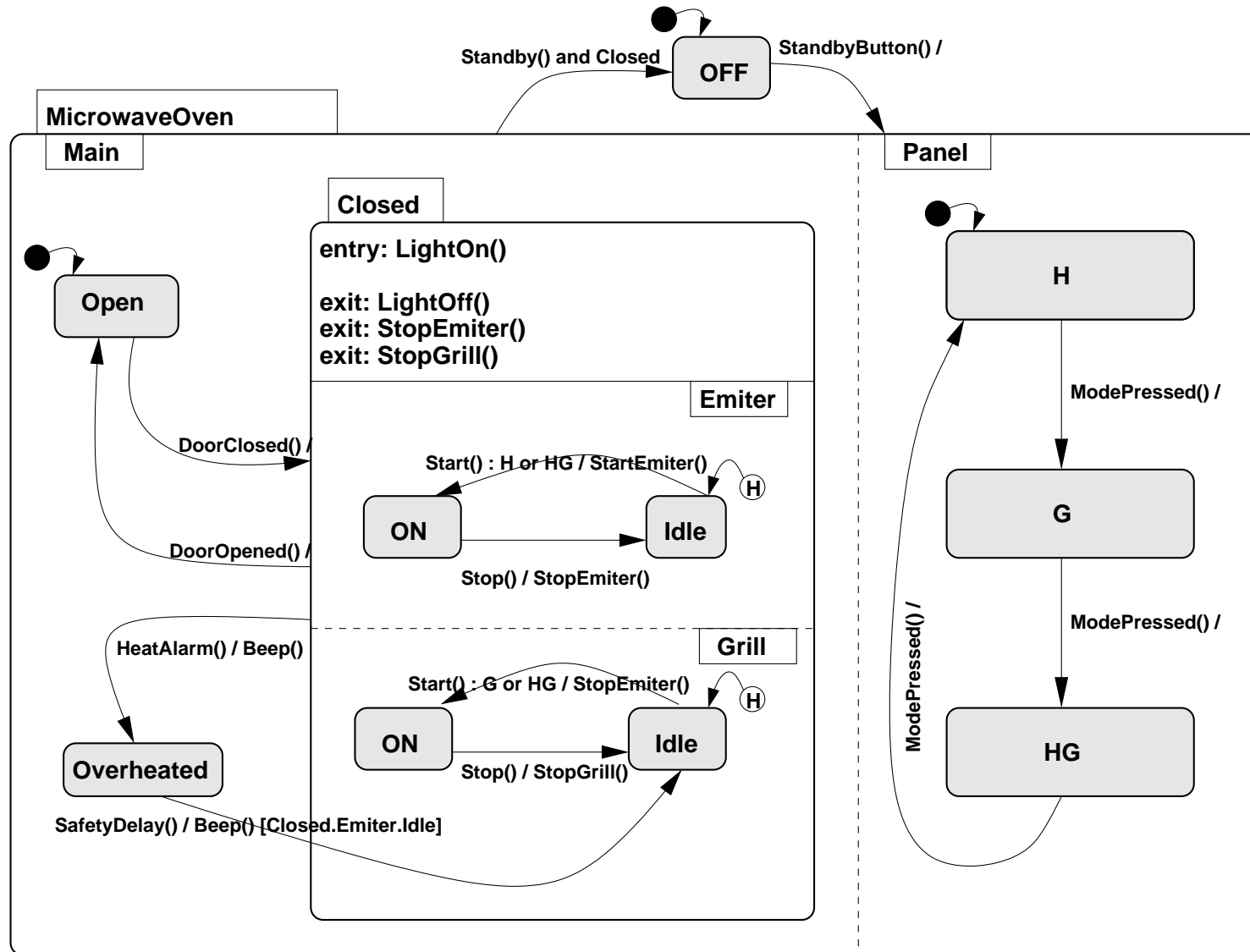


# Crossing the Gap between Semantics and Practice: Crafting a Compiler for Statecharts

Andrzej Wąsowski

22th November 2002

# Cooking a Statechart System



# Content

- Challenges in compiling statecharts
- Short survey of approaches
- Lessons learnt from implementation of SCOPE

# Statechart Compilation Challenges

Is it at all that challenging? An idiosyncratic list of hard problems:

- Meeting constraints: binary size, runtime memory consumption, time (often speed), power consumption
- Controlling trade offs = meeting constraints possibly cheaply (for example: balance speed and size)
- Support automatic tools: model-checkers, schedulability analysis, memory consumption analysis
- Separation of reaction speed from model size:
  - Natural for Java programs!
- From explicit states to implicit states: Algorithm abstraction and model minimization via transformations.

