Risk Assessment for One-Counter Threads

A. Ponse
M.B. van der Zwaag

Report PRG0608 October 2006

A. Ponse

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

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

M.B. van der Zwaag

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7584
e-mail: mbz@science.uva.nl

Programming Research Group Electronic Report Series

Risk Assessment for One-Counter Threads

Alban Ponse & Mark B. van der Zwaag
Programming Research Group
Informatics Institute
University of Amsterdam
`http://www.science.uva.nl/research/prog/`

October 24, 2006

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 a certain form of risk assessment for one-counter threads is proved. This 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. This property is based on a result for ω-one counter machines by Rosier and Yen (1987).

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 [3], 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).

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 [4], 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 [4].

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 [7]. 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 [12] 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, and some auxiliary definitions. Then, in Section 3, we discuss one-counter threads and the above-mentioned reachability result. In Section 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. Finally, in Section 5 we end with some conclusions.

2 Threads and Services

In this section we introduce the basics of thread algebra and thread-service composition. For a general introduction to these matters we refer to [11]. Finally we present some auxiliary definitions that are used in the remainder of the paper.

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 *postconditional composition* operator $\_ \unlhd a \unrhd \_$.

We use *action prefixing* $a \circ P$ as an abbreviation for $P \unlhd a \unrhd P$ 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`.

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

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

Following the metric theory of [1] in the form developed as the basis of the introduction of processes in [5], 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:[2]

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

For a detailed account of this construction see [2, 13].

Overloading notation, we now 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 $R_{n+1} = P_n \trianglelefteq a \trianglerighteq Q_n$,
3. $\pi_n((P_m)_{m\in\mathbb{N}}) = (P_0, \ldots, P_{n-1}, P_n, P_n, P_n \ldots)$.

The elements of BTA are included in BTA^∞ by a mapping following this definition. 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 $\mathit{Res}(P)$ of *residual threads* of P inductively as follows:

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

[1]For the definition of threads we completely abstract from the environment. In Section 2.2 we define services which model (part of) the environment, and thread-service composition.

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

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 finite linear recursive specification over BTA^{∞} is a set

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

of equations for a finite 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 recursive specifications represent continuous operators having unique fixed points [13]. In reasoning with finite linear 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 $x = a \circ x$.

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

Proof. Easy, see e.g., [13, 11]. $\dashv$

2.2 Services and Thread-Service Composition

Services model (part of) the execution environment of threads. In order to define the interaction between a thread and a service, we let actions be of the form $c.m$ where c is the so-called *channel* or *focus*, and m is the *co-action* or *method*.

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

For a service $\mathcal{H} = \langle \Sigma, F \rangle$ and a thread P, $P /_c \mathcal{H}_\sigma$ represents P *using the service* $\mathcal{H}_\sigma$ via channel c. The defining rules are:

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

The operator $/_c$ is called the *use operator* and stems from [6]. An expression $P /_c \mathcal{H}_\sigma$ is sometimes referred to as *thread-service composition.*

The use operator is expanded to infinite threads P by stipulating

$$P /_c \mathcal{H}_\sigma = (\pi_n(P) /_c \mathcal{H}_\sigma)_{n \in \mathbb{N}}.$$

As a consequence, $P /_c \mathcal{H} = \mathsf{D}$ if for every n, $\pi_n(P) /_c \mathcal{H} = \mathsf{D}$. Note that this is the case if all actions of P are processed by the reply function of $\mathcal{H}$ (via channel c).

In Section 3 we give some simple examples of thread-service composition.

2.3 Auxiliary Definitions

In order to describe various notions of reachability we use certain action relations:

Definition 1 (Reachability and Alphabet) *One-step action relations:*

$$x \trianglelefteq a \trianglerighteq y \xrightarrow{a} x, \quad x \trianglelefteq a \trianglerighteq y \xrightarrow{a} y$$

Let $\rightarrow$ be the union of all action relations, and let $\rightarrow^*$ be the reflexive transitive closure of $\rightarrow$.

Let furthermore $\overset{\sigma}{\twoheadrightarrow}$ be defined by

$$x \overset{\epsilon}{\twoheadrightarrow} x, \quad \frac{x \xrightarrow{a} y}{x \overset{a}{\twoheadrightarrow} y} \quad \text{and} \quad \frac{x \overset{\sigma}{\twoheadrightarrow} y \quad y \overset{\rho}{\twoheadrightarrow} z}{x \overset{\sigma\rho}{\twoheadrightarrow} z}$$

where ϵ is the empty sequence and σ and ρ range over sequences of actions.

