CS6848 - Principles of Programming Languages
Exceptions

V. Krishna Nandivada

IIT Madras

Recap

- Flow analysis using 0-CFA and some simple improvements.
- Closure conversion (revisit).

What you should be able to answer? (necessary not sufficient)

- Given a set of flow constraints solve them to get the flow sets.
- Translate closures in Scheme to C.

Reminder

- Assignment due in 3 days.
- Four more classes to go (Last instructional day for CS6848 - 18th April)
- Final exam on 28th May
- Portion - Post mid-term.

Exceptions

- Real-world programming – a function needs to signal to its caller (or some one in the call chain) that it is not able perform some task. Examples:
  - Division by zero, array out of bounds, out of memory, etc.
- One option is to return a special value. Issue:
  - Every caller has to now look for the special value explicitly.
- Option 2: Automatic transfer of program control. Multiple variants exist:
  - Abort the program when an exception occurs.
  - “throw” the exception – trap + recover (aka “caught”)
  - Pass programmer specified data along with the exception – Programmer defined exceptions.

Extending simply typed lambda calculus with errors

- Errors - abort the program.
- Recall: Extending the language requires - extension to syntax, values, type rules and operational semantics.
- Expressions

$$e ::= \cdots | \texttt{error}$$

- Values – we don’t add any new values (discussion to follow)
- Types. What should be the type of `error`?Do we need any special types?

Type rules

- There is no restriction on the return type of a function.
- Any function can throw an `error`.
- So for each function $s \to t$, we want the type of `error` $: t$
- For the program to typecheck:
  - If we allow subtyping: then `error` $: \bot$.
  - If we allow polymorphism: then `error` $: \forall X.X$

Operational semantics

- We need rules for only application.

$$\begin{array}{ll} \texttt{error}\ e \to \texttt{error} & AppError1 \\ v\ \texttt{error} \to \texttt{error} & AppError2 \end{array}$$

- Summary: abandon the work if there is an error (during the evaluation of the argument or the function).
- Q: Can we get a situation where we get: `error error` ?
  - NO. Because, `error` is not a value.
- Also note, the evaluation order.

Modification to type soundness

- Recall: Progress lemma: If e is a closed expression, and $A \vdash e : t$ then either e is a value or `error`, or there exists e' such that $e \to_V e'$.

Exceptions. Variant 2

- Let us “catch” the exception and do something relevant.
- Extension to syntax

$$e ::= \cdots | \texttt{try}\ e\ \texttt{with}\ e$$

- New typing rules:

$$\text{Type-Try-With}\ \frac{A \vdash e_1 : t \qquad A \vdash e_2 : t}{A \vdash \texttt{try}\ e_1\ \texttt{with}\ e_2 : t}$$

Operational semantics

- Evaluating expressions that don't result in `error`.

$$\texttt{try}\ v\ \texttt{with}\ e \rightarrow v$$

- Evaluating an expression that evaluates to an error.

$$\texttt{try error with}\ e \rightarrow e$$

- Step

$$\frac{e_1 \rightarrow e_1'}{\texttt{try}\ e_1\ \texttt{with}\ e_2 \rightarrow \texttt{try}\ e_1'\ \texttt{with}\ e_2}$$

Exceptions variant 3 - User defined

- The program point where the exception is thrown may want to pass information.
- The handler may use this information - to take relevant action (such as recovery, reversal, display some relevant message, and so on).
- Extension to syntax

$$e ::= \cdots |\texttt{throw}\ e\ |\texttt{try}\ e\ \texttt{with}\ e$$

- New typing rules:

$$\text{Type-throw}\frac{A \vdash e_1 : t}{A \vdash \texttt{throw}\ e_1\ : t}$$

$$\text{Type-Try-With}\frac{A \vdash e_1 : t \qquad A \vdash e_2 : t_1 \rightarrow t}{A \vdash \texttt{try}\ e_1\ \texttt{with}\ e_2 : t}$$

Operational semantics

- Application of a throw.

$$(\texttt{throw}\ v)e \rightarrow \texttt{throw}\ v$$

- throw as an argument.

$$v_1(\texttt{throw}\ v_2) \rightarrow \texttt{throw}\ v_2$$

- throw of throw

$$\texttt{throw}\ (\texttt{throw}\ v) \rightarrow \texttt{throw}v$$

- Step throw.

$$\frac{e_1 \rightarrow e_2}{\texttt{throw}\ e_1 \rightarrow \texttt{throw}\ e_2}$$

Operational semantics (contd)

- try with no exception.

$$\texttt{try}\ v\ \texttt{with}\ e \rightarrow v$$

- Evaluating an expression that throws an expression

$$\texttt{try throw}\ v\ \texttt{with}\ e \rightarrow ev$$

- Step try.

$$\frac{e_1 \rightarrow e_1'}{\texttt{try}\ e_1\ \texttt{with}\ e_2 \rightarrow \texttt{try}\ e_1'\ \texttt{with}\ e_2}$$

Recap

- Exceptions
- Reason about programs with exceptions.
- Type rules and operational semantics for languaes with exceptions.