Risk assessment for one-counter threads

Ponse, A.; van der Zwaag, M.B.

DOI
10.1007/s00224-007-9034-5
Publication date
2008

Published in
Theory of Computing Systems

Theory Comput Syst (2008) 43: 563–582
DOI 10.1007/s00224-007-9034-5

Risk Assessment for One-Counter Threads

Alban Ponse · Mark B. van der Zwaag

Published online: 26 July 2007

Abstract Threads as contained in a thread algebra are used for the modeling of sequential program behavior. A thread that may use a counter to control its execution is called a 'one-counter thread'. In this paper the decidability of risk assessment (a certain form of action forecasting) for one-counter threads is proved. This relates to Cohen's impossibility result on virus detection (Comput. Secur. 6(1), 22–35, 1984). Our decidability result follows from a general property of the traces of one-counter threads: if a state is reachable from some initial state, then it is also reachable along a path in which all counter values stay below a fixed bound that depends only on the initial and final counter value. A further consequence is that the reachability of a state is decidable. These properties are based on a result for ω-one counter machines by Rosier and Yen (SIAM J. Comput. 16(5), 779–807, 1987).

Keywords One-counter systems · Thread algebra · Reachability · Risk assessment

1 Introduction

Threads as contained in a thread algebra are used for the modeling of sequential program behavior. Threads may make use of services such as a counter or a stack to control their execution. A regular thread using a counter is called a 'one-counter thread'. One-counter threads form a proper subclass of the *pushdown threads* as studied in [6], i.e. regular threads that may use a stack over a finite sort. In that paper it is shown that equality of pushdown threads is decidable, while their inclusion is not. The latter undecidability result even holds for one-counter threads (of course, a counter is similar to a stack over a one-element data type).

A. Ponse · M.B. van der Zwaag (✉)
Programming Research Group, Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
e-mail: mbz@science.uva.nl

In this paper the decidability of a certain form of risk assessment for one-counter threads is proved. We assume that risks are modeled by the execution of an action *risk*, and the question is to forecast the execution of *risk* given the specification of the thread under execution. For pushdown threads this type of forecasting is decidable: rename the action(s) to be forecast and decide whether the thread thus obtained equals the original one. Forecasting becomes much more complicated if a thread may contain test actions that yield a reply according to the result of this type of forecasting. For example, assume that a thread may contain a test action *ok* that yields true if its true-branch does not execute *risk*, and false otherwise. Then a current test action *ok* in the code to be inspected may yield true because a future one will yield false. The reply to these *ok* actions can be modeled using a *risk assessment* service. For regular threads, such a service has a decidable reply function [7], while for pushdown threads this is still an open question. In this paper we show that the associated service is decidable for one-counter threads, thereby answering one of the open questions in [7].

Our modeling of risk assessment—that is, the existence of a test action *ok* and a risk assessment service as described above—was introduced in [5]. This modeling was inspired by Cohen's well-known impossibility result on virus detection [8], first described in 1984. Essentially, Cohen's result establishes the non-existence of a test that decides whether or not a code fragment is virus-free. This point will be elaborated on in our conclusions.

In technical terms the decidability result of the present paper is based on a reachability result for so-called ω-one counter machines, proved by Rosier and Yen [13] in 1987. This result implies a general property of the traces of one-counter threads: if a state is reachable from some initial state, then it is also reachable along a path in which all counter values stay below a fixed bound that depends only on the initial and final counter value.

The paper is structured as follows. In the next section we introduce the basics of thread algebra and services. Then, in Sect. 3, we discuss one-counter threads and the above-mentioned reachability result. In Sect. 4 we recall that risk assessment for regular threads is decidable and come up with a new short proof of this fact. Furthermore, we use our proof technique to show the decidability of risk assessment for one-counter threads. In Sect. 5 we prove some further reachability results for one-counter threads. Finally, in Sect. 6 we end with some conclusions.

2 Threads and Services

We introduce the basics of thread algebra and thread-service composition. For a general introduction we refer to [12].

2.1 Basic Thread Algebra

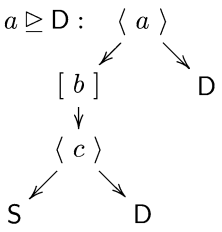
Basic Thread Algebra (BTA) is a form of process algebra which is tailored to the description of sequential program behavior. Based on a set A of *actions*, it has the following constants and operators:

- the *termination* constant S,
- the *deadlock* or *inaction* constant D,
- for each $a \in A$, a binary *post-conditional 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 $n \geq 1$ we define $a^n \circ P$ by $a^1 \circ P = a \circ P$ and $a^{n+1} \circ P = a \circ (a^n \circ P)$.

The operational intuition is that each action represents a command which is to be processed by the execution environment of the thread. The processing of a command may involve a change of state of this environment.[1] At completion of the processing of the command, the environment produces a reply value `true` or `false`. The thread $P \trianglelefteq a \trianglerighteq Q$ proceeds as P if the processing of a yields `true`, and it proceeds as Q if the processing of a yields `false`. Note that a single action is not a thread, and that threads as introduced here always "end" in S or D. We can depict threads as in the following example:

$b \circ (\mathsf{S} \trianglelefteq c \trianglerighteq \mathsf{D}) \trianglelefteq a \trianglerighteq \mathsf{D}$:



Here, an action symbol between angular brackets represents the execution of the action, with an arrow descending to the left leading to the subsequent execution in case the reply value for the action is `true`, and an arrow descending to the right leading to the subsequent execution in case the reply value for the action is `false`. In the case that these lead to the same state (action prefix) the descent is vertical, and the action symbol is placed between square brackets.

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. The *approximation operator* $\pi : \mathbb{N} \times \mathrm{BTA} \to \mathrm{BTA}$ gives the behavior up to a specified depth. It is defined by

1. $\pi(0, P) = \mathsf{D}$,
2. $\pi(n+1, \mathsf{S}) = \mathsf{S}$, $\pi(n+1, \mathsf{D}) = \mathsf{D}$,
3. $\pi(n+1, P \trianglelefteq a \trianglerighteq Q) = \pi(n, P) \trianglelefteq a \trianglerighteq \pi(n, Q)$,

for $P, Q \in \mathrm{BTA}$ and $n \in \mathbb{N}$. We further write $\pi_n(P)$ instead of $\pi(n, P)$. We find that for every $P \in \mathrm{BTA}$, there exists an $n \in \mathbb{N}$ such that

$$\pi_n(P) = \pi_{n+1}(P) = \cdots = P.$$

2.2 Infinite Threads

Following the metric theory of [9] in the form developed as a model for the process algebra ACP in [3], BTA has a completion BTA^∞ which comprises also the in-

[1] For the definition of threads we completely abstract from the environment. In Sect. 2.3 we define services which model (part of) the environment, and thread-service composition.

finite threads. Standard properties of the completion technique yield that we may take BTA^∞ as the cpo consisting of all so-called *projective* sequences of finite threads:[2]

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

We give an exposition of this construction containing all definitions required for this paper; for a detailed account and further motivation we refer to [1, 14].

Overloading notation, we define the constants and operators of BTA on BTA^∞:

1. $\mathsf{D} = (\mathsf{D}, \mathsf{D}, \ldots)$ and $\mathsf{S} = (\mathsf{D}, \mathsf{S}, \mathsf{S}, \ldots)$,
2. $(P_n)_{n\in\mathbb{N}} \trianglelefteq a \trianglerighteq (Q_n)_{n\in\mathbb{N}} = (R_n)_{n\in\mathbb{N}}$ with $R_0 = \mathsf{D}$ and $R_{n+1} = P_n \trianglelefteq a \trianglerighteq Q_n$.

