One-Counter Threads

Reachability and Action Forecasting

Alban Ponse

Section Software Engineering
Informatics Institute
University of Amsterdam

PAM - June 13, 2007

Contents

1. Basics
2. Services and one-counter threads
3. Action forecasting (including risk assessment)
4. Conclusion, digression and discussion (PAM's future?)

1. Basics

Given a set A of actions, basic thread algebra (BTA) has the following constants and operators:

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

Execution of an action yields a reply value `true` or `false`.

The postconditional composition $P \trianglelefteq a \trianglerighteq Q$ represents action a followed by thread P if `true` was replied, and a followed by Q otherwise.

Action prefix: $a \circ P \stackrel{\text{def}}{=} P \trianglelefteq a \trianglerighteq P$

Action prefix binds stronger than postconditional composition.

The approximation operator $\pi_n(\_)$ gives the behavior of a thread up to depth n ($n \in \mathbb{N}$).

1. $\pi_0(P) = \mathsf{D}$
2. $\pi_{n+1}(\mathsf{S}) = \mathsf{S}$
3. $\pi_{n+1}(\mathsf{D}) = \mathsf{D}$
4. $\pi_{n+1}(P \trianglelefteq a \trianglerighteq Q) = \pi_n(P) \trianglelefteq a \trianglerighteq \pi_n(Q)$

Example: $\pi_2(b \circ c \circ \mathsf{S} \trianglelefteq a \trianglerighteq \mathsf{S}) = b \circ \mathsf{D} \trianglelefteq a \trianglerighteq \mathsf{S}$

Every thread in BTA is finite: there is a finite upper bound to the number of consecutive actions it can perform.
So, for every $P \in$ BTA there exists $n \in \mathbb{N}$ such that

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

Infinite threads

We define BTA^∞, the set of projective sequences of BTA terms:

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

We turn the set BTA^∞ into an algebra by defining operations on it. Overloading notation, let

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

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

Regular threads

Informally, a thread is regular if it has finitely many *states*.

The regular threads are exactly the threads that can be defined by a finite linear recursive specification, i.e., a set of equations

$$x_i = t_i$$

for $i \in I$ with I some finite index set, variables x_i, and all t_i terms of the form S, D, or $x_j \trianglelefteq a \trianglerighteq x_k$ with $j, k \in I$.

Fact

- *Variables in these specifications have unique solutions (fixed points).*
- *The finite threads form a proper subset of the regular threads, which form a proper subset of BTA^{∞}.*

Convention

We shall identify variables in linear recursive specifications and their fixed points.

For example, we say that P is the thread defined by $P = a \circ P$ instead of stating that P equals the fixed point for x in the specification $\{x = a \circ x\}$.

Convention

We shall identify variables in linear recursive specifications and their fixed points.

For example, we say that P is the thread defined by $P = a \circ P$ instead of stating that P equals the fixed point for x in the specification $\{x = a \circ x\}$.

Example

We define regular thread P by

$$
\begin{aligned}
P &= Q \trianglelefteq a \trianglerighteq R \\
Q &= b \circ P \\
R &= T \trianglelefteq c \trianglerighteq P \\
T &= \mathsf{S}
\end{aligned}
$$

Note the finite *graphical representation* of P. [Draw on blackboard]

2. Services and one-counter threads

We have assumed that a thread is executed in an environment that supplies reply values for actions.
We can model (part of) this environment as one or more services.

A typical example of such a service is a stack: for $n \in \mathbb{N}$, $\mathcal{S}_n$ is a service that

- holds a value in $\{0, \ldots, n\}^*$,
- and is controlled by $2n + 3$ methods ($i \leq n$):

  push:i pushes i onto the stack and yields `true`,

  topeq:i tests whether i is on top of the stack,

  pop pops the stack with reply `true` if it is non-empty, and yields `false` otherwise (while the stack contents is preserved).

We write $\mathcal{S}_n(\alpha)$ for a stack with contents $\alpha \in \{0, \ldots, n\}^*$, and initially the stack is empty ($\mathcal{S}_n = \mathcal{S}_n(\epsilon)$ with ϵ the empty sequence).

