Synthesizing Safe Smart Contracts using Session Types

ANONYMOUS AUTHOR(S)

Abstract

CCS Concepts: •**Software and its engineering** → **General programming languages;** •**Social and professional topics** → *History of programming languages;*

Additional Key Words and Phrases: keyword1, keyword2, keyword3

ACM Reference format:
Anonymous Author(s). 2017. Synthesizing Safe Smart Contracts using Session Types. *PACM Progr. Lang.* 1, 1, Article 1 (January 2017), 4 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 NETWORK SEMANTICS

We define the valid transitions on the entire system, assuming abstract specification of the clients and the (single) server.

$$
\begin{array}{ll}
types : State, Event, TX; \quad i \in Agent & \\
\quad c_i : State & \Sigma_i^c : State \times \text{list of Event} \times (State \times TX) \\
\quad q_i : \text{queue of TX} & \Sigma^s : State \times (Agent \times TX) \longrightarrow (State \times Event) \\
\quad \sigma : State & \\
\quad e : \text{list of Event} &
\end{array}
$$

$$
\frac{(c_i, e, (c_i', tx)) \in \Sigma_i^c \quad \forall i \neq j, c_j = c_j'}{(c, q, \sigma, e) \rightsquigarrow (c', enque_i(q, tx), \sigma, e)} Send
$$

$$
\frac{\Sigma^s(\sigma, (i, peek(q_i)) = (\sigma', e')}{(c, q, \sigma, e) \rightsquigarrow (c, deque_i(q), \sigma', e' :: e)} Perform
$$

A note.

2017. 2475-1421/2017/1-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

1.1 Server Language

We define a language SL whose programs $p \in SL$ has meaning

$$[\![p]\!] \in State \times (Agent \times TX) \rightarrow (State \times Event)$$

$$
\begin{aligned}
\langle cmd \rangle \quad &::= \text{match yield ev } \langle \textit{case list} \rangle \text{ end} \\
&\mid \text{if * then } \langle cmd \rangle \text{ else } \langle cmd \rangle \text{ end} \\
&\mid \text{while * do } \langle cmd \rangle \text{ end} \\
&\mid \text{proc(); } \langle cmd \rangle \\
&\mid \text{fail} \\
&\mid \text{require *; } \langle cmd \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{case list} \rangle &::= \text{\_ => end} \\
&\mid \text{(i, *) => } \langle cmd \rangle \text{; } \langle \textit{case list} \rangle
\end{aligned}
$$

The intention is that `yield` expressions will be the only entry points, and evaluation of such an expression always translates to the sequence:

(1) finish execution with ev as an output event
(2) wait for the network to send a request
(3) inspect the request and continue execution

$$\frac{-}{[\![\text{match yield ev Cs end}]\!] = (\text{match tx Cs end}, ev)}\,Yield$$

$$\frac{-}{[\![\text{if * then C1 else C2}]\!] = [\![C1]\!]}\,Then \qquad \frac{-}{[\![\text{(i, *) => p; Cs}]\!] = [\![p]\!]}\,CaseGo$$

$$\frac{-}{[\![\text{if * then C1 else C2}]\!] = [\![C2]\!]}\,Else \qquad \frac{-}{[\![\text{(i, *) => p; Cs}]\!] = [\![Cs]\!]}\,CaseDrop$$

$$\frac{-}{[\![\text{while * do p end; p'}]\!] = [\![p']\!]}\,WhileDone \qquad \frac{[\![p]\!] = (p', ev)}{[\![\text{while * do p end}]\!] = (\text{p'; while * do p end}, ev)}\,While$$

$$E : \text{Append-only list}$$
$$R_i : \text{A Queue for actor } i$$

$$\begin{array}{lcl}
\langle cmd \rangle & ::= & \text{publish } \langle V \rangle \\
& | & \text{yield; take } \langle T \rangle \\
& | & \text{if * then } \langle cmdList \rangle \text{ else } \langle cmdList \rangle \\
& | & \text{while * do } \langle cmdList \rangle
\end{array}$$

$$\frac{-}{(E, (i, L, M) :: R, \Sigma, \text{take L}.P) \rightsquigarrow (E, R, \Sigma, P)}\,TAKE \qquad \frac{\Sigma \vdash v \rightsquigarrow x}{(E, R, \Sigma, \text{publish v}.P) \rightsquigarrow (x :: E, R, \Sigma, P)}\,PUB$$

$$\frac{L' \neq L}{(E, (i, L', M) :: R, \Sigma, \text{take L}.P) \rightsquigarrow (E, R, \Sigma, \text{take L}.P)}\,DROP \qquad \frac{}{(E, R, \Sigma, \text{yield; take T}.P) \rightsquigarrow (E, R, \Sigma, \text{takes T}.P)}\,YI$$

1.2 Client Language

We define a language CL whose programs $p \in CL$ has meaning

$$[\![p]\!] \in State \times \text{list of Event} \times (State \times TX)$$

$$\begin{array}{lcl}
\langle cmd \rangle & ::= & \text{send } \langle V \rangle\text{: } \langle T \rangle \\
& | & \text{read latest } \langle T \rangle \\
& | & \text{deq } \langle T \rangle \\
& | & \text{if * then } \langle cmdList \rangle \text{ else } \langle cmdList \rangle \\
& | & \text{while * do } \langle cmdList \rangle
\end{array}$$

$$\frac{\forall M', (L, M') \notin E'}{(E'.(L, M).E, R_i, \Phi, \text{read latest L}.P) \rightsquigarrow (E, R_i, \Phi, P)}\,RL \qquad \frac{\Sigma \vdash v \rightsquigarrow x}{(E, R_i, \Phi, \text{send v: L}.P) \rightsquigarrow (E, (i, L, x) :: R_i, \Phi, P)}\,SEND$$

$$\frac{}{((T, M) :: E, R_i, \Phi, \text{deq T}.P) \rightsquigarrow (E, R_i, \Phi, P)}\,DEQ \qquad \frac{-}{-}\,YIELD$$

2 NOTATIONS

We write _ to denote an immaterial value, which is implicitly existentially quantified, and $\perp$ to denote the undefined value. We denote the *size* (number of elements) of a set A by $|A|$. We write $f : A \to B$ and $f : A \rightharpoonup B$ to denote a *total*, respectively, *partial*, function from A to B. We denote the *domain of definition* and *range* of a function $f : A \rightharpoonup B$ by $\mathrm{dom}(f)$ and $\mathrm{range}(f)$, respectively, i.e., $\mathrm{dom}(f) = \{a \in A \mid f(a) \neq \perp\}$ and $\mathrm{range}(f) = \{b \in B \mid \exists a.\ f(a) = b\}$. We write $f : A \rightharpoonup_{fin} B$ to denote that f has a finite domain. We denote the set of natural numbers (including zero) by $\mathbb{N}$. We write $\{m..n\}$, for some $m, n \in \mathbb{N}$, to denote the set of integers $\{i \in \mathbb{N} \mid m \leq i \wedge i \leq n\}$. A *sequence* $\pi = a_1, \ldots, a_n$ *over a set* A is a function $\pi : \{1..n\} \to A$, from $\{1..n\}$, for some $n \in \mathbb{N}$, to A. We denote the *length* of π by $|\pi| = |\mathrm{dom}(\pi)|$, and its ith element, for $i \in \{1..|\pi|\}$, by $\pi(i)$. We denote the *empty sequence* by ϵ, and the concatenation of sequences π_1 and π_2 by $\pi_1 \cdot \pi_2$. We denote the set of sequences over a set A by $\overline{A}$.

REFERENCES