프로그램 조각별 따로분석의 이론적 틀

이준협 이광근

서울대학교 프로그래밍 연구실 (ROPAS)

풀고자 한 문제

프로그램 조각이 주어졌을 때, 분석을 해 놓고 기다리자!

- 목표: $e_1 \propto e_2$의 결과 어림잡기.
- 가정: e_1이 무엇을 내보내는지 일부만 알고 있음.
- 질문: e_2를 미리 분석해놓고, e_1을 **따로** 분석해서 **합치면** 전체 결과를 **안전하게** 어림잡을 수 있을까?

따로분석을 위한 의미구조를 엄밀하게 정의해보자!

모듈이 있는 언어의 정의

겉모습 (Untyped λ+Modules)

$$
\begin{array}{rrcll}
\text{Identifiers} & x, M & \in & \text{Var} & \\
\text{Expression} & e & \rightarrow & x & \text{value identifier} \\
 & & \mid & \lambda x.e & \text{function} \\
 & & \mid & e\ e & \text{application} \\
 & & \mid & e \propto e & \text{linked expression} \\
 & & \mid & \varepsilon & \text{empty module} \\
 & & \mid & M & \text{module identifier} \\
 & & \mid & \texttt{let}\ x\ e\ e & \text{binding expression} \\
 & & \mid & \texttt{let}\ M\ e\ e & \text{binding module}
\end{array}
$$

속내용 (T, tick에 대해 매개화된)

$$
\begin{array}{rrcl}
\text{Time} & t & \in & T \\
\text{Context} & C & \in & \text{Ctx}(T) \\
\text{Value(Expr)} & v & \in & \text{Val}(T) \triangleq \text{Expr} \times \text{Ctx}(T) \\
\text{Value(Expr/Mod)} & V & \in & \text{Val}(T) \uplus \text{Ctx}(T) \\
\text{Memory} & m & \in & \text{Mem}(T) \triangleq T \xrightarrow{\text{fin}} \text{Val}(T) \\
\text{State} & s & \in & \text{State}(T) \triangleq \text{Ctx}(T) \times \text{Mem}(T) \times T \\
\text{Result} & r & \in & \text{Result}(T) \triangleq (\text{Val}(T) \uplus \text{Ctx}(T)) \times \text{Mem}(T) \times T \\
\text{Tick} & \text{tick} & \in & \text{Tick}(T) \triangleq (\text{State}(T) \times \text{Var} \times \text{Val}(T)) \rightarrow T \\
\text{Context} & C & \rightarrow & [] \\
 & & \mid & (x, t) :: C \\
 & & \mid & (M, C) :: C \\
\text{Value(Expr)} & v & \rightarrow & \langle \lambda x.e, C \rangle
\end{array}
$$

$\text{addr}(C, x)$: C의 가장 위에 있는 x의 주소(시간).

$\text{ctx}(C, M)$: C의 가장 위에 있는 M의 값(환경).

실행의미 (Operational)

$$\boxed{(e, C, m, t) \rightsquigarrow_{\text{tick}} (V, m', t') \text{ or } (e', C', m', t')}$$

$$\text{[ExprVar]}\ \frac{t_x = \text{addr}(C, x) \quad v = m(t_x)}{(x, C, m, t) \rightsquigarrow (v, m, t)} \qquad \text{[Fn]}\ \frac{}{(\lambda x.e, C, m, t) \rightsquigarrow (\langle \lambda x.e, C \rangle, m, t)}$$

$$\text{[AppL]}\ \frac{}{(e_1\ e_2, C, m, t) \rightsquigarrow (e_1, C, m, t)} \qquad \text{[AppR]}\ \frac{(e_1, C, m, t) \rightsquigarrow (\langle \lambda x.e_\lambda, C_\lambda \rangle, m_\lambda, t_\lambda)}{(e_1\ e_2, C, m, t) \rightsquigarrow (e_2, C, m_\lambda, t_\lambda)}$$

$$\text{[AppBody]}\ \frac{(e_1, C, m, t) \rightsquigarrow (\langle \lambda x.e_\lambda, C_\lambda \rangle, m_\lambda, t_\lambda) \quad (e_2, C, m_\lambda, t_\lambda) \rightsquigarrow (v, m_a, t_a)}{(e_1\ e_2, C, m, t) \rightsquigarrow (e_\lambda, (x, t_a) :: C_\lambda, m_a[t_a \mapsto v], \text{tick}((C, m_a, t_a), x, v))}$$