The elements of BTA are included in BTA^∞ by a mapping following this definition:

$$\mathsf{D} \mapsto (\mathsf{D}, \mathsf{D}, \mathsf{D}, \ldots), \qquad \mathsf{S} \mapsto (\mathsf{D}, \mathsf{S}, \mathsf{S}, \ldots),$$

and if $P \mapsto (P_n)_{n\in\mathbb{N}}$ and $Q \mapsto (Q_n)_{n\in\mathbb{N}}$, then

$$P \trianglelefteq a \trianglerighteq Q \mapsto (R_n)_{n\in\mathbb{N}}$$

with $R_0 = \mathsf{D}$ and $R_{n+1} = P_n \trianglelefteq a \trianglerighteq Q_n$.

It is not difficult to show that the projective sequence of $P \in \mathrm{BTA}$ defined thus equals $(\pi_n(P))_{n\in\mathbb{N}}$. We further use this inclusion of finite threads in BTA^∞ implicitly and write $P, Q, \ldots$ to denote elements of BTA^∞.

We define the set $Res(P)$ of *residual threads* of P inductively as follows:

1. $P \in Res(P)$,
2. $Q \trianglelefteq a \trianglerighteq R \in Res(P)$ implies $Q \in Res(P)$ and $R \in Res(P)$.

A residual thread may be reached (depending on the execution environment) by performing zero or more actions. A thread P is *regular* if $Res(P)$ is finite.

A *linear specification* over BTA^∞ is a set

$$\{x = t_x \mid x \in Var\}$$

of recursive equations for a set Var of variables, where all t_x are terms of the form S, D, or $y \trianglelefteq a \trianglerighteq z$ with $y, z \in Var$. Finite linear specifications represent continuous operators; each variable has a unique fixed point as solution [14].

Proposition 1 *For all $P \in \mathrm{BTA}^\infty$, P is regular if and only if P is the solution of a finite linear specification.*

Proof For the if-part: Assume that P is the solution of a finite linear recursive specification. Because the variables in a finite linear specification have unique fixed points, we know that there are threads $P_1, \ldots, P_n \in \mathrm{BTA}^\infty$ with $P = P_1$, and for every

[2]The cpo is based on the partial ordering $\sqsubseteq$ defined by $\mathsf{D} \sqsubseteq P$, and $P \sqsubseteq P'$, $Q \sqsubseteq Q'$ implies $P \trianglelefteq a \trianglerighteq Q \sqsubseteq P' \trianglelefteq a \trianglerighteq Q'$.

$i \in \{1,\ldots,n\}$, either $P_i = \mathsf{D}$, $P_i = \mathsf{S}$, or $P_i = P_j \trianglelefteq a \trianglerighteq P_k$ for some $j,k \in \{1,\ldots,n\}$. We find that $Q \in Res(P)$ if and only if $Q = P_i$ for some $i \in \{1,\ldots,n\}$. So $Res(P)$ is finite, and P is regular.

For the only-if-part: Suppose P is regular. Then $Res(P)$ is finite, so P has residual threads $P_1,\ldots,P_n$ with $P = P_1$. We construct a linear specification with variables $x_1,\ldots,x_n$ as follows:

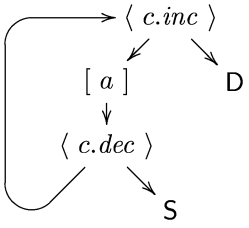
$$x_i = \begin{cases} \mathsf{D} & \text{if } P_i = \mathsf{D}, \\ \mathsf{S} & \text{if } P_i = \mathsf{S}, \\ x_j \trianglelefteq a \trianglerighteq x_k & \text{if } P_i = P_j \trianglelefteq a \trianglerighteq P_k. \end{cases}$$ □

We shall further describe regular threads by finite linear specifications. Moreover, in reasoning with such specifications, we shall identify variables and their fixed points. For example, we say that x *is* the thread defined by $x = a \circ x$ instead of stating that some thread P equals the fixed point for x in the finite linear specification $x = a \circ x$.

Example 1 Consider the finite linear specification defined by the following five equations:

$$\begin{aligned} x_1 &= x_2 \trianglelefteq c.inc \trianglerighteq x_4, \\ x_2 &= a \circ x_3, \\ x_3 &= x_1 \trianglelefteq c.dec \trianglerighteq x_5, \\ x_4 &= \mathsf{D}, \\ x_5 &= \mathsf{S}. \end{aligned}$$

The actions $c.inc$ and $c.dec$ are requests to a counter service as will be explained below; we come back to this specification in Example 2. The regular thread x_1 can be depicted as follows (cf. the picture in Sect. 2.1):



Here the state with the action $c.inc$ represents the regular thread x_1, and the state with the action a represents the regular thread x_2, etc.

Observe that each finite linear specification can be depicted as a finite flowchart (in the style as displayed above) in which each state represents a regular thread.

2.3 Services and Thread-Service Composition

A thread $P \trianglelefteq a \trianglerighteq Q$ is non-deterministic in the sense that upon execution of the action a, both P and Q are possible subsequent behaviors. Which one is executed depends on the reply value for the action that is returned by the execution environment, and in general we do not know what this reply will be. We now define the composition of a thread and a *service* that models a part of the environment of the thread. In order to define the interaction between threads and services, we let all actions be of the form $f.m$ where f is the so-called *focus* and m is the *method*. Furthermore, let $focus(\_)$ be the mapping that extracts the focus of an action:

$$focus(f.m) = f.$$

We interpret an action $f.m$ as a request to a service that is known to the thread as f. A service answering to such requests is defined as follows.

A *service* is a pair $\mathcal{H} = \langle M, F\rangle$ consisting of a set M of methods and a reply function F that gives for each non-empty finite sequence of methods from M a reply value `true` or `false`. These sequences represent the *history* of the service: on input $v = m_1 \ldots m_{k+1}$, for some $k \geq 0$, the function F gives the reply value for method m_{k+1} if $m_1, \ldots, m_k$ were called before. We shall write $\mathcal{H}_v$ for service $\mathcal{H}$ with history v, and $\mathcal{H}$ for the service $\mathcal{H}_\epsilon$ where ϵ is the empty sequence, so this is the service in its initial state.

For a service $\mathcal{H} = \langle M, F\rangle$ and a thread P, $P \mathbin{/_f} \mathcal{H}_v$ represents P using the service $\mathcal{H}_v$ via focus f. The defining rules for threads in BTA are:

$$\begin{aligned}
\mathsf{S} \mathbin{/_f} \mathcal{H}_v &= \mathsf{S},\\
\mathsf{D} \mathbin{/_f} \mathcal{H}_v &= \mathsf{D},\\
(P \trianglelefteq g.m \trianglerighteq Q) \mathbin{/_f} \mathcal{H}_v &= (P \mathbin{/_f} \mathcal{H}_v) \trianglelefteq g.m \trianglerighteq (Q \mathbin{/_f} \mathcal{H}_v) \quad \text{if } g \neq f,\\
(P \trianglelefteq f.m \trianglerighteq Q) \mathbin{/_f} \mathcal{H}_v &= P \mathbin{/_f} \mathcal{H}_{vm} \quad \text{if } m \in M \text{ and } F(vm) = \texttt{true},\\
(P \trianglelefteq f.m \trianglerighteq Q) \mathbin{/_f} \mathcal{H}_v &= Q \mathbin{/_f} \mathcal{H}_{vm} \quad \text{if } m \in M \text{ and } F(vm) = \texttt{false},\\
(P \trianglelefteq f.m \trianglerighteq Q) \mathbin{/_f} \mathcal{H}_v &= \mathsf{D} \quad \text{if } m \notin M.
\end{aligned}$$

The operator $/_f$ is called the *use operator* and stems from [4]. An expression $P \mathbin{/_f} \mathcal{H}_v$ is sometimes referred to as a *thread-service composition*. The use operator is expanded to infinite threads in BTA^∞ by defining

$$(P_n)_{n\in\mathbb{N}} \mathbin{/_f} \mathcal{H}_v = \bigsqcup_{n\in\mathbb{N}} P_n \mathbin{/_f} \mathcal{H}_v.$$

(Cf. [2].) It follows that the rules for finite threads are valid for infinite threads as well. Observe that the requests to the service do not occur as actions in the behavior of a thread-service composition. So the composition not only reduces the above-mentioned non-determinism of the thread, but also hides the associated actions.

