Decision Problems for Pushdown Threads

J.A. Bergstra
I. Bethke
A. Ponse

Report PRG0502 June 2005

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

I. Bethke

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7583
e-mail: inge@science.uva.nl

A. Ponse

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7592
e-mail: alban@science.uva.nl

Programming Research Group Electronic Report Series

Decision Problems for Pushdown Threads

Jan A. Bergstra[1,2], Inge Bethke[1], and Alban Ponse[1]

[1] University of Amsterdam, Faculty of Science, Programming Research Group*
[2] Utrecht University, Department of Philosophy, Applied Logic Group

Abstract. Threads as contained in a thread algebra emerge from the behavioral abstraction from programs in an appropriate program algebra. Threads may make use of services such as stacks, and a thread using a single stack is called a pushdown thread. Equivalence of pushdown threads is shown decidable whereas pushdown thread inclusion is undecidable. This is again an example of a borderline crossing where the equivalence problem is decidable, whereas the inclusion problem is not.

1 Introduction

A challenging question in language theory is to decide whether the languages accepted by two different machines in some given class are the same. This question is called the *equivalence problem.* Another important question, known as the *inclusion problem*, is that of determining whether one language is a subset of another. The most obvious connection between these two problems is that the latter implies the former, that is, that any algorithm that decides inclusion for some family of languages can also be used to decide equivalence. The question then arises whether the converse holds, i.e., whether there are natural examples of language families with a decidable equivalence problem and an undecidable inclusion problem.

In 1973, Bird [8] found that the languages accepted by two-tape Rabin and Scott machines possess a decidable equivalence problem and an undecidable inclusion problem. Valiant explored this question further, finding two other families exhibiting this feature: the languages accepted by deterministic finite-turn pushdown automata [23] and deterministic one-counter pushdown automata [24]. In 1976, Friedman [10] investigated another subclass of deterministic pushdown automata—simple machines that have only one state and operate in real-time—and showed that these languages indeed have an undecidable inclusion problem. More recently, e.g. erasing and nonerasing pattern languages [14, 18] and deterministic context-free languages [9] have been added to this growing list.

In this paper we investigate yet another class: *pushdown threads*, a form of processes describing sequential program behaviour and using the services offered by a single stack. In this approach, threads as contained in a thread algebra emerge from the behavioral abstraction of sequential programs. A *basic* thread

* `www.science.uva.nl/~`{janb, inge, alban}. We thank the organizers of the *Logic Colloquium 2005* in Athens for providing the opportunity to present this paper.

models a finite program behaviour to be controled by some execution environment: upon each action (e.g. a request for some service), a reply `true` or `false` from the environment determines further execution. Any execution trace of a basic thread ends either in the (successful) termination state or in the deadlock state. Both these states are modeled as special thread constants. *Regular* threads extend basic threads by comprising loop behaviour, and are reminiscent of flowcharts [15, 11]. Threads may make use of services, i.e., devices that control (part of) their execution by consuming actions, providing the appropriate reply, and suppressing observable activity. Regular threads using the service of a single stack are called *pushdown* threads. Apart from the distinction between deadlock and termination, pushdown threads are comparable to pushdown automata. We show that quivalence of pushdown threads is decidable, whereas pushdown thread inclusion is undecidable.

The paper is structured as follows: in Section 2, we outline the fundamental properties of thread algebra. In Section 3, we show that equivalence between pushdown threads is decidable by reducing the equivalence problem for deterministic pushdown automata [19, 21, 22] to our equivalence problem. In Section 4, we prove that inclusion is undecidable for pushdown threads. Here we reduce the halting problem for Minsky machines to the inclusion problem—an approach also taken in Jančar et al. [13]. In Section 5 we shortly discuss a programming notation for the specification of pushdown threads. The paper ends with some conclusions in Section 6.

2 Threads and services

Basic thread algebra [6][3], BTA, is a form of process algebra which is tailored for the description of sequential program behaviour. Based on a finite set of *actions* A, it has the following constants and operators:

- the *termination* constant S,
- the *deadlock* or *inaction* constant D,
- for each $a \in A$, a binary *postconditional composition* operator $\_ \trianglelefteq a \trianglerighteq \_$.

We use *action prefixing* $a \circ P$ as an abbreviation for $P \trianglelefteq a \trianglerighteq P$ and take $\circ$ to bind strongest. Furthermore, for $n \in \mathbb{N}$ we define $a^n \circ P$ by $a^0 \circ P = P$ and $a^{n+1} \circ P = a \circ (a^n \circ P)$.

The operational intuition behind this algebraic framework is that each action represents a command which is to be processed by the execution environment of the thread. More specifically, an action is taken as a command for a service offered by the environment. The processing of a command may involve a change of state of this environment. At completion of the processing of the command, the service concerned produces a reply value. This reply is either `true` or `false` and is returned to the thread under execution. The thread $P \trianglelefteq a \trianglerighteq Q$ will then proceed

[3] In [5], basic thread algebra is introduced under the name *basic polarized process algebra*.

as P if the processing of a leads to the reply `true` indicating the successful processing of a, and it will proceed as Q if the processing of a leads to the unsuccessful reply `false`.

BTA can be equipped with a partial order and an *approximation operator* in the following way:

1. $\sqsubseteq$ is the partial ordering on BTA generated by the clauses
   (a) for all $P \in \text{BTA}$, $\mathsf{D} \sqsubseteq P$, and
   (b) for all $P_1, P_2, Q_1, Q_2 \in \text{BTA}$, $a \in A$,
$$P_1 \sqsubseteq Q_1 \ \&\ P_2 \sqsubseteq Q_2 \Rightarrow P_1 \trianglelefteq a \trianglerighteq P_2 \sqsubseteq Q_1 \trianglelefteq a \trianglerighteq Q_2.$$
2. $\pi : \mathbb{N} \times \text{BTA} \to \text{BTA}$ is the approximation operator determined by the equations
   (a) for all $P \in \text{BTA}$, $\pi(0, P) = \mathsf{D}$,
   (b) for all $n \in \mathbb{N}$, $\pi(n+1, \mathsf{S}) = \mathsf{S}$, $\pi(n+1, \mathsf{D}) = \mathsf{D}$, and
   (c) for all $P, Q \in \text{BTA}, n \in \mathbb{N}$,
$$\pi(n+1, P \trianglelefteq a \trianglerighteq Q) = \pi(n, P) \trianglelefteq a \trianglerighteq \pi(n, Q).$$

   We further write $\pi_n(P)$ instead of $\pi(n, P)$.

The operator π finitely approximates every thread in BTA. That is, for all $P \in$ BTA,

$$\exists n \in \mathbb{N}\ \pi_0(P) \sqsubseteq \pi_1(P) \sqsubseteq \cdots \sqsubseteq \pi_n(P) = \pi_{n+1}(P) = \cdots = P.$$

Every thread in BTA is finite in the sense that there is a finite upper bound to the number of consecutive actions it can perform. Following the metric theory of [1] in the form developed as the basis of the introduction of processes in [4], BTA has a completion BTA^∞ which comprises also the infinite threads. Standard properties of the completion technique yield that we may take BTA^∞ as the cpo consisting of all so-called *projective* sequences. That is,

$$\text{BTA}^\infty = \{(P_n)_{n \in \mathbb{N}} \mid \forall n \in \mathbb{N}\ (P_n \in \text{BTA} \ \&\ \pi_n(P_{n+1}) = P_n)\}$$

with

$$(P_n)_{n \in \mathbb{N}} \sqsubseteq (Q_n)_{n \in \mathbb{N}} \Leftrightarrow \forall n \in \mathbb{N}\ P_n \sqsubseteq Q_n$$

and

$$(P_n)_{n \in \mathbb{N}} = (Q_n)_{n \in \mathbb{N}} \Leftrightarrow \forall n \in \mathbb{N}\ P_n = Q_n.$$

For a detailed account of this construction see [2]. In this cpo structure, finite linear recursive specifications represent continuous operators having as unique fixed points *regular* threads, i.e., threads which can only reach finitely many states. A finite linear recursive specification over BTA^∞ is a set of equations

$$X_i = t_i(\overline{X})$$

for $i \in I$ with I some finite index set and all $t_i(\overline{X})$ of the form S, D, or $X_{i_l} \trianglelefteq a_i \trianglerighteq X_{i_r}$ for $i_l, i_r \in I$.

Example 1. We define the regular threads

1. $a^n \circ \mathsf{D}$,
2. $a^n \circ \mathsf{S}$ and
3. a^∞ (this informal notation will be often used in the sequel)

as the fixed points for X_1 in the specifications

1. $X_1 = a \circ X_2, \ldots, X_{n-1} = a \circ X_n, X_n = \mathsf{D}$,
2. $X_1 = a \circ X_2, \ldots, X_{n-1} = a \circ X_n, X_n = \mathsf{S}$,
3. $X_1 = a \circ X_1$, respectively.

