Constructing Set Constraints for ReScript

서울대학교 전기·정보공학부 이재호, 이준협

1 Definition of set expressions

This proposal extends upon the results of [3] and attempts to address the limitations of the tool *ReAnalyze*.

In [3], the set-based constraint solving method for program analysis is used to track unhandled exceptions. However, the analysis simply collects all instances of a function being called. This is appropriate when there are no wrapper functions for raise. The sad fact is that there are multiple functions that can serve as raise in ReScript programs; when those functions are used, the original method leads to impractical conclusions. The analysis will always warn that all raised exceptions might be raised in every location. Thus, the set variables must keep track of where the arguments to the functions were supplied. This is why our definition of the set variable corresponding to the expression e is extended from V_e, P_e to $V(e,\sigma), P(e,\sigma)$. σ maps free variables in e to code locations where the arguments were supplied.

In *ReAnalyze*, when exceptions are not raised explicitly, it fails to analyze what kind of exceptions might be raised. An example is when exception values are wrapped inside a variant Error e and is raised as raise e. This is addressed by adding σ to set variables and tracking the possible values of expressions, not only the possible exception packets that might be raised. Also, when there is a division operator, *ReAnalyze* alerts that Divide_by_zero might be raised regardless of the denominator. This is addressed by adding conditional set constraints and tracking whether the denominator might turn to 0.

$$
\begin{array}{lcll}
se & ::= & \emptyset & \textit{empty set} \\
 & \mid & \_ & \textit{maximum set} \\
 & \mid & () & \textit{unit} \\
 & \mid & n & \textit{integer} \\
 & \mid & b & \textit{boolean} \\
 & \mid & \langle \lambda x.e, \sigma \rangle & \textit{closure} \\
 & \mid & \mathsf{loc} & \textit{memory location} \\
 & \mid & V(e,\sigma) & \textit{set variable corresponding to the possible values of e under } \sigma \\
 & \mid & P(e,\sigma) & \textit{set variable corresponding to the possible exn packets of e under } \sigma \\
 & \mid & \mathsf{app}_V(se, e) & \textit{values that se can spit out when se is applied to e} \\
 & \mid & \mathsf{app}_P(se, e) & \textit{exn packets that se can spit out when se is applied to e} \\
 & \mid & \mathsf{con}(\kappa, se) & \textit{construct, including exceptions} \\
 & \mid & \mathsf{fld}(se, l) & \textit{contents of the field l of a record se} \\
 & \mid & \mathsf{cnt}(se) & \textit{contents of a reference} \\
 & \mid & \mathsf{bop}(se, se) & \textit{binary operators, where } \mathsf{bop} \in \{+,-,\times,\div,=,<,>\} \\
 & \mid & f_{(i)}^{-1}(se) & \textit{projection onto the i-th argument of f} \\
 & \mid & se \cup se & \textit{union} \\
 & \mid & se \cap se & \textit{intersection} \\
 & \mid & \overline{se} & \textit{complement} \\
 & \mid & se \Rightarrow se & \textit{conditional expression}
\end{array}
$$

The definition of the conditional set expression needs clarification.

$$(se_1 \Rightarrow se_2) := \begin{cases} \emptyset & (se_1 = \emptyset) \\ se_2 & (o.w) \end{cases}$$

The conditional set expression is a naive approximation for pattern matching. Consider the case when we want to match an expression against the record pattern with fields x and y. The constraints describing the record $r = \{x = 1, y = 2\}$ are $1 \subseteq \mathsf{fld}(V(r,\sigma_r), x) \wedge 2 \subseteq \mathsf{fld}(V(r,\sigma_r), y)$ with $\sigma_r = [r \mapsto \{x = 1, y = 2\}]$. To pattern-match r against $\{x, y\}$, we want the fields x and y of $V(r,\sigma_r)$ to be nonempty. Thus, the value of "*match r with* $\{x, y\} \to e$" is $\mathsf{fld}(V(r,\sigma_r), x) \Rightarrow (\mathsf{fld}(V(r,\sigma_r), y) \Rightarrow V(e,\sigma_e))$.