A (natural number) *counter* $\mathcal{C}$ can be defined as a service with methods *inc* and *dec*, where *inc* increases the counter value and yields reply `true`, and *dec* decreases

the value with reply `true` if the current value is larger than 0, and yields `false` otherwise (while the counter value remains 0). It is sufficient and more convenient to represent the state of $\mathcal{C}$ by its number value rather than as a method history, so we write $\mathcal{C}(n)$ for a counter service $\mathcal{C}$ holding value $n \geq 0$. Troughout this paper we assume that threads use focus c to address a counter service, and that *inc* and *dec* are the only requests made to it. By the defining equations for thread-service composition we find that, for any thread x,

$$(c.inc \circ x) \,/_c\, \mathcal{C}(n) = x \,/_c\, \mathcal{C}(n+1),$$

and

$$(x \trianglelefteq c.dec \trianglerighteq y) \,/_c\, \mathcal{C}(n) = \begin{cases} y \,/_c\, \mathcal{C}(0) & \text{if } n = 0, \\ x \,/_c\, \mathcal{C}(n-1) & \text{if } n \geq 1. \end{cases}$$

Example 2 Consider again the thread x_1 defined in Example 1 by

$$\begin{aligned} x_1 &= x_2 \trianglelefteq c.inc \trianglerighteq x_4, \\ x_2 &= a \circ x_3, \\ x_3 &= x_1 \trianglelefteq c.dec \trianglerighteq x_5, \\ x_4 &= \mathsf{D}, \\ x_5 &= \mathsf{S}, \end{aligned}$$

and assume that $focus(a) \neq c$. For any $n \in \mathbb{N}$, we find that

$$\begin{aligned} x_1 \,/_c\, \mathcal{C}(n) &= x_2 \,/_c\, \mathcal{C}(n+1) \\ &= a \circ (x_3 \,/_c\, \mathcal{C}(n+1)) \\ &= a \circ (x_1 \,/_c\, \mathcal{C}(n)). \end{aligned}$$

So this thread performs an infinite loop consisting of the action a. Observe how the composition with the counter service hides the counter actions $c.inc$ and $c.dec$.

Example 3 The equation $x = c.inc \circ x$ defines the infinite thread $(P_n)_{n\in\mathbb{N}}$ with $P_0 = \mathsf{D}$ and $P_{n+1} = (c.inc)^{n+1} \circ \mathsf{D}$. We find that $P_n \,/_c\, \mathcal{C}(0) = \mathsf{D}$ for all $n \in \mathbb{N}$, so by definition $x \,/_c\, \mathcal{C}(0) = \mathsf{D}$.

This last example implies that we identify "livelock" and deadlock, and explains why we also use the name *inaction* for this state of affairs. Although this may seem to illustrate a weakness in expressive power, it proves to be a nice abstraction characterizing the absence of observable activity (any action or termination).

In the next example we show that the use of a counter service may turn regular threads into non-regular ones.

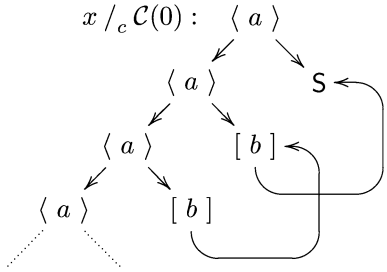
Example 4 Consider the regular thread x defined by[3]

$$x = c.inc \circ x \trianglelefteq a \trianglerighteq y, \qquad y = b \circ y \trianglelefteq c.dec \trianglerighteq \mathsf{S},$$

where $focus(a) \neq c \neq focus(b)$. Then, for all $n \in \mathbb{N}$,

$$\begin{aligned} x \mathbin{/_c} \mathcal{C}(n) &= (c.inc \circ x \trianglelefteq a \trianglerighteq y) \mathbin{/_c} \mathcal{C}(n) \\ &= (x \mathbin{/_c} \mathcal{C}(n+1)) \trianglelefteq a \trianglerighteq (y \mathbin{/_c} \mathcal{C}(n)), \\ y \mathbin{/_c} \mathcal{C}(0) &= \mathsf{S}, \\ y \mathbin{/_c} \mathcal{C}(n+1) &= b \circ (y \mathbin{/_c} \mathcal{C}(n)). \end{aligned}$$

The thread $x \mathbin{/_c} \mathcal{C}(0)$ can be depicted as follows (cf. the examples in Sects. 2.1 and 2.2):



It is not hard to see that $x \mathbin{/_c} \mathcal{C}(0)$ is an infinite thread with the property that for all n, a trace of $n+1$ a-actions produced by n replies `true` and one reply `false` is followed by $b^n \circ \mathsf{S}$. This yields a *non-regular* thread: if $x \mathbin{/_c} \mathcal{C}(0)$ were regular, it would be a fixed point of some finite linear specification, say with k equations. But specifying a trace $b^k \circ \mathsf{S}$ already requires $k+1$ linear equations $x_1 = b \circ x_2, \ldots, x_k = b \circ x_{k+1}, x_{k+1} = \mathsf{S}$, which contradicts the assumption. So $x \mathbin{/_c} \mathcal{C}(0)$ is not regular.

3 One-Counter Threads

In this section we introduce one-counter threads, i.e., threads that are reminiscent of various one-counter systems that occur in the literature (cf. [10, 11, 13, 15]). We focus on reachability under bounded counter values and transform a result by Rosier and Yen [13] to the setting of thread algebra.

3.1 One-Counter Threads and their Linear Specifications

A *one-counter thread* is a possibly non-regular thread that can be defined as the composition of a regular thread with a single counter service. We have found (cf. Example 4) that one-counter threads are not necessarily regular, so need not be specifiable by a finite linear specification. We define the *infinite* linear specification $E_{\mathcal{C}}$ derived from a finite linear specification E used to define a one-counter thread.

[3] Observe that a *linear* specification of x contains (at least) five equations.

Definition 1 Let E be a finite linear specification of a thread using a counter via focus c. Define

$$E_{\mathcal{C}} = \{x(n) = t_{x,n} \mid x \in \mathit{Var}(E),\ n \in \mathbb{N}\}$$

as the least set satisfying for all $(x = t_x) \in E$ and for all $n \in \mathbb{N}$,

- if $t_x = \mathsf{S}$ then $t_{x,n} = \mathsf{S}$,
- if $t_x = \mathsf{D}$ then $t_{x,n} = \mathsf{D}$,
- if $t_x = y \trianglelefteq a \trianglerighteq z$ with $\mathit{focus}(a) \neq c$, then $t_{x,n} = y(n) \trianglelefteq a \trianglerighteq z(n)$,
- if $t_x = y \trianglelefteq c.\mathit{inc} \trianglerighteq z$, then $t_{x,n} = c.\mathit{inc} \circ y(n+1)$,
- if $t_x = y \trianglelefteq c.\mathit{dec} \trianglerighteq z$, then $t_{x,0} = c.\mathit{dec} \circ z(0)$ and $t_{x,n+1} = c.\mathit{dec} \circ y(n)$.

