B} \quad \frac{A, \Gamma \vdash B}{!A, \Gamma \vdash B} \qquad \frac{\Gamma \vdash B}{!A, \Gamma \vdash B} \quad \frac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B}$$

Figure 2 The S-ILL sequent calculus.

$$\frac{}{A \vdash A} \quad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B} \quad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C} \qquad \frac{!^\star \Gamma \vdash B}{!^\star \Gamma \vdash !^m B}\ m \preccurlyeq \star \quad \frac{A, \Gamma \vdash B}{!^m A, \Gamma \vdash B} \qquad \frac{\Gamma \vdash B}{!^u A, \Gamma \vdash B}\ u \in \mathcal{U} \quad \frac{!^u A, !^u A, \Gamma \vdash B}{!^u A, \Gamma \vdash B}\ u \in \mathcal{U}$$

Figure 3 The S-IMSELL$_\Lambda$ sequent calculus with $(\Lambda, \mathcal{U}, \preccurlyeq)$.

In the sequent rules of the S-IMSELL$_\Lambda$ calculus of Fig. 3, the three leftmost rules are common with S-ILL, there is no rule for the additive conjunction & since it does belong to the fragment, and the modal rules have changed a bit. We skip over the two middle rules for the moment and consider the rightmost structural rules of weakening and contraction which generalise the corresponding rules of S-ILL, except that their use is limited to unbounded modalities ($u \in \mathcal{U}$). Back to the two middle rules, the bottom dereliction rule applies to every modality, so a direct generalisation of the corresponding rule of S-ILL. However, the promotion rule (reproduced below on the left)

$$\frac{!^\star \Gamma \vdash B}{!^\star \Gamma \vdash !^m B}\ m \preccurlyeq \star \qquad \frac{!^{k_1} A_1, \ldots, !^{k_n} A_n \vdash B}{!^{k_1} A_1, \ldots, !^{k_n} A_n \vdash !^m B}\ m \preccurlyeq k_1, \ldots, m \preccurlyeq k_n \qquad \frac{!^m \Gamma \vdash B}{!^m \Gamma \vdash !^m B}$$

is somehow more complicated and deserves further explanations. The $\star$ notation represents a multiset $k_1, \ldots, k_n$ of modalities and $!^\star\Gamma$ represents the multiset $!^{k_1} A_1, \ldots, !^{k_n} A_n$. The constraint $m \preccurlyeq \star$ imposes that m is lower than every modality in $\{k_1, \ldots, k_n\}$. Using these more explicit notations, we reframe it as in the above displayed middle rule. Finally, the (uniform) instance where $m = k_1 = \cdots = k_n$ (the rightmost above; the constraint $m \preccurlyeq \star$ holds by reflexivity), matches the promotion rule of S-ILL.

Considered independently, all modalities behave like ILL modalities, satisfying dereliction and promotion rules, while only unbounded modalities allow for contraction and weakening. However, depending on the relation $\preccurlyeq$, the promotion rule allows for *non-trivial interactions* between modalities. Given an unbounded modality $\infty \in \mathcal{U}$ and replacing ! with $!^\infty$, one can trivially embed the multiplicative fragment of ILL and recover intuitionistic multiplicative and exponential linear logic (IMELL), of which the (un)decidability of entailment is a notoriously difficult open problem [14, 20].

5.2 Embedding in S-ILL vs. S-IMSELL (file `ndMM2_IMSELL.v`)

For the reduction from MM$_{\text{nd}}$ to IMSELL$_\Lambda$ to work out properly, we need at least three modality $\{a, b, \infty\}$ where ∞ is the only unbounded modality ($\infty \in \mathcal{U}$ and $a, b \notin \mathcal{U}$), $a \preccurlyeq \infty$, $b \preccurlyeq \infty$ and a and b are incomparable, i.e. $a \not\preccurlyeq b$ and $b \not\preccurlyeq a$. As a consequence, ∞ is also strictly above a and b. From now on, we assume that Λ satisfies these requirements. The coming discussion can also be understood in the minimal case where $\Lambda_3 = \{a, b, \infty\}$ and we denote IMSELL$_3$ for either of these logics.

In this section, we review the encoding of $\mathsf{MM}_{\mathrm{nd}}$ sequents into both ILL and IMSELL_3, and explain how, while mostly similar, *they noticeably differ on how they handle zero tests combined with jumps*. Notice that the encoding targeting ILL can be adapted to n registers Minsky machines (as done in [8]), while in the case of IMSELL_3, working with two counters only is critically important to the construction.

Identifying the exponential ! with the unbounded modality $!^\infty$ allows to discuss IMELL, ILL and IMSELL_3 in a common syntactic framework, avoiding cumbersome notations for trivial embeddings. We show the derivability of the two following rules: generalised weakening and customised absorption.