Both $a^n \circ \mathsf{D}$ and $a^n \circ \mathsf{S}$ are finite threads; a^∞ is the infinite thread corresponding to the projective sequence $(P_n)_{n\in\mathbb{N}}$ with $P_0 = \mathsf{D}$ and $P_{n+1} = a \circ P_n$. Observe that e.g. $a^n \circ \mathsf{D} \sqsubseteq a^n \circ \mathsf{S}$, $a^n \circ \mathsf{D} \sqsubseteq a^\infty$ but $a^n \circ \mathsf{S} \not\sqsubseteq a^\infty$.

In reasoning with finite linear specifications, we shall from now on identify variables and their fixed points. For example, we say that P *is* the regular thread defined by $P = a \circ P$ instead of stating that P equals the fixed point for X in the finite linear specification $X = a \circ X$. Furthermore, in a finite linear specification

$$P_1 = t_1(\overline{P}), ..., P_n = t_n(\overline{P}), \tag{1}$$

the threads P_i will also be referred to as *states.* As another example following this convention, we show that for regular threads P and Q, $P \sqsubseteq Q$ is decidable (recall that $\sqsubseteq$ is decidable for finite threads). Because one can always take the disjoint union of two recursive specifications, it suffices to show that $P_i \sqsubseteq P_j$ is decidable in (1) above. This follows with finite approximations from the following assertion (the idea being that any loop has at most length n):

$$\forall i, j \leq n \;\; \pi_n(P_i) \sqsubseteq \pi_n(P_j) \Rightarrow P_i \sqsubseteq P_j \tag{2}$$

where $\pi_l(P_k)$ is defined by $\pi_l(t_k(\overline{P}))$. To prove (2), assume that $n > 1$ (otherwise the implication follows trivially). Choose i, j and assume that $\pi_n(P_i) \sqsubseteq \pi_n(P_j)$. Suppose $P_i \not\sqsubseteq P_j$. Then for some $k > n$, $\pi_k(P_i) \not\sqsubseteq \pi_k(P_j)$ while $\pi_{k-1}(P_i) \sqsubseteq \pi_{k-1}(P_j)$. So there exists a trace of length k from P_i of the form

$$P_i \xrightarrow{a_{\mathtt{true}}} P_{i'} \xrightarrow{b_{\mathtt{false}}} ...$$

that is not a trace of P_j, while by the assumption the first n actions of this trace are a trace of P_j. These n actions are connected by $n{+}1$ states, and because there are only n different states, a repetition occurs in this sequence of states. So the trace witnessing $\pi_k(P_i) \not\sqsubseteq \pi_k(P_j)$ can be made shorter, contradicting k's minimality and hence the supposition. Thus $P_i \sqsubseteq P_j$. As a corollary, also $P = Q$ is decidable for regular threads P and Q.

A *service*[4] is a pair $\langle \Sigma, F\rangle$ consisting of a set Σ of special actions and a reply function F. The reply function F of a service $\langle \Sigma, F\rangle$ is a mapping that gives for each finite sequence of actions from Σ the reply produced by the service. This reply is a Boolean value `true` or `false`.

Example 2. Services that will occur in Section 3 and 4 are

1. $C = \langle \Sigma, F\rangle$ with $\Sigma = \{\mathtt{inc}, \mathtt{dec}\}$ consisting of the increase and decrease actions of a natural number counter and the reply function F which always replies `true` to increase actions and `false` to decrease actions if and only if the counter is zero. We denote by $C(n)$ a counter with value n.
2. $S = \langle \Sigma, F\rangle$ with $\Sigma = \{\mathtt{push}{:}i, \mathtt{topeq}{:}i, \mathtt{empty}, \mathtt{pop} \mid i = \mathtt{1}, ..., \mathtt{n}\}$ for some `n` where $\mathtt{push}{:}i$ pushes i onto the stack and yields reply `true`, the action $\mathtt{topeq}{:}i$ tests whether i is on top of the stack, `empty` tests whether the stack is empty, and `pop` pops the stack if it is non-empty with reply `true` and yields `false` otherwise. We denote by $S(\alpha)$ a stack with contents $\alpha \in \{\mathtt{1}, ..., \mathtt{n}\}^*$. Observe that counters can be seen as particular stacks (take $\mathtt{n} = 1$).

In order to provide a specific description of the interaction between a thread and a service, we will use for actions the general notation `c.a` where `c` is the so-called *channel* and `a` is the so-called *co-action*. In particular, we will write e.g. `c.inc` to denote the action which increases a counter via channel `c` and `s.pop` to denote the action which pops a stack via channel `s` . For a service $\mathcal{S} = \langle \Sigma, F\rangle$ and a finite thread P, the defining rules for $P/_{\mathsf{c}}\, \mathcal{S}$ (the thread P *using the service* $\mathcal{S}$ *via channel c*) are:

$$\begin{aligned}
\mathsf{S}/_{\mathsf{c}}\, \mathcal{S} &= \mathsf{S},\\
\mathsf{D}/_{\mathsf{c}}\, \mathcal{S} &= \mathsf{D},\\
(P \trianglelefteq \mathsf{c}'.\mathsf{a} \trianglerighteq Q)/_{\mathsf{c}}\, \mathcal{S} &= (P/_{\mathsf{c}}\, \mathcal{S}) \trianglelefteq \mathsf{c}'.\mathsf{a} \trianglerighteq (Q/_{\mathsf{c}}\, \mathcal{S}) \text{ if } \mathsf{c}' \neq \mathsf{c},\\
(P \trianglelefteq \mathsf{c.a} \trianglerighteq Q)/_{\mathsf{c}}\, \mathcal{S} &= P/_{\mathsf{c}}\, \mathcal{S}' \text{ if } \mathsf{a} \in \Sigma \text{ and } F(\mathsf{a}) = \mathtt{true},\\
(P \trianglelefteq \mathsf{c.a} \trianglerighteq Q)/_{\mathsf{c}}\, \mathcal{S} &= Q/_{\mathsf{c}}\, \mathcal{S}' \text{ if } \mathsf{a} \in \Sigma \text{ and } F(\mathsf{a}) = \mathtt{false},\\
(P \trianglelefteq \mathsf{c.a} \trianglerighteq Q)/_{\mathsf{c}}\, \mathcal{S} &= \mathsf{D} \text{ if } \mathsf{a} \notin \Sigma.
\end{aligned}$$

where $\mathcal{S}' = \langle \Sigma, F'\rangle$ with $F'(\sigma) = F(\mathsf{a}\sigma)$ for all co-action sequences $\sigma \in \Sigma^+$. The use operator is expanded to infinite threads P by stipulating

$$P/_{\mathsf{c}}\, \mathcal{S} = (\pi_n(P)/_{\mathsf{c}}\, \mathcal{S})_{n\in\mathbb{N}}.$$

As a consequence, $P/_{\mathsf{c}}\mathcal{S} = \mathsf{D}$ if for any n, $\pi_n(P)/_{\mathsf{c}}\mathcal{S} = \mathsf{D}$. Finally, repeated applications of the use operator bind to the left, thus $P/_{\mathtt{c0}}\mathcal{S}_0/_{\mathtt{c1}}\mathcal{S}_1 = (P/_{\mathtt{c0}}\mathcal{S}_0)/_{\mathtt{c1}}\mathcal{S}_1$.

Example 3. We consider again the threads $a^n \circ \mathsf{D}$, $a^n \circ \mathsf{S}$ and a^∞ from Example 1 but now in the versions $\mathsf{c.a}^n \circ \mathsf{D}$ (short for $(\mathsf{c.a})^n \circ \mathsf{D}$), $\mathsf{c.a}^n \circ \mathsf{S}$ and $\mathsf{c.a}^\infty$ for some channel `c` and some service $\mathcal{S} = \langle \Sigma, F\rangle$ with $\mathsf{a} \in \Sigma$. Then $(\mathsf{c.a}^n \circ \mathsf{D})/_{\mathsf{c}}\, \mathcal{S} = \mathsf{D}$, $(\mathsf{c.a}^n \circ \mathsf{S})/_{\mathsf{c}}\, \mathcal{S} = \mathsf{S}$ but $\mathsf{c.a}^\infty/_{\mathsf{c}}\, \mathcal{S} = \mathsf{D}$. The last identity can also be retrieved as

[4] In [7] a service is called a *state machine*.

follows: if $P = \mathsf{c.a} \circ P$, then $P/_{\mathsf{c}}\,\mathcal{S} = (\mathsf{c.a} \circ P)/_{\mathsf{c}}\,\mathcal{S} = P/_{\mathsf{c}}\,\mathcal{S}'$. In this way, the computation will diverge, characterizing the rule: *"If for a regular thread P the defining rules for $P/_{\mathsf{c}}\,\mathcal{S}$ fail to prove $P/_{\mathsf{c}}\,\mathcal{S} = \mathsf{S}$ or $\pi_1(P/_{\mathsf{c}}\,\mathcal{S}) = b \circ \mathsf{D}$ for some action b, then $P/_{\mathsf{c}}\,\mathcal{S} = \mathsf{D}$".*

In the next example we show that the use of services may turn regular threads into irregular ones.