The conditional set expression can also be used to define conditional set constraints [1].

$$\left(se \Rightarrow \left(\bigwedge_{i=1}^{n} X_i \subseteq Y_i\right)\right) := \bigwedge_{i=1}^{n} (se \Rightarrow X_i) \subseteq Y_i$$

2 Constructing set constraints

Now we are in a position to define constraint construction rules for our ReScript-like language. Hopefully this would be reasonably fast when implemented and be accurate enough...

Notation: $\sigma_i := \sigma|_{e_i}$, E_x: expression variable, Σ_x: environment variable, all *Expr*s are tagged with their location

$$[\textsc{Unit, Int, Bool}]\ \frac{}{[\,] \rhd c :\ V(e,\sigma) \supseteq c}\ c = (), n, b$$

$$[\textsc{App}]\ \frac{\sigma_1 \rhd e_1 :\ C_1 \qquad \sigma_2 \rhd e_2 :\ C_2}{\sigma \rhd e_1\ e_2 :\quad \begin{array}{l} V(e,\sigma) \supseteq \mathsf{app}_V(V(e_1,\sigma_1), e_2) \land \\ P(e,\sigma) \supseteq \mathsf{app}_P(V(e_1,\sigma_1), e_2) \cup P(e_1,\sigma_1) \cup P(e_2,\sigma_2) \land C_1 \land C_2 \end{array}}$$

$$[\textsc{Fn}]\ \frac{\sigma \cup [x \mapsto E_x] \rhd e' :\ C'}{\sigma \rhd \lambda x.e' :\ V(e,\sigma) \supseteq \langle \lambda x.e', \sigma\rangle \land \quad \begin{array}{l} \mathsf{app}_V(\langle \lambda x.e', \sigma\rangle, E_x) \supseteq V(e', [x \mapsto E_x] \cup \sigma) \land \\ \mathsf{app}_P(\langle \lambda x.e', \sigma\rangle, E_x) \supseteq P(e', [x \mapsto E_x] \cup \sigma) \land C' \end{array}}$$

$$[\textsc{Var}]\ \frac{}{\sigma \rhd x :\ V(x,\sigma) \supseteq V(\sigma(x), \Sigma_x)} \qquad [\underline{\textsc{Var}}]\ \frac{}{\sigma \rhd x :\ V(x,\sigma) \supseteq P(\sigma(x), \Sigma_x)}\ \text{when } \sigma(x) \text{ is underlined}$$

$$[\textsc{Let}]\ \frac{\sigma_1 \cup \overbrace{[x \mapsto e_1]}^{\text{support rec}} \rhd e_1 :\ C_1 \qquad \sigma_2 \cup [x \mapsto e_1] \rhd e_2 :\ C_2}{\sigma \rhd \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 :\quad \begin{array}{l} V(e,\sigma) \supseteq V(e_2, \sigma_2 \cup [x \mapsto e_1]) \land \\ P(e,\sigma) \supseteq P(e_1, \sigma_1 \cup [x \mapsto e_1]) \cup P(e_2, \sigma_2 \cup [x \mapsto e_1]) \land C_1 \land C_2 \end{array}}$$

$$[\textsc{Bop}]\ \frac{\sigma_1 \rhd e_1 :\ C_1 \qquad \sigma_2 \rhd e_2 :\ C_2}{\sigma \rhd e_1\ \mathsf{bop}\ e_2 :\quad \begin{array}{l} V(e,\sigma) \supseteq \mathsf{bop}(V(e_1,\sigma_1), V(e_2,\sigma_2)) \land \\ P(e,\sigma) \supseteq P(e_1,\sigma_1) \cup P(e_2,\sigma_2) \land \underbrace{P(e,\sigma) \supseteq (V(e_2,\sigma_2) \cap 0 \Rightarrow \mathsf{exn}(\mathsf{Divide\_by\_zero}, ()))}_{\text{when bop is } \div} \land C_1 \land C_2 \end{array}}$$

$$[\textsc{Con/Exn}]\ \frac{\sigma \rhd e' :\ C'}{\sigma \rhd \mathsf{con|exn}\ \kappa\ e' :\ V(e,\sigma) \supseteq \mathsf{con}(\kappa, V(e',\sigma)) \land P(e,\sigma) \supseteq P(e',\sigma) \land C'}$$