Given $x \overset{\sigma}{\twoheadrightarrow} y$, alphabet$(\sigma)$ is the set of actions occurring in σ.

Note that for a thread x defined by a linear specification E and a service $\mathcal{H} = \langle \Sigma, F \rangle$,

$$x /_c \mathcal{H} \overset{\sigma}{\twoheadrightarrow} y /_c \mathcal{H}_\rho$$

implies that in E, either $y = z \trianglelefteq a \trianglerighteq z'$ for some $z, z' \in \mathit{Var}(E)$ and $a \neq c.m$, or $y = \mathsf{D}$ or $y = \mathsf{S}$. In other words, the target state y of a relation $x \overset{\sigma}{\twoheadrightarrow} y$ never issues an action that is processed by the reply function of $\mathcal{H}$. Of course, a similar remark is in order for the case of repeated thread-service compositions.

For threads we also distinguish so-called action sets.

Definition 2 (Action sets) *For finite threads we define*

$$\begin{aligned} \mathit{actions}(\mathsf{S}) &= \mathit{actions}(\mathsf{D}) = \emptyset, \\ \mathit{actions}(x \trianglelefteq a \trianglerighteq y) &= \{a\} \cup \mathit{actions}(x) \cup \mathit{actions}(y). \end{aligned}$$

For potentially infinite threads we define

$$\mathit{actions}(P) = \bigcup_{n \in \mathbb{N}} \mathit{actions}(\pi_n(P)).$$

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. [9, 12, 14, 10]). We focus on reachability under bounded counter values and transform the result by Rosier and Yen [12] to the setting of thread algebra.

3.1 One-Counter Threads and their Linear Specifications

A (natural number) *counter* $\mathcal{C}$ can be defined as a service with co-actions *inc* and *dec*, where *inc* increases the counter value and yields `true`, and *dec* decreases the value with reply `true` if the current value is larger than 0, and yields `false` otherwise. 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 $n \geq 0$.

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

Example 1 *By the defining equations for thread-service composition it follows that for any thread x,*

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

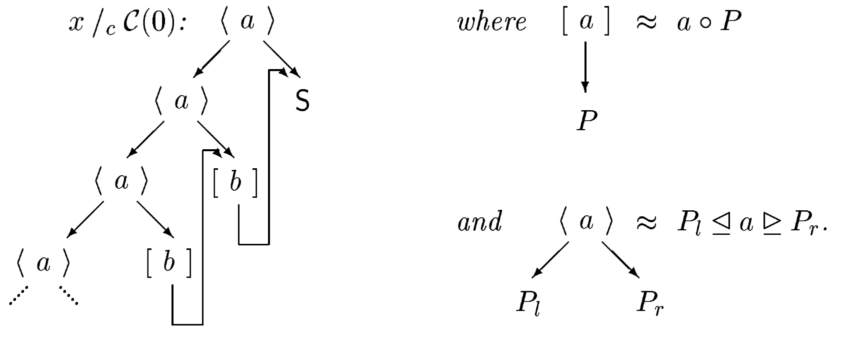
Now consider the regular thread x defined by

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

where actions a and b do not use focus c. Then, for all $n \in \mathbb{N}$,

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

and $y /_c \mathcal{C}(n+1) = b \circ (y /_c \mathcal{C}(n))$. The thread $x /_c \mathcal{C}(0)$ can be depicted as follows:



It is not hard to see that $x /_c \mathcal{C}(0)$ is an infinite thread with the property that for all n, a trace of $n+1$ a-actions produced by n positive and one negative reply on a is followed by $b^n \circ \mathsf{S}$. This yields an non-regular *thread: if $x /_c \mathcal{C}(0)$ were regular, it would be a fixed point of some finite linear recursive 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 $x /_c \mathcal{C}(0)$ is not regular.*

We have found that one-counter threads are not necessarily regular, so need not be specifiable by a finite linear specification. We first define the *infinite* linear specification of a one-counter thread. Let E be a finite linear specification. We define the infinite specification $E_{\mathcal{C}}$

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

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

Observe that

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

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

Definition 3 (Finite Approximation of a Linear Specification) *Given an infinite linear recursive specification $E_{\mathcal{C}}$ for E as defined above, 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 recursive specification, which we shall denote by $\pi(N, E_{\mathcal{C}})$.*

Remark: in the above definition, we also could have chosen S or in fact any legal term for $t_{x,n}$ and $n > N$ (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 Reachability and Bounded Counter Values

For the definition of a risk assessment service for $E_{\mathcal{C}}$ we will use a result by Rosier and Yen [12] for ω-one counter machines. In our setting this amounts to the following lemma:

Lemma 1 (Bounded Counter Values) *Let a finite linear specification E be given. For any state $x(n)$ of $E_{\mathcal{C}}$, if*

$$x(n) \to^* x'(n'),$$

then, for some $k \geq 0$, there is a trace

$$x(n) = x_0(n_0) \to x_1(x_1) \to \cdots \to x_k(n_k) = x'(n')$$

with for all $i \leq k$, $n_i \leq 3 \cdot (4 \cdot |\mathit{Var}(E)|)^3 + \max(n, n')$.