Example 4. Let $a \in A$. Consider the following regular process P: [5]

$$P = (\mathtt{c0.inc} \circ P) \trianglelefteq a \trianglerighteq ((P \trianglelefteq \mathtt{c0.dec} \trianglerighteq \mathsf{D}) \trianglelefteq a \trianglerighteq (\mathsf{D} \trianglelefteq \mathtt{c0.dec} \trianglerighteq P)).$$

With the counter C defined in Example 2, define for $n \in \mathbb{N}$ the thread $\mathcal{P}_n$ by

$$\mathcal{P}_n = P/_{\mathtt{c0}}\, C(n).$$

This definition yields the following infinite recursive specification:

$$\begin{aligned}\mathcal{P}_0 &= \ \ \mathcal{P}_1 \ \trianglelefteq a \trianglerighteq (\mathsf{D} \trianglelefteq a \trianglerighteq \mathcal{P}_0)\\ \mathcal{P}_{k+1} &= \mathcal{P}_{k+2} \trianglelefteq a \trianglerighteq (\mathcal{P}_k \trianglelefteq a \trianglerighteq \mathsf{D}) \text{ for } k \in \mathbb{N}.\end{aligned}$$

Now assume that $\mathcal{P}_n \sqsubseteq \mathcal{P}_m$ for some $n > m$. Then, by definition of $\mathcal{P}_m$ (following the rightmost branches of $\trianglelefteq a \trianglerighteq$),

$$(\mathcal{P}_{n-m-1} \trianglelefteq a \trianglerighteq \mathsf{D}) \sqsubseteq (\mathsf{D} \trianglelefteq a \trianglerighteq \mathcal{P}_0).$$

It is clear that $\sqsubseteq$ does not hold, thus we obtained a contradiction and $\mathcal{P}_n \not\sqsubseteq \mathcal{P}_m$. In the same way it follows that $\mathcal{P}_n \not\sqsupseteq \mathcal{P}_m$ if $n > m$. We conclude that $\mathcal{P}_0$ is not regular: for each $k \in \mathbb{N}$, the state $\mathcal{P}_k$ can be reached, and if $n \neq m$, then the states $\mathcal{P}_n$ and $\mathcal{P}_m$ are different. [6]

We call a regular thread that uses a stack or a counter as described in Example 2 a *pushdown thread* (typically, any thread $\mathcal{P}_k$ defined in Example 4 is a pushdown thread).

3 Decidable equality

In this section we prove the decidability of $P/_{\mathsf{s}}\, S(\alpha) = Q/_{\mathsf{s}}\, S(\beta)$ for regular threads P and Q, and a stack S over some finite data type with contents α, respectively β. We first discuss deterministic pushdown automata (dpda's) and recall the decidability result establishing dpda-equivalence (Sénizergues [19, 21]). We base our approach on Stirling's account of this result in [22], as this is closer to our setting. In the sequel we write ϵ for the empty sequence over any alphabet.

[5] Note that a *linear* recursive specification requires (at least) six equations in this case.

[6] Note that for all n, $\mathcal{P}_n \sqsubseteq a^\infty$ (this follows from $\pi_k(\mathcal{P}_n) \sqsubseteq \pi_k(a^\infty) = a^k \circ \mathsf{D}$ for all k) and $\mathcal{P}_n \not\sqsubseteq \mathsf{S}$. We will use these properties in Section 4.

A *pushdown automaton* (pda) $\mathcal{A}$ is given by a finite set $\mathcal{P}$ of states, a finite set $\mathbb{S}$ of stack symbols, a finite alphabet $\mathbb{A}$, and a finite set of basic transitions of the form

$$Px \xrightarrow{a} Q\alpha$$

with $P, Q \in \mathcal{P}$, $x \in \mathbb{S}$, $a \in \mathbb{A} \cup \{\epsilon\}$, and $\alpha \in \mathbb{S}^*$. Expressions of the form Px are further called *source-configurations.* A *configuration* of $\mathcal{A}$ is any expression $P\alpha$ whose behaviour is determined by the basic transitions and the prefix rule

$$\textit{if } Px \xrightarrow{a} Q\alpha \textit{ then } Px\beta \xrightarrow{a} Q\alpha\beta.$$

The language accepted by a configuration $P\alpha$, notation

$$L(\mathcal{A}, P\alpha),$$

is $\{w \in \mathbb{A}^* \mid \exists Q \in \mathcal{P}. P\alpha \xrightarrow{w} Q\}$ where the extended transitions for words are defined as expected. Note that ϵ-transitions are swallowed in the usual fashion and that acceptance is by empty stack.

A *deterministic* pushdown automaton (dpda) $\mathcal{A}'$ has three restrictions on its basic transitions:

- if $Px \xrightarrow{a} Q\alpha$ and $Px \xrightarrow{a} R\beta$ for $a \in \mathbb{A}$, then $Q = R$ and $\alpha = \beta$,
- if $Px \xrightarrow{\epsilon} Q\alpha$ and $Px \xrightarrow{a} R\lambda$ then $a = \epsilon$,
- if $Px \xrightarrow{\epsilon} Q\alpha$ then $\alpha = \epsilon$ (so ϵ-transitions can only pop the stack; in [21] this case is referred to as a *normalized* dpda).

Note that the language accepted by any configuration of a dpda $\mathcal{A}'$ is prefix-free: if w is accepted then no proper prefix of w is accepted. For dpda's it is decidable whether $L(\mathcal{A}', P\alpha) = L(\mathcal{A}', Q\beta)$ ([19, 21]). With this decidability result we can prove the main result of this section:

Theorem 1. *For regular threads P and Q, and a stack S over a finite data type it is decidable whether*

$$P/_{\mathtt{s}}\, S(\alpha) = Q/_{\mathtt{s}}\, S(\beta),$$

where α, β represent the contents of S.

Proof. Let S be the (empty) stack controlled by the actions

$$\{\mathtt{s.push}{:}i, \mathtt{s.topeq}{:}i, \mathtt{s.empty}, \mathtt{s.pop} \mid i = \mathtt{1}, ..., \mathtt{n}\}$$

for some $\mathtt{n} \geq 1$. We write $S(\alpha)$ if the stack contains the elements from sequence $\alpha \in \{\mathtt{1}, ..., \mathtt{n}\}^*$ with the leftmost element of α on top. Furthermore, $\mathtt{s.push}{:}i$ pushes i onto the stack and yields reply `true`, the action $\mathtt{s.topeq}{:}i$ tests whether i is on top of the stack, `s.empty` tests whether the stack is empty, and `s.pop` pops the stack if it is non-empty (reply `true`) and yields `false` otherwise. Finally, we assume that in P and Q there are external actions $a_0, ..., a_m$, and that P and Q are given by a single finite linear specification. Using five transformations, we reduce the dpda-equivalence problem as discussed above to the question $P/_{\mathtt{s}}\, S(\alpha) = Q/_{\mathtt{s}}\, S(\beta)$.

Adapting the stack contents. In order to use the dpda-equivalence result, the stack should be empty upon termination and non-empty upon the start because language acceptance is by empty stack, and defined on configurations with non-empty stack. This can be achieved as follows:

- Extend the stack S to S_0 as follows:
  - in order to start with a non-empty stack, add an extra stack symbol $\mathtt{0}$ and extend all reply functions to actions over stack symbols $\{\mathtt{0}, \mathtt{1}, ..., \mathtt{n}\}$ in the obvious way, except for `s.pop` that leaves $\mathtt{0}$ on the stack and yields in this case reply `false`,
  - define a new action `s.pop:all` that pops any symbol in $\{\mathtt{0}, \mathtt{1}, ..., \mathtt{n}\}$ from the stack with reply `true`, and yields reply `false` if the stack is empty.
- In the specification of P and Q,
  - replace each occurrence of `s.empty` by `s.topeq:0` (so, $\mathtt{0}$ acts as an empty stack marker),
  - replace any equation $R = \mathsf{S}$ by $R = \mathtt{s.pop{:}all} \circ \overline{\mathsf{S}}$ (where $\overline{\mathsf{S}}$ is fresh),
  - add the equation $\overline{\mathsf{S}} = \overline{\mathsf{S}} \unlhd \mathtt{s.pop{:}all} \unrhd \mathsf{S}$.

Call the resulting threads P_0 and Q_0, respectively, and assume these are given by an appropriate adaptation of the linear specification of P and Q (note that `s.push:0` and `s.empty` do not occur in this specification). It follows straightforwardly that for $\alpha, \beta \in \{\mathtt{1}, ..., \mathtt{n}\}^*$,

$$P /_{\mathtt{s}} S(\alpha) = Q /_{\mathtt{s}} S(\beta) \Leftrightarrow P_0 /_{\mathtt{s}} S_0(\alpha\mathtt{0}) = Q_0 /_{\mathtt{s}} S_0(\beta\mathtt{0})$$

and that upon S (i.e., termination), the stack S_0 is empty.