$$[\textsc{Decon}]\ \frac{\sigma \rhd e' :\ C'}{\sigma \rhd \mathsf{decon}\ \kappa\ e' :\quad \begin{array}{l} V(e,\sigma) \supseteq \mathsf{con}^{-1}_{(2)}(\underbrace{V(e',\sigma) \cap \mathsf{con}(\kappa, \_)}_{\text{filter}}) \land \\ P(e,\sigma) \supseteq P(e',\sigma) \land C' \end{array}}\ \text{in case}$$

$$[\underline{\textsc{Decon}}]\ \frac{\sigma \rhd e' :\ C'}{\sigma \rhd \mathsf{decon}\ \kappa\ \underline{e'} :\ V(e,\sigma) \supseteq \mathsf{con}^{-1}_{(2)}(\underbrace{P(e',\sigma) \cap \mathsf{con}(\kappa, \_)}_{\text{filter}}) \land C'}\ \text{in handle}$$

$$[\textsc{Record}]\ \frac{\sigma_i \rhd e_i :\ C_i\ (1 \le i \le n)}{\sigma \rhd \{l_1 = e_i, ..., l_n = e_n\} :\ \bigwedge_{i=1}^{n} (\mathsf{fld}(V(e,\sigma), l_i) \supseteq V(e_i,\sigma_i)) \land P(e,\sigma) \supseteq \bigcup_{i=1}^{n} P(e_i,\sigma_i) \land \bigwedge_{i=1}^{n} C_i}$$

$$[\textsc{Field}]\ \frac{\sigma \rhd e' :\ C'}{\sigma \rhd e'.l :\ V(e,\sigma) \supseteq \mathsf{fld}(V(e',\sigma), l) \land P(e,\sigma) \supseteq P(e',\sigma) \land C'}$$

$$[\textsc{Ref}]\ \frac{\sigma \rhd e' :\ C'}{\sigma \rhd \mathsf{ref}\ e' :\ V(e,\sigma) \supseteq \mathsf{loc} \land \mathsf{cnt}(\mathsf{loc}) \supseteq V(e',\sigma) \land P(e,\sigma) \supseteq P(e',\sigma) \land C'}\ \text{new loc}$$

$$[\textsc{Update}]\ \frac{\sigma_1 \rhd e_1 :\ C_1 \qquad \sigma_2 \rhd e_2 :\ C_2}{\sigma \rhd e_1\ := e_2 :\ \mathsf{cnt}(V(e_1,\sigma_1)) \supseteq V(e_2,\sigma_2) \land P_e \supseteq P(e_1,\sigma_1) \cup P(e_2,\sigma_2) \land C_1 \land C_2}$$

$$[\textsc{Bang}]\ \frac{\sigma \rhd e' :\ C'}{\sigma \rhd !e' :\ V(e,\sigma) \supseteq \mathsf{cnt}(V(e',\sigma)) \land P(e,\sigma) \supseteq P(e',\sigma) \land C'}$$

$$[\textsc{Seq}]\ \frac{\sigma_1 \rhd e_1 : C_1 \qquad \sigma_2 \rhd e_2 : C_2}{\sigma \rhd e_1; e_2 : V(e,\sigma) \supseteq V(e_2,\sigma_2) \wedge P(e,\sigma) \supseteq P(e_1,\sigma_1) \cup P(e_2,\sigma_2) \wedge C_1 \wedge C_2}$$

We define an auxilary function for generating constraints out of pattern matching. If we want to figure out the constraint for the expression e = "match e' with $p \to e''$" under σ:

$$\mathsf{case}(e',\sigma,p,e'') := \begin{cases} V(e,\sigma) \supseteq V(e'',\sigma|_{e''} \cup [x \mapsto e']) \wedge P(e,\sigma) \supseteq P(e'',\sigma|_{e''} \cup [x \mapsto e']) & (p = x) \\ V(e,\sigma) \supseteq V(e'',\sigma|_{e''}) \wedge P(e,\sigma) \supseteq P(e'',\sigma|_{e''}) & (o.w) \end{cases}$$