Not all are solvable, some not solved, some not yet solved satisfactorily.

# Methods Overview

Only methods compiling *for execution* (not for model-checking, etc)

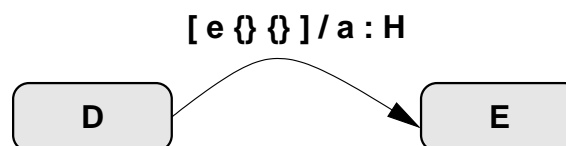
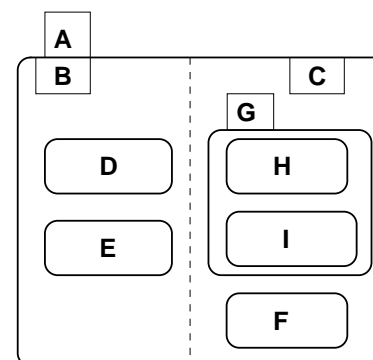
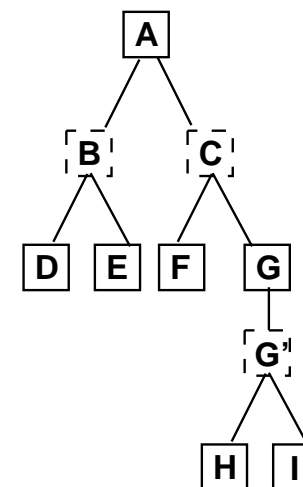
- Simple interpretive
  - Transformation based
  - Flattening
  - Object-oriented
- Reducing to SAT problem (BDD based)
- Synchronous

Disclaimer: tool and authors references are mostly examples of use, not contributions.

More in upcoming survey report on compiling statecharts (time analysis oriented, verification oriented, hardware oriented, other target formalisms, etc).

# Language of Statecharts

- State hierarchy:
  - parallel and sequential decompositions
  - The *root* is an and-state
  - Basic states (leaves) are and-states
  - State type alternation
- Entry/exit actions.
- Transitions:
  - condition side: event + guard
  - executable side: action + targets



# Language of Statecharts (II)

Abundance of additional constructs:

- initial, history and deep history states
- internal reactions
- join and fork transitions
- do reactions, final states and termination transitions
- event-less transitions
- special events, f.eg.  $en(s)$ ,  $ex(s)$ , timer events
- special actions (timer actions)
- call events and signal events

# Dynamic Semantics

- Semantics of statecharts is usually defined in operational way
- Decomposed into several relations:
  - macrostep
  - microstep
  - fire
  - exit
  - enter
  - execute action/evaluate guard
  - init
- Details differ among authors (no standard semantics)



# Simple Interpretive Method

- Remain close to semantics.
- Code Generator: runtime representations of
  - hierarchy,
  - transitions,
  - queues,
  - state vectors, etc.
- Runtime library: counterparts of behavioral relations:

```
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
```

# Simple Interpretive Method (II)

- Advantages:
  - Easier to understand and communicate
  - Raises confidence in correctness
- Disadvantage: complicated runtime logics
- In generic form does not addresses any of the challanges

[Behrmann/Kirstoffersen/Larsen NWPT'99]

[STARC, Erpenbach, PhD thesis, Paderborn, 2000]

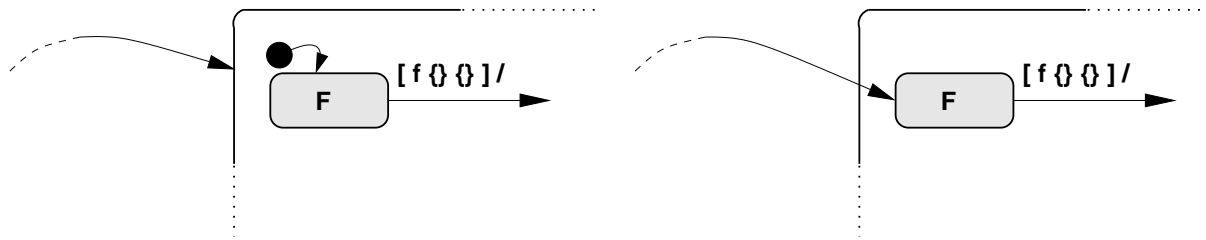
[SCOPE, Wasowski, IT Copenhagen, 2002]

