Towards a Cost-Effective Estimation of Uncaught Exceptions in SML Programs

Kwang Yi & Sukyoung Ryu
`http://compiler.kaist.ac.kr/~(kwang,puppy)`

9/8/97

Dept. of Computer Science
KAIST (Korea Advanced Institute of Science & Technology)

Exception Handling in SML

```
exception Empty
exception Error of int * (int -> unit)

raise Empty
raise Error(lineno, echo)

e handle Empty => ()
       | Error(n, echo) => echo n
```

Problem

- SML exceptions are first-class objects:

```
fun f(x) = ···raise !x···
```

- Precise exception analysis needs a precise call-graph estimation:

```
fun f(g) = ···g(x) handle E =>···
```

- Conversely, precise call-graph estimation needs a precise exception analysis:

```
fun f(x) = ···e handle E(g) => g(x)···
```

Caveat

Wrong for programs that utilize the generative nature of exception declarations

```
fun fact n =
    let
      exception ZERO
    in
      if n <= 0 then raise ZERO
         else n * (fac(n-1) handle ZERO => 1)
    end
```

Past Approaches

- effect-type system [Guzmán & Suárez 94]
- collecting analysis from an abstract interpreter [Yi 94]
- constraint-type system [Fahndrich & Aiken 97]

Monolithic approaches maybe an overkill because functions and exceptions rarely interfere.

$$\boxed{(\mathcal{A}+\mathcal{B})^3 > \mathcal{A}^3 + \mathcal{B}^3}$$

program	exns	w/ args	arg types	ftn arg
KB.sml	1	1	string	0
ml-lex.sml	8	1	int list * string	0
SML/NJ 109	339	34	string, string*int, int list, intmap, System.Unsafe.object list, symbol list, exn, unit→unit	1 (the `Found` exception in `debug/run.sml`)
HOL	60	18	string, int, record of string/int	0

Our Approach

Decouple the two analyses

- Control Flow Analysis:
  - simple case analysis
  - help from types: accuracy price for decoupling
  - correctness proof
- Exception Flow Analysis
  - focus on exceptions; use the CFA result
  - formalize as set constraint systems
  - medium-sized granularity
  - correctness proof

Control Flow Analysis

- "`(fn x` e`) 0`" calls the immediate ftn
- "`x 0`" calls ftns bound to `x` when λ`x.`e is called
- "`(f 0) 1`" calls ftns that f's body represents
- "`(decon` e`) 0`" calls ftns paired with the data of e:
  - look up construction expressions "`(exn` κ e'`)`"
  - include only ftns(e') whose types can **unify** with the call expression
- Similarly for "`(!`e`) 0`"

Exception Flow Analaysis

Two classes of set constraints for each function f:

- Set variable P_f for function `f`'s uncaught exceptions
  - `fun f(x) =` e`(0)+1`
    P_f includes P_g where g is called at the "e`(0)`."
  - `fun f(x) = raise x`
    P_f includes the exceptions X_f passed to `f`.
- Set variable X_f for exception values during f's application.
  - `fun f(x) = raise x`
    X_f includes X_g where g may call `f`.

Language L

Mono-morphic, eager language

$$
\begin{array}{rcl}
e & ::= & x \quad | \quad \mathtt{1} \\
 & | & \lambda x_\tau.e \\
 & | & e_1\ e_2 \\
 & | & \mathtt{exn}\ \kappa\ e \\
 & | & \mathtt{decon}\ e \\
 & | & \mathtt{case}\ e_1\ \kappa\ e_2\ e_3 \\
 & | & \mathtt{fix}\ f\ \lambda x_\tau.e_1\ \mathtt{in}\ e_2 \\
 & | & \mathtt{raise}\ e \\
 & | & \mathtt{+raise}\ e\ \kappa \\
 & | & \mathtt{-raise}\ e\ \kappa_1 \cdots \kappa_n \\
 & | & \mathtt{handle}\ e_1\ \lambda x_\tau.e_2 \\
\tau & ::= & \tau\ exn \\
 & | & \tau \to \tau \\
 & | & \iota
\end{array}
$$

SML Programs in L (I)

```
e handle ERROR => 1
       | FAIL => 2
```

$$\Downarrow$$

```
handle e λx_exn.
    case x
      ERROR 1
      FAIL  2
  _   -raise x ERROR FAIL
```

SML Programs in L (II)

```
exception E of int list
      ...
 e handle E nil => 1
```

$$\Downarrow$$

handle e $\lambda\, x_{(\iota list)exn}$.
    case x
    E ($\lambda\, y_{\iota list}$.
        case y
         NIL 1
       _ +raise x E) (decon x)
    _ -raise x E

And, every functor applications are in-lined.