If we want to figure out the constraint for the expression e = "try e' with $p \to e''$" under σ:

$$\mathsf{hndl}(e',\sigma,p,e'') := \begin{cases} V(e,\sigma) \supseteq V(e'',\sigma|_{e''} \cup [x \mapsto \underline{e'}]) \wedge P(e,\sigma) \supseteq P(e'',\sigma|_{e''} \cup [x \mapsto \underline{e'}]) & (p = x) \\ V(e,\sigma) \supseteq V(e'',\sigma|_{e''}) \wedge P(e,\sigma) \supseteq P(e'',\sigma|_{e''}) & (o.w) \end{cases}$$

$$[\textsc{Case}]\ \frac{\sigma|_{e'} \rhd e' : C' \qquad \sigma_i \rhd e_i : C_i\ (1 \le i \le n)}{\sigma \rhd \mathsf{case}\ e'\ (p_i \to e_i)_{i=1}^{n} : \bigwedge_{i=1}^{n} \mathsf{case}(e',\sigma,p_i,e_i) \wedge P(e,\sigma) \supseteq P(e',\sigma|_{e'}) \wedge C' \wedge \bigwedge_{i=1}^{n} C_i}$$

$$[\textsc{Handle}]\ \frac{\sigma|_{e'} \rhd e' : C' \qquad \sigma_i \rhd e_i : C_i\ (1 \le i \le n)}{\sigma \rhd \mathsf{handle}\ e'\ (p_i \to e_i)_{i=1}^{n} : \begin{array}{l} P(e,\sigma) \supseteq (P(e',\sigma|_{e'}) \cap \bigcap_{i=1}^{n} \overline{\mathsf{con}(p_i,\_)}) \wedge \\ V(e,\sigma) \supseteq V(e',\sigma|_{e'}) \wedge \\ \bigwedge_{i=1}^{n} \mathsf{hndl}(e',\sigma,p_i,e_i) \wedge C' \wedge \bigwedge_{i=1}^{n} C_i \end{array}}$$

$$[\textsc{Raise}]\ \frac{\sigma \rhd e' : C'}{\sigma \rhd \mathsf{raise}\ e' : P(e,\sigma) \supseteq V(e',\sigma) \cup P(e',\sigma) \wedge C'}$$

$$[\textsc{For}]\ \frac{\sigma_1 \rhd e_1 : C_1 \qquad \sigma_2 \rhd e_2 : C_2 \qquad \sigma_3 \cup [x \mapsto e_1] \rhd e_3 : C_3 \qquad \sigma_3 \cup [x \mapsto x+1] \rhd e_3 : C_4}{\sigma \rhd \mathsf{for}\ x\ e_1\ e_2\ e_3 : \begin{array}{l} V(e,\sigma) \supseteq () \wedge V(x,[x \mapsto e_1]) \supseteq V(e_1,\sigma_1) \wedge \\ V(x,[x \mapsto x+1]) \supseteq +(V(e_1,\sigma_1),1) \cup +(V(x,[x \mapsto x+1]),1) \wedge \\ P(e,\sigma) \supseteq P(e_1,\sigma_1) \cup P(e_2,\sigma_2) \cup P(e_3,\sigma_3 \cup [x \mapsto e_1]) \cup P(e_3,\sigma_3 \cup [x \mapsto x+1]) \wedge \\ C_1 \wedge C_2 \wedge C_3 \wedge C_4 \end{array}}$$

$$[\textsc{While}]\ \frac{\sigma_1 \rhd e_1 : C_1 \qquad \sigma_2 \rhd e_2 : C_2}{\sigma \rhd \mathsf{while}\ e_1\ e_2 : V(e,\sigma) \supseteq () \wedge P(e,\sigma) \supseteq P(e_1,\sigma_1) \cup P(e_2,\sigma_2) \wedge C_1 \wedge C_2}$$

3 Example

$$\underbrace{\text{let } f = \overbrace{\lambda x.\underbrace{\text{Error}\ \overbrace{(\underbrace{f}_{e''_{11}}\ \underbrace{x}_{e''_{12}})}^{e''_1}}_{e'_1}}^{e_1} \text{ in } \overbrace{\text{raise}\ \underbrace{(\overbrace{f}^{e'_{21}}\ \overbrace{\text{Fail}}^{e'_{22}})}_{e'_2}}^{e_2}}_{e}$$

The above program is type checked as $f : \mathsf{exn} \to \mathsf{exn}$, yet it does not terminate. The (simplified) set constraints generated for this program are:

Constraints		From	By	No.
$P(e, [])$	$\supseteq P(e_2, [f \mapsto e_1])$	e	[Let]	(1)
$V(e_1, [f \mapsto e_1])$	$\supseteq \langle \lambda x.e'_1, [f \mapsto e_1] \rangle$	e_1	[Fn]	(2)
$\mathsf{app}_V(\langle \lambda x.e'_1, [f \mapsto e_1] \rangle, E_x)$	$\supseteq V(e'_1, [f \mapsto e_1; x \mapsto E_x])$	e_1	[Fn]	(3)
$V(e'_1, [f \mapsto e_1; x \mapsto E_x])$	$\supseteq \mathsf{exn}(\mathsf{Error}, V(e''_1, [f \mapsto e_1; x \mapsto E_x]))$	e'_1	[Exn]	(4)
$V(e''_1, [f \mapsto e_1; x \mapsto E_x])$	$\supseteq \mathsf{app}_V(V(e''_{11}, [f \mapsto e_1]), e''_{12})$	e''_1	[App]	(5)
$V(e''_{11}, [f \mapsto e_1])$	$\supseteq V(e_1, \Sigma_f)$	e''_{11}	[Var]	(6)
$V(e''_{12}, [x \mapsto E_x])$	$\supseteq V(E_x, \Sigma_x)$	e''_{12}	[Var]	(7)
$P(e_2, [f \mapsto e_1])$	$\supseteq V(e'_2, [f \mapsto e_1])$	e_2	[Raise]	(8)
$V(e'_2, [f \mapsto e_1])$	$\supseteq \mathsf{app}_V(V(e'_{21}, [f \mapsto e_1]), e'_{22})$	e'_2	[App]	(9)
$V(e'_{21}, [f \mapsto e_1])$	$\supseteq V(e_1, \Sigma_f)$	e'_{21}	[Var]	(10)
$V(e'_{22}, [])$	$\supseteq \mathsf{exn}(\mathsf{Fail}, ())$	e'_{22}	[Exn]	(11)

To "solve" this system of constraints, multiple reduction steps are needed.

(1)	$V(e'_2, [f \mapsto e_1])$	$\supseteq \mathsf{app}_V(V(e'_{21}, [f \mapsto e_1]), e'_{22})$	($\because$(9))
		$\supseteq \mathsf{app}_V(V(e_1, \Sigma_f), e'_{22})$	($\because$(10))
		$\supseteq \mathsf{app}_V(V(e_1, [f \mapsto e_1]), e'_{22})$	(unify with (2))
		$\supseteq \mathsf{app}_V(\langle \lambda x.e'_1, [f \mapsto e_1] \rangle, e'_{22})$	($\because$(2))
		$\supseteq V(e'_1, [f \mapsto e_1; x \mapsto e'_{22}])$	($\because$(3))
(2)	$V(e'_1, [f \mapsto e_1; x \mapsto e'_{22}])$	$\supseteq \mathsf{exn}(\mathsf{Error}, V(e''_1, [f \mapsto e_1; x \mapsto e'_{22}]))$	($\because$(4))
(3)	$V(e''_1, [f \mapsto e_1; x \mapsto e'_{22}])$	$\supseteq \mathsf{app}_V(V(e''_{11}, [f \mapsto e_1]), e''_{12})$	($\because$(5))
		$\supseteq \mathsf{app}_V(V(e_1, \Sigma_f), e''_{12})$	($\because$(6))
		$\supseteq \mathsf{app}_V(V(e_1, [f \mapsto e_1]), e''_{12})$	(unify with (2))
		$\supseteq \mathsf{app}_V(\langle \lambda x.e'_1, [f \mapsto e_1] \rangle, e''_{12})$	($\because$(2))
		$\supseteq V(e'_1, [f \mapsto e_1; x \mapsto e''_{12}])$	($\because$(3))
(4)	$V(e'_1, [f \mapsto e_1; x \mapsto e''_{12}])$	$\supseteq \mathsf{exn}(\mathsf{Error}, V(e''_1, [f \mapsto e_1; x \mapsto e''_{12}]))$	($\because$(4))
(5)	$V(e''_1, [f \mapsto e_1; x \mapsto e''_{12}])$	$\supseteq \mathsf{app}_V(V(e''_{11}, [f \mapsto e_1]), e''_{12})$	($\because$(5))
		$\supseteq V(e'_1, [f \mapsto e_1; x \mapsto e''_{12}])$	(as in 3)
(6)	Cannot reduce further as $V(e'_1, [f \mapsto e_1; x \mapsto e''_{12}])$ was already reduced in 4.		