# Reductions in Execution Model

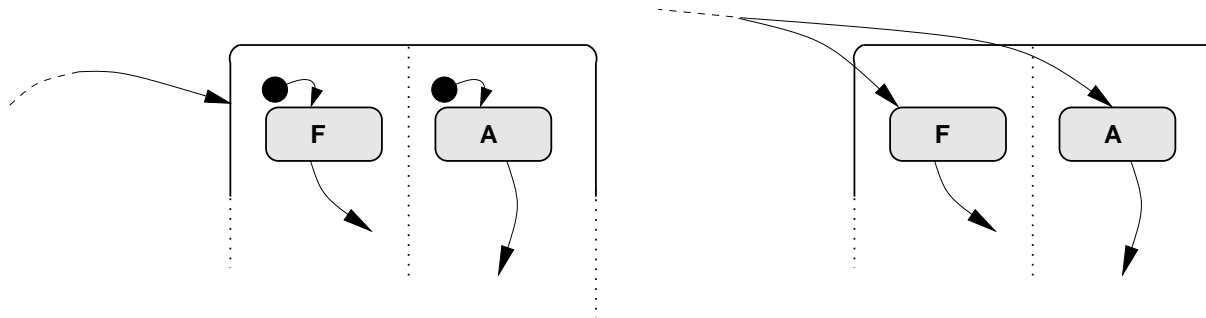
- Express some model elements in terms of simpler constructs
- Expand runtime representation (Code Generator)
- Simplify execution model (runtime interpreter)
- Example of reductions:
  - Initial states elimination
  - Entry/exit actions elimination
  - Hierarchy elimination
  - Message passing elimination
- Reduction calls for proof of correctness of respective transformation

# Initial States Elimination

- Trivial in absence of history and concurrency



- Explodes in presence of heavy concurrency

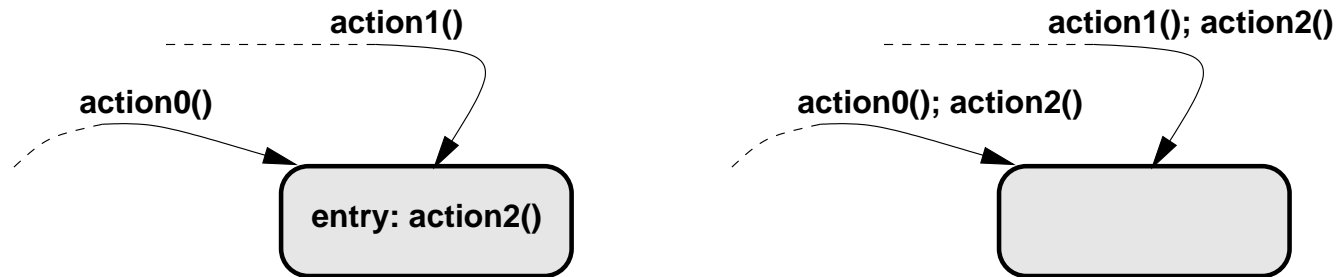


- In presence of history explodes even more.

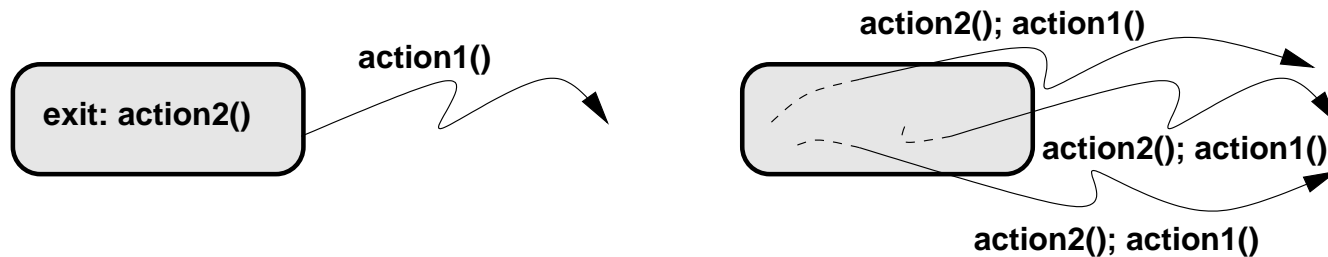
[iState, Sekerinski Zurob, McMaster University, Ontario, 2001]