Formally:

A service $\mathcal{H}$ is a pair $\langle M, F \rangle$ consisting of

- a set M of so-called methods, and
- a reply function F.

The reply function is a mapping that gives for each non-empty finite sequence of methods from M a reply `true` or `false`.

On input $m_1 \dots m_{k+1}$, function F gives the reply for m_{k+1} if $m_1, \dots, m_k$ (the history) were called before.
Write $\mathcal{H}_\nu$ for $\mathcal{H}$ with history ν.

Formally:

A service $\mathcal{H}$ is a pair $\langle M, F \rangle$ consisting of

- a set M of so-called methods, and
- a reply function F.

The reply function is a mapping that gives for each non-empty finite sequence of methods from M a reply `true` or `false`.

On input $m_1 \dots m_{k+1}$, function F gives the reply for m_{k+1} if $m_1, \dots, m_k$ (the history) were called before.
Write $\mathcal{H}_\nu$ for $\mathcal{H}$ with history ν.

The notation $\mathcal{S}_n(\alpha)$ is convenient and adequate, but not history-based.

Formally:

A service $\mathcal{H}$ is a pair $\langle M, F\rangle$ consisting of

- a set M of so-called methods, and
- a reply function F.

The reply function is a mapping that gives for each non-empty finite sequence of methods from M a reply `true` or `false`.

On input $m_1 \ldots m_{k+1}$, function F gives the reply for m_{k+1} if $m_1, \ldots, m_k$ (the history) were called before.
Write $\mathcal{H}_\nu$ for $\mathcal{H}$ with history ν.

The notation $\mathcal{S}_n(\alpha)$ is convenient and adequate, but not history-based.

Focus-method notation: Let actions be of the form $f.m$ where f is the focus, and m is the method.

E.g., $st.pop$ denotes the action which pops a stack via focus st.

Use-operator

$P \mathbin{/_f} \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} \mathbin{/_f} \mathcal{H}_\nu = \mathsf{S}$

Use-operator

$P /_f \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} /_f \mathcal{H}_\nu = \mathsf{S}$
2. $\mathsf{D} /_f \mathcal{H}_\nu = \mathsf{D}$

Use-operator

$P /_f \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} /_f \mathcal{H}_\nu = \mathsf{S}$
2. $\mathsf{D} /_f \mathcal{H}_\nu = \mathsf{D}$
3. $(P \trianglelefteq g.m \trianglerighteq Q) /_f \mathcal{H}_\nu = (P /_f \mathcal{H}_\nu) \trianglelefteq g.m \trianglerighteq (Q /_f \mathcal{H}_\nu) \quad \text{if } g \neq f$

Use-operator

$P /_f \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} /_f \mathcal{H}_\nu = \mathsf{S}$
2. $\mathsf{D} /_f \mathcal{H}_\nu = \mathsf{D}$
3. $(P \trianglelefteq g.m \trianglerighteq Q) /_f \mathcal{H}_\nu = (P /_f \mathcal{H}_\nu) \trianglelefteq g.m \trianglerighteq (Q /_f \mathcal{H}_\nu)$ if $g \neq f$
4. $(P \trianglelefteq f.m \trianglerighteq Q) /_f \mathcal{H}_\nu = \mathsf{D}$ if $m \notin M$

Use-operator

$P /_f \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} /_f \mathcal{H}_\nu = \mathsf{S}$
2. $\mathsf{D} /_f \mathcal{H}_\nu = \mathsf{D}$
3. $(P \trianglelefteq g.m \trianglerighteq Q) /_f \mathcal{H}_\nu = (P /_f \mathcal{H}_\nu) \trianglelefteq g.m \trianglerighteq (Q /_f \mathcal{H}_\nu)$ if $g \neq f$
4. $(P \trianglelefteq f.m \trianglerighteq Q) /_f \mathcal{H}_\nu = \mathsf{D}$ if $m \notin M$
5. $(P \trianglelefteq f.m \trianglerighteq Q) /_f \mathcal{H}_\nu = P /_f \mathcal{H}_{\nu m}$ if $m \in M$ and $F(\nu m) = \mathtt{true}$