Analysis Design Using Set Constraints

- Set expressions se: variables, constants, operators
- Scan the program and collect constraints of the form: $X \supseteq se$
- Compute the least model (constraint solution): map variables to sets.

Concrete Constraints

Start point: like the standard semantics

- Small granularity: set variables V_e and P_e for every expr e
- Set variable V_e ranges over standard values

$$
\begin{array}{rcll}
\mathcal{I}(V_e) & \subseteq & \mathit{Val} & = \mathit{Closure} + \mathit{Exn} + \{\mathtt{1}\} \\
 & & \mathit{Closure} & = \mathit{Lambda} \\
 & & \mathit{Exn} & = \mathit{Con} \times \mathit{Val} \\
 & & \mathit{Con} & = \{\kappa_1, \cdots, \kappa_N\} \\
\mathcal{I}(P_e) & \subseteq & \mathit{Packet} & = \mathit{Exn}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{I}(\mathit{var}(x)) & = & \{v \mid \text{“}e_1\, e_2\text{”} \in e, \lambda x_\tau.\bullet \in \mathcal{I}(V_1), v \in \mathcal{I}(V_2)\} \\
\mathcal{I}(\mathit{app}_V(V_1)) & = & \{v \mid \lambda x_\tau.e \in \mathcal{I}(V_1), v \in \mathcal{I}(V_e)\} \\
\mathcal{I}(\mathit{app}_P(V_1)) & = & \{\kappa\, v \mid \lambda x_\tau.e \in \mathcal{I}(V_1), \kappa\, v \in \mathcal{I}(P_e)\}
\end{array}
$$

$$\triangleright_1 x\colon \{V_x \supseteq var(x)\}$$

$$\frac{\triangleright_1 e_1\colon \mathcal{C}_1 \qquad \triangleright_1 e_2\colon \mathcal{C}_2}{\triangleright_1 e_1\ e_2\colon \{V_e \supseteq app_V(V_1), P_e \supseteq app_P(V_1) \cup P_1 \cup P_2\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

$$\frac{\triangleright_1 e_1\colon \mathcal{C}_1 \qquad \triangleright_1 e_2\colon \mathcal{C}_2}{\triangleright_1 \texttt{handle}\ e_1\ \lambda x_\tau.e_2\colon \begin{array}{l} \{\ V_e \supseteq app_V(\lambda x_\tau.e_2) \cup V_1, \\ \quad P_e \supseteq app_P(\lambda x_\tau.e_2), V_x \supseteq P_1 \\ \}\ \cup \mathcal{C}_1 \cup \mathcal{C}_2 \end{array}}$$

$$\frac{\triangleright_1 e_1\colon \mathcal{C}_1}{\triangleright_1 \texttt{raise}\ e_1\colon \{P_e \supseteq V_1 \cup P_1\} \cup \mathcal{C}_1}$$

Control Flow Analysis

- Notation:

$$e \to \lambda x.e'$$

iff a function expression e may evaluates into a closure of lambda expression $\lambda x.e'$.

- The CFA table Lam is

$$Lam(e) = \{\lambda x.e' \mid e \to \lambda x.e' \text{ is deduced}\}$$

CFA (I)

$$\frac{}{\lambda x.e \to \lambda x_\tau.e}$$

$$\frac{\text{“}\mathtt{fix}\ f\ \lambda x.e_1\ \mathtt{in}\ e_2\text{”} \in \text{pgm}}{f \to \lambda x.e_1}$$

$$\frac{\text{“}e_1\ e_2\text{”} \in \text{pgm}, \quad e_1 \to \lambda x.e_1', \quad e_2 \to \lambda y.e}{x \to \lambda y.e}$$

$$\frac{e_1 \to \lambda x.e_1', \quad e_1' \to \lambda y.e}{e_1 e_2 \to \lambda y.e}$$

