Constructing Set Constraints for ReScript

서울대학교 전기·정보공학부 2018-12602 이준협

1 Definition of set expressions

se	$::=$	$\emptyset$	*empty set*
	$\mid$	$\_$	*maximum set*
	$\mid$	$()$	*unit*
	$\mid$	n	*integer*
	$\mid$	b	*boolean*
	$\mid$	$\lambda x.e$	*function*
	$\mid$	loc	*memory location*
	$\mid$	V_e	*set variable corresponding to the possible values of e*
	$\mid$	P_e	*set variable corresponding to the possible exn packets of e*
	$\mid$	$\mathrm{body}_V(se)$	*values that se can spit out when se is applied to something*
	$\mid$	$\mathrm{body}_P(se)$	*exn packets that se can spit out when se is applied to something*
	$\mid$	$\mathrm{par}(se)$	*values that can be a parameter for se*
	$\mid$	κ	*constructor*
	$\mid$	l	*field of a record*
	$\mid$	$\mathrm{con}(\kappa, se)$	*construct*
	$\mid$	$\mathrm{exn}(\kappa, se)$	*exception*
	$\mid$	$\mathrm{fld}(se, l)$	*contents of the field l of a record se*
	$\mid$	$\mathrm{cnt}(se)$	*contents of a reference*
	$\mid$	$\mathrm{bop}(se, se)$	*binary operators, where* bop $\in \{+, -, \times, \div, =, <, >\}$
	$\mid$	$f^{-1}_{(i)}(se)$	*projection onto the i-th argument of f*
	$\mid$	$se \cup se$	*union*
	$\mid$	$se \cap se$	*intersection*
	$\mid$	$\overline{se}$	*complement*

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 $1 \subseteq \mathrm{fld}(V_r, x) \wedge 2 \subseteq \mathrm{fld}(V_r, y)$. To pattern-match r against $\{x, y\}$, we want the fields x and y of V_r to be nonempty. Thus, the value of "*match r with* $\{x, y\} \to e$" is $\mathrm{fld}(V_r, x) \Rightarrow (\mathrm{fld}(V_r, y) \Rightarrow V_e)$.

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

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

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...

$$[\text{UNIT, INT, BOOL}]\ \frac{}{\rhd c : V_e \supseteq c}\ c = (), n, b$$

$$[\text{APP}]\ \frac{\rhd e_1 : C_1 \qquad \rhd e_2 : C_2}{\rhd e_1\, e_2 : (V_e \supseteq \mathrm{body}_V(V_{e_1})) \wedge (P_e \supseteq (\mathrm{body}_P(V_{e_1}) \cup P_{e_1} \cup P_{e_2})) \wedge (\mathrm{par}(V_{e_1}) \supseteq V_{e_2}) \wedge C_1 \wedge C_2}$$

$$[\text{FN}]\ \frac{\rhd e' : C'}{\rhd \lambda x.e' : (V_e \supseteq \lambda x.e') \wedge (\mathrm{body}_V(V_e) \supseteq V_{e'}) \wedge (\mathrm{body}_P(V_e) \supseteq P_{e'}) \wedge (\mathrm{par}(V_e) \subseteq V_x) \wedge C'}$$

$$[\text{LET}]\ \frac{\rhd e_1 : C_1 \qquad \rhd e_2 : C_2}{\rhd \textit{let } x = e_1 \textit{ in } e_2 : (V_x \supseteq V_{e_1}) \wedge (V_e \supseteq V_{e_2}) \wedge (P_e \supseteq P_{e_1} \cup P_{e_2}) \wedge C_1 \wedge C_2}$$

$$[\text{BOP}]\ \frac{\rhd e_1 : C_1 \qquad \rhd e_2 : C_2}{\rhd e_1 \text{ bop } e_2 : (V_e \supseteq \mathsf{bop}(V_{e_1}, V_{e_2})) \wedge (P_e \supseteq P_{e_1} \cup P_{e_2}) \wedge \underbrace{(P_e \supseteq (V_{e_2} \cap 0 \Rightarrow \mathsf{exn}(\mathsf{Divide\_by\_zero}, ())))}_{\text{when bop is } \div} \wedge C_1 \wedge C_2}$$

$$[\text{CON}]\ \frac{\rhd e' : C'}{\rhd \mathsf{con}\ \kappa\ e' : (V_e \supseteq \mathsf{con}(\kappa, V_{e'})) \wedge (P_e \supseteq P_{e'}) \wedge C'}$$

$$[\text{EXN}]\ \frac{\rhd e' : C'}{\rhd \mathsf{exn}\ \kappa\ e' : (V_e \supseteq \mathsf{exn}(\kappa, V_{e'})) \wedge (P_e \supseteq P_{e'}) \wedge C'}$$

