On Efficient Program Synthesis from Statecharts

Andrzej Wąsowski

Joint work with Peter Sestoft

IT University of Copenhagen
Glentevej 67, 2400 Copenhagen NV, Denmark

wasowski@itu.dk
http://www.mini.pw.edu.pl/~wasowski/scope

12 June 2003
Content

- Brief overview of statecharts
- Hierarchical synthesis vs Flattening synthesis
- Implementation of SCOPE
- Theoretical and experimental evaluation
- Conclusion and future work
The Language of Statecharts

- **State hierarchy:**
  - parallel/sequential decompositions
  - The *root* is an and-state
  - Basic states (leaves) are and-states
  - Initial, history and deep history

- **Entry/exit actions.**

- **Transitions:** event + guard + action + targets
- **Dynamic semantics relations:** macrostep, microstep, fire, exit, enter, execute action, evaluate guard, init.
Runtime Overview

• Runtime engine

  macrostep : event * state -> state
  microstep : event * state * queue -> state * queue
  fire : tran * state * queue -> state * queue
  exit : orstate * state * queue -> state * queue
  enter : targets * orstate * state * queue -> state * queue

• Runtime data structures

  ![](diagram.png)
Flattening = Hierarchy Elimination

Refine guards conjuncting condition that A is active
Conjunct condition that D is active

Code generated: flat set of rules and state vector, very simple runtime

[visualSTATE] [Björklund, Lilius, Porres, Turku, Finland, 2001]

IT University of Copenhagen
SCOPE: Hierarchical Code Generator
• Regular type alternation
• Separate namespace for and-states and or-states
• Shorter state identifiers at runtime
• No runtime type-checks for states
• Simpler runtime library
Hierarchy tree (II)

• The tree can be splitted in two arrays (♯ is end-of-state):

```
[228x510]Hierarchy tree (II)

• The tree can be splitted in two arrays (# is end-of-state):

    A
   /\  \
  B  C
 /\ /\  \
D E F  G J
 /\ /\  \
G H I
```

or–states

and–states

• State names are naturals
• Cheaper of following is chosen automatically:
  – Offsets in the state array
  – Consecutive numbers with dictionary of offsets
Active State Set

- Only active *basic* state set
- Implemented as prioritized buffer (no gaps)
- Safety demands a bound
- Trivial bound: number of basic states
- Simple improvement:

  \[
  \begin{align*}
  \text{bound (Basic } s\text{)} &= 1 \\
  \text{bound (ORState } s\text{)} &= \text{maximum bound (children } s\text{)} \\
  \text{bound (ANDState } s\text{)} &= \text{sum bound (children } s\text{)}
  \end{align*}
  \]

- This is exact for strictly sequential or strictly flat models
Interval Labeling

- Trivial method needs ancestorship checks – use state labeling
- Label and-states in depth-first-search post-order, left to right visiting of children

Ancestorship is reduced to two comparisons: is given state in interval of descendants of \( s \)?
- Only *leftmost descendant* (LMD) needs to be saved in the array.
- *Exit-purity*
Dual labeling

Let's label or-states in order precisely dual to the one used for and-states
Removal of # marks

- Children lists are composed of monotonic sequences (increasing for or-states and decreasing for and-states).
- This can be used to remove end-of-state markers (#)
Scope of Transition

- Targets statically annotated with scopes:

\[
\begin{align*}
  t_1 & : [e \{D, F\} \{\}]/a : [B]E [C]G \\
  t_2 & : [f \{F\} \{\}]/- : [C]G \\
  t_3 & : [f \{G\} \{E\}]/- : [C]F
\end{align*}
\]
Dynamic Scope

- Three legal configurations activating the transition.
- All contain $D$.
- Also contain one of $F$, $H$ or $I$

- Scope of target $E$ is always $B$
- Scope of target $H$ depends on active configuration of $C$ ....
Dynamic Scope (II)

Runtime detection is slow and affects all transitions.
The problematic transition in our example:

\[
\begin{align*}
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{D} \\
\text{E} \\
\text{F} \\
\text{G} \\
\text{G'} \\
\text{H} \\
\text{I} \\
\end{array}
& \xrightarrow{a : H}
\begin{array}{c}
\text{C} \\
\text{G} \\
\text{H} \\
\text{I} \\
\end{array}
\end{align*}
\]

\[\left[ e \{ \} \{ \} \right] / a : H\]

\[\begin{align*}
\left[ e \{ D, F \} \{ \} \right] / a &: [B]E [C]H \\
\left[ e \{ D, G \} \{ \} \right] / a &: [B]E [G']H
\end{align*}\]

- Adding extra positive conditions can ensure static scopes.
- Scope performs this rewriting analyzing number of possible solutions to scope-equation (BDD based implementation)
Complexity Evaluation

\( n \) – number of states, \( t \) – number of transitions
\( d \) – model depths, \( m \) – maximum over number of targets