Let $X_1 \coloneqq V(e'_1, [f \mapsto e_1; x \mapsto e'_{22}])$, $X_2 \coloneqq V(e''_1, [f \mapsto e_1; x \mapsto e'_{22}])$, $X_3 \coloneqq V(e'_1, [f \mapsto e_1; x \mapsto e''_{12}])$, $X_4 \coloneqq V(e''_1, [f \mapsto e_1; x \mapsto e''_{12}])$. Then the constraints reduce to:

$$
\begin{aligned}
P(e,[]) \supseteq P(e_2,[f \mapsto e_1]) \supseteq V(e_2',[f \mapsto e_1]) & \quad (\because(1),(8)) \\
V(e_2',[f \mapsto e_1]) \supseteq X_1 & \quad (\because 1) \\
\Rightarrow P(e,[]) \supseteq X_1 & \\
X_1 \supseteq \mathsf{exn}(\mathrm{Error}, X_2) & \quad (\because 2) \\
X_2 \supseteq X_3 & \quad (\because 3) \\
X_3 \supseteq \mathsf{exn}(\mathrm{Error}, X_4) & \quad (\because 4) \\
X_4 \supseteq X_3 & \quad (\because 5)
\end{aligned}
$$

Note that constraint (7) was not used in the derivation of the above relations. This reflects the fact that the program does not terminate. There is absolutely no execution path evaluating what "Fail" is.

4 Why σ?

Section 3 demonstrated that even with σ, the set constraints may handle tricky expressions. It failed, however, to demonstrate how it improves upon [3]. In this section, an example program demonstrates how the accuracy of analysis is improved by separating when functions are called.

$$
\underbrace{\text{let id} = \overbrace{\lambda x.\,\underbrace{x}_{e_1'}}^{e_1} \text{ in } \overbrace{\underbrace{\overbrace{\text{id}}^{e_{211}}\ \overbrace{1}^{e_{212}}}_{e_{21}} + \underbrace{\overbrace{\text{id}}^{e_{221}}\ \overbrace{2}^{e_{222}}}_{e_{22}}}^{e_2}}_{e}
$$

The original method $\triangleright_1$ in [3] concludes that since id might spit out 1 or 2, (id 1) + (id 2) might be anything in {1+1, 1+2, 2+1, 2+2}. However, with σ, this ambiguity is dispelled.

Constraints			From	By	No.
$V(e,[])$	$\supseteq$	$V(e_2,[\text{id} \mapsto e_1])$	e	[LET]	(1)
$V(e_1,[])$	$\supseteq$	$\langle \lambda x.e_1',[] \rangle$	e_1	[FN]	(2)
$\mathsf{app}_V(\langle \lambda x.e_1',[] \rangle, E_x)$	$\supseteq$	$V(e_1',[x \mapsto E_x])$	e_1	[FN]	(3)
$V(e_1',[x \mapsto E_x])$	$\supseteq$	$V(E_x,\Sigma_x)$	e_1'	[VAR]	(4)
$V(e_2,[\text{id} \mapsto e_1])$	$\supseteq$	$+(V(e_{21},[\text{id} \mapsto e_1]), V(e_{22},[\text{id} \mapsto e_1]))$	e_2	[BOP]	(5)
$V(e_{21},[\text{id} \mapsto e_1])$	$\supseteq$	$\mathsf{app}_V(V(e_{211},[\text{id} \mapsto e_1]), e_{212})$	e_{21}	[APP]	(6)
$V(e_{211},[\text{id} \mapsto e_1])$	$\supseteq$	$V(e_1,\Sigma_{\text{id}})$	e_{211}	[VAR]	(7)
$V(e_{212},[])$	$\supseteq$	1	e_{212}	[INT]	(8)
$V(e_{22},[\text{id} \mapsto e_1])$	$\supseteq$	$\mathsf{app}_V(V(e_{221},[\text{id} \mapsto e_1]), e_{222})$	e_{22}	[APP]	(9)
$V(e_{221},[\text{id} \mapsto e_1])$	$\supseteq$	$V(e_1,\Sigma_{\text{id}})$	e_{221}	[VAR]	(10)
$V(e_{222},[])$	$\supseteq$	2	e_{222}	[INT]	(11)