▶ **Lemma 28.** *The two following rules are derivable in IMELL, and hence ILL and IMSELL$_3$:*

$$\frac{\Delta \vdash B}{!^\infty\Sigma, \Delta \vdash B} \qquad \frac{A, !^\infty\Sigma, \Delta \vdash B}{!^\infty\Sigma, \Delta \vdash B}\ A \in \Sigma$$

Proof. We obtain the left generalised weakening rule by repeating the weakening rule. For customised absorption, it is the combination of dereliction and contraction. ◀

These derived rules are essential tools for the reduction from $\mathsf{MM}_{\mathrm{nd}}$. Let us review the other tools. Recall that a $\mathsf{MM}_{\mathrm{nd}}$ sequent is of the form $\Sigma /\!/_{\mathrm{n}} x \oplus y \vdash p$. We encode this with an IMSELL_3 (or ILL) sequent of the form $!^\infty\overline{\Sigma}, \Delta \vdash \overline{p}$ where $\Delta := x\overline{\alpha}, y\overline{\beta}$ encodes the pair $(x, y) : \mathbb{N} \times \mathbb{N}$, i.e. $\overline{\alpha}$ (resp. $\overline{\beta}$) is repeated x (resp. y) times. Hence increment and decrement operations on the values x/y naturally correspond to the multiset operations. We do not need to specify what formulæ are $\overline{\alpha}$ and $\overline{\beta}$ for the moment, but these will differ in the ILL case compared to the IMSELL_3 case. On the other hand, $\overline{p}$ or $\overline{q}$ will always be logical variables.

First, we show how to simulate the $\mathtt{INC}_{\mathrm{n}}\,\alpha\,p\,q$ rule of $\mathsf{S\text{-}MM}_{\mathrm{nd}}$:

$$\frac{\Sigma /\!/_{\mathrm{n}} 1{+}x \oplus y \vdash q}{\Sigma /\!/_{\mathrm{n}} x \oplus y \vdash p}\ \mathtt{INC}_{\mathrm{n}}\,\alpha\,p\,q \in \Sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{!^\infty\overline{\Sigma}, \overline{\alpha}, \Delta \vdash \overline{q}}{!^\infty\overline{\Sigma}, \Delta \vdash \overline{\alpha} \multimap \overline{q}} \quad \dfrac{}{\overline{p} \vdash \overline{p}}}{(\overline{\alpha} \multimap \overline{q}) \multimap \overline{p}, !^\infty\overline{\Sigma}, \Delta \vdash \overline{p}}}{!^\infty\overline{\Sigma}, \Delta \vdash \overline{p}}\ (\overline{\alpha} \multimap \overline{q}) \multimap \overline{p} \in \overline{\Sigma}$$

and the $\mathtt{DEC}_{\mathrm{n}}\,\alpha\,p\,q$ rule of $\mathsf{S\text{-}MM}_{\mathrm{nd}}$:

$$\frac{\Sigma /\!/_{\mathrm{n}} x \oplus y \vdash q}{\Sigma /\!/_{\mathrm{n}} 1{+}x \oplus y \vdash p}\ \mathtt{DEC}_{\mathrm{n}}\,\alpha\,p\,q \in \Sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{}{\overline{\alpha} \vdash \overline{\alpha}} \quad \dfrac{!^\infty\overline{\Sigma}, \Delta \vdash \overline{q} \quad \dfrac{}{\overline{p} \vdash \overline{p}}}{\overline{q} \multimap \overline{p}, !^\infty\overline{\Sigma}, \Delta \vdash \overline{p}}}{\overline{\alpha} \multimap (\overline{q} \multimap \overline{p}), !^\infty\overline{\Sigma}, \overline{\alpha}, \Delta \vdash \overline{p}}}{!^\infty\overline{\Sigma}, \overline{\alpha}, \Delta \vdash \overline{p}}\ \overline{\alpha} \multimap (\overline{q} \multimap \overline{p}) \in \overline{\Sigma}$$

Notice that we only use the customised absorption rule, the left- and right-introduction rules for $\multimap$ and the identity (axiom) rule hence simulating $\mathtt{INC}_{\mathrm{n}}\,\alpha\,p\,q$ and $\mathtt{DEC}_{\mathrm{n}}\,\alpha\,p\,q$ can be performed within the IMELL fragment.

The axiom rule $\mathtt{STOP}_{\mathrm{n}}\,p$ of $\mathsf{S\text{-}MM}_{\mathrm{nd}}$ (acceptance of $(0, 0)$ at p) can also be simulated

$$\frac{}{\Sigma /\!/_{\mathrm{n}} 0 \oplus 0 \vdash p}\ \mathtt{STOP}_{\mathrm{n}}\,p \in \Sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\overline{p} \vdash \overline{p}}}{\vdash \overline{p} \multimap \overline{p}} \quad \dfrac{}{\overline{p} \vdash \overline{p}}}{(\overline{p} \multimap \overline{p}) \multimap \overline{p} \vdash \overline{p}}}{(\overline{p} \multimap \overline{p}) \multimap \overline{p}, !^\infty\overline{\Sigma} \vdash \overline{p}}}{!^\infty\overline{\Sigma}, \emptyset \vdash \overline{p}}\ (\overline{p} \multimap \overline{p}) \multimap \overline{p} \in \overline{\Sigma}$$

using the customised absorption rule, then the generalised weakening rule, the left- and right-introduction rules for $\multimap$ and the identity rule. Hence an IMELL proof as well.

The remaining rules of $\mathsf{MM}_{\mathrm{nd}}$, that of e.g. $\mathtt{ZERO}_{\mathrm{n}}\,\alpha\,p\,q$, a zero test *combined with* a jump instruction, are the problematic rules to encode in the IMELL fragment. This can however be done in ILL and in IMSELL_3, but the techniques for the two fragments diverge precisely on these $\mathtt{ZERO}_{\mathrm{n}}\,\alpha\,p\,q$ instructions.

