Modelica transformational Debugger and implementation in the OpenModelica Compiler

Martin Sjölund <martin.sjolund@liu.se>
Peter Fritzson <peter.fritzson@liu.se>
Linköping University, Sweden

OpenModelica Workshop
Feb 2012, Linköping University, Sweden
What Happens in a Modelica Compiler?
Example - RC Circuit (Diagram)
model RC
    Modelica.Electrical.Analog.Basic.Ground ground1;
    Modelica.Electrical.Analog.Basic.Resistor resistor1(R = 100);
    Modelica.Electrical.Analog.Basic.Capacitor capacitor1(C = 0.01);
    Modelica.Electrical.Analog.Sources.SineVoltage sinevoltage1(V = 240, freqHz = 50);

equation
    connect(capacitor1.n,ground1.p);
    connect(sinevoltage1.n,ground1.p);
    connect(resistor1.n,sinevoltage1.p);
    connect(resistor1.p,capacitor1.p);
end RC;
class RC // 24 equations and variables

... 
edition

equation

... 

ground1.p.v = 0.0;

0.0 = resistor1.p.i + resistor1.n.i;

resistor1.i = resistor1.p.i;

resistor1.T_heatPort = resistor1.T;

capacitor1.i = capacitor1.C * der(capacitor1.v);

capacitor1.v = capacitor1.p.v - capacitor1.n.v;

0.0 = capacitor1.p.i + capacitor1.n.i;

capacitor1.i = capacitor1.p.i;

... 
ed

end RC;
class RC // 24 equations and variables

... equation ...

ground1.p.v = 0.0;

0.0 = resistor1.p.i + resistor1.n.i;

resistor1.i = resistor1.p.i;

resistor1.T_heatPort = resistor1.T;

capacitor1.i = capacitor1.C * der(capacitor1.v);

capacitor1.v = capacitor1.p.v - capacitor1.n.v;

0.0 = capacitor1.p.i + capacitor1.n.i;

capacitor1.i = capacitor1.p.i;

... end RC;

class RC // 5 equations and variables

... equation

// 14 alias variables 5 constants

sinevoltage1.signalSource.y = sinevoltage1.signalSource.offset + (if time < sinevoltage1.signalSource.startTime then 0.0 else sinevoltage1.signalSource.amplitude * sin(6.28318530717959 * (sinevoltage1.signalSource.freqHz * (time - sinevoltage1.signalSource.startTime)) + sinevoltage1.signalSource.phase));

resistor1.v = capacitor1.v - sinevoltage1.signalSource.y;

capacitor1.i = -resistor1.v / resistor1.R_actual;

resistor1.LossPower = -resistor1.v * capacitor1.i;

0.0 = capacitor1.p.i + capacitor1.n.i;

capacitor1.i = capacitor1.p.i;

... end RC;
Debugging Equation Systems

- Modelica involves a lot of magic
  - Lots of math
  - Hidden to users
  - Users want to access this information
  - Some algorithms work better for certain input
  - Not intuitive
    - No explicit control flow
    - Numerical solvers
    - Linear/Non-linear blocks
    - Optimization
    - Events
Error solving nonlinear system 132

time = 0.002
residual[0] = 0.288956
x[0] = 1.105149
residual[1] = 17.000400
x[1] = 1.248448
...

Error solving nonlinear system 132 <more info>

time = 0.002
residual[0] = 0.288956
x[0] = 1.105149
residual[1] = 17.000400
x[1] = 1.248448
...

Several Levels

- (Graphical Representation)
- Source Code
- Flat Equation-System
- Optimized Equation-System
- Translated Code (typically C)

It should always be possible to go backwards

- Simple for flattened equation system to source
- Harder for optimized code
Symbolic Transformations

- From source code to flat equations
  - Most of the structure remains
  - Few symbolic manipulations (mostly simplification/evaluation)

- Equation System Optimization
  - Changes structure
  - Strong connected components
  - Variable replacements
  - … and more
Tracing Transformations

- Simple Idea
  - Store transformations as equation metadata
  - Works best for operations on single equations
- Each kind of transformation is different
  - Alias Elimination \((a = b)\)
  - Gaussian Elimination (linear systems, several equations)
  - Equation solving \((f_1(a,b) = f_2(a,b), \text{ solve for } a)\)
  - ...
OpenModelica Implementation (1)

- Equation source has an extra field for transformations
- Optimization modules add information to this field
  - Some operations now need to keep track of any changes made
  - Expression simplification changed to fix-point algorithm

