Closing open SDL-systems for model checking with DTSpin

Natalia Ioustinova
Natalia Sidorova
Martin Steffen

Dept. of Software Eng.
Centrum Wiskunde en Informatica
The Netherlands

Dept. of Math. and Comp. Science
Technische Universiteit Eindhoven
The Netherlands

Inst. für Informatik u. Prakt. Math.
Christian-Albrechts Univ. of Kiel
Germany

Model checking

- pro: automatic (“push-button”) verification method
- con: state-space explosion

Model checking

- pro: automatic ("push-button") verification method
- con: state-space explosion

Abstraction and decomposition techniques

Model checking

- pro: automatic ("push-button") verification method
- con: state-space explosion

Abstraction and decomposition techniques

- data abstraction:
  replace concrete domains by finite, abstract ones

Model checking

- pro: automatic (“push-button”) verification method
- con: state-space explosion

Abstraction and decomposition techniques

- data abstraction:
  replace concrete domains by finite, abstract ones
- control abstraction, i.e., add non-determinism

Model checking

- pro: automatic ("push-button") verification method
- con: state-space explosion

Abstraction and decomposition techniques

- data abstraction:
  replace concrete domains by finite, abstract ones
- control abstraction, i.e., add non-determinism
- assume-guarantee paradigm

Model checking in theory

Model checking in theory

- cut out a sub-component

Model checking in theory

- cut out a sub-component
- model its environment abstractly, i.e.,

Model checking in theory

- cut out a sub-component
- model its environment abstractly, i.e., add an environment *process* which
  - closes the sub-component
  - shows more behavior than the real environment ⇒ *in extremis:* add chaos-process

Model checking in theory

- cut out a sub-component
- model its environment abstractly, i.e., add an environment *process* which
  - closes the sub-component
  - shows more behavior than the real environment ⇒ *in extremis:* add chaos-process
- push the button. . .

Model checking in practice

- components and interfaces might be large

Model checking in practice

- components and interfaces might be large
- closing is tedious

Model checking in practice

- components and interfaces might be large
- closing is tedious
- model checkers often don’t work with abstract data

Model checking in practice

- components and interfaces might be large
- closing is tedious
- model checkers often don't work with abstract data
- with asynchronous communication adding external chaotic process leads to state space explosion

Model checking in practice

- components and interfaces might be large
- closing is tedious
- model checkers often don't work with abstract data
- with asynchronous communication adding external chaotic process leads to state space explosion

Embedding Chaos [Sidorova and Steffen, 2001]

Goal

- a tool implementing embedding closing ideas

Goal

- a tool implementing embedding closing ideas
- experiments to corroborate the usfulness of the approach

Goal

- a tool implementing embedding closing ideas
- experiments to corroborate the usfulness of the approach
- the tool is targeted towards the verification of SDL components with DTSpin

SDL

(Specification and Description Language)

- standardized (in various versions)
- standard spec. language for telecom applications
- characteristics:
  - control structure: communicating finite-state machines
  - communication: asynchronous message passing
  - data: various basic and composed types
  - timers and timeouts
  - bells and whistles: graphical notation, structuring mechanisms, OO, . . .

Model checking SDL systems

- three more specific problems
  1. asynchronous input queues: ⇒ state explosion
  2. infinite data domains
  3. chaotic timer behavior