Let us first review[5] the (idea behind the) legacy encoding of Minsky machines into linear logic [17], mechanized for ILL in [8]. The idea is to *fork* the $\mathtt{ZERO}_{\mathrm{n}}\,\alpha\,p\,q$ simulation into a proof-search branch where only a zero test on α is performed, and in the other branch, only a jump to q is performed:

$$\frac{\Sigma \mathbin{/\!\!/_{\mathrm{n}}} 0 \oplus y \vdash q}{\Sigma \mathbin{/\!\!/_{\mathrm{n}}} 0 \oplus y \vdash p}\ \mathtt{ZERO}_{\mathrm{n}}\,\alpha\,p\,q \in \Sigma \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dots}{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \underline{\alpha}} \quad !^\infty\overline{\Sigma}, y\overline{\beta} \vdash \overline{q}}{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \underline{\alpha} \,\&\, \overline{q}} \quad \dfrac{}{\overline{p} \vdash \overline{p}}}{(\underline{\alpha} \,\&\, \overline{q}) \multimap \overline{p}, !^\infty\overline{\Sigma}, y\overline{\beta} \vdash \overline{p}}}{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \overline{p}}\ (\underline{\alpha} \,\&\, \overline{q}) \multimap \overline{p} \in \overline{\Sigma}$$

Notice that $\underline{\alpha}$ and $\underline{\beta}$ denote fresh logical variables. Critically for this encoding, the additive conjunction & is used to copy the context into the two premises, implementing a fork. The zero test on left sub-branch can however by performed in IMELL only:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{!^\infty\overline{\Sigma}, \emptyset \vdash \underline{\alpha}}{\dots}}{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \underline{\alpha}} \quad \dfrac{}{\underline{\alpha} \vdash \underline{\alpha}}}{\underline{\alpha} \multimap \underline{\alpha}, !^\infty\overline{\Sigma}, y\overline{\beta} \vdash \underline{\alpha}} \quad \dfrac{}{\overline{\beta} \vdash \overline{\beta}}}{\overline{\beta} \multimap (\underline{\alpha} \multimap \underline{\alpha}), !^\infty\overline{\Sigma}, \overline{\beta}, y\overline{\beta} \vdash \underline{\alpha}}}{!^\infty\overline{\Sigma}, (1{+}y)\overline{\beta} \vdash \underline{\alpha}}\ \overline{\beta} \multimap (\underline{\alpha} \multimap \underline{\alpha}) \in \overline{\Sigma} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\underline{\alpha} \vdash \underline{\alpha}}}{\vdash \underline{\alpha} \multimap \underline{\alpha}} \quad \dfrac{}{\underline{\alpha} \vdash \underline{\alpha}}}{(\underline{\alpha} \multimap \underline{\alpha}) \multimap \underline{\alpha}, \emptyset \vdash \underline{\alpha}}}{(\underline{\alpha} \multimap \underline{\alpha}) \multimap \underline{\alpha}, !^\infty\overline{\Sigma}, \emptyset \vdash \underline{\alpha}}}{!^\infty\overline{\Sigma}, \emptyset \vdash \underline{\alpha}}}{\dots}\ (\underline{\alpha} \multimap \underline{\alpha}) \multimap \underline{\alpha} \in \overline{\Sigma}$$

Notice that the dots above $!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \underline{\alpha}$ mean repetition of the lower part of the proof until exhaustion of all the $\overline{\beta}$ from the context: this is implemented by an induction on y, and the base case where $y = 0$ corresponds to the upper part of the proof, starting at $!^\infty\overline{\Sigma}, \emptyset \vdash \underline{\alpha}$ and completed on the right hand side, simulating of a would be $\mathtt{STOP}_{\mathrm{n}}\,\underline{\alpha}$ instruction (see above).

We see that $\underline{\alpha}$ together with the formulæ $\overline{\beta} \multimap (\underline{\alpha} \multimap \underline{\alpha})$ and $(\underline{\alpha} \multimap \underline{\alpha}) \multimap \underline{\alpha}$ in $\overline{\Sigma}$ allow $\underline{\alpha}$ to perform the elimination of all the $\overline{\beta}$ from the context. However, $\underline{\alpha}$ will not allow the removal of any $\overline{\alpha}$ and hence, the zero test branch cannot be completed if Δ contains an occurrence of $\overline{\alpha}$, i.e. when $x \neq 0$. This encoding of the zero test using $\underline{\alpha}$, while it can already be performed in IMELL, is pertinent only for ILL because it is in combination with the fork in $(\underline{\alpha} \,\&\, \overline{q}) \multimap \overline{p}$ (see above) that it provides the ability to *conditionally jump on zero*.

