Debugging Symbolic Transformations in Equation Systems

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

4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools
September 2011, ETH Zürich, Switzerland

Debugging EOO Languages

- Not intuitive
  - No explicit control flow
  - Numerical solvers
  - Linear/Non-linear blocks
  - Optimization
  - Events

Typical OMC Error Message

Error solving nonlinear system 132

time = 0.002

residual[0] = 0.288956

x[0] = 1.105149

residual[1] = 17.000400

x[1] = 1.248448

...

Better Error Message

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

...

Origin

- 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 ($f_1(a,b) = f_2(a,b)$, solve for a)
  - ...

Alias Elimination

a = b

c = a + b

d = a - b

c = a + b (subst a=b) =>

c = b + b (simplify) =>

c = 2 * b

d = a - b (subst a=b) =>

d = b - b (simplify) =>

d = 0.0

- The alias relation a=b stored in variable a
- The equations are e.g. stored as (lhs,rhs,list<ops>)

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

Trace Example

0 = y + der(x * time * z); z = 1.0;

(1) subst:

y + der(x * (time * z))

=>

y + der(x * (time * 1.0))

(2) simplify:

y + der(x * (time * 1.0))

=>

y + der(x * time)

(3) expand derivative (symbolic diff):

y + der(x * time)

=>

y + (x + der(x) * time)

(4) solve:

0.0 = y + (x + der(x) * time)

=>

der(x) = ((-y) - x) / time

Trace of Dummy Derivatives Alg.

differentiation:

d/dtime L ^ 2.0

=>

0.0

differentiation:

d/dtime x ^ 2.0 + y ^ 2.0

=>

2.0 * (der(x) * x + der(y) * y)

subst:

2.0 * (der(x) * x + der(y) * y)

=>

2.0 * ($DER.x * x + $DER.y * y)

=>

2.0 * (u * x + $DER.y * y)

=>

2.0 * (u * x + v * y)

=>

2.0 * (u * xloc[1] + v * xloc[0])

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