The infinite linear specification $E_{\mathcal{C}}$ mimicks the composition of a thread defined by E with a counter service. However, in contrast to the thread-service composition, the counter actions remain observable. By construction, these actions occur in the equations of $E_{\mathcal{C}}$ as action prefix only. In this way, these equations incorporate the reply to the action, and we shall further refer to the counter actions in $E_{\mathcal{C}}$ as *interactions* rather than requests. We find that

$$x \,/_c\, \mathcal{C}(n) = x(n) \,/_c\, \mathcal{C}(n) \quad \text{for all } x \in \mathit{Var}(E) \text{ and } n \in \mathbb{N}, \tag{1}$$

where on the right-hand side we have added a use application $/_c$ in order to hide the interactions with the counter. Further observe that in this last application the counter value is immaterial; it is also true that

$$x \,/_c\, \mathcal{C}(n) = x(n) \,/_c\, \mathcal{C}(m)$$

for all $m \in \mathbb{N}$.

We continue with the definition of action relations and *traces* between the states of a specification $E_{\mathcal{C}}$ as just defined. We write

$$x(n) \overset{a:b}{\rightarrow\!\triangleright} y(m)$$

to denote that state $x(n)$ can evolve into state $y(m)$ by the execution of the action a with reply value $b \in \{\mathtt{true}, \mathtt{false}\}$. We call these labels $a{:}b$ *polarized actions*, and we use the letter α to denote them. The action relations for $E_{\mathcal{C}}$ are defined by

$$x(n) \overset{a:\mathtt{true}}{\rightarrow\!\triangleright} y(m) \quad \text{and} \quad x(n) \overset{a:\mathtt{false}}{\rightarrow\!\triangleright} z(m)$$

for all $(x(n) = y(m) \trianglelefteq a \trianglerighteq z(m)) \in E_{\mathcal{C}}$. A *trace* $r : x(n) \overset{\sigma}{\rightarrow\!\triangleright} x'(n')$ is a sequence

$$x_0(n_0) \overset{\alpha_0}{\rightarrow\!\triangleright} x_1(n_1) \overset{\alpha_1}{\rightarrow\!\triangleright} \cdots \overset{\alpha_{k-1}}{\rightarrow\!\triangleright} x_k(n_k)$$

for some $k \geq 0$ with $\sigma = \alpha_0 \ldots \alpha_{k-1}$, $x = x_0$, $n = n_0$, $x' = x_k$, and $n' = n_k$. For such a trace we further define $\mathit{labels}(r) = \{\alpha_0, \ldots, \alpha_{k-1}\}$.

We end this section with the definition of finite approximations of an infinite linear specification. Such approximations are used in Sect. 4.2 to prove the main result of this paper.

Definition 2 Given the infinite linear specification $E_{\mathcal{C}}$ for E we construct the approximation of $E_{\mathcal{C}}$ up to finite depth N, by redefining $t_{x,n}$ for $n > N$ as D, and consequently identifying all $x(n)$ for $n > N$. This yields a finite linear specification, which we shall denote by $Appr(N, E_{\mathcal{C}})$.

Remark In the above definition, we also could have chosen S or in fact any legal term for $t_{x,n}$ with $n > N$ (e.g., $x(n) = a \circ x(n)$ or $x(n) = c.inc \circ x(n)$). This is the case because we will use these approximations only in applications in which the $x(n)$ for $n > N$ play no role.

3.2 ω-One Counter Machines

We briefly introduce the ω-one counter machine (ω-1CM for short) as defined in [13]. Let Q be a set of states, Σ an input alphabet, and Z and B symbols indicating whether the counter value is zero or positive, respectively. Then the operation of an ω-1CM is defined by a transition function

$$\delta : Q \times \Sigma \times \{Z, B\} \to 2^{Q \times \{-1,0,1\}}.$$

Here, the value -1, 0, or 1 at the target states indicates whether the counter value decreased, remained unchanged, or increased during the transition. For example, if $\delta(q, a, Z) = \delta(q, a, B) = \{(q', 0)\}$, then the execution of a in state q leads to state q'. Moreover, this execution does not depend on the counter value and does not have an effect on it. In this case $\delta(q, a, Z)$ and $\delta(q, a, B)$ both define one transition. The definition of an ω-1CM is completed by specifying an initial state and a set of designated state sets (the latter one is used to express various notions of fairness, see [13]), however, we will not use these two ingredients.

The computations of an ω-1CM using a counter are formalized as follows. A *configuration* of an ω-1CM is a 3-tuple (q, j, n) where $q \in Q$ is the current state, j is the current length of the computation (initially 0), and n is the value of the counter. The transition function δ defines the *one-step computations*: $c \vdash^a c'$ denotes that configuration c can lead to configuration c' in a single step that reads symbol $a \in \Sigma$. The definition is as follows: $(q, j, n) \vdash^a (q', j + 1, n')$ if, and only if, a is the $(j + 1)$-th action symbol that is read, and either

- $n = n' = 0$ and $(q', 0) \in \delta(q, a, Z)$, or
- $n = n' > 0$ and $(q', 0) \in \delta(q, a, B)$, or
- $n = 0$, $n' = n + 1$ and $(q', 1) \in \delta(q, a, Z)$, or
- $n > 0$, $n' = n + 1$ and $(q', 1) \in \delta(q, a, B)$, or
- $n > 0$, $n' = n - 1$ and $(q', -1) \in \delta(q, a, B)$.

A *computation* $c \vdash^\sigma c'$ is a sequence

$$c_0 \vdash^{a_0} c_1 \vdash^{a_1} \ldots \vdash^{a_{k-1}} c_k$$

for some $k \geq 0$ with $c = c_0$, $c' = c_k$ and $\sigma = a_0 a_1 \ldots a_{k-1}$.

The following lemma is a result by Rosier and Yen (Lemma 4.3 in [13]):

Lemma 1 *For any ω-1CM with transition function δ, if $r : (q, j, n) \vdash^{\sigma} (q', j', n')$ is a computation for some σ, then there exists a computation $r' : (q, j, n) \vdash^{\rho} (q', j'', n')$ with the following properties*:

1. *The set of transitions used in r is identical to the set of transitions used in r', and*
2. *Every configuration (p, i, m) in r' satisfies $m \leq 3 \cdot s^3 + \max(n, n')$, where s is the number of transitions defined by δ.*

3.3 From Threads to Machines

We construct an ω-1CM for a one-counter thread specification E such that the computations of the constructed ω-1CM correspond directly to the traces of the infinite specification $E_{\mathcal{C}}$ (see Definition 1). The counterpart of Lemma 1 for the traces of $E_{\mathcal{C}}$ then follows straightforwardly.