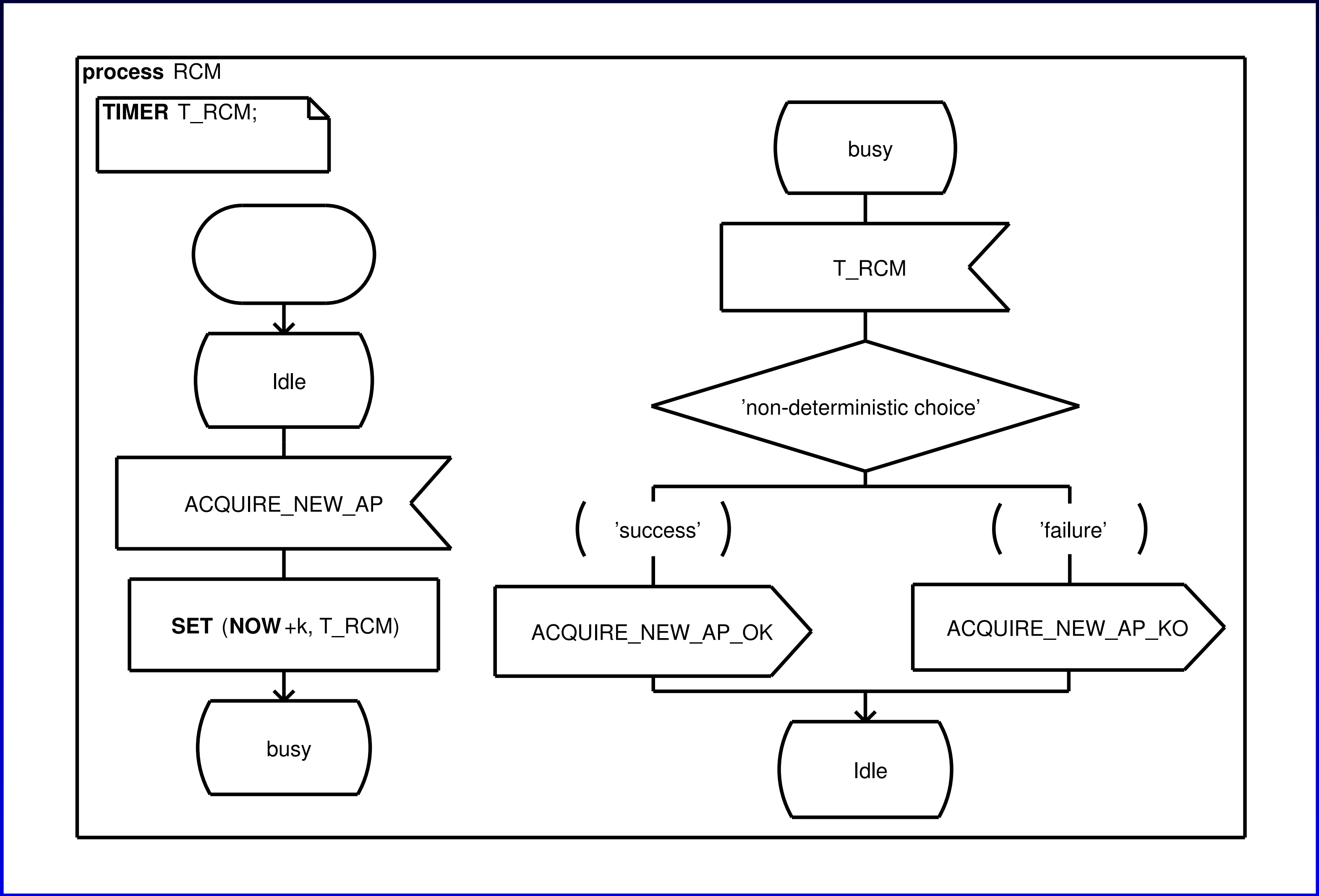
Model checking SDL systems

- three more specific problems
  1. asynchronous input queues: $\Rightarrow$ state explosion
  2. infinite data domains
  3. chaotic timer behavior
- three specific solutions
  1. embedding environment into the system
  2. one-valued data abstraction $\hat{=}$ no external data
  3. three-valued timer abstraction

Roadmap

1. (sketch of) syntax
2. SO-semantic rules
3. eliminating external data via data-flow analysis
4. dealing with chaotic timers
5. eliminating communication with environment
6. tools overview
7. experimental results

Syntax: Example



Syntax

- labelled edges $l \longrightarrow_\alpha \hat{l}$ connecting locations
- actions α:

input	$?s(x)$
output	$g \triangleright P!s(e)$
assignment	$g \triangleright x := e$

with guards g, signals s, processes P

Semantics (local)

- straightforward operational small-step semantics
  - interleaving semantics
  - top-level concurrency
- local process configuration:
  1. location/control state
  2. valuation of variables
  3. an input queue

⇒ labelled steps between configurations, e.g.

$$\frac{l \longrightarrow_{?s(x)} \hat{l} \in Edg}{(l, \eta, s(v)::q) \longrightarrow_{?s(x)} (\hat{l}, \eta_{[x \mapsto v]}, q)} \text{ INPUT}$$

Modelling SDL Timers

- discrete-time semantics

⇒ time evolves by ticking down (active) timer variables

- timer: active or deactivated
- timeout possible: if active timer has reached 0
- modelled by timeout guards