# Elimination of Entry/Exit Actions

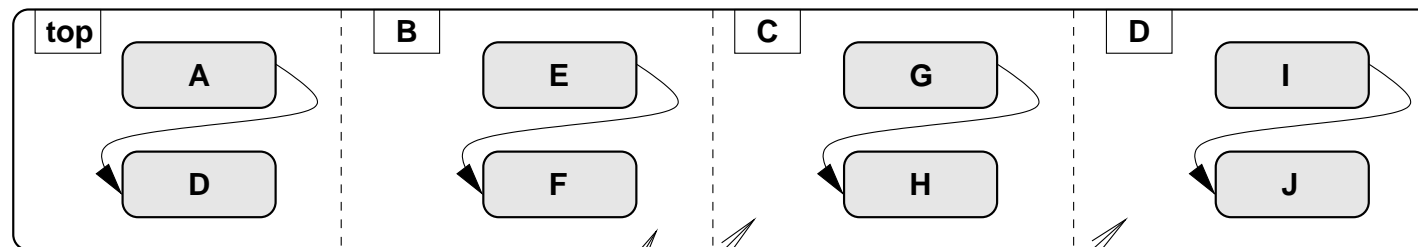
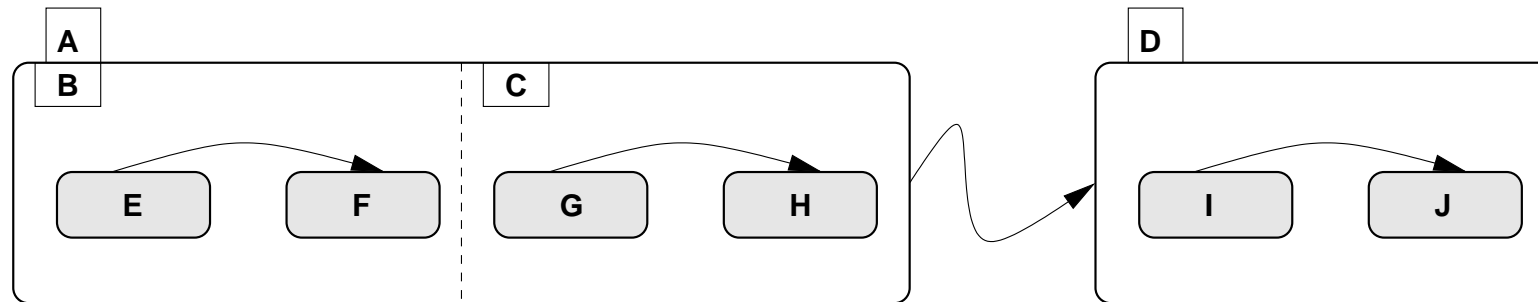
- Add entry actions of all states on the way (not only the last one!)
- Expensive for dense models (many entry actions + many transitions targetting the same state)



- Much more expensive for exit actions.