Replacement of D *by explicit loops.* In the specification of P_0 and Q_0, replace each equation $R = \mathsf{D}$ by $R = \mathtt{s.push{:}1} \circ L$ (where L is fresh) and add the equation $L = \mathtt{s.push{:}1} \circ L$. Call the resulting threads P_1 and Q_1, respectively, for some appropriate adaptation to a linear recursive specification. Again it is straightforward that for $\alpha, \beta \in \{\mathtt{1}, ..., \mathtt{n}\}^*$,

$$P_0 /_{\mathtt{s}} S_0(\alpha\mathtt{0}) = Q_0 /_{\mathtt{s}} S_0(\beta\mathtt{0}) \Leftrightarrow P_1 /_{\mathtt{s}} S_0(\alpha\mathtt{0}) = Q_1 /_{\mathtt{s}} S_0(\beta\mathtt{0})$$

and that upon S, the stack S_0 is empty.

Normalization of infinite traces. Let *halt* be a fresh external action. Replace in P_1 and Q_1's specification each equation $R = R_l \unlhd a \unrhd R_r$ with a an external action by $R = \overline{\mathsf{S}} \unlhd \mathit{halt} \unrhd (R_l \unlhd a \unrhd R_r)$. Call the resulting threads P_2 and Q_2, respectively. Again it is straightforward that for $\alpha, \beta \in \{\mathtt{1}, ..., \mathtt{n}\}^*$,

$$P_1 /_{\mathtt{s}} S_0(\alpha\mathtt{0}) = Q_1 /_{\mathtt{s}} S_0(\beta\mathtt{0}) \Leftrightarrow P_2 /_{\mathtt{s}} S_0(\alpha\mathtt{0}) = Q_2 /_{\mathtt{s}} S_0(\beta\mathtt{0})$$

and that upon S, the stack S_0 is empty. Moreover, each infinite sequence of external actions in $P_1 /_{\mathtt{s}} S_0(\alpha\mathtt{0})$ or $Q_1 /_{\mathtt{s}} S_0(\beta\mathtt{0})$ becomes after this transformation interlarded with $\mathit{halt} \circ \mathsf{S}$ exits, so gives rise to an infinite number of (finite) traces.

Transformation to pda-equivalence. From the linearized specification of P_2 and Q_2, construct a pda $\mathcal{A}_1$ as follows: for $\mathcal{P}$, the set of states, take those of the linear specification; for $\mathbb{S}$, the set of stack symbols, take $\{\mathtt{0},...,\mathtt{n}\}$; and for the alphabet $\mathbb{A}$ take $\{a_{\mathtt{true}}, a_{\mathtt{false}} \mid a \text{ an external action in } P_2 \text{ or } Q_2\}$. As for the basic transitions,

- for each state $R = R_l \trianglelefteq a \trianglerighteq R_r$ with a an external action and $i \in \{\mathtt{1},...,\mathtt{n}\}$, define transitions $Ri \xrightarrow{a_{\mathtt{true}}} R_l i$ and $Ri \xrightarrow{a_{\mathtt{false}}} R_r i$,
- for each state $R = R_l \trianglelefteq \mathtt{s.push}{:}j \trianglerighteq R_r$ and $i \in \{\mathtt{1},...,\mathtt{n}\}$, define transitions $Ri \xrightarrow{\epsilon} R_r ji$,
- for each state $R = R_l \trianglelefteq \mathtt{s.topeq}{:}j \trianglerighteq R_r$ and $i \in \{\mathtt{0},\mathtt{1},...,\mathtt{n}\} \setminus \{j\}$, define transitions $Rj \xrightarrow{\epsilon} R_l j$, and define a transition $Ri \xrightarrow{\epsilon} R_r i$,
- for each state $R = R_l \trianglelefteq \mathtt{s.pop} \trianglerighteq R_r$ and $i \in \{\mathtt{1},...,\mathtt{n}\}$, define transitions $Ri \xrightarrow{\epsilon} R_l$ and $R\mathtt{0} \xrightarrow{\epsilon} R_r \mathtt{0}$,
- for each state $R = R_l \trianglelefteq \mathtt{s.pop{:}all} \trianglerighteq R_r$ and $i \in \{\mathtt{0},\mathtt{1},...,\mathtt{n}\}$, define transitions $Ri \xrightarrow{\epsilon} R_l$.

It follows immediately that for $\alpha, \beta \in \{\mathtt{1},...,\mathtt{n}\}^*$,

$$P_2/_{\mathtt{s}}\, S_0(\alpha\mathtt{0}) = Q_2/_{\mathtt{s}}\, S_0(\beta\mathtt{0}) \Leftrightarrow L(\mathcal{A}_1, P_2\alpha\mathtt{0}) = L(\mathcal{A}_1, Q_2\beta\mathtt{0}).$$

Note that $\mathcal{A}_1$'s basic transitions with label ϵ are very limited: either a push or a pop, or they do not change the stack. Also, the transition relation is deterministic: for each source configuration Ri at most one transition is defined.

Transformation to dpda-equivalence. The only remaining problem is that $\mathcal{A}_1$ is not normalized, i.e., may have ϵ-transitions that do not pop the stack. However, the set of basic ϵ-transitions can be transformed to one that contains only popping ϵ-transitions while preserving language acceptance. Not swallowing ϵ-transitions, each source-configuration Ri in $\mathcal{A}_1$ with an ϵ-transition can be classified by tracing its consecutive transitions. Assume $\mathcal{A}_1$ has k states. Because $\mathcal{A}_1$ has $n{+}1$ stack symbols, it is sufficient to consider at most $k(n{+}1){+}1$ consecutive transitions. Let $\xrightarrow{\epsilon}\!\!\!\rightarrow$ stand for at least 0 and at most $k(n{+}1){-}1$ ϵ-transitions, then the following classes can be distinguished:

1. $Ri \xrightarrow{\epsilon} R'\alpha \xrightarrow{\epsilon}\!\!\!\rightarrow R''\alpha' \xrightarrow{a} R'''\beta$ where $R''\alpha' \xrightarrow{a} R'''\beta$ is the first occurring $\mathbb{A}$-transition,
2. $Ri \xrightarrow{\epsilon} R'\alpha \xrightarrow{\epsilon}\!\!\!\rightarrow R''\alpha' \xrightarrow{\epsilon} R'''$ where $R''\alpha' \xrightarrow{\epsilon} R'''$ is the first ϵ-transition from Ri that empties the stack,
3. Not 1 or 2, i.e., $Ri \xrightarrow{\epsilon} R'j\alpha \xrightarrow{\epsilon}\!\!\!\rightarrow R''m\alpha' \xrightarrow{\epsilon} R'''l\beta$ and in the sequence of associated source-configurations a repetition occurs (and no empty stack).

Now define a dpda $\mathcal{A}_2$ with $\mathcal{P}$, $\mathbb{S}$ and $\mathbb{A}$ as in $\mathcal{A}_1$, and with basic transitions as follows:

- in case 1, replace $Ri \xrightarrow{\epsilon} R'\alpha$ by $Ri \xrightarrow{a} R'''\beta$,
- in case 2, replace $Ri \xrightarrow{\epsilon} R'\alpha$ by $Ri \xrightarrow{\epsilon} R'''$,
- in case 3, simply omit the ϵ-transition from Ri,

– keep all basic transitions with a label in $\mathbb{A}$ from $\mathcal{A}_1$.

It is clear that $L(\mathcal{A}_1, R\alpha) = L(\mathcal{A}_2, R\alpha)$. In $\mathcal{A}_2$, language equivalence is decidable as $\mathcal{A}_2$ satisfies the dpda-definition given above (which is taken from [22]; note that the requirement in [22] that $|\alpha| < 3$ in a basic transition $Px \xrightarrow{a} Q\alpha$ can be trivially fulfilled by introducing auxiliary stack symbols). □

We note that in the proof above, the transformation step *Normalization of infinite traces* can be skipped if one considers *bisimulation equivalence* in the transition graph of a normalized dpda, which is decidable as well (Sénizergues [20]).