Proof. Based on Lemma 4.3 in [12] (alternatively, see Lemma 3.1 in [14]). The only thing left to explain is how a finite linear specification can be associated with a ω-one counter machine (ω-1CM for short) as defined in [12]. We introduce the specification of ω-1CMs on the fly: let Q be a set of states, Σ an input alphabet (in our case including $c.inc$ and $c.dec$) and Z and B symbols indicating whether a counter value is zero or positive, respectively. Then the operation of a ω-1CM is defined by a transition function

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

Given E, we take $Q = \mathit{Var}(E) \cup \{\mathit{Term}\}$ with Term a fresh termination state.

The input alphabet Σ and the transitions of the associated ω-1CM follow from the following transformation on E's equations:

- each equation $x = \mathsf{S}$ yields the two transitions $\delta(x, t, Z) = \delta(x, t, B) = \{(\mathit{Term}, 0)\}$, where t is a fresh action (relative to E) that signals termination and the 0 indicates that the counter value is not changed,
- each equation $x = \mathsf{D}$ yields the two transitions $\delta(x, d, Z) = \delta(x, d, B) = \{(\mathit{Term}, 0)\}$ where d is a fresh action that signals deadlock,
- each equation $x = y \trianglelefteq c.inc \trianglerighteq z$ yields the two transitions $\delta(x, c.inc, Z) = \delta(x, c.inc, B) = \{(y, 1)\}$, where the 1 indicates that the counter value increases,
- each equation $x = y \trianglelefteq c.dec \trianglerighteq z$ yields the two transitions $\delta(x, c.dec, Z) = \{(z, 0)\}$ and $\delta(x, c.dec, B) = \{(y, -1)\}$, where the -1 indicates that the counter value decreases,
- each equation $x = y \trianglelefteq a \trianglerighteq z$ for $a \notin \{d, t, c.inc, c.dec\}$ yields the four transitions $\delta(x, a, Z) = \delta(x, a, B) = \{(y, 0), (z, 0)\}$.

The result by Rosier and Yen states that if $(q, a, n) \vdash^{\sigma} (q', b, n')$ where q is the current state, a the current input symbol of Σ and n the current value of the counter and the symbol $\vdash^{\sigma}$ indicates that (q, a, n) has a computation that leads to (q', b, n') while reading σ, then $(q, a, n) \vdash^{\rho} (q', b, n')$ where

- the set of transitions used in $\vdash^{\sigma}$ is identical to the set of transitions used in $\vdash^{\rho}$, and
- every configuration (r, c, m) in $\vdash^{\rho}$ satisfies $m \leq 3 \cdot s^3 + \max(n, n')$, where s is the number of transitions defined by δ.

This result and the above transformation imply the result stated in the lemma. $\dashv$

Theorem 1 *Let E be a finite linear specification with $x, y \in \mathit{Var}(E)$. If*

$$x /_c \mathcal{C}(n) \overset{\sigma}{\twoheadrightarrow} y /_c \mathcal{C}(m)$$

then

$$x /_c \mathcal{C}(n) \overset{\rho}{\twoheadrightarrow} y /_c \mathcal{C}(m)$$

for some ρ such that $\mathit{alphabet}(\sigma) = \mathit{alphabet}(\rho)$ and the counter values in $\overset{\rho}{\twoheadrightarrow}$ do not exceed $3 \cdot (4 \cdot |\mathit{Var}(E)|)^3 + \max(n, m)$.

