Acta Informatica (2007) 44:75–90
DOI 10.1007/s00236-007-0040-5

ORIGINAL ARTICLE

Decision problems for pushdown threads

Jan A. Bergstra · Inge Bethke · Alban Ponse

Received: 22 March 2006 / Accepted: 23 January 2007 /
Published online: 22 March 2007

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

J. A. Bergstra · I. Bethke · A. Ponse (✉)
University of Amsterdam, Faculty of Science,
Programming Research Group, Amsterdam, The Netherlands
e-mail: alban@science.uva.nl
URL: http://www.science.uva.nl/∼alban

J. A. Bergstra
e-mail: janb@science.uva.nl
URL: http://www.science.uva.nl/∼janb

I. Bethke
e-mail: inge@science.uva.nl
URL: http://www.science.uva.nl/∼inge

J. A. Bergstra
Utrecht University, Department of Philosophy,
Applied Logic Group, Utrecht, The Netherlands

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 [9] 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 [20,21] 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 from programs in an appropriate program algebra. A *basic* thread models a finite program behaviour to be controlled 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 equivalence of pushdown threads is decidable, whereas pushdown thread inclusion is undecidable.

The paper is structured as follows: in Sect. 2, we outline the fundamental properties of thread algebra. In Sect. 3, we show that equivalence between pushdown threads is decidable by reducing this problem to the equivalence problem for deterministic pushdown automata [20–22]. In Sect. 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]. The paper is ended with some conclusions in Sect. 5.

2 Threads and services

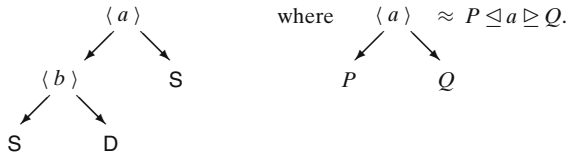
Basic thread algebra [6],[1] BTA, is a form of process algebra which is tailored for the description of sequential program behaviour. Based on a finite set of *actions* A, it has the following constants and operators:

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

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

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 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`.

Technically speaking, a basic thread can be viewed as a finite binary tree with leaves in $\{\mathsf{S}, \mathsf{D}\}$. As an example, we can depict the basic thread $(\mathsf{S} \trianglelefteq b \trianglerighteq \mathsf{D}) \trianglelefteq a \trianglerighteq \mathsf{S}$ in the following way:



The inclusion relation $\sqsubseteq$ on threads in BTA is the partial ordering generated by the clauses

1. For all $P \in \mathrm{BTA}$, $\mathsf{D} \sqsubseteq P$, and
2. For all $P_1, P_2, Q_1, Q_2 \in \mathrm{BTA}$, $a \in A$,

$$P_1 \sqsubseteq Q_1 \;\&\; P_2 \sqsubseteq Q_2 \Rightarrow (P_1 \trianglelefteq a \trianglerighteq P_2) \sqsubseteq (Q_1 \trianglelefteq a \trianglerighteq Q_2).$$

For example, $(\mathsf{S} \trianglelefteq b \trianglerighteq \mathsf{D}) \trianglelefteq a \trianglerighteq \mathsf{S}$ includes itself and six other threads.

Every thread in BTA is finite in the sense that there is a finite upperbound to the number of consecutive actions it can perform. The *approximation operator* $\pi : \mathbb{N} \times \mathrm{BTA} \to \mathrm{BTA}$ is determined by the equations

1. For all $P \in \mathrm{BTA}$, $\pi(0, P) = \mathsf{D}$,
2. For all $n \in \mathbb{N}$, $\pi(n+1, \mathsf{S}) = \mathsf{S}$,
3. For all $n \in \mathbb{N}$, $\pi(n+1, \mathsf{D}) = \mathsf{D}$, and
4. For all $P, Q \in \mathrm{BTA}, n \in \mathbb{N}$,

$$\pi(n+1, P \trianglelefteq a \trianglerighteq Q) = \pi(n, P) \trianglelefteq a \trianglerighteq \pi(n, Q).$$

We further write $\pi_n(P)$ instead of $\pi(n, P)$. The operator π finitely approximates every thread in BTA. That is, for all $P \in \mathrm{BTA}$,

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

Taking $P = (\mathsf{S} \trianglelefteq b \trianglerighteq \mathsf{D}) \trianglelefteq a \trianglerighteq \mathsf{S}$, we find $\pi_2(P) \neq \pi_3(P) = P$.