Contrary to the ILL encoding, IMSELL_3 does not require (and cannot use) forking but instead uses sub-modalities to prevent jumping when the zero test fails. In that case, $\overline{\alpha}$ and $\overline{\beta}$ are not atomic formulæ anymore: they contain the bounded modalities $!^a$ and $!^b$, and we define $\overline{\alpha} := !^a\alpha_0$ and $\overline{\beta} := !^b\beta_0$ where α_0, β_0 are fresh variables. In the following encoding,

[5] here we only discuss the ILL case, i.e. we do not replicate the former ILL mechanisation [8] in the code.

$$\frac{\Sigma \mathbin{/\!\!/_{\rm n}} 0 \oplus y \vdash q}{\Sigma \mathbin{/\!\!/_{\rm n}} 0 \oplus y \vdash p}\ \mathtt{ZERO}_{\rm n}\,\alpha\,p\,q \in \Sigma \qquad \rightsquigarrow \qquad \frac{\dfrac{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \overline{q}}{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash !^b\overline{q}} \quad \dfrac{}{\overline{p} \vdash \overline{p}}}{\dfrac{!^b\overline{q} \multimap \overline{p}, !^\infty\overline{\Sigma}, y\overline{\beta} \vdash \overline{p}}{!^\infty\overline{\Sigma}, y\overline{\beta} \vdash \overline{p}}\ !^b\overline{q} \multimap \overline{p} \in \overline{\Sigma}}$$

notice that the upper rule is an instance of the promotion rule of S-IMSELL$_3$. It is allowed because every formula on the left is prefixed either with the unbounded modality $!^\infty$ for those in $!^\infty\overline{\Sigma}$, or with the modality $!^b$ for those in $y\overline{\beta} = !^b\beta_0, \ldots, !^b\beta_0$, and we have both $b \preccurlyeq \infty$ and $b \preccurlyeq b$. On the other hand, an occurrence of $\overline{\alpha} = !^a\alpha_0$ in the context, corresponding to a non-zero value of x, would prevent the application of the promotion rule ($b \not\preccurlyeq a$). This interaction of modalities in the promotion rule of IMSELL$_3$ is the key to simulate zero tests.

▶ **Definition 29.** *Let us define* $\alpha_0 := 0$, $\beta_0 := 1$, $\overline{p} := 2 + p$, $\overline{\alpha} := !^a\alpha_0$ *and* $\overline{\beta} := !^b\beta_0$. *We encode* MM$_{\rm nd}$ *instructions as:*

$$\overline{\mathtt{STOP}_{\rm n}\,p} := (\overline{p} \multimap \overline{p}) \multimap \overline{p}$$

$$\begin{array}{lll} \overline{\mathtt{INC}_{\rm n}\,\alpha\,p\,q} := (\overline{\alpha} \multimap \overline{q}) \multimap \overline{p} & \overline{\mathtt{DEC}_{\rm n}\,\alpha\,p\,q} := \overline{\alpha} \multimap (\overline{q} \multimap \overline{p}) & \overline{\mathtt{ZERO}_{\rm n}\,\alpha\,p\,q} := !^b\overline{q} \multimap \overline{p} \\ \overline{\mathtt{INC}_{\rm n}\,\beta\,p\,q} := (\overline{\beta} \multimap \overline{q}) \multimap \overline{p} & \overline{\mathtt{DEC}_{\rm n}\,\beta\,p\,q} := \overline{\beta} \multimap (\overline{q} \multimap \overline{p}) & \overline{\mathtt{ZERO}_{\rm n}\,\beta\,p\,q} := !^a\overline{q} \multimap \overline{p} \end{array}$$

and then map $\overline{(\cdot)}$ *on the list* Σ *extensionally, i.e.* $\overline{[\sigma_1; \ldots; \sigma_n]} := \overline{\sigma_1}, \ldots, \overline{\sigma_n}$.

▶ **Lemma 30.** *If* $\Sigma \mathbin{/\!\!/_{\rm n}} x \oplus y \vdash p$ *can be derived in* S-MM$_{\rm nd}$ *then the sequent* $!^\infty\overline{\Sigma}, x\overline{\alpha}, y\overline{\beta} \vdash \overline{p}$ *is provable in* S-IMSELL$_3$.

Proof. The argument proceeds by induction on the derivation of $\Sigma \mathbin{/\!\!/_{\rm n}} x \oplus y \vdash p$, combining the proof skeletons of the above discussion in a direct way. ◀

5.3 Trivial Phase semantics for IMSELL (file `imsell.v`)

We define trivial phase semantics for IMSELL$_\Lambda$ and show soundness w.r.t. the S-IMSELL$_\Lambda$ calculus. We start with a commutative monoid $(M, \bullet, \epsilon)$. Typically, for the completeness of our reduction, we will only need to use the semantics for $M = (\mathbb{N}^2, +, \vec{0})$, i.e. vectors of natural numbers of length 2, but the semantics works for any commutative monoid. For any $X, Y \subseteq M$, we define the *point-wise extension* by $X \bullet Y := \{x \bullet y \mid x \in X \wedge y \in Y\}$ and its *linear adjunct* as $X \multimap Y := \{k \in M \mid \{k\} \bullet X \subseteq Y\}$ providing a residuated monoidal structure on the subset type $M \to \mathbb{P}$.

