CS6848 - Principles of Programming Languages
Exceptions

V. Krishna Nandivada

IIT Madras

Recap

- Exceptions

Announcements

- Assignment 6 is out - Due 30th (Not a Saturday!)
- Two more classes to go (Last instructional day for CS6848 - 18th April)
- Final exam on 28th May 11AM.
- Portion - Post mid-term.

Versioning Exceptions

- Traditional exceptions provide only transfer of control.
- Used typically for handling cases when unexpected conditions arise.
- The <u>store</u> (maps memory locations to values) is left untouched.
  - It is left to the programmer to manually undo any changes.
  - Q: Is handling the environment (maps variables to values) easy?
- Q: Can we provide transaction semantics to the non-local control flow of control-exceptions?
- Goal: Revert computation to a well-defined state in response to unexpected or undesirable conditions.

Versioning Exceptions

- Each code is protected by an exception handler (installed by `try`.
- A versioned exception ensures that the content of the store, when the exception is raised reflects the program state when the corresponding handler was installed.
- The data generated in the code protected by such exceptions are implicitly versioned.
- Each version is assocated with a particular generative exception value.
- When an exception is raised, the version corresponding to the associated exception value is is restored.
- A handler is provided, which lets the programmer to re-executed the protected code or print error message and so on.

Background needed

- When do you need store?
- Modeling store.

Extending the language with references

Extending the syntax

- $$e = \cdots | e;e | ref\ e | !e | e_1 := e_2 | unit$$
  - Creating a reference - creates a cell in memory.
  - The value stored in the cell is the value the expression e evaluates to.
  - Say, r is a reference, then `let` $s = r\ \ e$ makes s an alias to r.
    - Setting $r := 32$, will change the value of s and vice versa.

Extending types

- $$t := \cdots | Ref\ t | Unit$$
- **Extending values**
  $$v ::= \cdots | l | unit$$
- Think of $Unit$ as the `void` type of C.
- The result of evaluating an expression of type $Unit$ is the constant $unit$

Type rules

- Reference creation.
  $$\frac{A \vdash e : t}{A \vdash ref\ e : Ref\ t}$$
- Dereference
  $$\frac{A \vdash e : Ref\ t}{A \vdash !e : t}$$
- Assignment.
  $$\frac{A \vdash e_1 : Ref\ t_1 \qquad A \vdash e_2 : t_1}{A \vdash e_1 := e_2 : Unit}$$
  - Note: The left hand side is not necessarily a variable.

Modelling the store

- Store can be seen as array of values.
- Store can be seen as a map $L \to Values$, where L is the set of locations, and $Values$ is the set of values.
- We use σ to represent the store.
- Rules of operational semantics now will use σ.

Syntax for store

- $$\sigma ::= \Phi | \sigma, l = v$$

Typing store elements

$$\Sigma ::= \Phi | \Sigma, l : t$$

Evaluation rules

Defined over the reflexive, transitive closure of $\to_V$:

$$\to_V : \langle Expression, Store \rangle \to_V \langle Expression, Store \rangle$$

- Step - Application
  $$\frac{\langle e_1, \sigma \rangle \to_V \langle e_1', \sigma' \rangle}{\langle e_1 e_2, \sigma \rangle \to_V \langle e_1' e_2, \sigma' \rangle}$$
- Step - Arguments
  $$\frac{\langle e_2, \sigma \rangle \to_V \langle e_2', \sigma' \rangle}{\langle v_1 e_2, \sigma \rangle \to_V \langle v_1 e_2' \sigma' \rangle}$$
- Apply
  $$\langle (\lambda x.e)v, \sigma \rangle \to_V \langle e[x/v], \sigma \rangle$$

Evaluation rules

- Create reference

$$\langle ref\ v, \sigma\rangle \rightarrow_V \langle l, \sigma[l \mapsto v]\rangle, \text{ where } l \text{ is fresh}$$

- Step - reference

$$\frac{\langle e, \sigma\rangle \rightarrow_V \langle e', \sigma'\rangle}{\langle ref\ e, \sigma\rangle \rightarrow_V \langle ref\ e', \sigma'\rangle}$$

- Dereference a location

$$\langle !l, \sigma\rangle \rightarrow_V \langle \sigma(l), \sigma\rangle$$

- Step - Dereference

$$\frac{\langle e, \sigma\rangle \rightarrow_V \langle e', \sigma'\rangle}{\langle !e, \sigma\rangle \rightarrow_V \langle !e', \sigma'\rangle}$$

Evaluation rules

- Assignment.

$$\langle l := v, \sigma\rangle \rightarrow_V \langle unit, \sigma[l \mapsto v]\rangle$$

- Step - Assignment (lhs)

$$\frac{\langle e_1, \sigma\rangle \rightarrow_V \langle e_1', \sigma'\rangle}{\langle e_1 := e_2, \sigma\rangle \rightarrow_V \langle e_1' := e_2, \sigma'\rangle}$$

- Step - Assignment (rhs)

$$\frac{\langle e_2, \sigma\rangle \rightarrow_V \langle e_2', \sigma'\rangle}{\langle l := e_2, \sigma\rangle \rightarrow_V \langle l := e_2', \sigma'\rangle}$$

Versioning exceptions

Extensions to syntax

- $$e ::= \cdots \mid vExn(x) \mid try\ (y, e) \mid restore\ (p, q)$$
  - $vExn(x)$ – constructs a new exception. x is bound to a procedure that defines the handler for this exception.
  - $try\ (y, e)$ – evaluates y to an exception E, and then evaluates e.
  - $restore\ (p, q)$ – p evaluates to an exception (say E).
    - Raises exception E.
    - Control is transferred to the closest enclosing try expression for E.
    - the handler of E is evaluated with q as the argument.
    - Restores the state.

Q:How to construct try-expression with multiple catches?

Versioning exceptions

Extension to types

- $$t ::= \cdots | Exn(t_1 \rightarrow t_2)$$

Extension to type rules

- Exception construction.

$$\frac{A \vdash x : t_1 \rightarrow t_2}{A \vdash vExn(x) : Exn(t_1 \rightarrow t_2)}$$

- Try block

$$\frac{A \vdash x : Exn(t_1 \rightarrow t_2) \qquad A \vdash e : t_2}{A \vdash try(x, e) : t_2}$$

- Restore

$$\frac{A \vdash y : t_1 \qquad A \vdash x : Exn(t_1 \rightarrow t_2)}{A \vdash restore(x, y) : t_2}$$

Operational Semantics

- In the style of CE^2SK: Control, Environment, Exception-stack, Store, Continuation Pointer: Each evaluation is defined as a reflexive and transitive closure over $\rightarrow_V$

$$\rightarrow_V : State \rightarrow_V State$$

- Standard rules apply for non-exception expressions. For example:
  - $$(let\ x = c\ e, \rho, \Sigma, \sigma, k) \rightarrow_V (e, \rho[x \mapsto c], \Sigma, k, \sigma)$$
  - $$(x, \rho, \Sigma, \sigma, k) \rightarrow_V (k, \rho(x), \sigma, \Sigma)$$
  - $$(\{ret\langle x, e, p\rangle\} \oplus k, v, \sigma, \Sigma) \rightarrow_V (e, \rho[x \mapsto v], k, \sigma, \Sigma)$$

Operational semantics for versioning exceptions

See the hand out.