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,

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

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 $i_l, i_r \in I$. Before we provide some examples, we introduce *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 $a^{n+1} \circ P = a \circ (a^n \circ P)$. In the following example we consider some threads defined by a recursive specification.

Example 1 For $n \geq 0$ 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. $X_k = a \circ X_{k+1}$ for $1 \leq k \leq n$ and $X_{n+1} = \mathsf{D}$,
2. $X_k = a \circ X_{k+1}$ for $1 \leq k \leq n$ and $X_{n+1} = \mathsf{S}$,
3. $X_1 = a \circ X_1$, respectively.

Both $a^n \circ \mathsf{D}$ and $a^n \circ \mathsf{S}$ are finite threads. The thread a^∞ corresponds 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_1 in the finite linear specification $X_1 = a \circ X_1$. Furthermore, in a finite linear specification

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

(with $n \geq 1$) the threads P_i will sometimes be referred to as *states*.

Using these conventions, we can easily argue that $P \sqsubseteq Q$ is decidable for regular threads P and Q (of course, $\sqsubseteq$ is decidable for finite threads). Because one can always take the disjoint union of two recursive specifications it suffices to show that given (1) above, $P_i \sqsubseteq P_j$ is decidable. We prove this from the following assertion:

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

where $\pi_l(P_k)$ is defined by $\pi_l(t_k(\overline{P}))$. To prove (2), assume that $n > 1$ and $\pi_n(P_i) \sqsubseteq \pi_n(P_j)$ for certain i, j. Now 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, writing $a_{\texttt{true}}$ ($a_{\texttt{false}}$) for action a followed by reply `true` (`false`, respectively), there exists a trace of length k from P_i of the form

$$P_i \xrightarrow{a_{\texttt{true}}} P_{i'} \xrightarrow{b_{\texttt{false}}} \dots$$

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 traces 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.

Services model (part of) the execution environment of threads. A service 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 Sect. 3 and 4 are

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

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 finite threads P and Q, the defining rules for the *use operator* $/_{\texttt{c}}$, where $P/_{\texttt{c}}\, \mathcal{S}$ represents the thread P using the service $\mathcal{S}$ via channel c, are:

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

where $\mathcal{S}' = \langle \Sigma, F'\rangle$ with $F'(\sigma) = F(\texttt{a}\sigma)$ for all co-action sequences $\sigma \in \Sigma^+$. So, in a thread-service composition $P/_{\texttt{c}}\, \mathcal{S}$, the task of $\mathcal{S}$ is to support P in its execution: actions processed by $\mathcal{S}$ are solely used to control execution and are not observable (or 'external'). Upon S or D, the service is forgotten and so is the state it is in.

The use operator $/_{\texttt{c}}$ is expanded to infinite threads P by stipulating

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

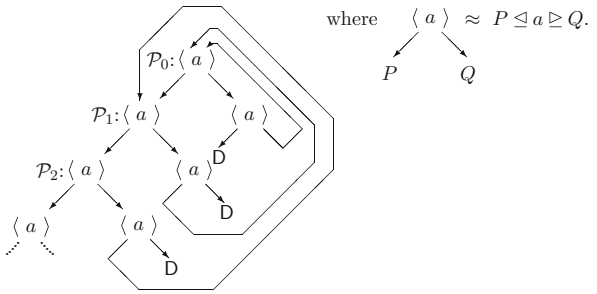


Fig. 1 The thread $\mathcal{P}_0$ as defined in Example 4

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

Thread-service composition was introduced in [7] (in that paper a service is called a *state machine*). In other papers on program and thread algebra (e.g., in [2,3,6,8,19]), channel and co-action are often referred to as *focus* and *method*, respectively.

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 $\mathtt{c.a}^n \circ \mathsf{D}$ (short for $(\mathtt{c.a})^n \circ \mathsf{D}$), $\mathtt{c.a}^n \circ \mathsf{S}$ and $\mathtt{c.a}^\infty$ for some channel $\mathtt{c}$ and some service $\mathcal{S} = \langle \Sigma, F\rangle$ with $\mathtt{a} \in \Sigma$. Then $(\mathtt{c.a}^n \circ \mathsf{D})/_{\mathtt{c}}\,\mathcal{S} = \mathsf{D}$ and therefore $\mathtt{c.a}^\infty/_{\mathtt{c}}\,\mathcal{S} = \mathsf{D}$, and $(\mathtt{c.a}^n \circ \mathsf{S})/_{\mathtt{c}}\,\mathcal{S} = \mathsf{S}$.