We conclude this section with a short comment. The restriction to a stack over a *finite* data type is not essential for the decidability of equality between pushdown threads: also for a stack $S_{\mathbb{N}}$ over the natural numbers $\mathbb{N}$ and regular threads P and Q it holds that $P/_{\mathtt{s}}\, S_{\mathbb{N}}(\alpha) = Q/_{\mathtt{s}}\, S_{\mathbb{N}}(\beta)$ is decidable. This can be seen by representing natural numbers in some unary notation and using a second data element as a separator. For example,

$$011101101111$$

represents the stack containing $2, 1, 3$ (so, the natural number n is represented by $n+1$ pushes of 1). For any $i \in \mathbb{N}$, the actions $\mathtt{s.push}{:}i$ and $\mathtt{s.topeq}{:}i$ can be expressed in this notation, be it a bit cumbersome. The action `s.pop` is easier to define, and `s.empty` need not be redefined. Thus given regular threads P and Q, there exists a transformation ϕ (depending on the stack-actions in P and Q) such that

$$P/_{\mathtt{s}}\, S_{\mathbb{N}}(\alpha) = Q/_{\mathtt{s}}\, S_{\mathbb{N}}(\beta) \Leftrightarrow \phi(P)/_{\mathtt{s}}\, S(\overline{\alpha}) = \phi(Q)/_{\mathtt{s}}\, S(\overline{\beta})$$

where S is the stack over symbols $\{\mathtt{0}, \mathtt{1}\}$ and $\overline{\alpha}$ transforms a sequence of natural numbers as indicated above.

Successor and predecessor for $S_{\mathbb{N}}$ can be easily defined using the representation discussed here. This is not the case for any action controlling $S_{\mathbb{N}}$. For instance, an action `swap` that exchanges the two top-elements of $S_{\mathbb{N}}$ is not definable because with this action the functionality of a Minsky machine is obtained (using the even stack positions to hold the values of the first counter, and the odd positions for those of the second counter). So with `swap` equality is not decidable. Neither is equality of the two top-elements a definable action. Of course, both these actions are definable for stacks over a finite data type.

4 Undecidable inclusion

The *Minsky machine* is a universal model of computation first used in [16, 17]. It is a simple imperative program consisting of a sequence of instructions labelled by natural numbers from `1` to some `L`. It starts from the instruction labelled `1`, halts if `stop` is reached and operates with two natural number counters $\mathtt{c_0}$ and $\mathtt{c_1}$. The instruction set consists of three types of instructions:

1. $\mathtt{l} : \mathtt{c_i} := \mathtt{c_i} + 1; \mathtt{goto}\ \mathtt{l'}$
2. $\mathtt{l} : \mathtt{if}\ \mathtt{c_i} = 0\ \mathtt{then}\ \mathtt{goto}\ \mathtt{l'}\ \mathtt{else}\ \mathtt{c_i} := \mathtt{c_i} - 1; \mathtt{goto}\ \mathtt{l''}$
3. $\mathtt{l} : \mathtt{stop}$

where $\mathtt{i} \in \{0, 1\}$ and $\mathtt{l}, \mathtt{l'}, \mathtt{l''} \in \{1, \ldots, \mathtt{L}\}$. It should be clear that the execution process is deterministic and has no failure. Any such process is either finished by the execution of the `stop` instruction or lasts forever. As expected, the halting problem for Minsky machines is undecidable:

Theorem 2. *([16, 17]) It is undecidable whether a Minsky machine halts when both counter values are initially zero.*

In our setting, a counter C is a service $\langle \Sigma, F \rangle$ with increase and decrease actions $\Sigma = \{\mathtt{inc}, \mathtt{dec}\}$ and a reply function F which always replies `true` to `inc` and `false` to `dec` if and only if the counter value is zero. A Minsky machine canonically defines the equations of a specification of a regular thread:

$$\begin{array}{lcl}
\mathtt{l} : \mathtt{c_i} := \mathtt{c_i} + 1; \mathtt{goto}\ \mathtt{l'} & \mapsto & M_l = \mathtt{ci.inc} \circ M_{l'} \\
\mathtt{l} : \mathtt{if}\ \mathtt{c_i} = 0\ \mathtt{then}\ \mathtt{goto}\ \mathtt{l'} & \mapsto & M_l = M_{l''} \trianglelefteq \mathtt{ci.dec} \trianglerighteq M_{l'} \\
\qquad\qquad \mathtt{else}\ \mathtt{c_i} := \mathtt{c_i} - 1; \mathtt{goto}\ \mathtt{l''} & & \\
\mathtt{l} : \mathtt{stop} & \mapsto & M_l = \mathsf{S}.
\end{array}$$

We call a thread M_l as defined above a *Minsky thread*, thus a regular thread over S and the counter actions `c0.inc`, `c0.dec`, `c1.inc`, `c1.dec` where the channels `c0` and `c1` refer to the two internal counters C_0 and C_1, respectively. [7] The halting problem for Minsky machines can now be rephrased to

Theorem 3. *It is undecidable whether for a Minsky thread M it holds that*

$$M /_{\mathtt{c0}}\, C_0(0) /_{\mathtt{c1}}\, C_1(0) = \mathsf{S}.$$

(Of course, if $M /_{\mathtt{c0}}\, C_0(0) /_{\mathtt{c1}}\, C_1(0) \neq \mathsf{S}$, it equals D.) The undecidability proof that follows consists of a reduction from the above halting problem to the inclusion problem for the use of a single internal counter—and thus to the inclusion problem for the use of a single internal stack.

For technical purposes we introduce the *norm* of a Minsky thread operating on counters $C_0(n)$ and $C_1(m)$ as the number of counter actions until termination occurs (possibly ∞). The norm is formally defined with help of a transformation θ of finite linear recursive specifications.

Definition 1 (norm). *Let M_1 be a Minsky thread defined by a finite linear recursive specification $E = \{M_n = t_n(\overline{M}) \mid n = 1, ..., k\}$ (for some $k > 0$) and let a be some action. Define $\theta(E) = \{\theta(M_n) = \theta(t_n(\overline{M})) \mid n = 1, ..., k\}$ and define $\theta(t_n(\overline{M}))$ by*

[7] We do not *have* to disambiguate these counters because of the different channel names; however, we do this to enhance readability.

1. $\theta(\mathsf{S}) = \mathsf{S}$
2. *For* $i \in \{0, 1\}$ *and* $\mathtt{b} \in \{\mathtt{inc}, \mathtt{dec}\}$,

$$\theta(M_l \unlhd \mathtt{c}i.\mathtt{b} \unrhd M_r) = a \circ \theta(M_l) \unlhd \mathtt{c}i.\mathtt{b} \unrhd a \circ \theta(M_r).$$

For $n, m \in \mathbb{N}$ *define the* norm $\|M_1, n, m\| \in \mathbb{N} \cup \{\infty\}$ *by* $\|M_1, n, m\| = 0$ *if* $\theta(M_1) = \mathsf{S} \in \theta(E)$, *and for* $k > 0$,

$$\|M_1, n, m\| = k \text{ if } \begin{cases} \pi_{k+1}(\theta(M_1)/_{\mathtt{c0}}\, C_0(n)/_{\mathtt{c1}}\, C_1(m)) = a^k \circ \mathsf{S} \text{ and} \\ \pi_k(\theta(M_1)/_{\mathtt{c0}}\, C_0(n)/_{\mathtt{c1}}\, C_1(m)) = a^{k-1} \circ \mathsf{D}. \end{cases}$$

For $n, m \in \mathbb{N}$, $\|M_1, n, m\| = \infty$ *if for no* $k \in \mathbb{N}$, $\|M_1, n, m\| = k$.

Note that

$$\|M, n, m\| \in \mathbb{N} \Leftrightarrow M/_{\mathtt{c0}}\, C_0(n)/_{\mathtt{c1}}\, C_1(m) = \mathsf{S}. \tag{3}$$

So the question whether $\|M, n, m\| \in \mathbb{N}$ is undecidable.

We now introduce a transformation ψ of Minsky threads which replaces specific runs with the second counter $C_1(0)$ by regular threads where the C_1-actions are simulated by a single external action a in such a way that termination behaviour is preserved. Moreover, C_0-actions are preceded by the simulation of a $\mathtt{c1.inc} \circ \mathtt{c1.dec}$-prefix. The reason for this is that divergence on C_0-actions (as in $\mathtt{c0.inc}^\infty$) then also yields an infinite sequence of a-actions, which enables us to use a smooth proof strategy.

Definition 2 (simulation). *Let* M_1 *be a Minsky thread defined by a finite linear recursive specification* $E = \{M_n = t_n(\overline{M}) \mid n = 1, ..., k\}$ *(for some* $k > 0$*). Define* $\psi(E) = \{\psi(M_n) = \psi(t_n(\overline{M})) \mid n = 1, ..., k\}$ *and define* $\psi(t_n(\overline{M}))$ *by*

1. $\psi(\mathsf{S}) = \mathsf{S}$
2. $\psi(M_i \unlhd \mathtt{c1.inc} \unrhd M_j) = \psi(M_i) \unlhd a \unrhd a^\infty$
3. $\psi(M_i \unlhd \mathtt{c1.dec} \unrhd M_j) = a^\infty \unlhd a \unrhd (\psi(M_i) \unlhd a \unrhd \psi(M_j))$
4. $\psi(M_i \unlhd \mathtt{c0.b} \unrhd M_j) =$

$$[a^\infty \unlhd a \unrhd a \circ (\psi(M_i) \unlhd \mathtt{c0.b} \unrhd \psi(M_j))] \unlhd a \unrhd a^\infty$$

for $\mathtt{b} \in \{\mathtt{inc}, \mathtt{dec}\}$.