To interpret the modal structure $(\Lambda, \mathcal{U}, \preccurlyeq)$, we further require for each modality $m \in \Lambda$, a subset $K_m \subseteq M$ i.e. a predicate $K_m : M \to \mathbb{P}$. We assume that the map $m \mapsto K_m$ is monotonically decreasing w.r.t. $\preccurlyeq$ (on the left below) and satisfies the three extra following rightmost axioms:

$$\forall m\,k,\ m \preccurlyeq k \to K_k \subseteq K_m \qquad \forall m,\ \epsilon \in K_m \qquad \forall m,\ K_m \bullet K_m \subseteq K_m \qquad \forall u \in \mathcal{U},\ K_u \subseteq \{\epsilon\}$$

Given any semantic interpretation $[\![\cdot]\!] \subseteq M$ of logical variables, we extend it inductively to IMSELL$_\Lambda$ sequents via *trivial phase semantics*:[6]

$$[\![A \multimap B]\!] := [\![A]\!] \multimap [\![B]\!] \qquad [\![!^m A]\!] := [\![A]\!] \cap K_m \qquad [\![A_1, \ldots, A_n]\!] := [\![A_1]\!] \bullet \cdots \bullet [\![A_n]\!]$$

[6] The *trivial* qualifier refers to the use of the identity closure $\mathrm{cl}(X) = X$ in the interpretation of modalities, i.e. $[\![!^m A]\!] := [\![A]\!] \cap K_m$ instead of the more general $[\![!^m A]\!] := \mathrm{cl}([\![A]\!] \cap K_m)$ where $\mathrm{cl}(\cdot) : (M \to \mathbb{P}) \to (M \to \mathbb{P})$ is a stable closure operator. This also applies to the (implicit) multiplicative conjunction where $[\![A_1, \ldots, A_n]\!] := [\![A_1]\!] \bullet \cdots \bullet [\![A_n]\!]$ instead of $[\![A_1, \ldots, A_n]\!] := \mathrm{cl}([\![A_1]\!] \bullet \cdots \bullet [\![A_n]\!])$. Notice that trivial phase semantics is sound but *not complete* for IMELL, ILL and IMSELL$_\Lambda$; see [13] for details.

Notice that because we work with commutative monoids, the above semantic interpretation of lists is invariant under permutations, hence is suitable for multisets. An IMSELL_Λ sequent $\Gamma \vdash A$ is *valid in that interpretation* if $[\![\Gamma]\!] \subseteq [\![A]\!]$, or (equivalently) if $\epsilon \in [\![\Gamma]\!] \multimap\!\bullet\, [\![A]\!]$.

▶ **Theorem 31.** *Trivial phase semantics is sound: any sequent $\Gamma \vdash A$ provable in S-IMSELL$_\Lambda$ must satisfy $\epsilon \in [\![\Gamma]\!] \multimap\!\bullet\, [\![A]\!]$ for any possible trivial phase semantics interpretation.*

Proof. We proceed by structural induction on the $\mathsf{S\text{-}IMSELL}_\Lambda$ derivation of $\Gamma \vdash A$. In the code, the proof is limited to the case where $M = (\mathbb{N}^n, +, \vec{0})$ for some $n : \mathbb{N}$. Compared to the soundness of trivial phase semantics for S-ILL [8], the only interesting new case is that of the promotion rule. In that case, we observe that $m \preccurlyeq k_1, \ldots, k_n$ implies $K_{k_1} \bullet \cdots \bullet K_{k_n} \subseteq K_m$. ◀

5.4 The completeness of the reduction (file `ndMM2_IMSELL.v`)

▶ **Lemma 32.** *If the sequent $!^\infty\overline{\Sigma}, x\overline{\alpha}, y\overline{\beta} \vdash \overline{p}$ is provable in S-IMSELL$_3$, then there is a derivation of $\Sigma /\!\!/_{\mathrm{n}} x \oplus y \vdash p$ in S-MM$_{\mathrm{nd}}$.*

Proof. We use a soundness argument for trivial phase semantics in place of reasoning by induction on focused derivation in MSELL as done in [2]. We consider the monoid of vectors $M = (\mathbb{N}^2, +, [0;0])$ of length 2 of natural numbers. We define the following interpretation for modalities, $K_m\,[x;y] := (a \preccurlyeq m \to y = 0) \wedge (b \preccurlyeq m \to x = 0) \wedge (m \in \mathcal{U} \to x = 0 \wedge y = 0)$, and as a consequence, we can check that K_m satisfies the required axioms as well as $K_a = \{[x;0] \mid x \in \mathbb{N}\}$, $K_b = \{[0;y] \mid y \in \mathbb{N}\}$, and $K_\infty = \{[0;0]\}$. We interpret logical variables as:

$$[\![\alpha_0]\!] := \{[1;0]\} \quad \text{and} \quad [\![\beta_0]\!] := \{[0;1]\} \quad \text{and} \quad [\![\overline{p}]\!] = \{[x;y] \mid \Sigma /\!\!/_{\mathrm{n}} x \oplus y \vdash p\}$$

and thus we have $[\![\overline{\alpha}]\!] = [\![!^a \alpha_0]\!] = [\![\alpha_0]\!] \cap K_a = \{[1;0]\}$ and $[\![\overline{\beta}]\!] = \{[0;1]\}$. Consequently, we get $[\![x\overline{\alpha}, y\overline{\beta}]\!] = \{[x;y]\}$.