CFA (II)

$$\frac{\text{“}\texttt{exn}\,\kappa\,e_1\text{”} \in \text{pgm}, \quad e_1 \rightarrow \lambda x.e', \quad \mathit{type}(e_1) = \mathit{type}(\texttt{decon}\,e)}{\texttt{decon}\,e \rightarrow \lambda x.e'}$$

$$\frac{e_2 \rightarrow \lambda x.e}{\texttt{case}\,e_1\,\kappa\,e_2\,e_3 \rightarrow \lambda x.e} \qquad \frac{e_3 \rightarrow \lambda x.e}{\texttt{case}\,e_1\,\kappa\,e_2\,e_3 \rightarrow \lambda x.e}$$

$$\frac{e_1 \rightarrow \lambda x.e}{\texttt{handle}\,e_1\,\lambda x.e_2 \rightarrow \lambda x.e} \qquad \frac{e_2 \rightarrow \lambda x.e}{\texttt{handle}\,e_1\,\lambda x.e_2 \rightarrow \lambda x.e}$$

Correctness

Proposition 1 $\rhd_1 e{:}\mathcal{C}$ *implies that, for every function-typed expression* e' *of the program* e, $Lam(e') \supseteq lm(\mathcal{C})(V_{e'})$.

Exception Constraints

- Small granularity: set variables X_e and P_e for every expr e
- Set variable X_e range over only exception names

$$
\begin{array}{rcll}
\mathcal{I}(X_e) & \subseteq & Exn & = \{\kappa_1, \cdots, \kappa_N\} \\
\mathcal{I}(P_e) & \subseteq & Packet & = Exn
\end{array}
$$

$$\triangleright_2\, x\colon\ \{X_x \supseteq \mathit{var}(x)\}$$

$$\frac{\triangleright_2\, e_1\colon\ \mathcal{C}_1 \qquad \triangleright_2\, e_2\colon\ \mathcal{C}_2}{\triangleright_2\, e_1\, e_2\colon\ \{X_e \supseteq \mathit{app}_X(e_1), P_e \supseteq \mathit{app}_P(e_1) \cup P_1 \cup P_2\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

$$\frac{\triangleright_2\, e_1\colon\ \mathcal{C}_1 \qquad \triangleright_2\, e_2\colon\ \mathcal{C}_2}{\triangleright_2\, \texttt{handle}\, e_1\, \lambda x_\tau.e_2\colon\ \begin{array}{l}\{\ X_e \supseteq \mathit{app}_X(\lambda x_\tau.e_2) \cup X_1,\\ \quad P_e \supseteq \mathit{app}_P(\lambda x_\tau.e_2), X_x \supseteq P_1\\ \}\ \cup \mathcal{C}_1 \cup \mathcal{C}_2\end{array}}$$

$$\frac{\triangleright_2\, e_1\colon\ \mathcal{C}_1}{\triangleright_2\, \texttt{raise}\, e_1\colon\ \{P_e \supseteq X_1 \cup P_1\} \cup \mathcal{C}_1}$$

$$\frac{\triangleright_2\, e_1\colon\ \mathcal{C}_1}{\triangleright_2\, \texttt{-raise}\, e_1\, \kappa_1 \cdots \kappa_n\colon\ \{P_e \supseteq (X_1 \setminus_{e_1} \{\kappa_1, \cdots, \kappa_n\}) \cup P_1\} \cup \mathcal{C}_1}$$