Proof. Immediate by the proof of Lemma 1 and observation (1). $\dashv$

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 [7, 4]. 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 recursive specification

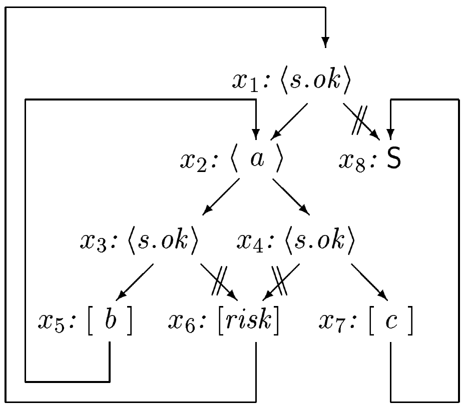
$$E = \{x = t_x \mid x \in \mathit{Var}(E)\},$$

where $\mathit{Var}(E)$ is a finite set of variables. There are two special actions among the actions of E. First, there is the action *risk* representing behaviour that poses a security risk. Secondly, there is the action $s.ok$ with s a channel name and ok a request to a *risk assessment service*. We call a variable $x \in \mathit{Var}(E)$ with t_x of the form $y \trianglelefteq s.ok \trianglerighteq z$ a *test state* with true-branch y and false-branch z. 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 [4]).

Example 2 *Suppose E is defined by the eight equations below. Then, in the regular thread x_1 the various $s.ok$ tests are evaluated as follows:*

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

This situation can be illustrated as follows:



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

Definition 4 *The set $Risk(E) \subseteq Var(E)$ of* risk states *of E is defined inductively as the least set satisfying, for all $x \in Var(E)$,*

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

Membership of $Risk(E)$ is decidable since $Var(E)$ is finite.

For instance, for E as defined in Example 2, $Risk(E) = \{x_6\}$. 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 Section 2.2.[3] 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.

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

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

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

We can now state the property of risk states mentioned above: in risk states possible future execution of *risk* cannot be avoided by $\mathcal{S}(E)$.

Lemma 2 *Let E be a finite linear specification. Then*

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

[3]In [4], 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.

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

The next theorem states that this 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, $risk \notin actions(x /_s \mathcal{S}(E))$.

Theorem 2 *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 $risk \in actions(x/_s\mathcal{S}(E))$ if and only if $x \in Risk(E)$. The if-part follows from Lemma 2. For the only-if-part, suppose that $risk \in actions(x/_s\mathcal{S}(E))$. 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 $x_1 = y \trianglelefteq risk \trianglerighteq z$ 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 $x_k \in Risk(E)$.

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

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

Clearly, the thread $T = x_1 /_s \mathcal{S}(E)$ as defined and depicted in Example 2 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 Section 3.1. For the definition of a risk assessment service for $E_{\mathcal{C}}$ we exploit Lemma 1. 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 a finite linear specification E to E' by replacing each equation

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

for some x' not in $Var(E)$. It is easily seen that a $risk$-free trace σ from a state $x(n)$ to a state $x''(m)$ in which $risk$ can be performed exists in $E_{\mathcal{C}}$ if and only if $x(n)$ has a $risk$-free trace $\sigma\rho$ in $E'_{\mathcal{C}}$ from $x(n)$ to $x'''(0)$ in which $risk$ can be performed and where ρ consists of zero or more $c.dec$ actions.

Thus, in the remainder of this section, 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. Given such a specification E, it follows from Lemma 1 that for any state $x(n)$ of $E_{\mathcal{C}}$ there is a certain value

$$N_n = 3 \cdot (4 \cdot |\, Var(E)|)^3 + n$$

such that any *risk*-free trace to a state in which *risk* can be performed, has an associated trace in which the counter value does not exceed N_n. So, for risk assessment it is sufficient to construct the finite approximation $\pi(N_n, E_{\mathcal{C}})$ of the infinite specification $E_{\mathcal{C}}$ (see Definition 3). For this finite linear specification membership of *Risk* is decidable. From now on we abbreviate $\pi(N_n, E_{\mathcal{C}})$ to $E^n_{\mathcal{C}}$.

Now, 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

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

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

Theorem 3 *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 $E_{\mathcal{C}}$,*

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

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

Proof. The if-part follows from Lemma 2. 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 1, $n_i \leq N_n$ 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 that $x_{k-1}(n_{k-1}) \in Risk(E^n_{\mathcal{C}})$. By (*): $x_k = y \trianglelefteq a \trianglerighteq z$ 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 $x_{k-1}(n_{k-1}) \in \mathit{Risk}(E^n_\mathcal{C})$ 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 $y(n_k) \in \mathit{Risk}(E^n_\mathcal{C})$ and $x_{k-1} = z$. So both $y(n_k) \in \mathit{Risk}(E^n_\mathcal{C})$ and $z(n_k) \in \mathit{Risk}(E^n_\mathcal{C})$ and therefore by definition of $\mathit{Risk}(E^n_\mathcal{C})$ also $x_k(n_k) \in \mathit{Risk}(E^n_\mathcal{C})$. $\dashv$

Our final 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 $\mathcal{C}(n)$. Adopting 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 3, the risk assessment service $\mathcal{S}(E_\mathcal{C})$ provides the correct reply to $x(n)$ as defined by $E_\mathcal{C}$. Because in the infinite specification $E_\mathcal{C}$ all equations that contain counter actions are deterministic, it follows that this reply is also correct for $x(n) /_c \mathcal{C}(n)$, 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 3 decidable and correct.