Take state set $Q = Var(E)$. The input alphabet Σ (consisting of polarized actions) and the transitions of the ω-1CM follow from the following transformation on the equations in E. Each equation $x = y \trianglelefteq c.inc \trianglerighteq z$ yields the two transitions

$$\delta(x, c.inc{:}\texttt{true}, Z) = \delta(x, c.inc{:}\texttt{true}, B) = \{(y, 1)\},$$

each equation $x = y \trianglelefteq c.dec \trianglerighteq z$ yields the two transitions

$$\delta(x, c.dec{:}\texttt{false}, Z) = \{(z, 0)\} \quad \text{and} \quad \delta(x, c.dec{:}\texttt{true}, B) = \{(y, -1)\},$$

and each equation $x = y \trianglelefteq a \trianglerighteq z$ for $a \notin \{c.inc, c.dec\}$ yields the four transitions

$$\delta(x, a{:}\texttt{true}, Z) = \delta(x, a{:}\texttt{true}, B) = \{(y, 0)\}$$

and

$$\delta(x, a{:}\texttt{false}, Z) = \delta(x, a{:}\texttt{false}, B) = \{(z, 0)\}.$$

The action relations of $E_{\mathcal{C}}$ correspond by definition to the one-step computations of the machine constructed for E:

$$x(n) \xrightarrow{\alpha} y(n') \quad \text{if and only if} \quad (x, j, n) \vdash^{\alpha} (y, j + 1, n') \quad \text{for all } j \geq 0.$$

Clearly this correspondence extends to traces, i.e.,

$$x(n) \xrightarrow{\sigma} y(n') \quad \text{if and only if} \quad (x, 0, n) \vdash^{\sigma} (y, |\sigma|, n'), \tag{2}$$

so that the counterpart of Lemma 1 for traces of $E_{\mathcal{C}}$ reads as follows:

Lemma 2 *Let E be a finite linear specification. If $r : x(n) \xrightarrow{\sigma} y(n')$ is a trace in $E_{\mathcal{C}}$, then there is a trace $r' : x(n) \xrightarrow{\rho} y(n')$ with the following properties*:

1. *$labels(\sigma) = labels(\rho)$, and*
2. *Every state $z(m)$ of r' satisfies $m \leq 3 \cdot (4 \cdot |Var(E)|)^3 + \max(n, n')$.*

Proof By Lemma 1, and the construction and (2) given above. Observe that $s \leq 4 \cdot |Var(E)|$ where s the number of transitions of the ω-1CM constructed for E. □

This reachability result for $E_{\mathcal{C}}$ traces suffices for the correctness of risk assessment for one-counter threads (the main result of this article) presented in Sect. 4. In Sect. 5 we address reachability results for one-counter threads with states of the form $x \,/_c\, \mathcal{C}(n)$ rather than for the states $x(n)$ of $E_{\mathcal{C}}$. These results will follow from Lemma 2.

4 Risk Assessment for Regular and One-Counter Threads

In this section we discuss risk assessment services. We introduce this topic for regular threads, although the decidability of a risk assessment service for regular threads was recorded earlier in [5, 7]. However, the proof described here is much shorter and more elegant. Furthermore, this proof can easily be generalized to the decidability of a risk assessment service for one-counter threads, which establishes the main result of this paper.

4.1 Risk Assessment for Regular Threads

Assume a finite linear specification

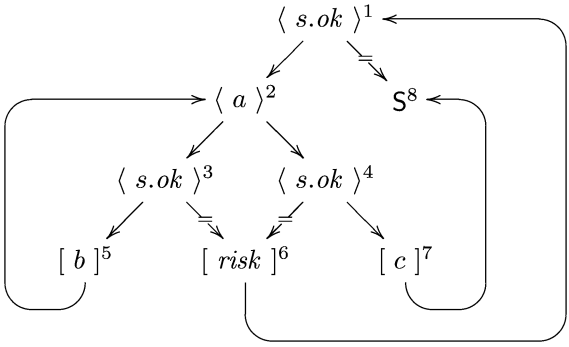
$$E = \{x = t_x \mid x \in Var(E)\}.$$

There are two special actions among the actions of E. First, there is the action *risk* representing behavior that poses a security risk. Secondly, there is the action $s.ok$ with s a focus name and *ok* a request to a *risk assessment service*. We call a variable $x \in Var(E)$ with t_x of the form $y \trianglelefteq s.ok \trianglerighteq z$ a *test state* with true-branch y and false-branch z. We assume that $focus(risk) \neq s$. A correct risk assessment service will reply `true` to the *ok* test, if, and only if, *risk* is not a possible future behavior of the true-branch composed with the service. We start with an example (taken from [7]).

Example 5 Suppose E is defined by the eight equations below, where the actions a, b and c do not use focus s. Then, in the regular thread x_1 the various $s.ok$ tests are evaluated as follows:

$$\begin{aligned}
x_1 &= x_2 \trianglelefteq s.ok \trianglerighteq x_8, && \texttt{(true)}\\
x_2 &= x_3 \trianglelefteq a \trianglerighteq x_4,\\
x_3 &= x_5 \trianglelefteq s.ok \trianglerighteq x_6, && \texttt{(true)}\\
x_4 &= x_6 \trianglelefteq s.ok \trianglerighteq x_7, && \texttt{(false)}\\
x_5 &= b \circ x_2,\\
x_6 &= risk \circ x_1,\\
x_7 &= c \circ x_8,\\
x_8 &= \mathsf{S}.
\end{aligned}$$

This situation can be illustrated as follows, where the superscript i on states corresponds to the subscript of the defining variable x_i of the specification:



Before giving the definition of a risk assessment service, we define the *risk states* of a finite linear specification.

Definition 3 The set $Risk(E) \subseteq Var(E)$ of risk states of a finite linear specification E is defined inductively as the least set satisfying, for all $(x = t_x) \in E$,

1. If $t_x = y \trianglelefteq risk \trianglerighteq z$, then $x \in Risk(E)$,
2. If $t_x = y \trianglelefteq s.ok \trianglerighteq z$, and both $y \in Risk(E)$ and $z \in Risk(E)$, then $x \in Risk(E)$,
3. If $t_x = y \trianglelefteq a \trianglerighteq z$ for $a \neq risk, s.ok$, and $y \in Risk(E)$ or $z \in Risk(E)$, then $x \in Risk(E)$.

Membership of $Risk(E)$ is decidable since $Var(E)$ is finite. For instance, for E as defined in Example 5, $Risk(E) = \{x_6\}$.

The idea is that in risk states, the possibility of a future *risk* execution cannot be avoided by a risk assessment service answering to the *ok* tests, so that the service should try to avoid risk states. Before we make this precise we do some preprocessing on the specification of the thread. We assume that to define the reply function of the risk assessment service, we can analyze the specification E. For the answer to a particular *ok* test, it must then be known which part of the specification is to be assessed. This is made possible by *annotating* the *ok* tests with variables from $Var(E)$. Let E^a be the specification obtained from E by replacing all equations of the form

$$x = y \trianglelefteq s.ok \trianglerighteq z \quad \text{by} \quad x = y \trianglelefteq s.ok{:}y \trianglerighteq z.$$

This preprocessing allows us to define the service as a history-based service in the format defined in Sect. 2.3.[4] Having defined this preprocessing step, we immediately adopt the convention to blur the distinction between E and E^a; we assume that the annotation is implicitly visible in a use application.

[4] In [7], a *general* risk assessment tool SHRAT (security hazard risk assessment tool) is defined as a more powerful kind of service that upon a request *s.ok* loads both the specification and the identity of the state to be tested. A disadvantage of that approach is that SHRAT does not necessarily commute with other thread-service applications.

We now define the risk assessment service $\mathcal{S}(E)$ for the specification E as the service with signature $\{ok\}$, or actually with signature $\{ok{:}x \mid x \in Var(E)\}$ by the convention just introduced, and with reply function

$$F(\nu(ok{:}x)) = \begin{cases} \texttt{true} & \text{if } x \notin Risk(E), \\ \texttt{false} & \text{otherwise,} \end{cases}$$

for all $x \in Var(E)$ (the history ν is never used).