Use-operator

$P /_f \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} /_f \mathcal{H}_\nu = \mathsf{S}$
2. $\mathsf{D} /_f \mathcal{H}_\nu = \mathsf{D}$
3. $(P \trianglelefteq g.m \trianglerighteq Q) /_f \mathcal{H}_\nu = (P /_f \mathcal{H}_\nu) \trianglelefteq g.m \trianglerighteq (Q /_f \mathcal{H}_\nu)$ if $g \neq f$
4. $(P \trianglelefteq f.m \trianglerighteq Q) /_f \mathcal{H}_\nu = \mathsf{D}$ if $m \notin M$
5. $(P \trianglelefteq f.m \trianglerighteq Q) /_f \mathcal{H}_\nu = P /_f \mathcal{H}_{\nu m}$ if $m \in M$ and $F(\nu m) = \texttt{true}$
6. $(P \trianglelefteq f.m \trianglerighteq Q) /_f \mathcal{H}_\nu = Q /_f \mathcal{H}_{\nu m}$ if $m \in M$ and $F(\nu m) = \texttt{false}$

Use-operator

$P \,/_f\, \mathcal{H}_\nu$ models thread P using the service $\mathcal{H}_\nu$ via focus f.

Let $\mathcal{H} = \langle M, F \rangle$. We define for threads in BTA:

1. $\mathsf{S} \,/_f\, \mathcal{H}_\nu = \mathsf{S}$
2. $\mathsf{D} \,/_f\, \mathcal{H}_\nu = \mathsf{D}$
3. $(P \trianglelefteq g.m \trianglerighteq Q) \,/_f\, \mathcal{H}_\nu = (P \,/_f\, \mathcal{H}_\nu) \trianglelefteq g.m \trianglerighteq (Q \,/_f\, \mathcal{H}_\nu)$ if $g \neq f$
4. $(P \trianglelefteq f.m \trianglerighteq Q) \,/_f\, \mathcal{H}_\nu = \mathsf{D}$ if $m \notin M$
5. $(P \trianglelefteq f.m \trianglerighteq Q) \,/_f\, \mathcal{H}_\nu = P \,/_f\, \mathcal{H}_{\nu m}$ if $m \in M$ and $F(\nu m) = \texttt{true}$
6. $(P \trianglelefteq f.m \trianglerighteq Q) \,/_f\, \mathcal{H}_\nu = Q \,/_f\, \mathcal{H}_{\nu m}$ if $m \in M$ and $F(\nu m) = \texttt{false}$

The use operator is expanded to infinite threads in BTA$^\infty$ by defining

$$(P_n)_{n\in\mathbb{N}} \,/_f\, \mathcal{H}_\nu = \bigsqcup_{n\in\mathbb{N}} P_n \,/_f\, \mathcal{H}_\nu$$

(For P defined by a linear specification, $/_f \mathcal{H}_\nu$ works nice and easy...)

One-counter threads

A counter service $\mathcal{C}$ holds a value in $\mathbb{N}$ (determined by its history) and is controlled by 2 methods:

inc increases the value of the counter and yields `true`,

dec decreases the value of the counter with reply `true` if it is positive, and yields `false` otherwise (while the counter value remains 0).

We write $\mathcal{C}(n)$ for a counter with value n, and initially the counter has value 0 ($\mathcal{C} = \mathcal{C}(0)$).

One-counter threads

A counter service $\mathcal{C}$ holds a value in $\mathbb{N}$ (determined by its history) and is controlled by 2 methods:

inc increases the value of the counter and yields `true`,

dec decreases the value of the counter with reply `true` if it is positive, and yields `false` otherwise (while the counter value remains 0).

We write $\mathcal{C}(n)$ for a counter with value n, and initially the counter has value 0 ($\mathcal{C} = \mathcal{C}(0)$).

A one-counter thread is a regular thread that uses a single counter.