실행의미 (Collecting)

$$\boxed{\text{Step}(A) \triangleq \left\{ \ell \rightsquigarrow_{\text{tick}} \rho, (\rho, \text{tick}) \middle| \frac{A'}{\ell \rightsquigarrow_{\text{tick}} \rho} \wedge A' \subseteq A \wedge (\ell, \text{tick}) \in A \right\}}$$

$$\boxed{[\![e]\!]S \triangleq \text{lfp}(\lambda X.\text{Step}(X) \cup \{((e, s), \text{tick}) | (s, \text{tick}) \in S\})}$$

모르는 정보가 있어도, 불완전한 증명나무를 저장하고 있다.

"합치기"의 정의

정리 (Concrete Linking)

$|[\![e_1]\!]S| \cong S_1 \triangleright S_2$임이 확인됐을 때,

$$|[\![e_1 \propto e_2]\!]S| \cong |S_1 \propto [\![e_2]\!]S_2|$$

이다.

- $|[\![e_1]\!]S|$: e_1이 내보내는 환경이
- $S_1 \triangleright S_2$: 가정된 S_2와 부족했던 S_1으로 쪼개질 수 있다.
- $|\cdot|$: 최종 상태에 관심이 있다.
- $\cong$: "동일한 이름들을 내보내는" 의미구조면 된다.

$\cong$의 정의

C, m : <u>이름</u>(x, M)으로만 접근 가능한 구조들.

$$
\begin{aligned}
\text{Step}_m(G) &\triangleq \{C \xrightarrow{x} t, t | C \in G \wedge t = \text{addr}(C, x)\} \\
&\cup \{C \xrightarrow{M} C', C' | C \in G \wedge C' = \text{ctx}(C, M)\} \\
&\cup \{t \xrightarrow{e} C, C | t \in G \wedge \langle e, C \rangle = m(t)\} \\
\underline{C, m} &\triangleq \text{lfp}(\lambda X.\text{Step}_m(X) \cup \{C\})
\end{aligned}
$$

$\underline{C, m}$: 이름으로 뽑아낼 수 있는 모든 정보를 담은 rooted graph.

$(C_1, m_1) \cong (C_2, m_2) \triangleq \underline{C_1, m_1} \cong \underline{C_2, m_2}$: 그래프가 동형

$\triangleright$, $\propto$의 정의

$(s_1, \text{tick}_1) \triangleright (s_2, \text{tick}_2) \triangleq (s_+, \text{tick}_+)$

- s_+는 $s_1 = (C_1, m_1, t_1)$를 $s_2 = (C_2, m_2, t_2)$에 **끼워넣은** 것.
- tick_+는 $t \in T_1 + T_2$를 $t \in T_1$의 경우 tick_1으로, $t \in T_2$의 경우 **끼워넣기 전** 상태를 바탕으로 tick_2로 증가시키는 함수.

$$\boxed{S \propto A \triangleq \underbrace{\text{lfp}(\lambda X.\text{Step}(X) \cup \overbrace{(S \triangleright A)}^{\text{끼워넣고}})}_{\text{나머지를 채운다}}}$$

도움정리 (Advance)

따로따로 나눌 수 있는 곳에서 출발하면, 미리 하고 합친 것과 같다.

$$[\![e]\!](S_1 \triangleright S_2) = S_1 \propto [\![e]\!]S_2$$

분석을 위해서

tick : 무한히 증가하는 시간 생성 (이미 있는 주소와 안 겹치게)

$\text{tick}^\#$: 유한한 시간 생성

$\alpha : T \rightarrow T^\#$:

tick