$$[\text{DECON}]\ \frac{\rhd e' : C'}{\rhd \mathsf{decon}\ \kappa\ e' : (V_e \supseteq \mathsf{con}^{-1}_{(2)}(\underbrace{V_{e'} \cap \mathsf{con}(\kappa, \_)}_{\text{filter}}) \cup \mathsf{exn}^{-1}_{(2)}(\underbrace{V_{e'} \cap \mathsf{exn}(\kappa, \_)}_{\text{filter}})) \wedge (P_e \supseteq P_{e'}) \wedge C'}\ \text{as in SML-NJ's Lambda}$$

$$[\text{RECORD}]\ \frac{\rhd e_i : C_i\ (1 \le i \le n)}{\rhd \{l_1 = e_i, ..., l_n = e_n\} : \bigwedge_{i=1}^{n} (\mathsf{fld}(V_e, l_i) \supseteq V_{e_i}) \wedge (P_e \supseteq \bigcup_{i=1}^{n} P_{e_i}) \wedge \bigwedge_{i=1}^{n} C_i}$$

$$[\text{FIELD}]\ \frac{\rhd e' : C'}{\rhd e'.l : (V_e \supseteq \mathsf{fld}(V_{e'}, l)) \wedge (P_e \supseteq P_{e'}) \wedge C'}$$

$$[\text{REF}]\ \frac{\rhd e' : C'}{\rhd \mathsf{ref}\ e' : (V_e \supseteq \mathsf{loc}) \wedge (\mathsf{cnt}(\mathsf{loc}) \supseteq V_{e'}) \wedge (P_e \supseteq P_{e'}) \wedge C'}\ \text{new loc}$$

$$[\text{UPDATE}]\ \frac{\rhd e_1 : C_1 \qquad \rhd e_2 : C_2}{\rhd e_1 := e_2 : (\mathsf{cnt}(V_{e_1}) \supseteq V_{e_2}) \wedge (P_e \supseteq P_{e_1} \cup P_{e_2}) \wedge C_1 \wedge C_2}$$

$$[\text{BANG}]\ \frac{\rhd e' : C'}{\rhd !e' : (V_e \supseteq \mathsf{cnt}(V_{e'})) \wedge (P_e \supseteq P_{e'}) \wedge C'}$$

We define an auxilary function for generating constraints out of pattern matching. If we want to figure out the constraint for the value X of "match Y with $p \to e$":

$$\mathsf{case}(X, Y, p, e) := \begin{cases} (Y \subseteq V_x) \wedge (V_e \subseteq X) & (p = x) \\ (\mathsf{fld}(Y, l_1) \Rightarrow ... \Rightarrow \mathsf{fld}(Y, l_n) \Rightarrow V_e) \subseteq X & (p = \{l_i\}_{i=1}^{n}) \\ (\kappa \cap (\mathsf{con}^{-1}_{(1)}(Y) \cup \mathsf{exn}^{-1}_{(1)}(Y))) \Rightarrow V_e \subseteq X & (p = \kappa) \\ (Y \cap c) \Rightarrow V_e \subseteq X & (p = \mathsf{constant}) \\ V_e \subseteq X & (p = \_) \end{cases}$$

$$[\text{CASE}]\ \frac{\rhd e' : C' \qquad \rhd e_i : C_i\ (1 \le i \le n)}{\rhd \mathsf{case}\ e'\ (p_i \to e_i)_{i=1}^{n} : \bigwedge_{i=1}^{n} \mathsf{case}(V_e, V_{e'}, p_i, e_i) \wedge (P_e \supseteq \bigcup_{i=1}^{n} P_i \cup P_{e'}) \wedge C' \wedge \bigwedge_{i=1}^{n} C_i}$$

$$[\text{HANDLE}]\ \frac{\rhd e' : C' \qquad \rhd e_i : C_i\ (1 \le i \le n)}{\rhd \mathsf{handle}\ e'\ (p_i \to e_i)_{i=1}^{n} : (P_e \supseteq (P_{e'} \cap \bigcap_{i=1}^{n} \overline{\mathsf{exn}(p_i, \_)}) \cup \bigcup_{i=1}^{n} P_{e_i}) \wedge (V_e \supseteq V_{e'} \cup \bigcup_{i=1}^{n} V_{e_i}) \wedge C' \wedge \bigwedge_{i=1}^{n} C_i}$$

$$[\text{RAISE}]\ \frac{\rhd e' : C'}{\rhd \mathsf{raise}\ e' : (P_e \supseteq V_{e_1} \cup P_{e_1}) \wedge C'}$$

$$[\text{FOR}]\ \frac{\rhd e_1 : C_1 \qquad \rhd e_2 : C_2 \qquad \rhd e_3 : C_3}{\begin{array}{ll} \rhd \mathsf{for}\ x\ e_1\ e_2\ e_3 : & (>(V_x, V_{e_1}) \cap \texttt{true} \Rightarrow () \subseteq V_e) \\ & \wedge(>(V_x, V_{e_1}) \cap \texttt{false} \Rightarrow (C_3 \wedge (V_x \supseteq +(V_x, 1)) \wedge (P_e \supseteq P_{e_3}))) \\ & \wedge(V_x \supseteq V_{e_1}) \wedge (P_e \supseteq P_{e_1} \cup P_{e_2}) \wedge C_1 \wedge C_2 \end{array}}$$