Examples

1. $(c.inc \circ P) /_c \mathcal{C}(n) = P /_c \mathcal{C}(n{+}1)$
2. $(P \trianglelefteq c.dec \trianglerighteq S) /_c \mathcal{C}(0) = S$
3. $(P \trianglelefteq c.dec \trianglerighteq S) /_c \mathcal{C}(n{+}1) = P /_c \mathcal{C}(n)$

Obtaining non-regularity using a counter

Consider the regular thread

$$Q = c.inc \circ Q \trianglelefteq a \trianglerighteq R, \quad R = b \circ R \trianglelefteq c.dec \trianglerighteq \mathrm{S},$$

where actions a and b do not use focus c.

Obtaining non-regularity using a counter

Consider the regular thread

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

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

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

Obtaining non-regularity using a counter

Consider the regular thread

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

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

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

$$R /_c \mathcal{C}(n) = \begin{cases} b \circ R /_c \mathcal{C}(n-1) & \text{if } n > 0 \\ \mathsf{S} & \text{otherwise.} \end{cases}$$

Obtaining non-regularity using a counter

Consider the regular thread

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

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

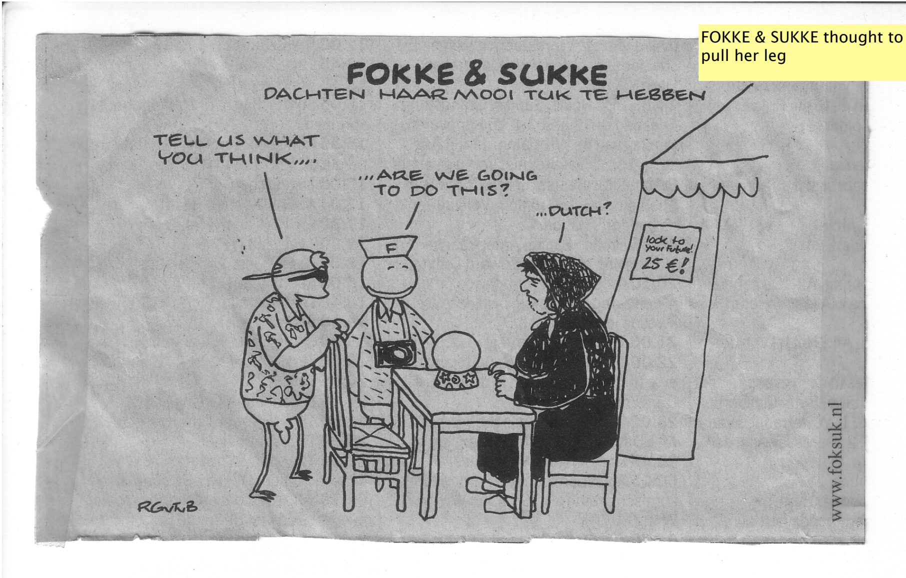
$$\begin{aligned} Q /_c \mathcal{C}(n) &= (c.inc \circ Q \trianglelefteq a \trianglerighteq R) /_c \mathcal{C}(n) \\ &= (Q /_c \mathcal{C}(n+1)) \trianglelefteq a \trianglerighteq (R /_c \mathcal{C}(n)) \end{aligned}$$

$$R /_c \mathcal{C}(n) = \begin{cases} b \circ R /_c \mathcal{C}(n-1) & \text{if } n > 0 \\ \mathsf{S} & \text{otherwise.} \end{cases}$$

So $Q /_c \mathcal{C}(0)$ is an infinite thread such that a trace of $n+1$ a's produced by n positive and one negative reply on a is followed by $b^n \circ \mathsf{S}$.

This yields a non-regular thread: the one-counter thread $Q /_c \mathcal{C}(0)$ is not definable by a *finite* linear recursive specification.