We note that $a^\infty/_{\mathtt{c0}}\, C_0(n) = a^\infty$ and therefore we omit any such use-application in reasoning about $\psi(M)/_{\mathtt{c0}}\, C_0(n)$.

In order to prove undecidability of inclusion we use the family $\mathcal{P}_k$ ($k \in \mathbb{N}$) of pushdown threads discussed in Example 4:

Definition 3. *For* $n \in \mathbb{N}$, *define* $\mathcal{P}_n = P/_{\mathtt{c0}}\, C_0(n)$ *by*

$$P = (\mathtt{c0.inc} \circ P) \unlhd a \unrhd ((P \unlhd \mathtt{c0.dec} \unrhd \mathsf{D}) \unlhd a \unrhd (\mathsf{D} \unlhd \mathtt{c0.dec} \unrhd P)).$$

This definition yields the following infinite recursive specification:

$$\begin{aligned}\mathcal{P}_0 &= \quad \mathcal{P}_1 \ \trianglelefteq a \trianglerighteq (\mathsf{D} \trianglelefteq a \trianglerighteq \mathcal{P}_0)\\ \mathcal{P}_{k+1} &= \mathcal{P}_{k+2} \trianglelefteq a \trianglerighteq (\mathcal{P}_k \trianglelefteq a \trianglerighteq \mathsf{D}) \text{ for } k \in \mathbb{N}.\end{aligned}$$

Note that for all n, $\mathcal{P}_n \sqsubseteq a^\infty$ (see Example 4) and $\mathcal{P}_n \not\sqsubseteq \mathsf{S}$. For future reference we derive the following identities:

$$\mathcal{P}_0 = [\mathcal{P}_2 \trianglelefteq a \trianglerighteq (\mathcal{P}_0 \trianglelefteq a \trianglerighteq \mathsf{D})] \trianglelefteq a \trianglerighteq (\mathsf{D} \trianglelefteq a \trianglerighteq \mathcal{P}_0) \tag{4}$$

and for $m > 0$,

$$\mathcal{P}_m = [\mathcal{P}_{m+2} \trianglelefteq a \trianglerighteq (\mathcal{P}_m \trianglelefteq a \trianglerighteq \mathsf{D})] \trianglelefteq a \trianglerighteq (\mathcal{P}_{m-1} \trianglelefteq a \trianglerighteq \mathsf{D}) \tag{5}$$

We shall prove that for any Minsky thread M,

$$M/_{\mathtt{c0}}\, C_0(0)/_{\mathtt{c1}}\, C_1(0) = \mathsf{S} \Leftrightarrow \mathcal{P}_0 \not\sqsubseteq \psi(\mathcal{M})/_{\mathtt{c0}}\, C_0(0).$$

We first prove this equivalence for Minsky threads that use counters with arbitrary initial values (Corollaries 1 and 2, respectively). To enhance readability we shall write in proofs often

$$\begin{aligned}&M/n/m \;\text{ for }\; M/_{\mathtt{c0}}\, C_0(n)/_{\mathtt{c1}}\, C_1(m) \text{ , and}\\ &\psi(M)/n \;\text{ for }\; \psi(M)/_{\mathtt{c0}}\, C_0(n).\end{aligned}$$

Lemma 1. *Let $k \in \mathbb{N}$. For all Minsky threads M and for all $n, m \in \mathbb{N}$,*

$$\|M, n, m\| = k \Rightarrow \mathcal{P}_m \not\sqsubseteq \psi(M)/_{\mathtt{c0}}\, C(n). \tag{6}$$

Proof. By induction on k using case ramification, i.e., considering all possible forms of the equation specifying M.

• $k = 0$. If $\|M, n, m\| = 0$, then $M = \mathsf{S}$ must be M's defining equation and (6) follows immediately.

• $k > 0$. If $\|M, n, m\| = k$, there are four possible forms for the equation specifying M:

a) $M = M_i \trianglelefteq \mathtt{c0.inc} \trianglerighteq M_j$. Then $\|M_i, n{+}1, m\| = k{-}1$ and

$$\psi(M)/n = (a^\infty \trianglelefteq a \trianglerighteq a \circ \psi(M_i)/n{+}1) \trianglelefteq a \trianglerighteq a^\infty. \tag{7}$$

By induction $\mathcal{P}_m \not\sqsubseteq \psi(M_i)/n{+}1$. Assume $\mathcal{P}_m \sqsubseteq \psi(M)/n$. Then by (4) or (5) and (7), $\mathcal{P}_m \sqsubseteq \psi(M_i)/n{+}1$, contradicting the induction hypothesis. Hence $\mathcal{P}_m \not\sqsubseteq \psi(M)/n$.

b) $M = M_i \trianglelefteq \mathtt{c0.dec} \trianglerighteq M_j$. This case is proved similarly, making a case distinction between $n = 0$ and $n > 0$.

c) $M = M_i \trianglelefteq \mathtt{c1.inc} \trianglerighteq M_j$. Then $\|M_i, n, m{+}1\| = k{-}1$ and

$$\psi(M)/n = \psi(M_i)/n \trianglelefteq a \trianglerighteq a^\infty. \tag{8}$$

By induction $\mathcal{P}_{m+1} \not\sqsubseteq \psi(M_i)/n$. Assume $\mathcal{P}_m \sqsubseteq \psi(M)/n$. Then by definition of $\mathcal{P}_m$ and (8), $\mathcal{P}_{m+1} \sqsubseteq \psi(M_i)/n$, contradicting the induction hypothesis. Hence $\mathcal{P}_m \not\sqsubseteq \psi(M)/n$.

d) $M = M_i \trianglelefteq \texttt{c1.dec} \trianglerighteq M_j$. Then

$$\psi(M)/n = a^\infty \trianglelefteq a \trianglerighteq (\psi(M_i)/n \trianglelefteq a \trianglerighteq \psi(M_j/n). \tag{9}$$

Let $m = 0$. Then $\|M_j, n, 0\| = k{-}1$ and by induction $\mathcal{P}_0 \not\sqsubseteq \psi(M_j)/n$. Assume $\mathcal{P}_0 \sqsubseteq \psi(M)/n$. Then by definition of $\mathcal{P}_0$ and (9), $\mathcal{P}_0 \sqsubseteq \psi(M_j)/n$, contradicting the induction hypothesis. Hence $\mathcal{P}_0 \not\sqsubseteq \psi(M)/n$.

Let $m > 0$. Then $\|M_i, n, m{-}1\| = k{-}1$ and by induction $\mathcal{P}_{m-1} \not\sqsubseteq \psi(M_i)/n$. Assume $\mathcal{P}_m \sqsubseteq \psi(M)/n$. Then by definition of $\mathcal{P}_m$ and (9), $\mathcal{P}_{m-1} \sqsubseteq \psi(M_i)/n$, contradicting the induction hypothesis. Hence $\mathcal{P}_m \not\sqsubseteq \psi(M)/n$. □

Combining (3) and Lemma 1 immediately yields

Corollary 1. *For all Minsky threads M and for all $n, m \in \mathbb{N}$,*

$$M/_{\texttt{c0}}\, C_0(n)/_{\texttt{c1}}\, C_1(m) = \mathsf{S} \Rightarrow \mathcal{P}_m \not\sqsubseteq \psi(M)/_{\texttt{c0}}\, C_0(n).$$

The next result is about the inclusion of finite approximations.

Lemma 2. *For all $k \in \mathbb{N}$, Minsky threads M and $n, m \in \mathbb{N}$,*

$$M/_{\texttt{c0}}\, C_0(n)/_{\texttt{c1}}\, C_1(m) = \mathsf{D} \Rightarrow \pi_k(\mathcal{P}_m) \sqsubseteq \pi_k(\psi(M)/_{\texttt{c0}}\, C_0(n)). \tag{10}$$

Proof. By induction on k with base cases $k = 0$ and $k = 1$.

If $k = 0$ then (10) follows trivially.

If $k = 1$ and $M/n/m = \mathsf{D}$, then $M = \mathsf{S}$ is not M's defining equation. Therefore, $\pi_1(\psi(M)/n) = a \circ \mathsf{D} = \pi_1(\mathcal{P}_m)$, which proves (10). (Note that already in this case it is essential that ψ introduces a-actions in the transformation of $\texttt{c0}$-equations.)