In the next example we show that the use of services may turn regular threads into non-regular ones. This example will play an important role in Sect. 4.

Example 4 Let $a \in A$. Consider the following regular thread P: [2]

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

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:

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

The thread $\mathcal{P}_0$ can be depicted as in Fig. 1.

Now assume that $\mathcal{P}_n \sqsubseteq \mathcal{P}_m$ for some $n \neq m$, say $n > m$. Then, by the recursive equations above,

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

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 $\mathcal{P}_n \not\sqsupseteq \mathcal{P}_m$. 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.

We finish this example with the observation that for all $n \in \mathbb{N}$, $\mathcal{P}_n \sqsubseteq a^\infty$ (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 Sect. 4.

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.

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

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). Then we exploit the decidability result establishing dpda-equivalence by Sénizergues in [20,21]. We base our proof on Stirling's formulation of this result in [22], as this is closer to the equivalence of pushdown threads. Henceforth we shall write ϵ for the empty sequence over any alphabet.

A *pushdown automaton* (pda) $\mathcal{A}$ is given by a finite set $\mathbb{P}$ 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 $P, Q \in \mathbb{P}$, $x \in \mathbb{S}$, $a \in \mathbb{A} \cup \{\epsilon\}$, and $\alpha \in \mathbb{S}^*$. 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 $\{w \in \mathbb{A}^* \mid \exists Q \in \mathbb{P}.P\alpha \xrightarrow{w} Q\}$ 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. Also note that $L(\mathcal{A}, P\epsilon) = \emptyset$.

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

- if $Px \xrightarrow{a} Q\alpha$ and $Px \xrightarrow{a} R\beta$ for $a \in \mathbb{A} \cup \{\epsilon\}$, then $Q = R$ and $\alpha = \beta$,
- if $Px \xrightarrow{\epsilon} Q\alpha$ and $Px \xrightarrow{a} R\lambda$ then $a = \epsilon$ (and $Q = R$ and $\alpha = \lambda$).

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 $L(\mathcal{A}', P\alpha) = L(\mathcal{A}', Q\beta)$, as was proved in [22]. With this decidability result we can easily prove the main result of this section: we only have to show how to deal with an initially empty stack, with a non-empty stack upon termination (S), and with both forms of divergence, i.e. "infinite words" (an infinite trace of external actions) and deadlock (D).

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

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

where α, β represent the contents of S.

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

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

for some $\texttt{n} \geq 1$. We write $S(\alpha)$ if the stack contains the elements from sequence $\alpha \in \{1, \ldots, \texttt{n}\}^*$ with the leftmost element of α on top. Furthermore, s.push:i pushes i onto the stack and yields reply true, the action 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, assume without loss of generality that both P and Q are defined by a single finite linear specification. Using four transformations, we reduce our question $P/_{\mathrm{s}}\, S(\alpha) = Q/_{\mathrm{s}}\, S(\beta)$ to the the dpda-equivalence problem as discussed above.

Adapting the stack contents. In order to use the dpda-equivalence result in [22], the stack should be non-empty at the start and empty upon termination (S) because language acceptance is by empty stack and starts from configurations with non-empty stack. This can be achieved as follows:

- Let S_0 be the stack over $\{0, 1, \ldots, n\}$ (the stack symbol 0 will be used as an empty stack marker).
- In the specification of P and Q,
  1. Replace each equation $R = R_l \trianglelefteq \mathtt{s.empty} \trianglerighteq R_r$ by $R = R_l \trianglelefteq \mathtt{s.topeq{:}0} \trianglerighteq R_r$,
  2. Replace each equation $R = R_l \trianglelefteq \mathtt{s.pop} \trianglerighteq R_r$ by $R = R_r \trianglelefteq \mathtt{s.topeq{:}0} \trianglerighteq \mathtt{s.pop} \circ R_l$,
  3. Replace each equation $R = \mathsf{S}$ by $R = \mathtt{s.pop} \circ \overline{\mathsf{S}}$ (where $\overline{\mathsf{S}}$ is fresh), and
  4. Add the equation $\overline{\mathsf{S}} = \overline{\mathsf{S}} \trianglelefteq \mathtt{s.pop} \trianglerighteq \mathsf{S}$.