We continue with the property of risk states mentioned above: in risk states possible future execution of *risk* cannot be avoided by $\mathcal{S}(E)$. First, for thread x, possible future execution of *risk* is defined as

$$risk \in actions(x),$$

where *actions*(_) is defined for BTA threads by

$$actions(\mathsf{S}) = actions(\mathsf{D}) = \emptyset,$$
$$actions(P \trianglelefteq a \trianglerighteq Q) = \{a\} \cup actions(P) \cup actions(Q),$$

and this is extended to BTA^∞ threads by

$$actions((P_n)_{n\in\mathbb{N}}) = \bigcup_{n\in\mathbb{N}} actions(P_n).$$

For example, if $E = \{x = \mathsf{D},\ y = x \trianglelefteq a \trianglerighteq y\}$ then $actions(x) = \emptyset$ while $actions(y) = \{a\}$.

Lemma 3 *Let E be a finite linear specification. Then*

$$x \in Risk(E) \quad implies \quad risk \in actions(x \mathbin{/_s} \mathcal{S}(E)).$$

Proof Easy, following the inductive definition of $Risk(E)$. □

The next theorem states that the service $\mathcal{S}(E)$ is a *correct* risk assessment service for E, in the sense that its reply to $ok{:}x$ is `true`, if, and only if, $risk \notin actions(x \mathbin{/_s} \mathcal{S}(E))$.

Theorem 1 *Let E be a finite linear specification. Then $\mathcal{S}(E)$ as defined above is a correct risk assessment service for E.*

Proof We must prove that $risk \in actions(x \mathbin{/_s} \mathcal{S}(E))$ if and only if $x \in Risk(E)$. The if-part follows from Lemma 3. For the only-if-part, suppose that $risk \in actions(x \mathbin{/_s} \mathcal{S}(E))$. Then there must be a trace of x leading to a state where *risk* can be performed, corresponding to a sequence

$$x_k, x_{k-1}, \ldots, x_1 \in Var(E)$$

with $x_k = x$, $k \geq 1$, such that $x_1 = y \trianglelefteq \mathit{risk} \trianglerighteq z$ for some y, z, with the following property (†): For $i = 2, \ldots, k$, $x_i = y \trianglelefteq a \trianglerighteq z$ in E for some y, z with $x_{i-1} \in \{y, z\}$, and $a = s.ok$ implies $x_i /_s \mathcal{S}(E) = x_{i-1} /_s \mathcal{S}(E)$.

We now prove by induction on k that $x_k \in \mathit{Risk}(E)$.

Base case. If $k = 1$ then $x_k \in \mathit{Risk}(E)$ because $x_1 = y \trianglelefteq \mathit{risk} \trianglerighteq z$ for some y, z.

Induction step. Let $k > 1$ and assume that $x_{k-1} \in \mathit{Risk}(E)$. By (†): $x_k = y \trianglelefteq a \trianglerighteq z$ in E with $x_{k-1} \in \{y, z\}$, and $a = s.ok$ implies $x_k /_s \mathcal{S}(E) = x_{k-1} /_s \mathcal{S}(E)$. If $a \neq s.ok$, then $x_k \in \mathit{Risk}(E)$ by definition. If $a = s.ok$, then $x_k /_s \mathcal{S}(E) = x_{k-1} /_s \mathcal{S}(E)$, and since $x_{k-1} \in \mathit{Risk}(E)$ we know by definition of $\mathcal{S}(E)$ that it must be that it answered `false` to the $s.ok$ test, and therefore that $y \in \mathit{Risk}(E)$ and $x_{k-1} = z$. So both $y \in \mathit{Risk}(E)$ and $z \in \mathit{Risk}(E)$ and therefore by definition of $\mathit{Risk}(E)$ also $x_k \in \mathit{Risk}(E)$. □

Clearly, the thread $T = x_1 /_s \mathcal{S}(E)$ as defined and depicted in Example 5 satisfies $T = b \circ T \trianglelefteq a \trianglerighteq c \circ \mathsf{S}$.

4.2 Risk Assessment for One-Counter Threads

We now turn to the construction of a risk assessment service for one-counter threads. Consider a finite linear specification E and its infinite version $E_{\mathcal{C}}$ constructed as in Sect. 3.1. Assume that distinct foci s and c are used to access the risk assessment service and the counter service, respectively, and that $\mathit{focus}(\mathit{risk}) \notin \{s, c\}$. For the definition of a risk assessment service for $E_{\mathcal{C}}$ we exploit Lemma 2. First we show that we can restrict to specifications in which *risk* can only be performed when the counter value is zero. To this end we modify E to E' by replacing each equation of the form

$$x = y \trianglelefteq \mathit{risk} \trianglerighteq z$$

by the two equations

$$x = x \trianglelefteq c.\mathit{dec} \trianglerighteq x' \quad \text{and} \quad x' = y \trianglelefteq \mathit{risk} \trianglerighteq z$$

for some fresh variable x' for x. Observe that, for any state $x(n)$ of $E_{\mathcal{C}}$ in which *risk* can be performed, $x(n)$ in $E'_{\mathcal{C}}$ cannot perform *risk* immediately, but that the consecutive execution of $n + 1$ $c.\mathit{dec}$ actions leads to state $x'(0)$ in which *risk* can be performed. We restrict, without loss of generality, to finite linear specifications with the property that *risk* can only be performed when the counter value is zero: we assume that E has been modified as just described.

Given such a specification E, it follows from Lemma 2 that for any state $x(n)$ of $E_{\mathcal{C}}$ there is a certain value

$$N_n = 3 \cdot (4 \cdot |\mathit{Var}(E)|)^3 + n$$

such that any trace to a state in which *risk* can be performed, has a trace in which the counter value does not exceed N_n. So, for risk assessment it is sufficient to construct the finite approximation $\mathit{Appr}(N_n, E_{\mathcal{C}})$ of the infinite specification $E_{\mathcal{C}}$ (see Definition 2). For this finite linear specification membership of *Risk* is decidable. In the remainder of this section we abbreviate $\mathit{Appr}(N_n, E_{\mathcal{C}})$ to $E^n_{\mathcal{C}}$.

So, let $x \in Var(E)$ be a test state, with $x = y \trianglelefteq s.ok \trianglerighteq z$ in E. What is the reply to the test in a state $x(n)$ of $E_{\mathcal{C}}$? Well, construct $E^n_{\mathcal{C}}$, and reply `true` if and only if $y(n)$ is not a risk state in this finite specification. We assume as before that the identity of the true-branch is visible in the test (so we have to reply to a method of the form $ok{:}y{:}n$), and define the risk assessment service $\mathcal{S}(E_{\mathcal{C}})$ with reply function

$$F(v(ok{:}y{:}n)) = \begin{cases} \texttt{true} & \text{if } y(n) \notin Risk(E^n_{\mathcal{C}}), \\ \texttt{false} & \text{otherwise,} \end{cases}$$

for all states $y(n)$ of $E_{\mathcal{C}}$.

Theorem 2 *Let E be a finite linear specification. Then $\mathcal{S}(E_{\mathcal{C}})$ as defined above is a correct risk assessment service for $E_{\mathcal{C}}$, that is, for any state $x(n)$ of $E_{\mathcal{C}}$,*

$$risk \in actions(x(n) \,/_s\, \mathcal{S}(E_{\mathcal{C}})) \quad \text{if and only if} \quad x(n) \in Risk(E^n_{\mathcal{C}}).$$

The proof is similar to the proof for the regular case in Sect. 4.1:

Proof For the if-part: using Lemma 3 we find that

$$risk \in actions(x(n) \,/_s\, \mathcal{S}(E^n_{\mathcal{C}}),$$

and it is straightforward to prove that

$$actions(x(n) \,/_s\, \mathcal{S}(E^n_{\mathcal{C}})) \subseteq actions(x(n) \,/_s\, \mathcal{S}(E_{\mathcal{C}})).$$

For the only-if-part, suppose that $risk \in actions(x(n) \,/_s\, \mathcal{S}(E_{\mathcal{C}}))$. Then there must be a trace of $x(n)$ leading to a state where *risk* can be performed, corresponding to a sequence of states

$$x_k(n_k), x_{k-1}(n_{k-1}), \ldots, x_1(n_1)$$

with $k \geq 1$, $x_k = x$, $n_k = n$, $x_1 = y \trianglelefteq risk \trianglerighteq z$ for some y, z, and $n_1 = 0$, with the following properties:

(†) For $i = 2, \ldots, k$, $x_i = y \trianglelefteq a \trianglerighteq z$ in E for some y, z with $x_{i-1} \in \{y, z\}$, and $a = s.ok$ implies $x_i(n_i) \,/_s\, \mathcal{S}(E_{\mathcal{C}}) = x_{i-1}(n_{i-1}) \,/_s\, \mathcal{S}(E_{\mathcal{C}})$.