Operators in Exception Constraints

$$
\begin{array}{lcl}
\mathcal{I}(var(x)) & = & \{\kappa \mid \text{“}e_1\, e_2\text{”} \in e, \lambda x_\tau.\bullet \in Lam(e_1), \kappa \in \mathcal{I}(X_2)\} \\
\mathcal{I}(app_X(e_1)) & = & \{\kappa \mid \lambda x_\tau.e \in Lam(e_1), \kappa \in \mathcal{I}(X_e)\} \\
\mathcal{I}(app_P(e_1)) & = & \{\kappa \mid \lambda x_\tau.e \in Lam(e_1), \kappa \in \mathcal{I}(P_e)\} \\
isExn(\tau) & = & \text{true iff } \tau = \tau'\ exn
\end{array}
$$

$$
\mathcal{I}(X_1 \setminus_e \{\kappa_1, \cdots, \kappa_n\}) = \begin{cases} \mathcal{I}(X_1) \text{ if } type(e) = \tau'\, exn\ \wedge\ isExn(\tau') \\ \mathcal{I}(X_1) \setminus \{\kappa_1, \cdots, \kappa_n\}\ \text{ o.w.} \end{cases}
$$

```
(* where x is Exit(Error) *)
case x of
  Error => 1
  _     => -raise x Error
```

Consistency: $\rhd_2$ and $\rhd_1$

Proposition 2 *If Lam is safe with respect to $\rhd_1$ then for every sub-expression $e_i \in e$:*

$$\mathcal{I}_2(X_i) \supseteq |\mathcal{I}_1(V_i)| \quad \textit{and} \quad \mathcal{I}_2(P_i) \supseteq |\mathcal{I}_1(P_i)|$$

where $|\mathcal{V}|$ is the set of exception names constituting exception values in $\mathcal{V}$

$$|\mathcal{V}| \stackrel{def}{=} \{\kappa \mid \kappa\, v \in \mathcal{V}\} \cup \big|\{v \mid \kappa\, v \in \mathcal{V}\}\big|.$$

Function's Exception Constraints

- Medium granularity: set variables X_f and P_f for every function f

$$\begin{array}{rcl} \mathcal{I}(X_f) & \subseteq & \mathit{Exn} \quad = \{\kappa_1, \cdots, \kappa_N\} \\ \mathcal{I}(P_f) & \subseteq & \mathit{Packet} = \mathit{Exn} \end{array}$$

$$f \;\triangleright_3\; x\colon \{X_f \supseteq var(x)\}$$

$$\frac{f \;\triangleright_3\; e_1\colon \mathcal{C}_1 \qquad f \;\triangleright_3\; e_2\colon \mathcal{C}_2}{f \;\triangleright_3\; e_1\, e_2\colon \{X_f \supseteq app_X(e_1, X_f), P_f \supseteq app_P(e_1, X_f)\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

$$\frac{g \;\triangleright_3\; e_g\colon \mathcal{C}_1 \qquad h \;\triangleright_3\; e_2\colon \mathcal{C}_2}{f \;\triangleright_3\; \texttt{handle}\, e_g\, \lambda_h x_\tau.e_2\colon \;\begin{array}{l}\{\; X_f \supseteq app_X(\lambda_h x_\tau.e_2, P_g) \cup X_g, \\ \quad P_f \supseteq app_P(\lambda_h x_\tau.e_2, P_g) \\ \} \cup \mathcal{C}_1 \cup \mathcal{C}_2\end{array}}$$

$$\frac{f \;\triangleright_3\; e_1\colon \mathcal{C}_1}{f \;\triangleright_3\; \texttt{raise}\, e_1\colon \{P_f \supseteq X_f\} \cup \mathcal{C}_1}$$

$$\frac{f \;\triangleright_3\; e_1\colon \mathcal{C}_1}{f \;\triangleright_3\; \texttt{-raise}\, e_1\, \kappa_1 \cdots \kappa_n\colon \{P_f \supseteq X_f \setminus_{e_1} \{\kappa_1, \cdots, \kappa_n\}\} \cup \mathcal{C}_1}$$

Operators in Function's Exception Constraints

$$
\begin{array}{lcl}
Owner(x) & = & f \quad \text{where} \quad \lambda_f x_\tau.e \\
\mathcal{I}(var(x)) & = & \mathcal{I}(X_{Owner(x)}) \\
\mathcal{I}(app_X(e_1, \mathcal{X})) & = & \\
\quad \{\kappa \mid \lambda_g x_\tau.e \in Lam(e_1),\ \kappa \in \mathcal{I}(X_g),\ \mathcal{I}(X_g) \supseteq \mathcal{I}(\mathcal{X})\} & & \\
\mathcal{I}(app_P(e_1, \mathcal{X})) & = & \\
\quad \{\kappa \mid \lambda_g x_\tau.e \in Lam(e_1),\ \kappa \in \mathcal{I}(P_g),\ \mathcal{I}(X_g) \supseteq \mathcal{I}(\mathcal{X})\} & &
\end{array}
$$