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 $\alpha, \beta \in \{1, \ldots, n\}^*$,

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

and that upon S, 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 $\alpha, \beta \in \{1, \ldots, n\}^*$,

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

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 $R = R_l \trianglelefteq a \trianglerighteq R_r$ with a an external action by $R = \overline{\mathsf{S}} \trianglelefteq \mathit{halt} \trianglerighteq (R_l \trianglelefteq a \trianglerighteq R_r)$. Call the resulting threads P_2 and Q_2, respectively. Again it is straightforward that for $\alpha, \beta \in \{1, \ldots, n\}^*$,

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

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

Transformation to dpda-equivalence. From the linearized specification of P_2 and Q_2, construct a pda $\mathcal{A}$ as follows: for $\mathbb{P}$, the set of states, take those of the linear specification; for $\mathbb{S}$, the set of stack symbols, take $\{0, \ldots, n\}$; 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 equation $R = R_l \trianglelefteq a \trianglerighteq R_r$ with a an external action and $i \in \{0, \ldots, n\}$, define transitions $Ri \xrightarrow{a_{\mathtt{true}}} R_l i$ and $Ri \xrightarrow{a_{\mathtt{false}}} R_r i$,

- for each equation $R = R_l \trianglelefteq \mathtt{s.push{:}j} \trianglerighteq R_r$ and $i \in \{\mathtt{1},\ldots,\mathtt{n}\}$, define transitions $Ri \xrightarrow{\epsilon} R_l ji$,
- for each equation $R = R_l \trianglelefteq \mathtt{s.topeq{:}j} \trianglerighteq R_r$ and $i \in \{\mathtt{0},\mathtt{1},\ldots,\mathtt{n}\} \setminus \{j\}$, define transitions $Ri \xrightarrow{\epsilon} R_r i$, and define a transition $Rj \xrightarrow{\epsilon} R_l j$,
- for each equation $R = R_l \trianglelefteq \mathtt{s.pop} \trianglerighteq R_r$ and $i \in \{\mathtt{0},\ldots,\mathtt{n}\}$, define transitions $Ri \xrightarrow{\epsilon} R_l$.

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

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

Finally, observe that $\mathcal{A}$'s transition relation is deterministic. Therefore $\mathcal{A}$ is a dpda and we are done. □

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, albeit a bit cumbersome. The action $\mathtt{s.pop}$ is easier to define, and $\mathtt{s.empty}$ 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 $\{\mathtt{0},\mathtt{1}\}$ 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 $\mathtt{swap}$ that exchanges the two top-elements of $S_{\mathbb{N}}$ is not definable because with this action the functionality of a Minsky machine (discussed in Sect. 4) is obtained: use the bottom stack position to hold the value of the first counter, and the top position for the value of the second counter. So with $\mathtt{swap}$ equality is not decidable.

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 $\mathtt{1}$ to some $\mathtt{L}$. It starts from the instruction labelled $\mathtt{1}$, halts if $\mathtt{stop}$ 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. $\mathtt{l : c_i := c_i + 1; goto\ l'}$
2. $\mathtt{l : if\ c_i = 0\ then\ goto\ l'\ else\ c_i := c_i - 1; goto\ l''}$
3. $\mathtt{l : stop}$

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 (Minsky [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 $\langle \Sigma, F\rangle$ as defined in Example 2.1, i.e., 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. We write $C(n)$ for a counter C with value n. A Minsky machine canonically defines the equations of a specification of a regular thread:

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

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. The halting problem for Minsky machines can now be rephrased as follows:

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 $E = \{M_n = t_n(\overline{M}) \mid n = 1,\ldots,k\}$ (for some $k > 0$) and let a be some action. Define $\theta(E) = \{\theta(M_n) = \theta(t_n(\overline{M})) \mid n = 1,\ldots,k\}$ and define $\theta(t_n(\overline{M}))$ by

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

$$\theta(M_l \trianglelefteq \mathtt{ci.b} \trianglerighteq M_r) = a \circ \theta(M_l) \trianglelefteq \mathtt{ci.b} \trianglerighteq a \circ \theta(M_r).$$

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 $k > 0$,

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

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 external actions a. This transformation is such that the behaviour of a Minsky thread M is preserved in the following sense:

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

where $\mathcal{P}_0$ is the pushdown thread defined in Example 4. The reply choices to the a-actions in our simulated thread lead the way to the distinguishing properties $\mathsf{D} \sqsubseteq \mathsf{S}$, $\mathsf{D} \sqsubseteq a^\infty$, and $\mathcal{P}_0 \not\sqsubseteq \mathsf{S}$, $\mathcal{P}_0 \sqsubseteq a^\infty$. A small technical complication of our transformation is that each C_0-action is 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 in the original Minsky thread (as in $\mathtt{c0.inc}^\infty$) then also yields an infinite sequence of simulating 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 $E = \{M_n = t_n(\overline{M}) \mid n = 1,\ldots,k\}$ (for some $k > 0$). Define $\psi(E) = \{\psi(M_n) = \psi(t_n(\overline{M})) \mid n = 1,\ldots,k\}$ and define $\psi(t_n(\overline{M}))$ by

1. $\psi(\mathsf{S}) = \mathsf{S}$,
2. $\psi(M_i \trianglelefteq \mathtt{c1.inc} \trianglerighteq M_j) = \psi(M_i) \trianglelefteq a \trianglerighteq a^\infty$,
3. $\psi(M_i \trianglelefteq \mathtt{c1.dec} \trianglerighteq M_j) = a^\infty \trianglelefteq a \trianglerighteq (\psi(M_i) \trianglelefteq a \trianglerighteq \psi(M_j))$, and
4. $\psi(M_i \trianglelefteq \mathtt{c0.b} \trianglerighteq M_j) = [a^\infty \trianglelefteq a \trianglerighteq a \circ (\psi(M_i) \trianglelefteq \mathtt{c0.b} \trianglerighteq \psi(M_j))] \trianglelefteq a \trianglerighteq a^\infty$ 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

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

This definition yields the following infinite recursive specification:

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

Note that for all n, $\mathcal{P}_n \sqsubseteq a^\infty$ and $\mathcal{P}_n \not\sqsubseteq \mathsf{S}$ (see Example 4). 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 $m > 0$,

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

As stated before, we shall prove that for any Minsky thread M,

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

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

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

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) $M = M_i \trianglelefteq \mathtt{c1.dec} \trianglerighteq M_j$. 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 the following corollary:

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

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

The next result is about the inclusion of finite approximations.

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

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

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 $\mathtt{c0}$-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) $M = M_i \trianglelefteq \mathtt{c0.inc} \trianglerighteq M_j$. 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)

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

(b) $M = M_i \trianglelefteq \mathtt{c0.dec} \trianglerighteq M_j$. Assume $n = 0$. Then $M_j/0/m = \mathsf{D}$ and

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

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 $m > 0$,

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

The case $n > 0$ is proved similarly.

(c) $M = M_i \trianglelefteq \mathtt{c1.inc} \trianglerighteq M_j$. Then $M_i/n/m{+}1 = \mathsf{D}$ and

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

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

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

Assume $m > 0$. Then

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

(d) $M = M_i \trianglelefteq \mathtt{c1.dec} \trianglerighteq M_j$. 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 $\mathtt{c0}$-terms: the simulation of a "$\mathtt{c1.inc} \circ \mathtt{c1.dec}$-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*

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

Since counters can be seen as particular stacks, we can by Corollary 3 summarize the reduction from the halting problem for Minsky machines and threads (Theorems 2 and 3) to the main result of this section:

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

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

5 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 Sect. 3.

The undecidability of inclusion for pushdown threads is proved using a particular reduction of the halting problem for Minsky machines 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.[3] However, in our case 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) is more sequentially structured than the related nets/processes in [12,13].

[3] Indeed, distinguishing the class of *one-counter threads*, i.e., regular threads using a single counter, as a proper subclass of the pushdown threads, we have shown the slightly stronger result that inclusion for one-counter threads is undecidable. More results on one-counter threads can be found in [19].

We end this paper with a few remarks on the motivation for the use of our threads. We started off by stating that "threads as contained in a thread algebra emerge from the behavioral abstraction from programs in an appropriate program algebra". This is the source of our use of threads and merits some further elaboration. In [5], *program algebra* was introduced as an approach that connects various sequential programming notations, distinguishing the simple program algebra PGA as a basis. The crucial property of a PGA-program is that it represents nothing more than a (finite or infinite) sequence of primitive instructions. More advanced programming notations (e.g. comprising *goto*'s, *conditionals* or *while*'s) are defined on top of PGA. Any program in such an advanced notation is given semantics by a projection to PGA. The execution (or 'behavioral semantics') of PGA-programs is defined by so-called thread extraction: each PGA-program defines a regular thread (in BTA^{∞}), and conversely, each regular thread is specifiable as a PGA-program. Furthermore, a pushdown thread can be considered as resulting from an 'architecture' that governs the interaction between a program (text) that defines the regular control, and a stack service in some initial state that supports execution. The so-called analytic execution architecture then specifies their interaction upon execution. In [8] we study such execution architectures for program algebra and come up with some basic results about, amongst other things, the modeling of Turing machines and rational agents that support optimal behaviour.