We verify that the interpretation of the IMSELL_3 encoding $\overline{\sigma}$ of $\mathsf{MM}_{\mathrm{nd}}$ instructions in Σ contains the zero vector, i.e. $\forall \sigma,\, \sigma \in \Sigma \to [0;0] \in [\![\overline{\sigma}]\!]$. For instance, let us consider the case $\sigma = \mathtt{ZERO}_{\mathrm{n}}\, \alpha\, p\, q$. Then $\overline{\sigma} = !^b \overline{q} \multimap \overline{p}$ is interpreted as $([\![\overline{q}]\!] \cap K_b) \multimap\!\bullet\, [\![\overline{p}]\!]$. Hence $[0;0] \in [\![\overline{\sigma}]\!] \leftrightarrow [\![\overline{q}]\!] \cap K_b \subseteq [\![\overline{p}]\!]$, i.e. for any $[x;y] : \mathbb{N}^2$, if $\Sigma /\!\!/_{\mathrm{n}} x \oplus y \vdash q$ and $x = 0$ then $\Sigma /\!\!/_{\mathrm{n}} x \oplus y \vdash p$ which is precisely the instance of rule $\mathtt{ZERO}_{\mathrm{n}}\, \alpha\, p\, q \in \Sigma$ of $\mathsf{S\text{-}MM}_{\mathrm{nd}}$.

From the previous observation, we deduce $[0;0] \in [\![!^\infty \Sigma]\!]$. Now let us consider a sequent $!^\infty\overline{\Sigma}, x\overline{\alpha}, y\overline{\beta} \vdash \overline{p}$ which is provable in $\mathsf{S\text{-}IMSELL}_3$. By the soundness Theorem 31, we know that $[0;0] \in [\![!^\infty \Sigma, x\overline{\alpha}, y\overline{\beta}]\!] \multimap\!\bullet\, [\![\overline{p}]\!]$. Since $[x;y] = [0;0] + [x;y] \in [\![!^\infty \Sigma]\!] \bullet [\![x\overline{\alpha}, y\overline{\beta}]\!] = [\![!^\infty \Sigma, x\overline{\alpha}, y\overline{\beta}]\!]$, by the definition of $\multimap\!\bullet$ we deduce $[x;y] = [0;0] + [x;y] \in [\![\overline{p}]\!]$, and hence we conclude that $\Sigma /\!\!/_{\mathrm{n}} x \oplus y \vdash p$ holds. ◀

▶ **Theorem 33.** *Let $(\Lambda, \mathcal{U}, \preccurlyeq)$ contain three modalities a, b and ∞ such that $\infty \in \mathcal{U}$, $a, b \notin \mathcal{U}$, $a, b \preccurlyeq \infty$, $a \not\preccurlyeq b$ and $b \not\preccurlyeq a$. Then we have a reduction $\mathsf{MM}_{\mathrm{nd}} \preceq \mathsf{IMSELL}_\Lambda$, hence derivability in the S-IMSELL$_\Lambda$ calculus is undecidable.*

6 Related works and Implementation remarks

While Theorem 33 gives us a mechanised synthetic proof of the undecidability of IMSELL_3, neither its statement nor the arguments deployed directly provide hints towards a solution to the question of the decidability IMELL/MELL. As in the original pen and paper proof [2], the two bounded modalities $!^a$ and $!^b$, and their interaction with the unbounded modality $!^\infty$ in the promotion rule, play an essential role in the simulation of conditional jumps of

two counters Minsky machines. While the zero test can be implemented in MELL only, the provided implementation consumes its context and thus cannot conditionally branch at the same time, hence the fork used in the case of ILL [8].

However Theorem 33 does give indications that certain decidability arguments for MELL are bound to fail, e.g. those that would also apply to MSELL in general, or IMSELL_3 in particular. It is our understanding that the refutation [20] of the faulty proof attempt for the decidability of MELL [1] partly proceeds in showing how the claimed "proof" technique would easily generalise to MSELL. In the same vein, the lower bounds on the complexity of a would be decision procedure for MELL [14], and more recently the reachability problem for Petri nets themselves [4], indicate that a decision procedure for MELL must be of non-elementary complexity. The most recent investigations [16, 5] might very well confirm that this problem is Ackermann complete and hence not primitive recursive.

Considering formalisation issues, the growing Coq library of undecidability proofs [9] was of course of great help to this work. Indeed, at the time we decided to try to implement the undecidability of MSELL, the framework for certified programming with Minsky machines was already part of the library [8]. Hence, to get two counters Minsky machines, i.e. the seed of undecidability of the pen and paper proof [2], only a modest step from many counters machines was necessary and this was even alleviated by the results on the FRACTRAN language [12], factoring out the Gödel coding phase. In fact, we contributed the seed of two counters machines much ahead of the MSELL result, and in the meantime, this seed was used to establish to *undecidability uniform boundedness* for simple stack machines and then of the problem of *semi-unification* [6]. This illustrates a critical aspect of this undecidability framework: its extensive range of seed problems for plugging into it.

Indeed, there is an important issue to consider when proving undecidability by many-one reduction, by far the most used method in the field: even mechanised, your proof is only as strong as *the implementation* of your seed problem. Typical problems can exhibit subtleties that show up at the mechanisation level: for instance Turing machines are built on tapes, a potentially infinite structure of which it could be easy to corrupt the implementation. Choosing a seed already linked to the many-one equivalence class containing easy to describe problems such as e.g. the Post correspondence problem or FRACTRAN gives much more confidence that starting from an isolated seed, still to be mechanically checked undecidable.

