Translation Sketch

1 Expressions

E translates a Chalice expression into an equivalent SIL expression. Given an scoped identifier i, $\rho(i)$ denotes a globally unique identifier. E.g., $\rho(\texttt{someField}) = \texttt{SomeClass::someField}$

$$E\,[\![e_1\texttt{ ? }e_2\texttt{ : }e_3]\!]_{\text{Ch}} = [\![\texttt{if }E\,(e_1)\texttt{ then }E\,(e_2)\texttt{ else }E\,(e_3)]\!]_{\text{SIL}} \quad (1)$$
$$E\,[\![e_1\texttt{== }e_2]\!]_{\text{Ch}} = [\![\texttt{==}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (2)$$
$$E\,[\![e_1\texttt{!= }e_2]\!]_{\text{Ch}} = [\![\texttt{!=}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (3)$$
$$E\,[\![e_1\texttt{< }e_2]\!]_{\text{Ch}} = [\![\texttt{<}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (4)$$
$$E\,[\![e_1\texttt{<= }e_2]\!]_{\text{Ch}} = [\![\texttt{<=}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (5)$$
$$E\,[\![e_1\texttt{>= }e_2]\!]_{\text{Ch}} = [\![\texttt{>=}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (6)$$
$$E\,[\![e_1\texttt{> }e_2]\!]_{\text{Ch}} = [\![\texttt{>}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (7)$$
$$E\,[\![e_1\texttt{ in }e_2]\!]_{\text{Ch}} = [\![\texttt{in}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (8)$$
$$E\,[\![e_1\texttt{ !in }e_2]\!]_{\text{Ch}} = E\,[\![\texttt{!}(e_1\texttt{ in }e_2)]\!]_{\text{Ch}} \quad (9)$$
$$E\,[\![e_1\texttt{+ }e_2]\!]_{\text{Ch}} = [\![\texttt{+}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (10)$$
$$E\,[\![e_1\texttt{- }e_2]\!]_{\text{Ch}} = [\![\texttt{-}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (11)$$
$$E\,[\![e_1\texttt{* }e_2]\!]_{\text{Ch}} = [\![\texttt{*}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (12)$$
$$E\,[\![e_1\texttt{/ }e_2]\!]_{\text{Ch}} = [\![\texttt{/}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (13)$$
$$E\,[\![e_1\texttt{\% }e_2]\!]_{\text{Ch}} = [\![\texttt{\%}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (14)$$
$$E\,[\![\texttt{! }e]\!]_{\text{Ch}} = [\![\texttt{!}(E\,(e))]\!]_{\text{SIL}} \quad (15)$$
$$E\,[\![\texttt{- }e]\!]_{\text{Ch}} = [\![\texttt{-}(E\,(e))]\!]_{\text{SIL}} \quad (16)$$
$$E\,[\![e_1\texttt{[}e_2\texttt{]}]\!]_{\text{Ch}} = [\![\texttt{at}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (17)$$
$$E\,[\![e_1\texttt{[}e_2\texttt{..]}]\!]_{\text{Ch}} = [\![\texttt{drop}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (18)$$
$$E\,[\![e_1\texttt{[..}\ e_2\texttt{]}]\!]_{\text{Ch}} = [\![\texttt{take}(E\,(e_1),E\,(e_2))]\!]_{\text{SIL}} \quad (19)$$
$$E\,[\![e_1\texttt{.}id]\!]_{\text{Ch}} = [\![E\,(e_1)\texttt{ .}id]\!]_{\text{SIL}} \quad \text{Also assert that } e_1 \text{ is not null} \quad (20)$$
$$E\,[\![e_1\texttt{.}id\texttt{(}e\texttt{ ...)}]\!]_{\text{Ch}} = [\![\rho(id)(E\,(e_1),E\,(e\texttt{ ...}))]\!]_{\text{SIL}} \quad \text{Also assert that } e_1 \text{ is not null} \quad (21)$$
$$E\,[\![\texttt{true}]\!]_{\text{Ch}} = [\![\texttt{true}]\!]_{\text{SIL}} \quad (22)$$
$$E\,[\![\texttt{false}]\!]_{\text{Ch}} = [\![\texttt{false}]\!]_{\text{SIL}} \quad (23)$$
$$E\,[\![\texttt{null}]\!]_{\text{Ch}} = [\![\texttt{null}]\!]_{\text{SIL}} \quad (24)$$
$$E\,[\![\texttt{this}]\!]_{\text{Ch}} = \text{First method argument} \quad (25)$$
$$E\,[\![x]\!]_{\text{Ch}} = [\![x]\!]_{\text{SIL}} \quad \text{where } x \text{ is a numeric literal} \quad (26)$$