3. Action forecasting



FOKKE & SUKKE thought to pull her leg

Risk assessment

Risk assessment is the forecast that a certain action that models risky behavior (viruses etc.) will NOT be executed:

The test action *s.ok* in $P \trianglelefteq s.ok \trianglerighteq Q$ yields `true` if the action *risk* is not executed in *P* (its true-branch), and `false` otherwise

Risk assessment

Risk assessment is the forecast that a certain action that models risky behavior (viruses etc.) will NOT be executed:

The test action $s.ok$ in $P \unlhd s.ok \unrhd Q$ yields `true` if the action *risk* is not executed in P (its true-branch), and `false` otherwise

We shall model this as a thread-service composition

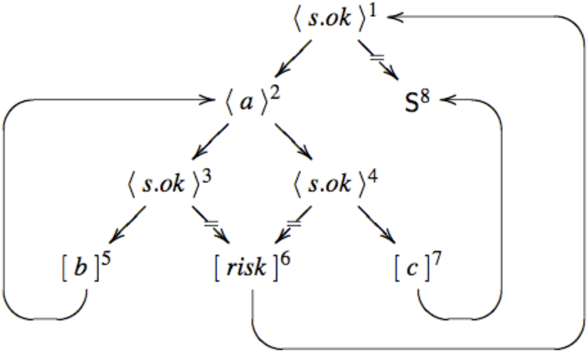
$$(P \unlhd s.ok \unrhd Q) /_s \mathcal{S}(E)$$

where the risk assessment service $\mathcal{S}(E)$

- has *ok* as its only method, and
- is aware of both the specification E that defines $P \unlhd s.ok \unrhd Q$ and the current execution state.

Risk assessment is non-trivial if the test action $s.ok$ occurs more than once in P, the thread to be assessed.

An example of risk assessment



Here the superscripts on states relate to a finite linear specification E:
$P_1 = P_2 \unlhd s.ok \unrhd P_8, \ldots, P_8 = \mathrm{S}$, and

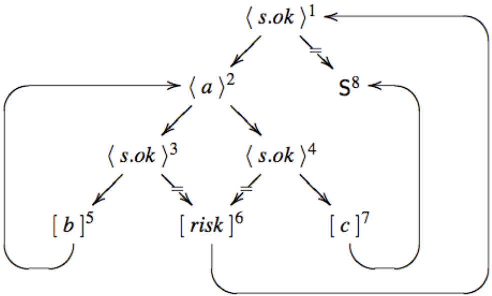
$$P_1 /_s \mathcal{S}(E) = \mathcal{T} \quad \text{with} \quad \mathcal{T} = b \circ \mathcal{T} \unlhd a \unrhd c \circ \mathrm{S}$$

Risk states

From risk states the execution of action *risk* cannot be avoided: for any equation in a given finite linear spec. E,

if	$x = y \unlhd risk \unrhd z$	then x is a risk state,
if	$x = y \unlhd s.ok \unrhd z$	and both y, z are risk states, then so is x,
if	$x = y \unlhd a \unrhd z$	and y or z is a risk state, then so is x.

In the example, P_6 is the only risk state:

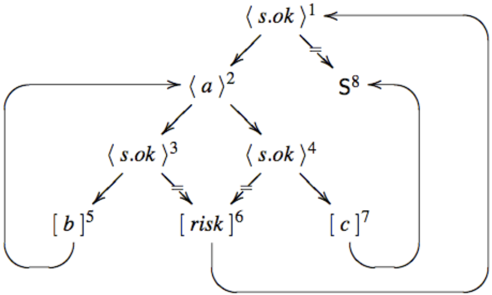


A risk assessment service for regular threads

From risk states the execution of *risk* cannot be avoided: a risk assessment $\mathcal{S}(E)$ should reply `true` to *s.ok* in

$$x = y \trianglelefteq s.ok \trianglerighteq z \qquad (1)$$

iff y is not a risk state. This can be resolved for any finite linear specification E: annotate *s.ok* to *s.ok:y* for equations of form (1). In the example, the crossed-out arrows illustrate this:



Risk assessment and Cohen's result

Observations:

1. The test action $s.ok$ is interpreted in the context of a postconditional composition (a thread specification E) and a resolving risk assessment service $\mathcal{S}(E)$.
2. The reply `false` to $s.ok$ in $P \trianglelefteq s.ok \trianglerighteq Q$ gives no clue about the execution of risk in Q.

