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.

$$types : State, Event, TX; \quad i \in Agent$$

$$\begin{aligned} c_i &: State & \Sigma_i^c &: State \times \text{list of Event} \times (State \times TX) \\ q_i &: \text{queue of TX} & \Sigma^s &: State \times (Agent \times TX) \longrightarrow (State \times Event) \\ \sigma &: State \\ e &: \text{list of Event} \end{aligned}$$

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

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

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

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

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

$$
\begin{array}{lcl}
\langle case\ list \rangle & ::= & \text{\_ => end} \\
 & | & \text{(i, *) => } \langle cmd \rangle\text{; } \langle case\ list \rangle
\end{array}
$$

The intention is that `yield` expressions will be the only entry points, and evaluation of such an expression always translates to the sequence (a)finish execution with ev as an output event (b) wait for the network to send a request (c) inspect the request and continue execution.

$$\frac{-}{[\![\text{match yield ev Cs end}]\!] = (Cs, ev)}\,Yield \qquad \frac{-}{[\![\text{(i, *) => p; Cs}]\!] = [\![p]\!]}\,CaseGo$$

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

$$\frac{-}{[\![\text{if * then C1 else C2}]\!] = [\![C1]\!]}\,Then \qquad \frac{-}{[\![\text{if * then C1 else C2}]\!] = [\![C2]\!]}\,Else$$

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

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

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

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

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 $f : A \rightarrow B$ 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 $\text{dom}(f)$ and $\text{range}(f)$, respectively, i.e., $\text{dom}(f) = \{a \in A \mid f(a) \neq \perp\}$ and $\text{range}(f) = \{b \in B \mid \exists a.\ f(a) = b\}$. 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 $\pi : \{1..n\} \rightarrow A$, from $\{1..n\}$, for some $n \in \mathbb{N}$, to A. We denote the *length* of π by $|\pi| = |\text{dom}(\pi)|$, 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