Assume $k \geq 2$ and $M/n/m = \mathsf{D}$. Again, $M = \mathsf{S}$ can not be M's defining equation. Consider the remaining four possibilities for M's defining equation:

a) $M = M_i \trianglelefteq \texttt{c0.inc} \trianglerighteq M_j$. Then $M_i/n{+}1/m = \mathsf{D}$ and

$$\pi_k(\psi(M)/n) = [\pi_{k-2}(a^\infty) \trianglelefteq a \trianglerighteq \pi_{k-2}(a \circ \psi(M_i)/n{+}1)] \trianglelefteq a \trianglerighteq \pi_{k-1}(a^\infty).$$

By induction, $\pi_l(\mathcal{P}_m) \sqsubseteq \pi_l(\psi(M_i)/n{+}1)$ for $l < k$. Assume $m = 0$. Then by (4)

$$\begin{aligned}\pi_k(\mathcal{P}_0) &= [\pi_{k-2}(\mathcal{P}_2) \trianglelefteq a \trianglerighteq \pi_{k-2}(\mathcal{P}_0 \trianglelefteq a \trianglerighteq \mathsf{D})] \trianglelefteq a \trianglerighteq \pi_{k-1}(\mathsf{D} \trianglelefteq a \trianglerighteq \mathcal{P}_0)\\ &\sqsubseteq [\pi_{k-2}(a^\infty) \trianglelefteq a \trianglerighteq \pi_{k-2}(a \circ \psi(M_i)/n{+}1)] \trianglelefteq a \trianglerighteq \pi_{k-1}(a^\infty)\\ &= \pi_k(\psi(M)/n).\end{aligned}$$

Assume $m > 0$. Then by (5)

$$\begin{aligned}\pi_k(\mathcal{P}_m) &= [\pi_{k-2}(\mathcal{P}_{m+2}) \unlhd a \unrhd \pi_{k-2}(\mathcal{P}_m \unlhd a \unrhd \mathsf{D})] \unlhd a \unrhd \pi_{k-1}(\mathcal{P}_{m-1} \unlhd a \unrhd \mathsf{D})\\ &\sqsubseteq [\pi_{k-2}(a^\infty) \unlhd a \unrhd \pi_{k-2}(a \circ \psi(M_i)/n{+}1)] \unlhd a \unrhd \pi_{k-1}(a^\infty)\\ &= \pi_k(\psi(M)/n).\end{aligned}$$

b) $M = M_i \unlhd \texttt{c0.dec} \unrhd M_j$. Assume $n = 0$. Then $M_j/0/m = \mathsf{D}$ and

$$\pi_k(\psi(M)/0) = [\pi_{k-2}(a^\infty) \unlhd a \unrhd \pi_{k-2}(a \circ \psi(M_j)/0)] \unlhd a \unrhd \pi_{k-1}(a^\infty).$$

By induction, $\pi_l(\mathcal{P}_m) \sqsubseteq \pi_l(\psi(M_j)/0)$ for $l < k$. As in case **a**), it follows that both for $m = 0$ and $m > 0$,

$$\begin{aligned}\pi_k(\mathcal{P}_m) &\sqsubseteq [\pi_{k-2}(a^\infty) \unlhd a \unrhd \pi_{k-2}(a \circ \psi(M_j)/0)] \unlhd a \unrhd \pi_{k-1}(a^\infty)\\ &= \pi_k(\psi(M)/n).\end{aligned}$$

The case $n > 0$ is proved similarly.

c) $M = M_i \unlhd \texttt{c1.inc} \unrhd M_j$. Then $M_i/n/m{+}1 = \mathsf{D}$ and

$$\pi_k(\psi(M)/n) = \pi_{k-1}(\psi(M_i)/n) \unlhd a \unrhd \pi_{k-1}(a^\infty).$$

By induction, $\pi_l(\mathcal{P}_{m+1}) \sqsubseteq \pi_l(\psi(M_i)/0)$ for $l < k$. Assume $m = 0$. Then

$$\begin{aligned}\pi_k(\mathcal{P}_0) &= \pi_{k-1}(\mathcal{P}_1) \unlhd a \unrhd \pi_{k-1}(\mathsf{D} \unlhd a \unrhd \mathcal{P}_0)\\ &\sqsubseteq \pi_{k-1}(\psi(M_i)/n) \unlhd a \unrhd \pi_{k-1}(a^\infty)\\ &= \pi_k(\psi(M)/n).\end{aligned}$$

Assume $m > 0$. Then

$$\begin{aligned}\pi_k(\mathcal{P}_m) &= \pi_{k-1}(\mathcal{P}_{m+1}) \unlhd a \unrhd \pi_{k-1}(\mathcal{P}_{m-1} \unlhd a \unrhd \mathsf{D})\\ &\sqsubseteq \pi_{k-1}(\psi(M_i)/n) \unlhd a \unrhd \pi_{k-1}(a^\infty)\\ &= \pi_k(\psi(M)/n).\end{aligned}$$

d) $M = M_i \unlhd \texttt{c1.dec} \unrhd M_j$. This case can be proved in a similar style, again making a case distinction between $m = 0$ and $m > 0$. □

In the proof above, the cases **a**) and **b**) clearly motivate ψ's definition on c0-terms: the simulation of a "$\texttt{c1.inc} \circ \texttt{c1.dec}$-prefix" generates in the case of divergence on C_0 the thread a^∞, which is needed to include $\mathcal{P}_m$. Lemma 2 immediately extends to the inclusion of infinite threads:

Corollary 2. *For all Minsky threads M and for all $n, m \in \mathbb{N}$,*

$$M/_{\mathtt{c0}}\, C_0(n)/_{\mathtt{c1}}\, C_1(m) = \mathsf{D} \Rightarrow \mathcal{P}_m \sqsubseteq \psi(M)/_{\mathtt{c0}}\, C_0(n).$$

Corollary 1 and Corollary 2 connect the halting problem for Minsky machines and inclusion between certain pushdown threads:

Corollary 3. *Let M be a Minsky thread. Then*

$$M/_{\mathsf{c0}}\, C_0(0)/_{\mathsf{c1}}\, C_1(0) = \mathsf{S} \Leftrightarrow \mathcal{P}_0 \not\sqsubseteq \psi(M)/_{\mathsf{co}}\, C_0(0).$$

Since counters can be seen as particular stacks, we can by Corollary 3 summarize the reduction from the halting problem for Minsky machines to

Theorem 4. *It is undecidable whether for a stack S and regular threads P and Q it holds that*

$$P/_{\mathsf{s}}\, S(\alpha) \sqsubseteq Q/_{\mathsf{s}}\, S(\beta).$$

5 Programming regular threads

Program Algebra [5], or PGA for short, is a program notation for regular threads. On PGA a hierarchy of program notations is founded, comprising languages that contain more and more sophisticated programming features, but that are all equally expressive. We will show that PGA exactly describes the class of regular threads.

Based on a finite set A of *basic instructions*, PGA has the binary operators "concatenation" and "repetition", and five kinds of instructions:

Concatenation, notation $\_;\_$. If X and Y are PGA-terms, so is $X;Y$.
Repetition, notation $(\_)^\omega$. If X is a PGA-term, so is X^ω.

Basic instruction $a \in A$. It is assumed that upon the execution of a basic instruction, the (executing) environment provides an answer `true` or `false`. However, in the case of a basic instruction, this answer is not used for program control. After execution of a basic instruction, the next instruction (if any) will be executed; if there is no next instruction, *inaction* (i.e., the thread D) will occur.

Positive test instruction $+a$ for $a \in A$. The instruction $+a$ executes like the basic instruction a. Upon `false`, the program skips its next instruction and continues with the instruction thereafter; upon `true` the program executes its next instruction. If there is no subsequent instruction to be executed, inaction occurs.

Negative test instruction $-a$ for $a \in A$. The instruction $-a$ executes like the basic instruction a. Upon `true`, the program skips its next instruction and continues with the instruction thereafter; upon `false` the program executes its next instruction. If there is no subsequent instruction to be executed, inaction occurs.

Termination instruction !. This instruction prescribes successful termination (it defines the thread S).

Jump instruction $\#k$ ($k \in \mathbb{N}$). This instruction prescribes execution of the program to jump k instructions forward; if there is no such instruction, inaction occurs. In the special case that $k = 0$, this prescribes a jump to the instruction itself and inaction occurs, in the case that $k = 1$ this jump acts as a *skip* and the next instruction is executed. In the case that the prescribed instruction is not available, inaction occurs.

Instruction sequence congruence for PGA-terms is axiomatized by the axioms PGA1-4 in Table 1. Here PGA2 is an axiom-scheme that is parametric in n where in PGA, $X^1 = X$ and $X^{k+1} = X; X^k$.

$(X;Y);Z = X;(Y;Z)$	(PGA1)	$X^\omega;Y = X^\omega$	(PGA3)
$(X^n)^\omega = X^\omega$ for $n > 0$	(PGA2)	$(X;Y)^\omega = X;(Y;X)^\omega$	(PGA4)

Table 1. Axioms for PGA's instruction sequence congruence

With the axioms PGA1-4 one easily derives *unfolding*, i.e., $X^\omega = X; X^\omega$.

Each closed PGA-term is considered a program of which the behaviour is a regular thread in BTA$^\infty$, viewing A as the set of actions. The *thread extraction* operator $|X|$ assigns a thread to a closed term X. Thread extraction is defined by the thirteen equations in Table 2, where $a \in A$ and u is an arbitrary instruction.