Acknowledgments We thank two reviewers for their valuable remarks and constructive suggestions. We thank the organizers of the *Logic Colloquium 2005* in Athens for providing the opportunity to present this paper.

References

1. de Bakker, J.W., Zucker, J.I.: Processes and the denotational semantics of concurrency. Information Control **54**(1/2), 70–120 (1982)
2. Bergstra, J.A., Bethke, I.: Polarized process algebra and program equivalence. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) Automata, Languages and Programming, 30th International Colloquium, ICALP, LNCS, vol. 2719, pp. 1–21. Springer, Eindhoven, 30 June–4 July 2003)
3. Bergstra, J.A., Bethke, I.: Predictable and reliable program code: virtual machine-based projection semantics. In: Electronic Report PRG0603, Programming Research Group, University of Amsterdam. Submitted for inclusion in the Handbook of Network and Systems Administration (2006)
4. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information Control **60**(1/3), 109–137 (1984)
5. Bergstra, J.A., Loots, M.E.: Program algebra for sequential code. J. Logic Algebraic Program. **51**(2), 125–156 (2002)
6. Bergstra, J.A., Middelburg, C.A.: A thread algebra with multi-level strategic interleaving. In: Cooper, S.B., Loewe, B., Torenvliet, L. (eds.) CiE 2005, Springer, LNCS, vol. 3526, pp. 35–48 (2005)
7. Bergstra, J.A., Ponse, A.: Combining programs and state machines. J. Logic Algebraic Program. **51**(2), 175–192 (2002)
8. Bergstra, J.A., Ponse, A.: Execution architectures for program algebra. J. Appl. Logic **5**(1), 170–192 (2007)
9. Bird, M.: The equivalence problem for deterministic two-tape automata. J. Comput. Syst. Sci. **7**(2), 218–236 (1973)
10. Friedman, E.P.: The inclusion problem for simple languages. Theoret. Comput. Sci. **1**, 297–316 (1976)
11. Greibach, S.: Theory of Program Structures: Schemes, Semantics, Verification, Springer, Heidelberg (1975)

12. Jančar, P.: Decidability questions for bisimilarity of Petri nets and some related problems. In: Proceedings of STACS94, LNCS, vol. 775, pp. 581–592. Springer, Heidelberg (1994)
13. Jančar, P., Moller, F., Sawa, Z.: Simulation problems for one-counter machines. Proceedings of SOFSEM'99: The 26th Seminar on Current Trends in Theory and Practice of Informatics, LNCS, vol. 1725, pp. 398–407. Springer, Heidelberg (1999)
14. Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Inclusion is undecidable for pattern languages. ICALP 93 LNCS, vol. 700, pp. 301–312. Springer, Heidelberg (1993)
15. Manna, Z.: Mathematical Theory of Computation. McGraw-Hill, New York (1974)
16. Minsky, M.L.: Recursive unsolvability of post's problem of "Tag" and other topics in theory of Turing machines. Ann. Math. **74**(3), 437–455 (1961)
17. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall International, Englewood Cliffs (1967)
18. Ohlebush, E., Ukkonen, E.: On the equivalence problem for E-pattern languages. Theoret. Comput. Sci. **186**(1/2), 231–248 (1997)
19. Ponse, A., van der Zwaag, M.B.: Risk assessment for one-counter threads. In: Electronic report PRG0608, Programming Research Group, University of Amsterdam. To appear in Theory of Computing Systems (2006)
20. Sénizergues, G.: $L(A) = L(B)$? In: Technical report 1161-97, LaBRI, Université Bordeaux (1997) Available at www.labri.u-bordeaux.fr.
21. Sénizergues, G.: L(A)=L(B)? decidability results from complete formal systems. Theoret. Comput. Sci. **251**, 1–166 (2001)
22. Stirling, C.: Decidability of DPDA equivalence. Theoret. Comput. Sci. **255**, 1–31 (2001)
23. Valiant, L.G.: The equivalence problem for deterministic finite-turn pushdown automata. Information Control **25**(2), 123–133 (1974)
24. Valiant, L.G., Patterson, M.: Deterministic one-counter automata. J. Comput. Syst. Sci. **10**(3), 340–350 (1975)