# Flattening = Hierarchy Elimination



*Refine guards conjuncting condition that A is active*

*Conjunct condition that D is active*

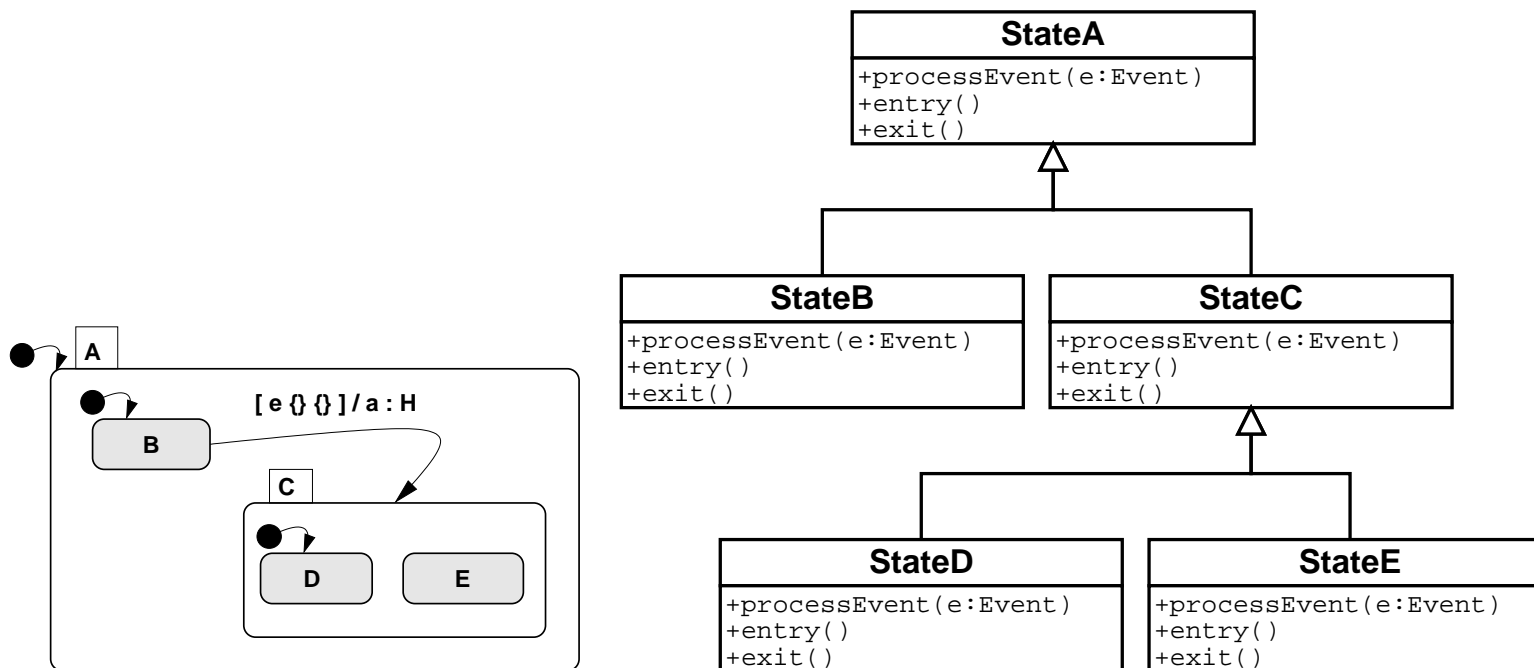
[Patent by Leerberg, Hulgaard, Lind-Nielsen, Andersen, Larsen, Kristoffersen, Behrmann, 1999]

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

# Object-Oriented Approach: New?

[Rhapsody (I-Logix), Rose RT (Rational Rose), Fujaba (Zündorf)...]

- Mostly variations of simple interpretive method
  - encode semantics using objects, methods and switch statements
- State pattern



- Reflect complete metamodel to obtain all features

[Hugo (Knapp, Merz, Munich, 2002)]

# Reducing to SAT (BDD based)

- Encode state diagram semantics as SAT problem.
- Transition:

$$t_i : [ e_i \ pos_i \ neg_i ] / a_i : s_1^i, \dots, s_k^i ]$$

a conjunction of boolean formulas representing elements.

- actions being an integer identifier (handle guards similarly)
- A separate set of variables for target states (in addition to one used in guard)
- System encoded as disjunction of all transitions:

$$\phi = \bigvee_{t_i \in Trans} (\phi_{e_i} \wedge \phi_{pos_i} \wedge \phi_{neg_i} \wedge \phi_{a_i} \wedge \phi_{s_1^i} \dots \wedge \phi_{s_k^i})$$

- Compile-time: build the BDD representing  $\phi$
- Runtime: traverse satisfying paths of BDD to find out the next state

BDD engine becomes an interpreter!



# Reducing to SAT (BDD based) (II)

- Peter Jacobsen (masters thesis, Technical Univeristy of Denmark, Lyngby 1999)
- Implemented for sublanguage
  - no internal signals, no variables and no hierarchy
- Advantage: cute theoretical formulation
- Drawback: Rather mean results (hardly beats visualSTATE)
- Useless if heavy synchronization via message passing employed.