(‡) By Lemma 2, $n_i \leq N_n$ for $i = 1, \ldots, k$, so that the trace is part of the finite $E^n_{\mathcal{C}}$.

We now prove by induction on k that $x_k(n_k) \in Risk(E^n_{\mathcal{C}})$.

Base case. If $k = 1$ then $x_k(n_k) \in Risk(E^n_{\mathcal{C}})$ because $x_1 = y \trianglelefteq risk \trianglerighteq z$ for some y, z.

Induction step. Let $k > 1$ and assume $x_{k-1}(n_{k-1}) \in Risk(E^n_{\mathcal{C}})$. By (†) we find that $x_k = y \trianglelefteq a \trianglerighteq z$ in E with $x_{k-1} \in \{y, z\}$, and $a = s.ok$ implies $x_k(n_k) \,/_s\, \mathcal{S}(E_{\mathcal{C}}) = x_{k-1}(n_{k-1}) \,/_s\, \mathcal{S}(E_{\mathcal{C}})$. If $a \neq s.ok$, then $x_k(n_k) \in Risk(E^n_{\mathcal{C}})$ by definition. If $a = s.ok$, then $x_k(n_k) \,/_s\, \mathcal{S}(E_{\mathcal{C}}) = x_{k-1}(n_{k-1}) \,/_s\, \mathcal{S}(E_{\mathcal{C}})$, and $n_k = n_{k-1}$. Since $x_{k-1}(n_{k-1}) \in Risk(E^n_{\mathcal{C}})$ we know by definition of $\mathcal{S}(E_{\mathcal{C}})$ and (‡) that it must be that it answered `false` to the $s.ok$ test, and therefore that $y(n_k) \in Risk(E^n_{\mathcal{C}})$ and $x_{k-1} = z$. So both $y(n_k) \in Risk(E^n_{\mathcal{C}})$ and $z(n_k) \in Risk(E^n_{\mathcal{C}})$, and therefore $x_k(n_k) \in Risk(E^n_{\mathcal{C}})$ by definition of $Risk(E^n_{\mathcal{C}})$. □

Our ultimate goal is to prove the decidability and correctness of a risk assessment service for any one-counter thread, given its finite linear specification E and counter value $\mathcal{C}(n)$. Using identity (1), i.e.,

$$x \,/_c\, \mathcal{C}(n) = x(n) \,/_c\, \mathcal{C}(n) \quad \text{for all } x \in \mathit{Var}(E) \text{ and } n \in \mathbb{N},$$

where $x(n)$ is defined by the infinite linear specification $E_{\mathcal{C}}$, we can now reason as follows: suppose $(x = y \trianglelefteq s.ok \trianglerighteq z) \in E$. By Theorem 2, the risk assessment service $\mathcal{S}(E_{\mathcal{C}})$ provides the correct reply to $x(n)$ as defined by $E_{\mathcal{C}}$. It follows that this reply is also correct for $x(n) \,/_c\, \mathcal{C}(n)$, because the counter actions in $E_{\mathcal{C}}$ equations occur as action prefix only, and thus for $x \,/_c\, \mathcal{C}(n)$ as defined by E.

We conclude that risk assessment for a one-counter thread $x \,/_c\, \mathcal{C}(n)$ can be made using its infinite specification $E_{\mathcal{C}}$, and is hence by Theorem 2 decidable and correct.

5 Reachability Results for One-Counter Threads

Let E be a finite linear specification and $x \in \mathit{Var}(E)$. In Sect. 3 we have defined action relations and traces for the states $x(n)$ of $E_{\mathcal{C}}$. In this section we give corresponding definitions for threads with states of the form $x \,/_c\, \mathcal{C}(n)$ and present the corresponding reachability results. We start with the action relations (note the different arrow heads).

For $(x = y \trianglelefteq a \trianglerighteq z) \in E$, with $\mathit{focus}(a) \neq c$,

$$x \,/_c\, \mathcal{C}(n) \overset{a:\mathtt{true}}{\rightarrow} y \,/_c\, \mathcal{C}(n) \quad \text{and} \quad x \,/_c\, \mathcal{C}(n) \overset{a:\mathtt{false}}{\rightarrow} z \,/_c\, \mathcal{C}(n).$$

In order to model the interactions with the counter we use an action relation with the special label ι, so for $(x = y \trianglelefteq c.\mathit{inc} \trianglerighteq z) \in E$,

$$x \,/_c\, \mathcal{C}(n) \overset{\iota}{\rightarrow} y \,/_c\, \mathcal{C}(n+1),$$

and for $(x = y \trianglelefteq c.\mathit{dec} \trianglerighteq z) \in E$,

$$x \,/_c\, \mathcal{C}(0) \overset{\iota}{\rightarrow} z \,/_c\, \mathcal{C}(0) \quad \text{and} \quad x \,/_c\, \mathcal{C}(n+1) \overset{\iota}{\rightarrow} y \,/_c\, \mathcal{C}(n).$$

A trace $r : x \,/_c\, \mathcal{C}(n) \overset{\sigma}{\rightarrow} y \,/_c\, \mathcal{C}(n')$ is a sequence

$$x_0 \,/_c\, \mathcal{C}(n_0) \overset{\alpha_0}{\rightarrow} x_1 \,/_c\, \mathcal{C}(n_1) \overset{\alpha_1}{\rightarrow} \cdots \overset{\alpha_{k-1}}{\rightarrow} x_k \,/_c\, \mathcal{C}(n_k),$$

with $x = x_0$, $n = n_0$, $y = x_k$, $n' = n_k$, and $\sigma = \alpha_0 \ldots \alpha_{k-1}$ with the α_i ranging over $\{\iota\} \cup \{a{:}\mathtt{true}, a{:}\mathtt{false} \mid a \in A\}$.

For actions not using focus c we find that

$$x(n) \overset{a:b}{\rightarrowtriangle} y(n) \quad \text{if and only if} \quad x \,/_c\, \mathcal{C}(n) \overset{a:b}{\rightarrow} y \,/_c\, \mathcal{C}(n),$$

while for interactions with the counter,

$$x(n) \overset{c.\mathit{inc}:\mathtt{true}}{\rightarrowtriangle} y(n+1) \quad \text{if and only if} \quad x \,/_c\, \mathcal{C}(n) \overset{\iota}{\rightarrow} y \,/_c\, \mathcal{C}(n+1),$$

$$x(n+1) \overset{c.dec:\mathtt{true}}{\rightarrowtriangle} y(n) \quad \text{if and only if} \quad x /_c \mathcal{C}(n+1) \xrightarrow{\iota} y /_c \mathcal{C}(n),$$

$$x(0) \overset{c.dec:\mathtt{false}}{\rightarrowtriangle} y(0) \quad \text{if and only if} \quad x /_c \mathcal{C}(0) \xrightarrow{\iota} y /_c \mathcal{C}(0).$$

This correspondence between the action relations of $E_{\mathcal{C}}$ and those defined for states of the form $x /_c \mathcal{C}(n)$ extends to traces: any trace $x(n) \overset{\sigma}{\rightarrowtriangle} y(n')$ of $E_{\mathcal{C}}$ corresponds to a trace $x /_c \mathcal{C}(n) \xrightarrow{\sigma'} y /_c \mathcal{C}(n')$ where σ' is obtained from σ by renaming the polarized counter actions to ι.