(1)	$V(e_{21}, [\text{id} \mapsto e_1])$	$\supseteq \text{app}_V(V(e_{211}, [\text{id} \mapsto e_1]), e_{212})$	$(\because(6))$
		$\supseteq \text{app}_V(V(e_1, \Sigma_{\text{id}}), e_{212})$	$(\because(7))$
		$\supseteq \text{app}_V(\langle \lambda x.e_1', [] \rangle, e_{212})$	$(\because(2))$
		$\supseteq V(e_1', [x \mapsto e_{212}])$	$(\because(3))$
(2)	$V(e_1', [x \mapsto e_{212}])$	$\supseteq V(e_{212}, \Sigma_x)$	$(\because(4))$
		$\supseteq V(e_{212}, [])$	(unify with (7))
		$\supseteq 1$	$(\because(8))$
(3)	$V(e_{22}, [\text{id} \mapsto e_1])$	$\supseteq \text{app}_V(V(e_{221}, [\text{id} \mapsto e_1]), e_{212})$	$(\because(9))$
		$\supseteq \text{app}_V(V(e_1, \Sigma_{\text{id}}), e_{222})$	$(\because(10))$
		$\supseteq \text{app}_V(\langle \lambda x.e_1', [] \rangle, e_{222})$	$(\because(2))$
		$\supseteq V(e_1', [x \mapsto e_{222}])$	$(\because(3))$
(4)	$V(e_1', [x \mapsto e_{222}])$	$\supseteq V(e_{222}, \Sigma_x)$	$(\because(4))$
		$\supseteq V(e_{222}, [])$	(unify with (11))
		$\supseteq 2$	$(\because(11))$

If we let $X_1 \coloneqq V(e_{21}, [\text{id} \mapsto e_1])$, $X_2 \coloneqq V(e_1', [x \mapsto e_{212}])$, $X_3 \coloneqq V(e_{22}, [\text{id} \mapsto e_1])$, $X_4 \coloneqq V(e_1', [x \mapsto e_{222}])$, the constraints reduce to:

$$\begin{aligned} V(e, []) \supseteq V(e_2, [\text{id} \mapsto e_1]) \supseteq +(X_1, X_3) & \qquad (\because(1),(5)) \\ \Rightarrow V(e, []) \supseteq +(X_1, X_3) & \\ X_1 \supseteq X_2 & \qquad (\because 1) \\ X_2 \supseteq 1 & \qquad (\because 2) \\ X_3 \supseteq X_4 & \qquad (\because 3) \\ X_4 \supseteq 2 & \qquad (\because 4) \end{aligned}$$

Comments: Note that even since id is called two times, there are two set variables corresponding to id, which are $V(e_{211}, [\text{id} \mapsto e_1])$ and $V(e_{221}, [\text{id} \mapsto e_1])$. This is because expressions are tagged by their location in code. In other words, we assume the existence of a hash table between expressions and their locations in code.

Also, note that all constraints are fully utilized in reducing the system of constraints, unlike in the previous section.

5 TODO

1. A proof that the constraint generation rules are sound
2. A program to generate (and print) set constraints from the Typedtrees.
3. An algorithm to reduce the constraints : based on the work of Nevin Heintze[2]

References

[1] Alexander Aiken. "Introduction to set constraint-based program analysis". In: *Science of Computer Programming* 35.2 (1999), pp. 79–111. ISSN: 0167-6423. DOI: `https://doi.org/10.1016/S0167-6423(99)00007-6`.

[2] Nevin Heintze. *A Decision Procedure for a Class of Set Constraints*. Tech. rep. 1991.

[3] Kwangkeun Yi and Sukyoung Ryu. "Towards a Cost-Effective Estimation of Uncaught Exceptions in SML Programs". In: *Proceedings of the 4th International Symposium on Static Analysis*. SAS '97. Berlin, Heidelberg: Springer-Verlag, 1997, pp. 98–113. ISBN: 3540634681.