# The Argos Way

[Florence Maraninchi, VERIMAG, 1991]

- Argos: a variant of statecharts
- No queues, no level-crossing transitions, no fancy elements
- Fully synchronous semantics (after model of Esterel)
- Semantics by expansion to single Mealy machine (flat&sequential).
- Implicitly encoded in system of boolean flow equations
- Flows are sequences of boolean values

# The Argos Way (II)

- $i := \text{EQU}(b, g, v)$  defines  $i$  to be

$$i_0 = \begin{cases} b_0 & \text{if } g_0 \\ v & \text{otherwise} \end{cases} \quad i_n = \begin{cases} b_n & \text{if } g_n \\ i_{n-1} & \text{otherwise} \end{cases}$$

- $i := \text{MEM}(b, g, v)$  defines  $i$  to be

$$i_0 = v \quad i_n = \begin{cases} b_{n-1} & \text{if } g_{n-1} \\ i_{n-1} & \text{otherwise} \end{cases}$$

- A boolean flow for state  $s$  is  $s := \text{MEM}(rp, \text{true}, \text{initial})$ , where

$$rp = \left( s \wedge \bigwedge_{(s,g,-,-) \in \text{Trans}} (\neg g) \right) \vee \left( \bigvee_{(s',g,-,s) \in \text{Trans}} (s' \wedge g) \right)$$

- A boolean flow for output  $s$  is  $o := \text{EQU}(rp, \text{true}, v)$ , where

$$rp = \bigwedge_{(s,g,o \in O, -) \in \text{Trans}} (s \wedge g)$$

# The Argos Way (III)

- A bit more complicated for full-blown Argos
  - Compositional (syntax directed)
  - Disjunct output string from concurrent components
  - Ensure exclusiveness of sequential components.
- Compiled to declarative format of flow equations (DC)
- The size of DC program is linear in size of Argos program (cool!)
- Original compiler had an option for controlling size-speed trade off.

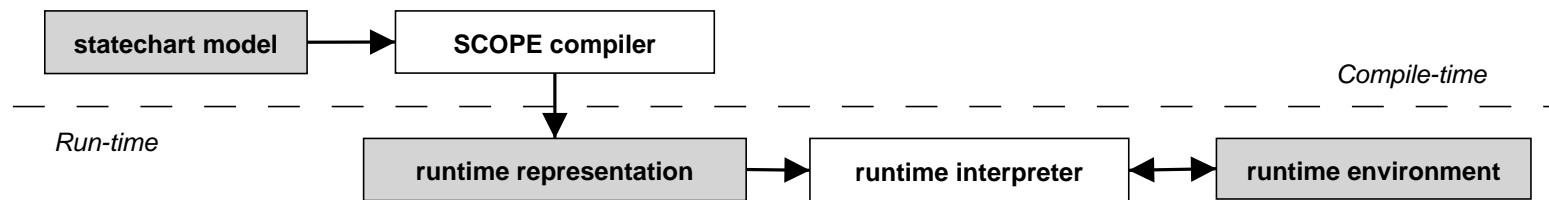
# Equations for Statecharts

[Beauvais, Gautier, Le Geurnic, Houdebine, Rutten, IRISA, 1998]

- Statecharts semantics is not fully synchronous
- Each step consists of microstep
- Discrete sequences with one clock are not enough
- Need flows occurring with various frequencies wrt each other
  - So you need Signal (rather DC+) with full clock-calculus
- Flows at macrostep level, microstep level, etc
- Easy to obtain synchronous semantics for statecharts!
- Translation from DC(+) to C
  - Automaton generation explodes exponentially  
Automaton  $\rightarrow$  equations  $\rightarrow$  automaton ?
  - Equation preserving method: 10  $\times$  slower, hundreds  $\times$  smaller  
[Amagbagnon, Besnard, Le Guernic, IRISA-INRIA, 1995]:  
canonize equations and organize in hierarchy  
hierarchy  $\rightarrow$  clock equations  $\rightarrow$  hierarchy ?