Syntax for timers

- guarded actions involving timers

set	$g \triangleright set\ t := e$	(re-)activate timer t for a period given by e
reset	$g \triangleright reset\ t$	deactivate t
timeout	$g_t \triangleright reset\ t$	perform a timeout, thereby deactivate t

- note: timeout is guarded by "timer-guard" g_t: $t = 0$

Parallel composition

- standard product construction
- message passing using the labelled steps
- note: tick step = counting down active timers:
  - is taken only when no other move is possible

$$\sigma \longrightarrow_{tick} \sigma[t \mapsto (t-1)] \quad \text{iff} \quad blocked(\sigma)$$

What's next

Goal:

- abstract data from outside:
  chaotic data value $\mathbb{T}$
- no external communication

What's next

Goal:

- abstract data from outside:
  chaotic data value $\mathbb{T}$
- no external communication

Side-condition

- verification with *DTSpin* model checker:
  - there are no abstract data
  - we cannot re-implement tick
- keep it simple

The need for data-flow analysis

The need for data-flow analysis

- abstractly: replace external $?s(x)$ by receiving $?s(\mathbb{T})$

The need for data-flow analysis

- abstractly: replace external $?s(x)$ by receiving $?s(\top)$
- better: remove communication parameters

The need for data-flow analysis

- abstractly: replace external $?s(x)$ by receiving $?s(\top)$
- better: remove communication parameters

$\Rightarrow$ remove all variable instances (potentially) influenced by x as well (and transitively so)

The need for data-flow analysis

- abstractly: replace external $?s(x)$ by receiving $?s(\mathbb{T})$
- better: remove communication parameters

$\Rightarrow$ remove all variable instances (potentially) influenced by x as well (and transitively so)

$\hat{=}$ forward slice/cone of influence

The need for data-flow analysis

- abstractly: replace external $?s(x)$ by receiving $?s(\top)$
- better: remove communication parameters

$\Rightarrow$ remove all variable instances (potentially) influenced by x as well (and transitively so)

$\hat{=}$ forward slice/cone of influence

eliminating external data

1. data-flow analysis: mark all variable instances potentially influenced by chaos
2. transform the program, using that marking

Data-flow analysis

- control-flow given by SDL-automata
- propagate $\mathbb{T}$ through control-flow graph, via abstract effect per action = node, e.g.:

$$f(?s(x)\eta^\alpha = \begin{cases} \eta^\alpha[x \mapsto \top] & s \in Sig_{ex} \\ \eta^\alpha[x \mapsto \bigvee\{[e]_{\eta^\alpha} \mid \alpha_{n'} = g \triangleright P!s(e)\}] & \text{else} \end{cases}$$

- constraint solving: minimal solution for

$$\eta^\alpha_{post}(n) \geq f_n(\eta^\alpha_{pre}(n))$$

$$\eta^\alpha_{pre}(n) \geq \bigvee\{\eta^\alpha_{post}(n') \mid (n', n) \text{ in flow relation}\}$$

Worklist algo (pseudo-code)

input : the flow−graph of the program

output : $\eta^{\alpha}_{pre}, \eta^{\alpha}_{post}$;

$\eta^{\alpha}(n) = \eta^{\alpha}_{init}(n)$;

$WL = \{n \mid \alpha_n =?s(x), s \in Sig_{ext}\}$;

repeat

  pick $n \in WL$;

  let $S = \{n' \in succ(n) \mid f_n(\eta^{\alpha}(n) \not\leq \eta^{\alpha}(n')\}$

  in

    for all $n' \in S$: $\eta^{\alpha}(n') := f(\eta^{\alpha}(n))$;

  if $n = g \rhd P!s(e)$

  then let $S' = \{n' \in P \mid n' =?s(x), \eta^{\alpha}(n)(e) \not\leq \eta^{\alpha}(n')(x)\}$

  $WL := WL \backslash n \cup S \cup S'$;

until $WL = \emptyset$;

$\eta^{\alpha}_{pre}(n) = \eta^{\alpha}(n)$;

$\eta^{\alpha}_{post}(n) = f_n(\eta^{\alpha}(n))$

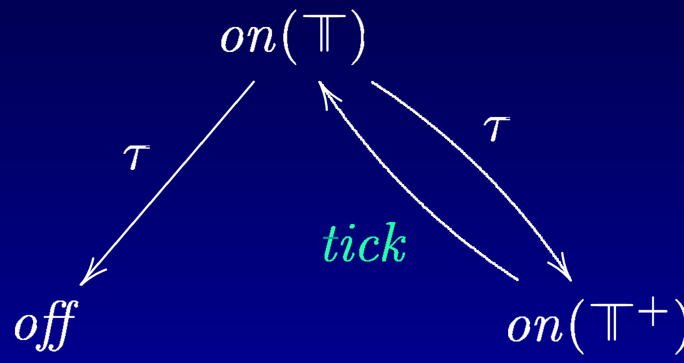
What about time?