Another aspect which is mostly overlooked in pen and paper proofs is the computability of the reduction function. The reason is that programming with low-level Turing complete models of computation is hard and painful, with encodings at every corner. To get a glimpse of the difficulty, think of a Turing machine working with logical formulas: because it only manipulates text written on tapes, it has to implement a syntax analyser, moreover proved correct. And only then can it start its real work. The general shortcut used in pen and paper proofs to avoid this kind of description is to speak about "algorithms" that manipulate high-level data-structures and rely on an informal and consensual understanding of what these are, hand-waving away the implementation issues completely.

In this regard, the synthetic computability framework allows, at the price of relying on the computability of Coq functions – e.g. by avoiding axioms, – to formally describe the reduction functions in a language strict enough to ensures their computability, but at the same time powerful enough to largely avoid complex encodings and hence get more natural correctness proofs following or inspired from pen and paper ones. Using a constructive framework like e.g. Coq or Agda is essential in that approach, because in classical frameworks, there is no direct way to automatically ensure the general computability of the defined (reduction) functions.

References

1. Katalin Bimbó. The decidability of the intensional fragment of classical linear logic. *Theoret. Comput. Sci.*, 597:1–17, 2015. doi:10.1016/j.tcs.2015.06.019.
2. Kaustuv Chaudhuri. Expressing additives using multiplicatives and subexponentials. *Math. Structures Comput. Sci.*, 28(5):651–666, 2018. doi:10.1017/S0960129516000293.
3. John H. Conway. *FRACTRAN: A Simple Universal Programming Language for Arithmetic*, pages 4–26. Springer New York, New York, NY, 1987.
4. Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 24–33, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316369.
5. Wojciech Czerwiński and Łukasz Orlikowski. Reachability in Vector Addition Systems is Ackermann-complete, 2021. arXiv:2104.13866.
6. Andrej Dudenhefner. Undecidability of Semi-Unification on a Napkin. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2020.9.
7. Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 38–51. ACM, 2019. doi:10.1145/3293880.3294091.
8. Yannick Forster and Dominique Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and Minsky machines. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 104–117. ACM, 2019. doi:10.1145/3293880.3294096.
9. Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq Library of Undecidable Problems. In *CoqPL 2020*, New Orleans, LA, United States, 2020. URL: https://github.com/uds-psl/coq-library-undecidability.
10. Max Kanovich. Linear Logic as a Logic of Computations. *Ann. Pure Appl. Logic*, 67(1–3):183–212, 1994.
11. Max Kanovich. The direct simulation of Minsky machines in linear logic. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, chapter 2, pages 123–145. Cambridge University Press, 1995.
12. Dominique Larchey-Wendling and Yannick Forster. Hilbert's Tenth Problem in Coq. In *4th International Conference on Formal Structures for Computation and Deduction*, volume 131 of *LIPIcs*, pages 27:1–27:20, February 2019.
13. Dominique Larchey-Wendling and Didier Galmiche. Nondeterministic Phase Semantics and the Undecidability of Boolean BI. *ACM Trans. Comput. Log.*, 14(1):6:1–6:41, 2013. doi:10.1145/2422085.2422091.
14. Ranko Lazić and Sylvain Schmitz. Non-Elementary Complexities for Branching VASS, MELL, and Extensions. *ACM Transactions on Computational Logic*, 16(3):20:1–20:30, 2015. doi:10.1145/2733375.
15. Jérôme Leroux and Sylvain Schmitz. Demystifying Reachability in Vector Addition Systems. In *LICS 2015: Proceedings of the 30th ACM/IEEE Symposium on Logic in Computer Science*, pages 56–67. IEEE, 2015. doi:10.1109/LICS.2015.16.
16. Jérôme Leroux. The Reachability Problem for Petri Nets is Not Primitive Recursive, 2021. arXiv:2104.12695.

17 Patrick Lincoln, John C. Mitchell, Andre Scedrov, and Natarajan Shankar. Decision problems for propositional linear logic. In *31st Annual Symposium on Foundations of Computer Science*, volume 2, pages 662–671. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89588.

18 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, page 238–246, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800076.802477.

19 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

20 Lutz Straßburger. On the decision problem for MELL. *Theoret. Comput. Sci.*, 768:91–98, 2019.

A Proof (sketch) of Proposition 17

Let us use the denotation $\overline{s}$ (resp. $\overline{d}$) for the dynamic value of register s (resp. d). Hence, the contents of the registers is represented by the vector $[\overline{s};\overline{d}]$ of length 2 with initial value $[x;0]$. Also, the initial value of the PC is $i_0 = i$.

The sub-program $\texttt{MULT\_CST}_{\text{a}}\, s\, d\, p\, i_0$ multiplies $\overline{s}$ with p and adds the result to the contents of d while emptying s, so the PC moves to i_1 and $\overline{s} = 0$ and $\overline{d} = px$. Then $\texttt{MOD\_CST}_{\text{a}}\, d\, s\, i_2\, i_4\, q\, i_1$ tests the divisibility of $\overline{d}$ by q, which succeeds under the assumption $qy = px$. By Proposition 15, this transfers the control to i_2 and now $\overline{d} = 0$ and $\overline{s} = px$. Then $\texttt{DIV\_CST}_{\text{a}}\, s\, d\, q\, i_2$ divides $\overline{s}$ with q while swapping the registers hence now $\overline{s} = 0$, $\overline{d} = y$ and the PC is at i_3. Then $\texttt{TRANSFER}_{\text{a}}\, d\, s\, i_3$ swaps s with d hence now $\overline{s} = y$ and $\overline{d} = 0$ and PC is now i_4. Finally, $\texttt{JUMP}_{\text{a}}\, j\, d$ transfers the control to j without altering the registers.