Before:

```plaintext
e2 = simplify(e1);
```

Now:

```plaintext
(e2, b) = simplify(e1);
source = addSymTSSimplify(b, source, e1, e2);
```
OpenModelica Implementation (2)

- Overhead?
  - It is so fast we enable tracing by default (1 extra comparison and/or cons operation per optimization)
  - No overhead unless you print the trace
    - +simCodeTarget=Dump
Alias Elimination

The alias relation $a=b$ stored in variable $a$

The equations are e.g. stored as (lhs,rhs,list<ops>)

\[
\begin{align*}
a &= b \\
c &= a + b \\
d &= a - b \\
\text{c = a + b (subst a=b) } &\Rightarrow \text{ c = b + b (simplify) } &\Rightarrow \text{ c = 2 * b} \\
d &= a - b (subst a=b) &\Rightarrow \text{ d = b - b (simplify) } &\Rightarrow \text{ d = 0.0}
\end{align*}
\]
Debugging Using the Trace

- **Text-file**
  - Initial implementation
  - Verify performance and correctness of the trace

- **Database (SQL/XML queries)**
  - Graphical debugging
  - Cross-referencing equations (dependents/parents)
  - Ability to see why a variable is solved in a particular way
  - Requires a schema
0 = y + \text{der}(x \times \text{time} \times z); z = 1.0;

(1) subst:
\[ y + \text{der}(x \times (\text{time} \times z)) \]
\[ \Rightarrow \]
\[ y + \text{der}(x \times (\text{time} \times 1.0)) \]

(2) simplify:
\[ y + \text{der}(x \times (\text{time} \times 1.0)) \]
\[ \Rightarrow \]
\[ y + \text{der}(x \times \text{time}) \]

(3) expand derivative (symbolic diff):
\[ y + \text{der}(x \times \text{time}) \]
\[ \Rightarrow \]
\[ y + (x + \text{der}(x) \times \text{time}) \]

(4) solve:
\[ 0.0 = y + (x + \text{der}(x) \times \text{time}) \]
\[ \Rightarrow \]
\[ \text{der}(x) = \frac{((-y) - x)}{\text{time}} \]
\[ \text{time} \neq 0 \]
<table>
<thead>
<tr>
<th>Differentiation</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{d}{dt} L^2.0 )</td>
<td>( 2.0 \times (\text{der}(x) \times x + \text{der}(y) \times y) )</td>
</tr>
<tr>
<td>=&gt; 0.0</td>
<td>=&gt; 2.0 \times (\text{DER}.x \times x + \text{DER}.y \times y)</td>
</tr>
<tr>
<td>Differentiation:</td>
<td></td>
</tr>
<tr>
<td>( \frac{d}{dt} x^2.0 + y^2.0 )</td>
<td>( 2.0 \times (u \times x + \text{DER}.y \times y) )</td>
</tr>
<tr>
<td>=&gt; ( 2.0 \times (\text{der}(x) \times x + \text{der}(y) \times y) )</td>
<td>=&gt; ( 2.0 \times (u \times x + v \times y) )</td>
</tr>
<tr>
<td></td>
<td>=&gt; ( 2.0 \times (u \times xloc[1] + v \times xloc[0]) )</td>
</tr>
</tbody>
</table>
Readability of Trace

- Most equations have very few transformations on them
- Most of the interesting equations have a few
  - Still rather readable

<table>
<thead>
<tr>
<th># Ops</th>
<th>Frequency</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>457</td>
<td>Parameters</td>
</tr>
<tr>
<td>1</td>
<td>89</td>
<td>Dummy eq &amp; know var</td>
</tr>
<tr>
<td>2</td>
<td>720</td>
<td>Alias vars</td>
</tr>
<tr>
<td>3</td>
<td>479</td>
<td>Alias vars</td>
</tr>
<tr>
<td>4</td>
<td>124</td>
<td>Alias after simplify</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>Alias after simplify</td>
</tr>
<tr>
<td>6</td>
<td>99</td>
<td>Alias after simplify</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
<td>Scalar eq</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>110</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>72</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>...</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>Known constant after many replacements</td>
</tr>
<tr>
<td>21</td>
<td>27</td>
<td>World object (3x3 matrix with many occurances of aliased vars)</td>
</tr>
</tbody>
</table>
Future Work

- Create database instead of text-file
- Graphical debugger
- Simulation runtime uses database
- Tracing in algorithmic code
- More operations recorded
  - Dead code elimination
  - Control flow and events
  - Forgotten optimization modules