Risk assessment and Cohen's result

Observations:

1. The test action $s.ok$ is interpreted in the context of a postconditional composition (a thread specification E) and a resolving risk assessment service $\mathcal{S}(E)$.
2. The reply `false` to $s.ok$ in $P \trianglelefteq s.ok \trianglerighteq Q$ gives no clue about the execution of risk in Q.

This brings us to a comparison with Cohen's seminal impossibility result on virus detection (1984), which in our setting reads:

> *There exists no predicate D on all programs (in a reasonable class) that determines whether a virus (cf. our action $risk$) is executed.*

Proof. Existence is contradicted by the program P defined by
$P = \texttt{if } D(P) \texttt{ then } \langle\text{safe behavior}\rangle \texttt{ else } \langle\text{spread virus}\rangle$.

First conclusions and a question

Some first conclusions:

1. While risk detection is impossible (à la Cohen), risk assessment is possible for regular threads.
2. Risk assessment is defined in terms of a test *s.ok* (using a r.a. service under focus *s*) that forecasts the absence of *risk* in its true-branch, resisting the form of self-reference used by Cohen.

First conclusions and a question

Some first conclusions:

1. While risk detection is impossible (à la Cohen), risk assessment is possible for regular threads.
2. Risk assessment is defined in terms of a test *s.ok* (using a r.a. service under focus *s*) that forecasts the absence of *risk* in its true-branch, resisting the form of self-reference used by Cohen.

Question. Up to which class of threads is risk assessment decidable?

Risk assessment for one-counter threads

Ponse & van der Zwaag (2006): Risk assessment is decidable for one-counter threads (to appear in ToCS).

This follows from a reachability result of Rosier and Yen (1987):

1. Let P be defined by finite linear specification E. If

$$P \mathbin{/_c} \mathcal{C}(n) \xrightarrow{\sigma} Q \mathbin{/_c} \mathcal{C}(m)$$

   then, for some ρ, $P \mathbin{/_c} \mathcal{C}(n) \xrightarrow{\rho} Q \mathbin{/_c} \mathcal{C}(m)$ with

   - *labels*$(\rho) =$ *labels*(σ), and
   - every intermediate state $R \mathbin{/_c} \mathcal{C}(n')$ satisfies

$$n' \leq 3(4|Var(E)|)^3 + \max(n, m).$$

2. E can be adapted so that *risk* is only performed at counter value 0 ($m = 0$).
3. Then, wrt. risk assessment, $P \mathbin{/_c} \mathcal{C}(n)$ can be faithfully approximated by a finite linear specification.

Pushdown threads

Pushdown thread: a regular thread that uses a stack.

Example (a pd thread, not an oc thread)

$x_1 \ /_{st} \ \mathcal{S}_1(\epsilon)$ with $\alpha \in \{0,1\}^*$ the contens of $\mathcal{S}_1(\alpha)$ and ϵ the empty sequence), and

$$x_1 = st.push{:}0 \circ x_1 \trianglelefteq a \trianglerighteq x_2, \quad x_3 = c \circ st.pop \circ x_3 \trianglelefteq st.topeq{:}1 \trianglerighteq x_4,$$
$$x_2 = st.push{:}1 \circ x_2 \trianglelefteq b \trianglerighteq x_3, \quad x_4 = d \circ st.pop \circ x_4 \trianglelefteq st.topeq{:}0 \trianglerighteq \mathrm{S}.$$

Pushdown threads

Pushdown thread: a regular thread that uses a stack.