2 Statements

S translates Chalice statements into equivalent SIL statements/expressions. $\tau_1, \tau_2, \ldots$ are temporary variables unique to each rule instantiation. Similarly, $\eta_1, \eta_2, \ldots$ are labels unique to each rule instantiation.

$$S[\![\texttt{assert } e_1]\!]_{\text{Ch}} = [\![\texttt{assert } E(e_1)]\!]_{\text{SIL}} \tag{27}$$

$$S[\![\texttt{assume } e_1]\!]_{\text{Ch}} = [\![\texttt{assume } E(e_1)]\!]_{\text{SIL}} \tag{28}$$

$$S[\![\{s \ldots\}]\!]_{\text{Ch}} = [\![S(s \ldots)]\!]_{\text{SIL}} \tag{29}$$

Need to flatten nested stmt blocks, since SIL doesn't have local variable scoping.

$$S[\![\ \texttt{var } id \texttt{ := } e_1]\!]_{\text{Ch}} = [\![id \texttt{ := } E(e_1)]\!]_{\text{SIL}} \quad \text{Keep } \texttt{const} \text{ and } \texttt{ghost} \text{ attributes} \tag{30}$$

$$S[\![\ \texttt{call } v \ldots \texttt{ := } r.m_1(e \ldots)]\!]_{\text{Ch}} = [\![\texttt{call } v \ldots \texttt{ := } \rho(m_1)(E(r), E(e \ldots))]\!]_{\text{SIL}} \tag{31}$$

$$\begin{aligned} S[\![\texttt{if}(e_1)\, s_1 \texttt{ else } s_2]\!]_{\text{Ch}} = [\![&\texttt{if } E(e_1) \texttt{ then goto } \eta_1 \texttt{ else } \eta_2 \texttt{ ;} \\ &\eta_1\texttt{: } S(s_1) \texttt{ ;} \\ &\quad \texttt{goto } \eta_3 \\ &\eta_2\texttt{: } S(s_2) \\ &\eta_3\texttt{: nop }]\!]_{\text{SIL}} \end{aligned} \tag{32}$$

`lockchange` for `while` loops ignored for now.

$$\begin{aligned} S[\![\texttt{while}(c) \texttt{ invariant } i \ldots \texttt{ ; } s]\!]_{\text{Ch}} = [\![\ &\eta_1\texttt{: } \tau_1 \texttt{ := } E(c) \texttt{ ;} \\ &\quad \texttt{exhale } E(i \ldots) \texttt{ ;} \\ &\quad \texttt{if } \tau_1 \texttt{ then goto } \eta_2 \texttt{ else } \eta_3 \texttt{ ;} \\ &\eta_2\texttt{: inhale } E(i \ldots) \texttt{ ;} \\ &\quad S(s) \texttt{ ;} \\ &\quad \texttt{goto } \eta_1 \texttt{ ;} \\ &\eta_3\texttt{: inhale } E(i \ldots)]\!]_{\text{SIL}} \end{aligned} \tag{33}$$

2.1 Fork-Join

There is no direct support for fork-join in SIL, so we'll have to simulate those asynchronous calls by exhaling the precondition at the fork-statement and returning a token object with certain properties. At the join-site, we verify that the token is valid and inhale the methods postcondition.

$c :: m_{\text{pre}}[e, a_1, a_2, \cdots, a_n, \pi_{\text{rd}}]$ stands for the precondition of the method m in class c with e substituted for the `this` reference and a_1 through a_n substituted for the formal parameters of the method. The value π_{rd}, finally, represents the permission fraction transferred to the callee to satisfy read permissions in the precondition of $c :: m$

$c :: m_{\text{post}}[e, a_1, a_2, \cdots, a_n, \pi_{\text{rd}}]$ is defined analogously for the method's postcondition.

