Exceptions: implementation and implications

Amer Diwan

Issues

• Since exceptions are for exceptional situations, they shouldn’t happen often
  – Should be “free” except when used
• Want to minimize the indirect impact of exceptions
• Will assume a simple exception model, a.l.a. Modula-3, Java
An example

```
TRY
  g();
EXCEPT
  e1 => <C1>;
  e2 => <C2>;
END

TRY
  h();
EXCEPT
  e1 => <C3>;
END;
```

```
IF (cond)
  RAISE e1;
ELSE
  RAISE e2;
END;
```

When a handler is not found in a routine

```
TRY
TRY
  g();
EXCEPT
  e1 => <C1>;
  e2 => <C2>;
END
g();
EXCEPT
  ELSE => <C6>
END
```

```
TRY
TRY
  h();
EXCEPT
  e1 => <C3>;
END;
EXCEPT
  ELSE => <C5>
END
```

```
TRY
  IF (cond)
    RAISE e1;
  ELSE
    RAISE e2;
  END;
```

What do <C4>, <C5>, and <C6> do?
Implementing exceptions

• In each scope, we know statically, what exceptions are handled
  – May have a “handler” for the “ELSE” exception too
• Make a table mapping exceptions to exception handlers for each scope

An Example

```
TRY
TRY
  p1: g();
  EXCEPT
    e1 => <C1>;
    e2 => <C2>;
    END
  p2: g();
  EXCEPT
    ELSE => <C6>
    END
TRY
TRY
  p3: h();
  EXCEPT
    e1 => <C3>;
    END;
  EXCEPT
    ELSE => <C5>
    END
IF (cond)
  RAISE e1;
ELSE
  p4: RAISE e2;
  END;
  EXCEPT
    ELSE => <C4>
    END
```

```
f
TRY
  TRY
    p1: g();
    EXCEPT
      e1 => <C1>;
      e2 => <C2>;
      END
    p2: g();
    EXCEPT
      ELSE => <C6>
    END
TRY
p1
  e1, <C1>
  e2, <C2>
  *, <C6>
p2
  *, <C6>
p3
  e1, <C3>
  *, <C5>
p4
  *, <C4>
```
An exception is raised

Path: p1, p3, p4

Observation: can use return address to locate exception table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td></td>
<td>Exception table entry: (&lt;C2&gt;) (i.e., e2’s exception handler)</td>
</tr>
<tr>
<td>g</td>
<td>return: p1</td>
<td>Exception table entry: (&lt;C5&gt;) (i.e., reraise)</td>
</tr>
<tr>
<td>h</td>
<td>return: p3</td>
<td>Exception table entry: (&lt;C4&gt;) (i.e., reraise)</td>
</tr>
</tbody>
</table>

Important properties of the technique

- Tables are built at compile time
  - no run-time overhead
- When an exception is raised, look for handler one activation record at a time
  - some handlers will reraise the exception
  - may need to unwind the stack
How exceptions affect the control flow in the program

• A program analysis must assume worst case about raising of exceptions

```
TRY
  x = y / z;
  t = foo();
EXCEPT
  div_0 =>
    <S1>
  | foo_dies1 =>
    <S2>
  | foo_dies2
    <S3>
END
```

Implications

• **Optimizations** cannot easily move code around instructions that may cause an exception
• **Optimizations** cannot easily move code that causes exceptions
Analysis to reduce pollution due to exceptions

• Can analyze programs to see what
  – statements will not raise an exception
    • e.g., if (d != 0) t = n/d;
    • e.g., if (p != NULL) *p = 0;
  – exceptions will be raised by each statement
    • E.g., foo can only raise foo_dies1 but never foo_dies2

Checked exceptions

• Checked exceptions alleviate this analysis
  – All “checked” exceptions that may be raised by a call have to be in the “throws” clause of callees
    • e.g., foo() throws {foo_dies1, foo_dies2} {...}
  – All exceptions in M-3 are checked. Some exceptions in Java are not checked
Other issues with exceptions

• If an exception causes a return from a function must invoke whatever cleanups are registered for that function
  – e.g., destructors for local variables
• Must register cleanups in exception-tables

Discussion

• Are implicit exceptions a good idea?
  – for div/0, *p, ... ?
  – for out of memory errors?
• Comparison of exception handling to “return” code
  – Return code typically restricts one to success/no success. More expressiveness in exceptions
  – Exceptions cannot be ignored
  – ...?
Summary

- The direct costs of exception handling are low
  - Tables built at compile time remove run-time overhead except when an exception is raised
- Indirect costs of exception handling may be high
  - Control-flow edges can severely restrict optimizations

Next two lectures

- First lecture: Continuations
  - Readings: See class web page for pointers
- Second lecture: Implementation of continuations