Example (a pd thread, not an oc thread)

$x_1 /_{st} \mathcal{S}_1(\epsilon)$ with $\alpha \in \{0,1\}^*$ the contens of $\mathcal{S}_1(\alpha)$ and ϵ the empty sequence), and

$$x_1 = st.push{:}0 \circ x_1 \trianglelefteq a \trianglerighteq x_2, \quad x_3 = c \circ st.pop \circ x_3 \trianglelefteq st.topeq{:}1 \trianglerighteq x_4,$$
$$x_2 = st.push{:}1 \circ x_2 \trianglelefteq b \trianglerighteq x_3, \quad x_4 = d \circ st.pop \circ x_4 \trianglelefteq st.topeq{:}0 \trianglerighteq \mathsf{S}.$$

Open question. Is risk assessment decidable for pushdown threads?

The proof for one-counter threads does not generalize: control decisions may occur at any stack contents (tests on identity of top value).

Other forms of forecasting

- Turing (1937): The Halting Problem, i.e.,

  > *Undecidability (unsolvability) of the question whether a Turing Machine halts on a certain input.*

  (This question can be modelled as a thread-service composition).
- Bergstra & Ponse (J'nal of Appl. Logic 5, 2007):
  - Forecasting Reactors: services that need a third truth value to escape paradoxes and give preference to reply `true`.
  - Rational Agents: services that intend to achieve an objective given a thread to be executed (e.g., get another service in an "optimal state").
  - Execution architectures (modelling threads & services) in which a service may be a forecaster of another one (Example: Newcomb Paradox).
- Goal assessment (with Mark van der Zwaag): decidable for one-counter threads.

Other results

Computable threads:

- Risk assessment is undecidable for computable threads (cf. the Halting Problem).

Pushdown threads:

- Equality is decidable.
- In risk assessment, recurrence of $s.ok$ is the difficult issue: if this is not the case, $s.ok$ yields `true` iff

$$(P \trianglelefteq s.ok \trianglerighteq Q) /_{st} \mathcal{S}_n(\alpha) = (\overline{P} \trianglelefteq s.ok \trianglerighteq Q) /_{st} \mathcal{S}_n(\alpha)$$

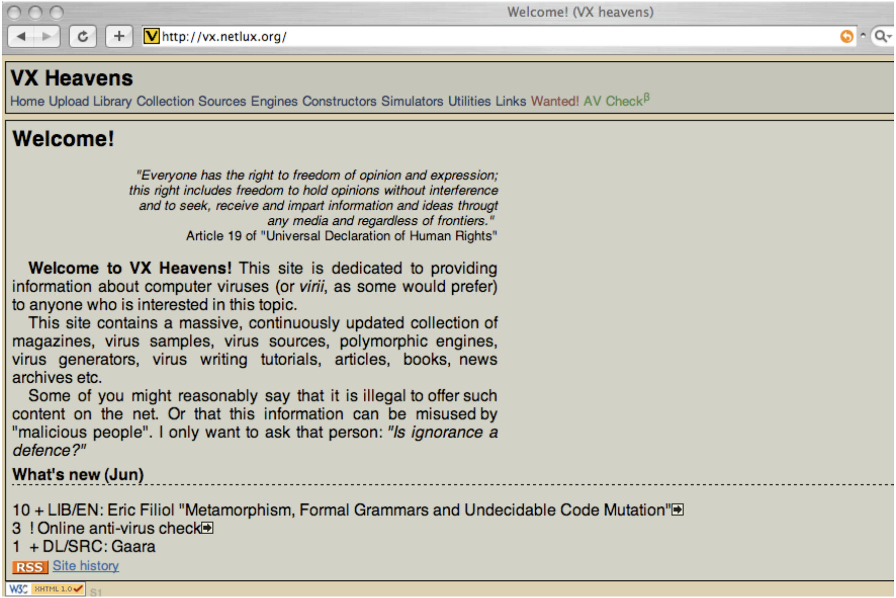
with in $\overline{P}$ all occurrences of *risk* replaced by a different action; this is decidable.

One-counter threads:

- Inclusion ($\sqsubseteq$) is undecidable.
- State reachability is preserved under bounded counter values.
- State reachability is decidable.