- so far: we ignored timers
- timers can be influenced by external data
- chaotic timeout for an active timer:
  1. it can happen now, or
  2. eventually in the future
- remember: time steps (ticks) have least priority!

Timer abstraction

Three abstract values:

1. *off* : deactivated
2. $on(\mathbb{T})$: arbitrarily active
3. $on(\mathbb{T}^+)$: active, but not 0 (no time-out possible)



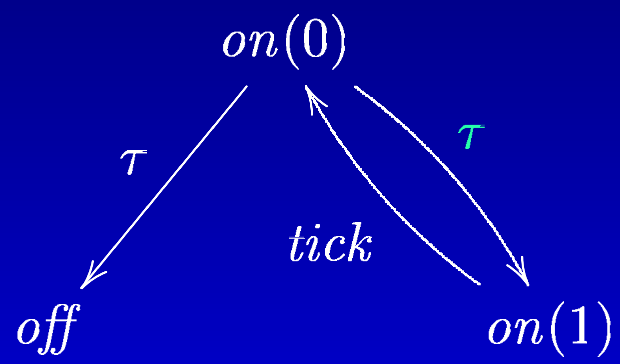
- arbitrary expiration time $\Rightarrow$ non-deterministic setting from $on(\mathbb{T})$ to $on(\mathbb{T}^+)$

Implementation: *off* , $on(0)$, $on(1)$

Transformation rules

- using result of the flow analysis
- inference rule(s) for each syntax construct, e.g.,

$$\frac{[\![t]\!]_{\eta_l^\alpha} = \top}{l \longrightarrow_{g_t \,\triangleright\, reset\ t} \longrightarrow_{set\ t:=\mathbf{1}} l \in Edg^\sharp}\ \textsc{T-NoTimeout}$$



- transformation yields a safe abstraction

What we have now

What we have now

- A safe abstraction of a given system

What we have now

- A safe abstraction of a given system
- No data involved in the communication with the environment

What we have now

- A safe abstraction of a given system
- No data involved in the communication with the environment
- But: the system is still open

Embedding chaos

Environment sends signals arbitrarily

$\Rightarrow$ Inputs from the environment are always potentially enabled

$\Rightarrow$ Replace them by $?none$ inputs ???

Embedding chaos

Environment sends signals arbitrarily

$\Rightarrow$ Inputs from the environment are always potentially enabled

$\Rightarrow$ Replace them by $?none$ inputs ???
Time won't progress!

Embedding chaos

Environment sends signals arbitrarily

⇒ Inputs from the environment are always potentially enabled

⇒ Replace them by $?none$ inputs ???
Time won't progress!

⇒ Regulate inputs from the environment by means of a special timer added to each process