$$[\text{WHILE}]\ \frac{\rhd e_1 : C_1 \qquad \rhd e_2 : C_2}{\begin{array}{ll} \rhd \mathsf{while}\ e_1\ e_2 : & (V_{e_1} \cap \texttt{true}) \Rightarrow (C_2 \wedge (P_e \supseteq P_{e_2})) \\ & \wedge((V_{e_1} \cap \texttt{false}) \Rightarrow () \subseteq V_e) \wedge (P_e \supseteq P_{e_1}) \wedge C_1 \end{array}}$$

3 Conditional expressions : a good approximation?

Case expressions are the cause for inaccuracy in approximating the program states. Take a look at the for loop.

```
let for x = match x > e2 with
            | true -> ()
            | false -> e3; for (x + 1)
in
for e1
```

The above program is equivalent to `for x = e1 to e2 do e3 done`. Translating this to set constraints is difficult as case statements partition the program states. That is, "`e3; for (x + 1)`" is evaluated under the constraint that $>(V_x, V_{e_2}) \subseteq \texttt{false}$ and "`()`" is evaluated under the constraint that $>(V_x, V_{e_2}) \subseteq \texttt{true}$.

These constraints obviously cannot be and-ed together, as the partitions are mutually disjoint. In other words, $x > e_2$ cannot evaluate to both true and false at the same time. Thus, it is straightforward that each set expression must have different "versions" of itself in each partition. Each case statement creates a "parallel world" where some set constraint becomes true.

The conditional expression approximation is very coarse, as can be observed in the [FOR] rule in section 2. Assume that $e_1 = 1, e_2 = 5, e_3 = ()$. Since $1 \subseteq V_x$, $\texttt{false} \subseteq> (V_x, 5) \subseteq> (V_x, V_{e_2})$, so the conditional expression is activated. Then $V_x \supseteq +(V_x, 1)$ becomes valid. Then the least model must map V_x to $\{x \in \mathbb{Z} | x \geq 1\}$. This overshoots the possible values that x may have, which is $\{1, 2, 3, 4, 5, 6\}$.

Why does this happen? It is because in the condition $V_x \supseteq +(V_x, 1)$, the V_x-s in different sides are in different partitions. The V_x on the left-hand side can be in either partition, as it is not matched against any pattern yet. However, the V_x on the right-hand side must be in the partition where $x > e_2$ is matched against `false`.

Then when are conditional expressions successful? In the case when $e_1 = 5$ and $e_2 = 1$, for example, the conditional expression eliminates the treacherous condition $V_x \supseteq +(V_x, 1)$. In the context of exception analysis, $P_e \supseteq P_{e_3}$ is not considered at all, so the analysis is a bit more accurate. That is, only when the program doesn't lay a foot on the wrong path is when conditional expressions succeed.

The obvious problem is that programs are often written so that all cases are reachable, maybe except for the case when some value is divided by a nonzero constant. Then the question is whether conditional expressions in the [CASE, FOR, WHILE] rules actually decrease false alarms. When implementing the analyzer it might not be a bad idea to eliminate the conditional expressions.

If the conditional expressions are eliminated, then the case function changes to

$$\text{case}(X, Y, p, e) := \begin{cases} (Y \subseteq V_x) \wedge (V_e \subseteq X) & (p = x) \\ V_e \subseteq X & (o.w) \end{cases}$$

and the [FOR, WHILE] rules become

$$[\text{FOR}] \quad \frac{\triangleright e_1 : C_1 \qquad \triangleright e_2 : C_2 \qquad \triangleright e_3 : C_3}{\begin{aligned}\triangleright \mathsf{for}\ x\ e_1\ e_2\ e_3 : \quad & (V_e \supseteq ()) \wedge (V_x \supseteq +(V_x, 1) \cup V_{e_1}) \\ & \wedge (P_e \supseteq P_{e_1} \cup P_{e_2} \cup P_{e_3}) \wedge C_1 \wedge C_2 \wedge C_3\end{aligned}}$$

$$[\text{WHILE}] \quad \frac{\triangleright e_1 : C_1 \qquad \triangleright e_2 : C_2}{\triangleright \mathsf{while}\ e_1\ e_2 : (V_e \supseteq ()) \wedge (P_e \supseteq P_{e_1} \cup P_{e_2}) \wedge C_1 \wedge C_2}$$

TODO:

1. How to solve the set constraints(first in the case when there are no conditional expressions) : based on the work of Nevin Heintze[2]
2. Maybe, how to formulate the "parellel worlds" approach?

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.