5 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 [7]). Secondly, exploiting a result of Rosier and Yen [12] 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 [4]). 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 such 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 (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 [3]),

2. Inclusion is undecidable (see also [3]), and

3. Reachability is preserved under bounded counter values (Theorem 1 in the present paper).

The following refinement of the last property may be interesting in its own right. In various papers on thread algebra (e.g., in [3]), a more refined notion of reachability is used, namely the relation $\overset{\sigma}{\twoheadrightarrow}$ as the reflexive transitive closure of action relations

$$x \trianglelefteq a \trianglerighteq y \xrightarrow{aT} x \quad \text{and} \quad x \trianglelefteq a \trianglerighteq y \xrightarrow{aF} y$$

which also record the reply value `true` respectively `false`. It is trivial to adapt the proof of Theorem 1 to this notion of traces and this does not change the number of transitions of the associated ω-1CM. Hence:

Corollary 1 *Let E be a finite linear specification with $x, y \in Var(E)$. If*

$$x /_c \mathcal{C}(n) \overset{\sigma}{\twoheadrightarrow} y /_c \mathcal{C}(m)$$

where σ is a trace in $\{aT, aF \mid a \in A\}^*$, then

$$x /_c \mathcal{C}(n) \overset{\rho}{\twoheadrightarrow} y /_c \mathcal{C}(m)$$

for some ρ such that $alphabet(\sigma) = alphabet(\rho)$ and the counter values in $\overset{\rho}{\twoheadrightarrow}$ do not exceed $3 \cdot (4 \cdot |Var(E)|)^3 + \max(n, m)$.

References

[1] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1-2):70–120,1982.

[2] J.A. Bergstra and I. Bethke. Polarized process algebra and program equivalence. In J.C.M. Baeten, J.K. Lenstra, J. Parrow, G.J. Woeginger, eds., *Proceedings of ICALP 2003*, Springer-Verlag, LNCS 2719:1–21, 2003.

[3] J.A. Bergstra, I. Bethke, and A. Ponse. Decision problems for pushdown threads. Electronic report PRG0502, Programming Research Group, University of Amsterdam, June 2005.

[4] J.A. Bergstra, I. Bethke, and A. Ponse. Thread algebra and risk assessment services. *Proceedings of the Logic Colloquium 2005*, Athens, Association for Symbolic Logic, to appear.

[5] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[6] J.A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming* 51(2):175-192, 2002.

[7] J.A. Bergstra and A. Ponse. A bypass of Cohen's impossibility result. In P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, eds., *Advances in grid computing – EGC 2005*, Springer-Verlag, LNCS 3470:1097–1106, 2005.

[8] F. Cohen. Computer viruses — theory and experiments. *Computers & Security*, 6(1):22–35, 1984. Available as `http://vx.netlux.org/lib/afc01.html`.

[9] P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. *Proceedings of SOFSEM'99: The 26th Seminar on Current Trends in Theory and Practice of Informatics*, Springer-Verlag, LNCS 1725:398-407, 1999.

[10] M.S. Paterson and L.G. Valiant. Deterministic one-counter automata. *Journal of Computer and System Science*, 10(3):340-350, 1975.

[11] A. Ponse and M.B. van der Zwaag. An introduction to program and thread algebra. In A. Beckmann et al. (editors), *Logical Approaches to Computational Barriers: Proceedings CiE 2006*, LNCS 3988, pages 445-458, Springer-Verlag, 2006.

[12] L.E. Rosier and H.-C. Yen. Logspace hierarchies, polynomial time and the complexity of fairness problems concerning ω-machines. *SIAM Journal on Computing*, 16(5):779–807, 1987.

[13] T.D. Vu. Metric denotational semantics for BPPA. Report PRG0503, Programming Research Group, University of Amsterdam, July 2005. Available at `www.science.uva.nl/research/prog/publications.html`.

[14] H.-C. Yen and L.-P. Yu. Decidability analysis of self-stabilization for infinite state systems. *Fundamenta Informaticae*, 70(4):387–402, 2006.