Embedding chaos

Environment sends signals arbitrarily

$\Rightarrow$ Inputs from the environment are always potentially enabled

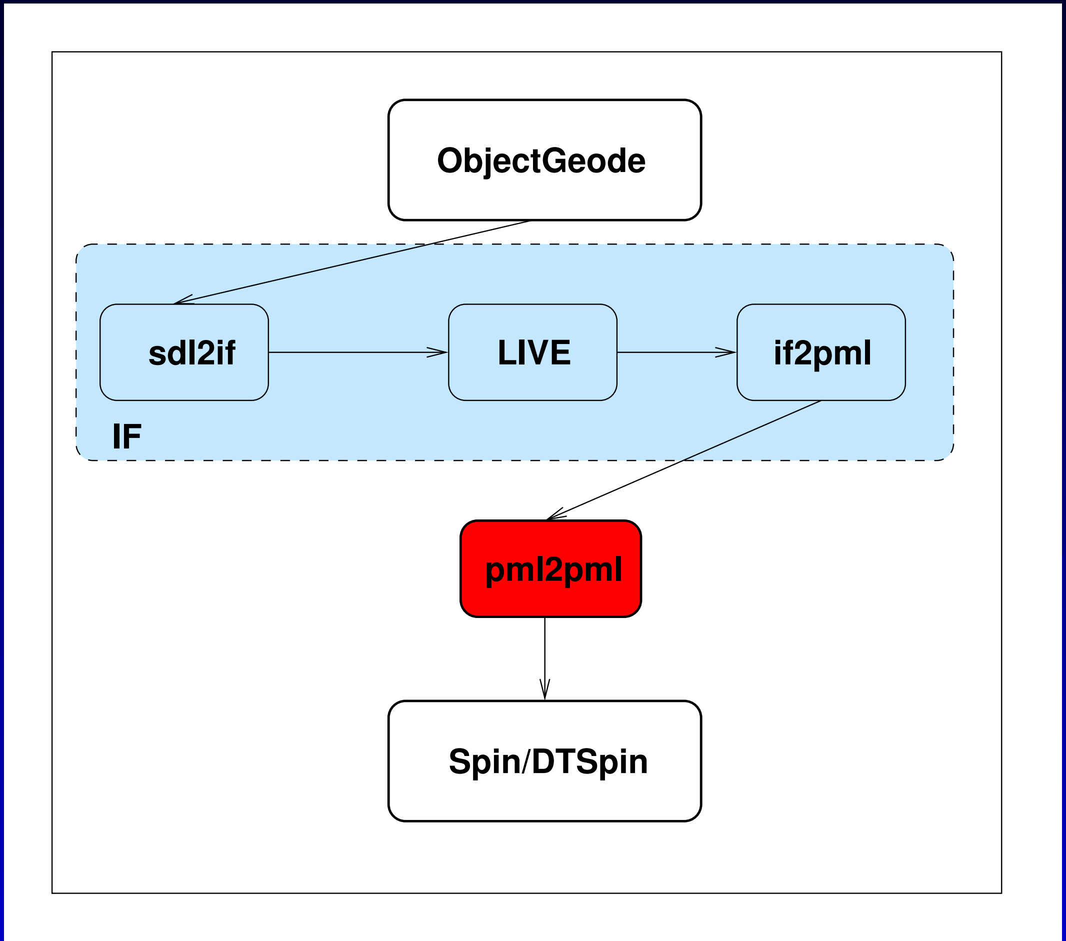
$\Rightarrow$ Replace them by $?none$ inputs ???
Time won't progress!

$\Rightarrow$ Regulate inputs from the environment by means of a special timer added to each process

$$\frac{l \longrightarrow_{?s(x)} \hat{l} \in Edg^{\top} \qquad s \in Sig_{ext}}{l \longrightarrow_{g_{t_P} \,\triangleright\, reset\ t_P} \longrightarrow_{set\ t_P:=0} \hat{l} \in Edg^{\sharp}} \text{ T-INPUT}$$

$$\frac{}{l \longrightarrow_{g_{t_P} \,\triangleright\, reset\ t_P} \longrightarrow_{set\ t_P:=1} l \in Edg^{\sharp}} \text{ T-NOINPUT}$$

Extending Vires toolset



Experimental results

Steady State Control of Mascara* closed with embedded chaos and model checked with *DTSpin*

	System with ext. chaos	System with embedded chaos
States	2.68938e+07	467555
Transitions	1.04753e+08	2.30307e+06
Memory	944.440	14.499
Time	1:59:39.76	2:12.91

*[Guoping and Graf],[Sidorova and Steffen, 2001b], [Bošnački et al., 2000]

Conclusion

- pml2pml automatically translates an open DTPromela model into

Conclusion

- pml2pml automatically translates an open DTPromela model into
  - a closed model

Conclusion

- pml2pml automatically translates an open DTPromela model into
  - a closed model
  - which is a safe abstraction of the original one

Conclusion

- pml2pml automatically translates an open DTPromela model into
  - a closed model
  - which is a safe abstraction of the original one
- experiments on Mascara confirm the usefulness of the embedding chaos approach

Related work

- software testing
- VERISOFT, C, untimed [Colby et al., 1998]
- filtering [Dwyer and Pasareanu, 1998] [Pasareanu, 2000]
- module checking:
  - checking open systems
  - e.g. MOCHA [Alur et al., 1998]

On-going work

- More refined abstractions

On-going work

- More refined abstractions
  - wrt. data abstraction

On-going work

- More refined abstractions
  - wrt. data abstraction
  - wrt. environment behaviour

On-going work

- More refined abstractions
  - wrt. data abstraction
  - wrt. environment behaviour
- extension of the approach to handle