Consistency: $\rhd_3$ and $\rhd_2$

Proposition 3 *If* $g \rhd_3 e'{:}\mathcal{C}'$ *occurs during*

$$\texttt{main} \rhd_3 e{:}\mathcal{C}_3$$

then

$$\mathcal{I}_3(X_g) \supseteq \mathcal{I}_2(X_{e'}) \quad \textit{and} \quad \mathcal{I}_3(P_g) \supseteq \mathcal{I}_2(P_{e'}).$$

Analysis Example (1)

```
(1)   fun m() = f(exn κ 1)
(2)   fun f(x) = handle g(x) λ_h y.1
(3)   fun g(x) = raise x
```

From line (1) $\begin{cases} X_m \supseteq \kappa \\ X_f \supseteq X_m,\ X_m \supseteq X_f \\ P_m \supseteq P_f \end{cases}$

From line (2) $\begin{cases} X_g \supseteq X_f,\ X_f \supseteq X_g \\ P_f \supseteq P_h,\ X_h \supseteq P_g \end{cases}$

From line (3) $P_g \supseteq X_g$

Corresponding equations:

$$\begin{array}{llll} X_m = \{\kappa\}, & X_f = X_m \cup X_g, & X_g = X_f, & X_h = P_g, \\ P_m = P_f, & P_f = P_h, & P_g = X_g. & \end{array}$$

The least solution is:

$$\begin{array}{lll} X_m = \{\kappa\}, & X_f = \{\kappa\}, & X_g = \{\kappa\}, \\ P_m = \emptyset, & P_f = \emptyset, & P_g = \{\kappa\}. \end{array}$$

Analysis Example (2)

```
exception Fail of int
fun f() = g() handle Fail(1) => true
fun g() = raise (exn Fail 1)
```

$$\Downarrow$$

```
fun f() = handle g() λ x. case x of
                          Fail (λ y. case y of
                                      1 true
                                      _ +raise x Fail)
                               (decon x)
                          _    -raise x Fail
```

Our analysis conclude that `f` has an escaping exception `Fail`.

Exception's Argument

The exception space becomes the set of tuples:

$$Exn = \{\kappa_1, \cdots, \kappa_N\} \times Expr$$

$$\frac{f \ \rhd_3 \ e_1 \colon \mathcal{C}_1}{f \ \rhd_3 \ \texttt{exn}\, \kappa\, e_1 \colon \{X_f \supseteq \langle \kappa, e \rangle\} \cup \mathcal{C}_1}$$

And $X_1 \setminus_e \{\kappa_1, \cdots, \kappa_n\}$ removes tuples headed by κ_i's.

Performance

program	lines	cfa+cs	solve	result
KB.sml	519	0.54	0.07	1 (1x,1r,10h)
ml-lex.sml	1204	0.89	0.47	3 (10x,19r,10h)
compiler.sml	3511	3.65	0.10	3 (11x,34r,4h)
instantiate.sml	1384	2.74	0.04	2 (7x,8r,18h)
typecheck.sml	648	6.07	0.03	0 (1x,2r,17h)
moduleutil.sml	847	4.37	0.08	3 (3x,25r,23h)
pathname.sml	426	0.09	0.01	4 (4x,6r,3h)
string-cvt.sml	454	0.13	0.03	1 (1x,10r,4h)

- Integrated within SML/NJ 108.13
- Print out: user ftn id $\rightarrow$ uncaught exceptions
- Run on DEC Alpha 1000 4/200 (128MB, CPU 21064-200)

Summing Up

- Optimistic that we are near to a "right" balance of the cost-accuracy tradeoff
- Consistency proves between constraint systems of increasing granularity
- Final goals:
  - Make the analysis overhead $\leq$ 10% of SML compilation time
  - Analyze modules in isolation
  - Consider primitive exceptions: `Subscript`, `Empty`, `Overflow`, etc.
- Expected to be included in SML/NJ 111