B Proof (sketch) of Proposition 18

As in the proof of Proposition 17, we reach the state where the PC is at i_1 and $\overline{s} = 0$ and $\overline{d} = px$. However now, $\texttt{MOD\_CST}_{\text{a}}\, d\, s\, i_2\, i_4\, q\, i_1$ gives a negative answer to the divisibility of d by s hence according to Proposition 14, the control is transferred to i_5 while $\overline{s} = px$ and $\overline{d} = 0$. Then $\texttt{DIV\_CST}_{\text{a}}\, s\, d\, p\, i_5$ divides the contents of s by p, reverting it to its initial value but there is a swap: PC is at i_6, $\overline{s} = 0$ and $\overline{d} = x$. Finally $\texttt{TRANSFER}_{\text{a}}\, d\, s\, i_6$ swaps again and reverts the registers to their initial values $\overline{s} = x$ and $\overline{d} = 0$ while the PC moves to the end of the sub-program at i_7.

C Proof (sketch) of Theorem 21

The implication $2 \implies 3$ is trivial. We show $1 \implies 2$ and $3 \implies 1$.

Let us start with $1 \implies 2$. As an instance of Lemma 19, if $Q \mathbin{/\!\!/_{\text{F}}} x \succ y$ holds then we have $(1, \texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1) \mathbin{/\!\!/_{\text{a}}} (1,[x;0]) \succ^+ (1,[y;0])$. By transitivity, from $Q \mathbin{/\!\!/_{\text{F}}} x \succ^* y$ we can deduce $(1, \texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1) \mathbin{/\!\!/_{\text{a}}} (1,[x;0]) \succ^* (1,[y;0])$.

Now assuming $Q \mathbin{/\!\!/_{\text{F}}} x{\downarrow}$, we get some y such that $Q \mathbin{/\!\!/_{\text{F}}} x \succ^* y$ and $Q \mathbin{/\!\!/_{\text{F}}} y \not\succ \star$. Hence we have $(1, \texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1) \mathbin{/\!\!/_{\text{a}}} (1,[x;0]) \succ^* (1,[y;0])$. By Lemma 20, as $Q \mathbin{/\!\!/_{\text{F}}} y \not\succ \star$, we get $(1, \texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1) \mathbin{/\!\!/_{\text{a}}} (1,[y;0]) \succ^* (|\texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1| + 1,[y;0])$. We deduce

$$(1, \texttt{FRAC\_MMA}_{\text{a}}\, Q) \mathbin{/\!\!/_{\text{a}}} (1,[x;0]) \succ^* (|\texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1| + 1,[y;0])$$

since $(1, \texttt{FRAC\_STEP}_{\text{a}}\ 1\ Q\ 1)$ is a sub-program of $(1, \texttt{FRAC\_MMA}_{\text{a}}\, Q)$. The nullifying code and the jump finish the computation and we get our proof that $(1, \texttt{FRAC\_MMA}_{\text{a}}\, Q) \mathbin{/\!\!/_{\text{a}}} (1,[x;0]) \rightsquigarrow (0,[0;0])$ holds.

Let us now finish with $3 \implies 1$ and assume $(1, \texttt{FRAC\_MMA}_\text{a}\, Q) \mathbin{/\!\!/_\text{a}} (1, [x; 0])\downarrow$. We show that $Q \mathbin{/\!\!/_\text{F}} x\downarrow$. Because $(1, \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1)$ is a sub-program of $(1, \texttt{FRAC\_MMA}_\text{a}\, Q)$, we also have $(1, \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1) \mathbin{/\!\!/_\text{a}} (1, [x; 0])\downarrow$. Hence there is $k, j : \mathbb{N}$ and $\vec{v} : \mathbb{N}^2$ such that $(1, \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1) \mathbin{/\!\!/_\text{a}} (1, [x; 0]) \succ^k (j, \vec{v})$ and $\texttt{out}\ j\ (1, \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1)$. We prove $Q \mathbin{/\!\!/_\text{F}} x\downarrow$ by strong induction on k. By Proposition 1, one can decide between two possibilities:

- either $Q \mathbin{/\!\!/_\text{F}} x \not\succ \star$ in which case $Q \mathbin{/\!\!/_\text{F}} x\downarrow$ is obvious;
- or there is y such that $Q \mathbin{/\!\!/_\text{F}} x \succ y$. We deduce $(1, \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1) \mathbin{/\!\!/_\text{a}} (1, [x; 0]) \succ^\delta (1, [y; 0])$ for some $\delta > 0$ by Lemma 19. Since the step relation is deterministic for Minsky machines, we have $(1, \texttt{FRAC\_STEP}_\text{a}\ 1\ Q\ 1) \mathbin{/\!\!/_\text{a}} (1, [y; 0]) \succ^{k-\delta} (j, \vec{v})$ hence we can apply the induction hypothesis ($k - \delta < k$) and we get $Q \mathbin{/\!\!/_\text{F}} y\downarrow$. Combining with $Q \mathbin{/\!\!/_\text{F}} x \succ y$, we conclude $Q \mathbin{/\!\!/_\text{F}} x\downarrow$.