On-going work

- More refined abstractions
  - wrt. data abstraction
  - wrt. environment behaviour
- extension of the approach to handle
  - procedures

On-going work

- More refined abstractions
  - wrt. data abstraction
  - wrt. environment behaviour
- extension of the approach to handle
  - procedures
  - dynamic process creation

On-going work

- More refined abstractions
  - wrt. data abstraction
  - wrt. environment behaviour
- extension of the approach to handle
  - procedures
  - dynamic process creation
- extension of pml2pml implementing synchronous closing [Sidorova and Steffen, 2002]

References

[Alur et al., 1998] Alur, R., Henzinger, T. A., Mang, F., Qadeer, S., Rajamani, S. K., and Tasiran, S. (1998). Mocha: Modularity in model checking. In Hu, A. J. and Vardi, M. Y., editors, *Proceedings of CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer-Verlag.

[Bošnački and Dams, 1998] Bošnački, D. and Dams, D. (1998). Integrating real time into Spin: A prototype implementation. In Budkowski, S., Cavalli, A., and Najm, E., editors, *Proceedings of Formal Description Techniques and Protocol Specification, Testing, and Verification (FORTE/PSTV'98)*. Kluwer Academic Publishers.

[Bošnački et al., 2000] Bošnački, D., Dams, D., Holenderski, L., and Sidorova, N. (2000). Verifying SDL in Spin. In Graf, S. and Schwartzbach, M., editors, *TACAS 2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag.

[Colby et al., 1998] Colby, C., Godefroid, P., and Jagadeesan, L. J. (1998). Automatically closing of open reactive systems. In *Proceedings of 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM Press.

[Guoping and Graf] J. Guoping and S. Graf. Verification experiments on the Mascara protocol. In M. B. Dwyer, editor, *Model Checking Software, Proceedings of the 8th International SPIN Workshop (SPIN 2001), Toronto, Canada*, Lecture Notes in Computer Science, pages 123–142. Springer-Verlag, 2001.

[DTSpin2000, 2000] DTSpin2000 (2000). Discrete-time Spin. `http://win.tue.nl/~dragan/DTSpin.html`.

[Dwyer and Pasareanu, 1998] Dwyer, M. B. and Pasareanu, C. S. (1998). Filter-based model checking of partial systems. In *Proceedings of the 6th ACM SIGSOFT Symposium on the Foundations of Software Engineering (SIGSOFT '98)*, pages 189–202.

[ObjectGeode 4.] ObjectGeode 4. http://www.csverilog.com/produ
2000.

[Pasareanu, 2000] Pasareanu, C. S. (2000). DEAO kernel: Environment modeling using LTL assumptions. Technical Report SASA-ARC-IC-2000-196, NASA Ames.

[SDL92, 1992] SDL92 (1992). Specification and Description Language SDL, blue book. CCITT Recommendation Z.100.

[Sidorova and Steffen, 2001a] Sidorova, N. and Steffen, M. (2001a). Embedding chaos. In Cousot, P., editor, *Proceedings of the 8th Static Analysis Symposium (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science*, pages 319–334. Springer-Verlag.

[Sidorova and Steffen, 2001b] Sidorova, N. and Steffen, M. (2001b). Verifying large SDL-specifications using model checking. In Reed, R. and Reed, J., editors, *Proceedings of the 10th International SDL Forum SDL 2001: Meeting UML*, volume 2078 of *Lecture Notes in Computer Science*, pages 403–416. Springer-Verlag.

[Sidorova and Steffen, 2001] N. Sidorova and M. Steffen. Embedding chaos. In P. Cousot, editor, *Proceedings of the 8th Static Analysis Symposium (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science*, pages 319–334. Springer-Verlag, 2001.

[Sidorova and Steffen, 2002] Sidorova, N. and Steffen. M. (2002). Synchronous Closing of Timed SDL Systems for Model Checking. In Cortesi, editor, *Proceedings of the Third International Workshop on Verification, Model Checking and Abstract Interpretation*, to appear in *Lecture Notes in Computer Science*. Springer-Verlag.

[TAU SDL] Telelogic TAU SDL Suite. http://www.telelogic.com/products/sdl/, 2002.

[VIRES, 2000] VIRES (1998-2000). Verifying industial reactive systems (VIRES), Esprit long-term research project LTR-23498. http://radon.ics.ele.tue.nl/~vires/.