The token's type contains the class and method that was invoked. This information is handled by the Chalice type checker and does not need to be encoded in SIL. On the other hand, the receiver and all method arguments need to be associated with the token at the fork-site in order to substitute them in the method's postcondition at the join-site later.

We probably have to pass read-permissions to the ghost fields of tokens around implicitly. Since these fields are never changed after the token is created, we can use fractional permissions to give away read-permissions to every method that receives the token.

$$
\begin{aligned}
S[\![t := \texttt{fork}\ e.m(a_1, a_2, \ldots, a_n)]\!]_{\text{Ch}} = [\![& \tau := E(e) \\
& t := \texttt{newobj} \\
& \texttt{inhale acc}(t.\texttt{joinable}, \texttt{write}) \\
& t.\texttt{joinable} := \texttt{True} \\
& \texttt{inhale acc}(t.\pi_{\text{rd}}, \texttt{write}) \\
& \texttt{inhale } 0 < t.\pi_{\text{rd}} < R \\
& \texttt{inhale acc}(t.\texttt{receiver}, \texttt{write}) \\
& t.\texttt{receiver} := \tau \\
& \texttt{inhale acc}\left(t.\texttt{arg}_1, \texttt{write}\right) \\
& t.\texttt{arg}_1 := E(a_1) \\
& \quad \vdots \\
& \texttt{inhale acc}\left(t.\texttt{arg}_n, \texttt{write}\right) \\
& t.\texttt{arg}_n := E(a_n) \\
& \texttt{exhale typeof}(e) :: m_{\text{pre}}[\tau, E(a_1, \cdots, a_n)] \\
&]\!]_{\text{SIL}}
\end{aligned}
\tag{34}
$$

where R = all permissions to fields $c :: m_{\text{pre}}$ mentions in read-access predicates.

$$
\begin{aligned}
S[\![o_1, \cdots, o_n := \texttt{join}\ t]\!]_{\text{Ch}} = [\![& \tau := E(t) \\
& \texttt{exhale } \tau.\texttt{joinable} == \texttt{True} \wedge \\
& \qquad \texttt{acc}(\tau.\texttt{joinable}, \texttt{write}) \\
& o_1 := \texttt{havoc} \\
& \quad \vdots \\
& o_n := \texttt{havoc} \\
& \texttt{inhale } c :: m_{\text{post}}\left[\tau.\texttt{receiver}, \tau.\texttt{arg}_{1,\cdots,n}, \tau.\pi_{\text{rd}}\right]]\!]_{\text{SIL}} \\
\text{where} \quad & c = \texttt{typeof}(\tau).\texttt{class} \\
& m = \texttt{typeof}(\tau).\texttt{method}
\end{aligned}
\tag{35}
$$

2.2 Predicates

$\sigma(o.p)$ looks up the (permission) expression associated with a predicate p on object o.

$$
\begin{aligned}
S[\![\texttt{fold}\ e_1.p_1]\!]_{\text{Ch}} = [\![& \texttt{exhale } \sigma\left(E(e_1).p_1\right) ; \\
& \texttt{inhale acc}\left(E(e_1).p_1, \texttt{write}\right)]\!]_{\text{SIL}}
\end{aligned}
\tag{36}
$$

$$
\begin{aligned}
S[\![\texttt{unfold}\ e_1.p_1]\!]_{\text{Ch}} = [\![& \texttt{exhale acc}\left(E(e_1).p_1, \texttt{write}\right) ; \\
& \texttt{inhale } \sigma\left(E(e_1).p_1\right)]\!]_{\text{SIL}}
\end{aligned}
\tag{37}
$$

2.3 Monitors

$\iota\,(e)$ denotes the (monitor) invariant of an object e.

$$S\,[\![\texttt{share}\ \ e_1]\!]_{\mathrm{Ch}} = [\![\texttt{exhale}\,\iota\,(E\,(e_1))]\!]_{\mathrm{SIL}} \tag{38}$$

$$S\,[\![\texttt{unshare}\ \ e_1]\!]_{\mathrm{Ch}} = [\![\texttt{exhale}\,\iota\,(E\,(e_1))]\!]_{\mathrm{SIL}} \tag{39}$$