- Linear size vs exponential size with flattening
- Scope resolution cost up to \( O(d^m) \) space but constant in practice.
- Size of current state (constant factor difference):
  - flattening: linear in number of state machines
  - SCOPE: linear in number of leaves
- Elimination of end-of-state markers – constant saving
- Ancestorship test – constant time
- Activity test: \( O(n) \) vs constant time of flattening
  - but exponentially less tests
- State update:
  - flattening: \( O(d) \)
  - hierarchical: at least \( O(nd) \), improved with exit-purity
# Experimental Evaluation

Pentium II 450 Mhz, GCC ver3.2, optimizing for size, bare executable sizes in bytes

<table>
<thead>
<tr>
<th>Model</th>
<th>states</th>
<th>transitions</th>
<th>VS size</th>
<th>SCP size</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>actions01</td>
<td>4</td>
<td>1</td>
<td>3 596</td>
<td>3 840</td>
<td>1.07</td>
</tr>
<tr>
<td>drusinsky89</td>
<td>19</td>
<td>14</td>
<td>3 976</td>
<td>4 288</td>
<td>1.08</td>
</tr>
<tr>
<td>lift</td>
<td>18</td>
<td>19</td>
<td>4 452</td>
<td>4 496</td>
<td>1.01</td>
</tr>
<tr>
<td>peer</td>
<td>275</td>
<td>192</td>
<td>12 644</td>
<td>10 352</td>
<td>0.82</td>
</tr>
<tr>
<td>trios01</td>
<td>1121</td>
<td>840</td>
<td>28 164</td>
<td>19 944</td>
<td>0.71</td>
</tr>
<tr>
<td>trios03</td>
<td>1121</td>
<td>840</td>
<td>60 196</td>
<td>23 008</td>
<td>0.38</td>
</tr>
</tbody>
</table>

Running time measured with $10^7$ random events, probability of reinitialization 0.01

<table>
<thead>
<tr>
<th>Model</th>
<th>states</th>
<th>transitions</th>
<th>VS time</th>
<th>SCP time</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>actions01</td>
<td>4</td>
<td>1</td>
<td>6.24</td>
<td>4.69</td>
<td>0.75</td>
</tr>
<tr>
<td>drusinsky89</td>
<td>19</td>
<td>14</td>
<td>8.11</td>
<td>6.28</td>
<td>0.77</td>
</tr>
<tr>
<td>lift</td>
<td>18</td>
<td>19</td>
<td>14.08</td>
<td>29.78</td>
<td>2.11</td>
</tr>
<tr>
<td>peer</td>
<td>275</td>
<td>192</td>
<td>21.40</td>
<td>29.42</td>
<td>1.37</td>
</tr>
<tr>
<td>trios01</td>
<td>1121</td>
<td>840</td>
<td>534</td>
<td>712</td>
<td>1.34</td>
</tr>
<tr>
<td>trios03</td>
<td>1121</td>
<td>840</td>
<td>1137</td>
<td>763</td>
<td>0.67</td>
</tr>
</tbody>
</table>
Summary

● Conclusion
  – Experimental and complexity-theoretical evaluation of code generation methods have been presented.
  – Hierarchical code generation is feasible speed-wise
  – Hierarchical code generation can yield much smaller code
  – It performs decently for small models and well for bigger ones

● Future work
  – Optimize the structure of transitions (decision diagrams)
  – Make algorithm adaptive (meeting constraints)
  – Evaluate against self-implemented flattening
  – Evaluate against state-pattern code generation method
A Statechart Cooker

MicrowaveOven

Main

Idle

OFF

ON

H

ON

Idle

Stop() / StopEmiter()
Start() : H or HG / StartEmiter()

Closed

entry: LightOn()
exit: LightOff()
exit: StopEmiter()
exit: StopGrill()

Emiter

ON

Idle

Stop() / StopEmiter()

Grill

ON

Idle

Start() : G or HG / StopGrill()
Stop() / StopEmiter()
Transitions – Trivia

• Hashed into buckets by activating event
• Each bucket simply contains a list of transitions to be checked (and fired).
• This can still be optimized \[\text{TBD}\]

• Scope of transition
• Semantics of firing: exit scope – execute action – enter scope
• Assume scopes can be computed statically
• There should be a scope saved for each transition
• But not for flat transitions, for which it can be cheaply computed
• C compiler may not remove redundant code
• So do not generate it!

• Some elements are used more often than others
• Example: Initial markers may not be kept at all (just reorder states)
• Example: transition source, event and target should come "for-free", while other attributes may be more costly
Internal Structure of SCOPE

- Statechart front-end
- Model transformations
- Internal translator
- Static data manager
- Code generator
- C pretty printer

Flow of transformations:
- Concrete syntax → Abstract syntax → Intermediate representation → IR + addressing data + int types → C abstract syntax → C concrete syntax

IT University of Copenhagen
Experiment conditions

- Pentium II, 450 Mhz, running Linux
- GCC ver 3.2, optimizing for size (-Os)
- Bare executable sizes in bytes.
- Only control algorithm (structures + runtime engine).
- External functions substituted with dummies.
- Size results similar for LCC on PC (non optimizing) and optimizing embedded systems compilers.
- Running time measured for feeding $10^7$ random events with probability of model initialization before each event equal to 0.01.