# StateChart cOmPiEr = SCOPE

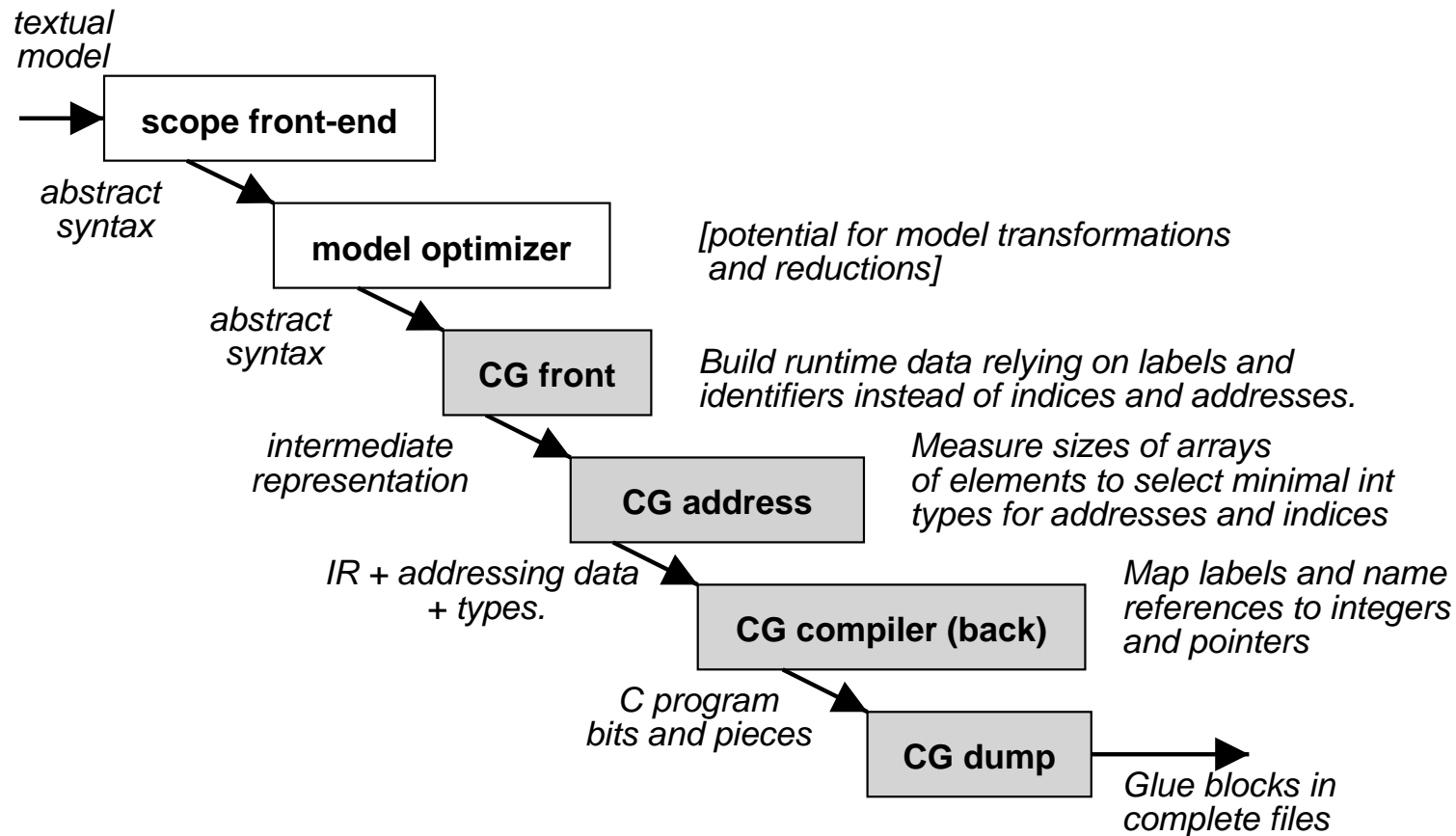
- Extremely high-level code generator, very close to operational semantics
- Compiles visualSTATE version of statecharts.



<http://www.mini.pw.edu.pl/~wasowski/scope>

- Code linear in size of the model.
- Hardly ever bigger than visualSTATE.
- For moderate and bigger models: 10%-50% smaller code.
- The speed is comparable with flattening (1-2 times slower)
- Open for further optimizations.

# SCOPE HOW-TO



# Summary

- Method Survey
  - Problems in domain of code synthesis from statecharts
  - Several fundamental methods
    - \* Still worse than hand-coding!
    - \* Difficulties in evaluation
  - Yet another semantics for statecharts?
- Experience with simple high level approach (SCOPE)
  - Works quite well
  - Platform for experimentation, optimization and comparison