4. Conclusion, digression and discussion

- Students like to program threads, also the secondary school ones doing our Webklas Informatica Wat is een programma?
  - The simple concepts in both program algebra and thread algebra appear to be appealing
  - Thread algebra (nice, compositional) can be seen/used as a semantics for program algebra (non-compositional, common programming constructs)
- Risk assessment: we made it to VX Heavens (site on viruses, on-line since Sept. 1999, some pictures on the next slides) in the category Theory, models and definitions (25 papers).
- Some advanced work in program and thread algebra:
  - Micro grids (concurrent hardware)
  - Tool set for PGA (including animation and multi-threading tools)
  - Predictable and Reliable Program Code: Virtual Machine-based Projection Semantics
  - Maurer computers (finite computer models) with pipelined instruction processing



Welcome! (VX heavens)

http://vx.netlux.org/

VX Heavens

Home Upload Library Collection Sources Engines Constructors Simulators Utilities Links Wanted! AV Check$^{\beta}$

Welcome!

"Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas througt any media and regardless of frontiers."
Article 19 of "Universal Declaration of Human Rights"

Welcome to VX Heavens! This site is dedicated to providing information about computer viruses (or *virii*, as some would prefer) to anyone who is interested in this topic.

This site contains a massive, continuously updated collection of magazines, virus samples, virus sources, polymorphic engines, virus generators, virus writing tutorials, articles, books, news archives etc.

Some of you might reasonably say that it is illegal to offer such content on the net. Or that this information can be misused by "malicious people". I only want to ask that person: *"Is ignorance a defence?"*

What's new (Jun)

10 + LIB/EN: Eric Filiol "Metamorphism, Formal Grammars and Undecidable Code Mutation"
3 ! Online anti-virus check
1 + DL/SRC: Gaara

RSS Site history

W3C XHTML 1.0 S1

VX Heavens

Home Upload Library Collection Sources Engines Constructors Simulators Utilities Links Wanted! AV Check$^{\beta}$

Library: Jan Bergstra

Jan Bergstra, Alban Ponse «A Bypass of Cohen's Impossibility Result» Σ [Abstract] 36.66Kb

W3C XHTML 1.0 S1

VX Heavens

Home Upload Library Collection Sources Engines Constructors Simulators Utilities Links Wanted! AV Check$^{\beta}$

Library: Alban Ponse

Universiteit van Amsterdam

Jan Bergstra, Alban Ponse «A Bypass of Cohen's Impossibility Result» Σ [Abstract] 36.66Kb
Homepage http://staff.science.uva.nl/~alban/

W3C XHTML 1.0 S1

VX Heavens

Home Upload Library Collection Sources Engines Constructors Simulators Utilities Links Wanted! AV Check$^{\beta}$

A Bypass of Cohen's Impossibility Result

Jan Bergstra, Alban Ponse
Advances in Grid Computing - EGC 2005, LNCS 3470, pages 1097-1106. Springer-Verlag, 2005
ISBN 3-540-26918-5
2005

Download PDF file (106.73Kb)

[Back to index]

TEX size 0 Scale

Extended Version for SSN - 29 November 2004

Jan Bergstra, Alban Ponse
University of Amsterdam, Programming Research Group, Kruislaan 403,
1098 SJ Amsterdam, The Netherlands
www.science.uva.nl/research/prog/

Jan Bergstra
Utrecht University, Department of Philosophy, Heidelberglaan 8,
3584 CS Utrecht, The Netherlands
www.phil.uu.nl/en/

Abstract

Detecting illegal resource access in the setting of network communication or grid computing is similar to the problem of virus detection as put forward by Fred Cohen in 1984. We disucuss Cohen's impossibility result on virus detection, and introduce "risk assessment of security hazards", a notion that is decidable for a large class of program behaviors.

Keywords: Malcode, Program algebra, Polarized process algebra, Virus, Worm.

1 Introduction

PAM's future?

PAM's future:

- At UvA perhaps?
- Which habitual audience?
- More open presentations/discussions?
- PhD-sessions?

Future work at CWI that might be of interest in this respect:

- ...

Other work at UvA's SSE that might be of interest in this respect:

- Process algebra: continuation of research, tool development etc.
- Algebraic specification (meadows, empty sorts & partial operations)