$\vert!\vert = \mathsf{S}$	$\vert!;X\vert = \mathsf{S}$	$\vert\#k\vert = \mathsf{D}$
$\vert a\vert = a \circ \mathsf{D}$	$\vert a;X\vert = a \circ \vert X\vert$	$\vert\#0;X\vert = \mathsf{D}$
$\vert+a\vert = a \circ \mathsf{D}$	$\vert+a;X\vert = \vert X\vert \trianglelefteq a \trianglerighteq \vert\#2;X\vert$	$\vert\#1;X\vert = \vert X\vert$
$\vert-a\vert = a \circ \mathsf{D}$	$\vert-a;X\vert = \vert\#2;X\vert \trianglelefteq a \trianglerighteq \vert X\vert$	$\vert\#k+2;u\vert = \mathsf{D}$
		$\vert\#k+2;u;X\vert = \vert\#k+1;X\vert$

Table 2. Equations for thread extraction on PGA

Two examples:

1. $|(\#0)^\omega| = |\#0;(\#0)^\omega| = \mathsf{D}$,
2. $$\begin{aligned} |-a;b;c| &= |\#2;b;c| \trianglelefteq a \trianglerighteq |b;c| \\ &= |\#1;c| \trianglelefteq a \trianglerighteq b \circ |c| \\ &= |c| \trianglelefteq a \trianglerighteq b \circ c \circ \mathsf{D} \\ &= c \circ \mathsf{D} \trianglelefteq a \trianglerighteq b \circ c \circ \mathsf{D}. \end{aligned}$$

In some cases, the equations in Table 2 can be applied from left to right yielding equations of the form $|X| = |Y| = ... = |X|$ and no behaviour, e.g.,

$$|(\#1)^\omega| = |\#1;(\#1)^\omega| = |(\#1)^\omega|,$$
$$|(\#2;a)^\omega| = |\#2;a;(\#2;a)^\omega| = |\#1;(\#2;a)^\omega| = |(\#2;a)^\omega|.$$

In such cases, the extracted thread is defined as D.

In all other cases, thread extraction yields a finite (linear) recursive specification, and thus a regular thread.

Example 5. Let $P = |(a; +b; \#3; -b; \#4)^\omega|$. Then

$$\begin{aligned} P &= a \circ Q \\ Q &= P \trianglelefteq b \trianglerighteq R \\ R &= P \trianglelefteq b \trianglerighteq R. \end{aligned}$$

From the above considerations it is clear that any PGA-program defines a regular thread. Conversely, each regular thread can be described as the thread extraction of a PGA term. We here only sketch why this is the case. First $\mathsf{S} = |!|$ and $\mathsf{D} = |\#0|$. Without loss of generality one can assume that every other thread P satisfies $\pi_1(P) = a \circ \mathsf{D}$ for some a, and solves X_1 in a finite linear recursive specification $X_1 = X_i \trianglelefteq a \trianglerighteq X_j$ in which S and D occur only in the last two equations. In Example 5, P is a fixed point for X_1 in the tailored specification

$$\begin{aligned} X_1 &= X_2 \trianglelefteq a \trianglerighteq X_2 \\ X_2 &= X_1 \trianglelefteq b \trianglerighteq X_3 \\ X_3 &= X_1 \trianglelefteq b \trianglerighteq X_3 \\ X_4 &= \mathsf{S} \\ X_5 &= \mathsf{D}. \end{aligned}$$

(Note that each fixed point for $X_1 - X_3$ does not refer to the equations for X_4 and X_5.) One can transform any such tailored specification to a program of the form $(u_1; ...; u_k)^\omega$ in the following way:

$$\begin{array}{rcl} X_1 = X_{1,l} \trianglelefteq a_1 \trianglerighteq X_{1,r} & \mapsto & (\ +a_1; \#c(1,l); \#c(1,r); \\ \vdots & & \vdots \\ X_i = X_{i,l} \trianglelefteq a_i \trianglerighteq X_{i,r} & & +a_i; \#c(i,l); \#c(i,r); \\ \vdots & & \vdots \\ X_n = \mathsf{S} & & !; \\ X_{n+1} = \mathsf{D} & & \#0)^\omega. \end{array}$$

If each value $c(i,j)$ is chosen such that the jump is to the position in the program that matches the transformation of $X_{i,l} = t_{i,l}(\overline{X})$, thread extraction yields the same recursive specification. Returning to Example 5,

$$\begin{array}{lcl} X_1 = X_2 \trianglelefteq a \trianglerighteq X_2 & \mapsto & (\ +a; \#2; \#1; \\ X_2 = X_1 \trianglelefteq b \trianglerighteq X_3 & & +b; \#7; \#1; \\ X_3 = X_1 \trianglelefteq b \trianglerighteq X_3 & & +b; \#4; \#9; \\ X_4 = \mathsf{S} & & !; \\ X_5 = \mathsf{D} & & \#0)^\omega. \end{array}$$

Clearly, $|(a; +b; \#3; -b; \#4)^\omega| = |(+a; \#2; \#1; +b; \#7; \#1; +b; \#4; \#9; !; \#0)^\omega|$.

6 Conclusions

Pushdown threads can be used in program algebra based semantics of sequential or object-oriented programming, for instance as described in [3]. In that approach, a single stack is used to store the arguments of a method call. Furthermore, pushdown threads are important for the theoretical foundation of program algebra itself, for instance admitting easy definitions of programming notations in which recursion can be expressed. This explains our interest in the decidability result proved in Section 3.

The undecidability of inclusion for pushdown threads is proved using a construction in which one of the counters is "weakly simulated". This method of Jančar is recorded first in 1994 [12], where it is used to prove various undecidability results for Petri nets. In 1999, Jančar et al. [13] used the same idea to prove the undecidability of simulation preorder for processes generated by one-counter machines, and this is most comparable to our approach. However, in the case of pushdown threads the inclusion relation itself is a little more complex than in process simulation or language theory because $\mathsf{D} \sqsubseteq P$ for any thread P. Moreover, threads have restricted branching, and therefore transforming a regular (control) thread into one that simulates one of the counters of a Minsky machine is more complex than in the related approaches referred to above. Also, the particular thread $\mathcal{P}_0$ used to prove our undecidability result (Corollary 3) has to be much more structured than the related nets/processes in [12, 13].

References

1. J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1/2):70-120, 1982.
2. J.A. Bergstra and I. Bethke. Polarized process algebra and program equivalence. In J.C.M. Baeten, J.K. Lenstra, J. Parrow, and G.J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4*, Springer-Verlag, LNCS 2719:1-21, 2003.
3. J.A. Bergsta, I. Bethke. Linear Projective Program Syntax. Logic Group Preprint Series 233, Utrecht University, Department of Philosophy, October 2004.
4. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60 (1/3):109-137, 1984.
5. J.A. Bergstra and M.E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125-156, 2002.
6. J.A. Bergstra and C.A. Middelburg. A thread algebra with multi-level strategic interleaving. In S.B. Cooper, B. Loewe and L. Torenvliet, editors, CiE 2005, Springer-Verlag, LNCS 3526:35-48, 2005.
7. J.A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming*, 51(2):175-192, 2002.
8. M. Bird. The equivalence problem for deterministic two-tape automata. *Journal of Computer and System Science*, 7(2):218-236, 1973.
9. S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143-148, 1995.
10. E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1:297-316, 1976.

11. S. Greibach. *Theory of Program Structures: Schemes, Semantics, Verification*, Springer-Verlag, 1975.
12. P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. *Proceedings of STACS94*, Springer-Verlag, LNCS 775:581-592, 1994.
13. P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. *Proceedings of SOFSEM'99: The 26th Seminar on Current Trends in Theory and Practice of Informatics*, Springer-Verlag, LNCS 1725:398-407, 1999.
14. T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Inclusion is undecidable for pattern languages. *ICALP 93*, Springer-Verlag, LNCS 700:301-312, 1993.
15. Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill, New-York, 1974.
16. M.L. Minsky. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437-455, 1961.
17. M.L. Minsky. *Computation: Finite and Infinite Machines*, Prentice-Hall International, Englewood Cliffs, New York, 1967.
18. E. Ohlebush and E. Ukkonen. On the equivalence problem for E-pattern languages. *Theoretical Computer Science*, 186(1/2):231-248, 1997.
19. G. Sénizergues. $L(A) = L(B)$? Technical report 1161-97, LaBRI, Université Bordeaux, 1997. Available at `www.labri.u-bordeaux.fr`.
20. G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. *Proceedings IEEE 39th FOCS*, 120-129, 1998.
21. G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, 251:1-166, 2001.
22. C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255:1-31, 2001.
23. L.G. Valiant. The equivalence problem for deterministic finite-turn pushdown automata. *Information and Control*, 25(2):123-133, 1974.
24. L.G. Valiant and M. Patterson. Deterministic one-counter automata. *Journal of Computer and System Science*, 10(3):340-350, 1975.

Electronic Reports Series of the Programming Research Group