The following two theorems for one-counter threads now follow as corollaries of Lemma 2.

Theorem 3 *Let E be a finite linear specification with $x, y \in Var(E)$. Then, for any $n, n' \in \mathbb{N}$ it is decidable whether the state $y /_c \mathcal{C}(n')$ can be reached from $x /_c \mathcal{C}(n)$.*

Proof Consider the infinite linear specification $E_{\mathcal{C}}$ as defined by Definition 1. Using identity (1), we find that the state $y /_c \mathcal{C}(n')$ is reachable from $x /_c \mathcal{C}(n)$ if, and only if, $y(n')$ is reachable from $x(n)$ in $E_{\mathcal{C}}$. According to Lemma 2 the latter is the case if and only if it is so in $Appr(N_{n,n'}, E_{\mathcal{C}})$, the finite approximation of $E_{\mathcal{C}}$ with

$$N_{n,n'} = 3 \cdot (4 \cdot |Var(E)|)^3 + \max(n, n').$$

This approximation is a regular thread, and reachability is decidable for regular threads (simply redefine the equation for $y(n')$ and decide whether equality is preserved), so the result follows immediately. □

Theorem 4 *Let E be a finite linear specification. If $r : x /_c \mathcal{C}(n) \xrightarrow{\sigma} y /_c \mathcal{C}(n')$ is a trace, then there is a trace*

$$r' : x /_c \mathcal{C}(n) \xrightarrow{\rho} y /_c \mathcal{C}(n')$$

such that $labels(\sigma) = labels(\rho)$, and $m \leq 3 \cdot (4 \cdot |Var(E)|)^3 + \max(n, n')$ for any state $z /_c \mathcal{C}(m)$ in r'.

We end this short section with a comment on the traces just defined. The action relation ι reflects the effect of thread-service composition: an interaction with the counter results in a single, discrete step. We use this as an auxiliary notion providing a simple definition of traces. Note however, that the equations that define thread-service composition (in Sect. 2.3) imply that the interaction between a thread and a service should not be visible (or 'observable'). On the level of traces, such invisibility can be obtained by taking together the source and target states of the ι steps, and consequently leaving ι out of the traces. A formal definition can easily be given. We then find, for a one-counter thread for which all actions are requests to the counter, that all traces collapse: for example, $x /_c \mathcal{C}(0)$ defined by $x = c.inc \circ x$ has no observable traces.[5] For the one-counter thread defined in Example 2, we find that it has an

[5] Cf. Example 3 and the remark about the identification of livelock and deadlock following this example.

observable trace

$$x_1 /_c \mathcal{C}(0) \xrightarrow{a^k} x_1 /_c \mathcal{C}(0)$$

for any $k > 0$. We further note that a complete account of the observable traces of threads should include the difference between termination and inaction, so that S and D are distinguished (for example by a special action relation that signals termination).

6 Conclusions

Risk assessment as investigated in this paper has two interesting characteristics. First, it offers an alternative to Cohen's seminal 1984-result on the impossibility of virus detection [8]. The crucial twist in our approach is that we use a test that only establishes whether its true-branch is risk-free and that does not evaluate its false-branch, thus resisting arguments that are based on self-reference (this form of risk assessment was first proposed in [5]). Secondly, exploiting a result of Rosier and Yen [13] we extended the class of threads for which risk assessment is decidable to the one-counter threads.

We expect that risk assessment is also decidable for pushdown threads (i.e., regular threads that may use a stack), but our proof does not generalize to this case (some partial results can be found in [7]). The essential problem is that control decisions depending on a stack over at least two elements can occur at any stack contents (e.g., a test on the identity of the top-element), while control decisions in the case of a counter can take place only if the counter has value 0. So, the decidability of a service for risk assessment of pushdown threads (or any similar form of action-forecasting) is still an open question.

Finally, apart from the decidability of risk assessment, the one-counter threads form a noteworthy class of threads. We conclude with a summary of some typical properties of this class of threads:

1. Equivalence is decidable (shown in [6]),
2. Inclusion is undecidable (see also [6]),
3. Reachability is decidable (Theorem 3 in the present paper), and
4. Reachability is preserved under bounded counter values and polarized action labels (Theorem 4 in the present paper).

Acknowledgements We are grateful for a valuable review.

References

1. 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.) Proceedings of ICALP 2003. Lecture Notes in Comput. Sci., vol. 2719, pp. 1–21. Springer, New York (2003)
2. Bergstra, J.A., Bethke, I.: Polarized process algebra with reactive composition. Theor. Comput. Sci. **343**(3), 285–304 (2005)
3. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Inf. Control **60**(1–3), 109–137 (1984)

4. Bergstra, J.A., Ponse, A.: Combining programs and state machines. J. Log. Algebr. Program. **51**(2), 175–192 (2002)
5. Bergstra, J.A., Ponse, A.: A bypass of Cohen's impossibility result. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A. (eds.) Advances in Grid Computing—EGC 2005. Lecture Notes in Comput. Sci., vol. 3470, pp. 1097–1106. Springer, New York (2005)
6. Bergstra, J.A., Bethke, I., Ponse, A.: Decision problems for pushdown threads. Acta Inf. **44**(2), 75–90 (2007)
7. Bergstra, J.A., Bethke, I., Ponse, A.: Thread algebra and risk assessment services. In: Dimitracopoulos, C., et al. (eds.) Logic Colloquium '05: Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic, Athens, Greece, July 28–August 3, 2005. ASL Lecture Notes in Logic, vol. 28 (to appear)
8. Cohen, F.: Computer viruses—theory and experiments. Comput. Secur. **6**(1), 22–35 (1984). Available as http://vx.netlux.org/lib/afc01.html
9. de Bakker, J.W., Zucker, J.I.: Processes and the denotational semantics of concurrency. Inf. Control **54**(1–2), 70–120 (1982)
10. Jančar, P., Moller, F., Sawa, Z.: Simulation problems for one-counter machines. In: Proceedings of SOFSEM'99: The 26th Seminar on Current Trends in Theory and Practice of Informatics. Lecture Notes in Comput. Sci., vol. 1725, pp. 398–407. Springer, New York (1999)
11. Paterson, M.S., Valiant, L.G.: Deterministic one-counter automata. J. Comput. Syst. Sci. **10**(3), 340–350 (1975)
12. Ponse, A., van der Zwaag, M.B.: An introduction to program and thread algebra. In: Beckmann, A. (ed.) Logical Approaches to Computational Barriers: Proceedings CiE 2006. Lecture Notes in Comput. Sci., vol. 3988, pp. 445–458. Springer, New York (2006)
13. Rosier, L.E., Yen, H.-C.: Logspace hierarchies, polynomial time and the complexity of fairness problems concerning ω-machines. SIAM J. Comput. **16**(5), 779–807 (1987)
14. Vu, T.D.: Metric denotational semantics for BPPA. Report PRG0503, Programming Research Group, University of Amsterdam (2005). To appear in J. Log. Algebr. Program.
15. Yen, H.-C., Yu, L.-P.: Decidability analysis of self-stabilization for infinite state systems. Fundam. Inf. **70**(4), 387–402 (2006)