Embedding Chaos

NATALIA SIDOROVA MARTIN STEFFEN

Dept. of Mathematics and Computer Science
Eindhoven University of Technology

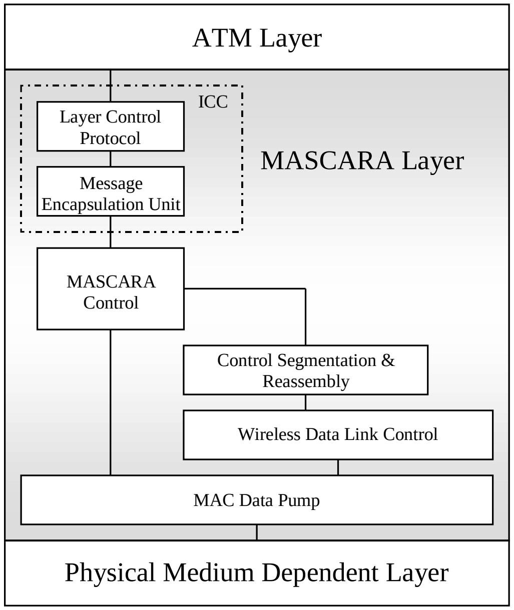
Inst. für Informatik u. Prakt. Mathematik
Christian-Albrechts Universtität zu Kiel

June 2001

AG Informatik, Logik und Mathematik

Motivation & starting point

- Verification/model checking of Mascara
  - wireless industial ATM medium-access protocol for LANs
  - developed within Wand industrial board
  - given in SDL



Model checking

+ “automatic” (“push-button”) program verification method

$$p \models \varphi$$

– state-space explosion

– How to obtain the model from a piece of software?

SDL

- "Specification Description Language" [SDL92, 1992]
- standardized (in various versions)
- standard spec. language for telecom applications
- language characteristics:
  * control structure: communicating finite-state machines
  * communication: asynchonous message passing
  * data: various basic and composed types
  * timers and time-outs[1]
  * bells and whistles: graphical notation, structuring mechanisms, OO, ...

[1] no real-time, not uncontroversial

Model checking SDL

- Various aggravations

1. it's about software (data)
2. it's about large[2] software
3. it's about open systems

- approaches:

1. abstraction:
   (a) data abstraction:replace concrete domains by finite, abstract ones
   (b) control abstraction, i.e., add non-determinism
2. decompose system along SDL-blocks

[2]well, depends

Model checking SDL in theory (and in practice)

- in theory
  1. cut out a sub-component
  2. model it's environment abstractly, i.e.,

  ⇒ add an enviroment *process* which
    - closes the sub-component
    - shows more behavior than the real environment ⇒ *in extremis:* add chaos-process
  3. push the button ...

- in practice
  - components and interfaces might be large
  - closing is tedious
  - SDL-tools (or others) don't often work with abstract data

Model checking open SDL systems

- three more specific problems

1. asynchronous input queues
2. (infinite data domains)
3. chaotic timer behavior

- three specific *practical* solutions

1. no external chaos process

"embedding chaos"

2. one-valued data abstraction (= no external data)
3. three-valued timer abstraction

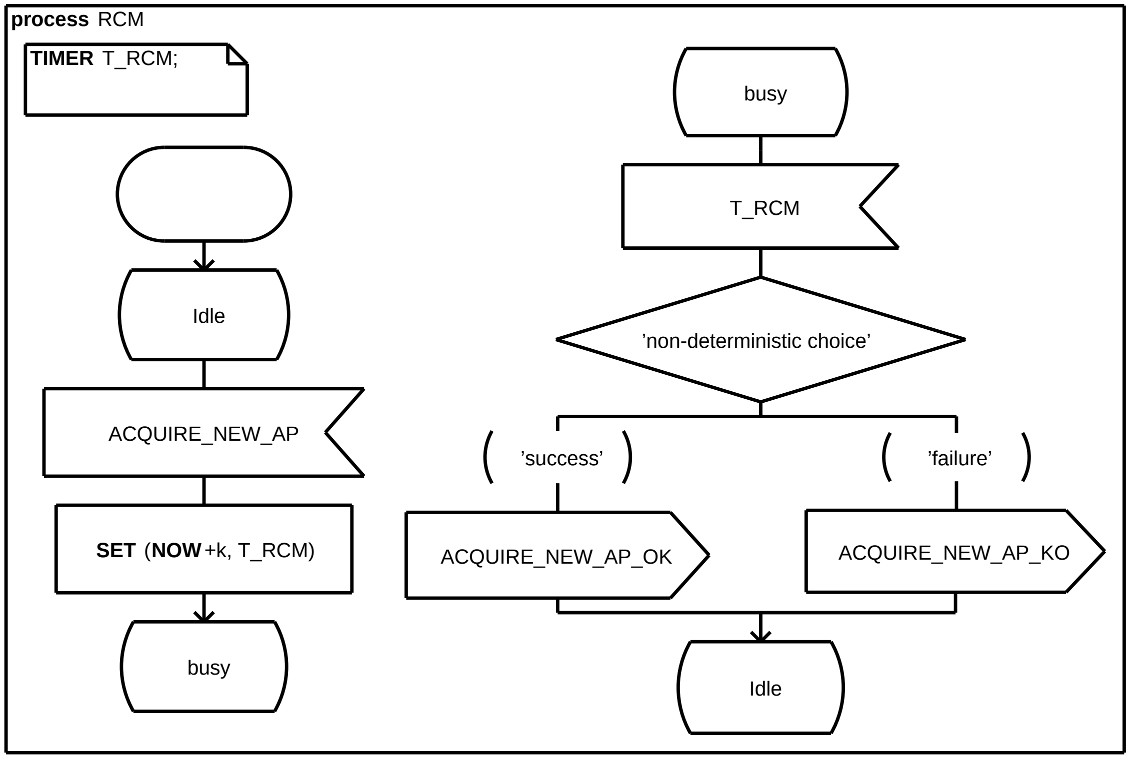
Goal

- Transformation
- automatic
- yielding a closed system
- safe abstraction
- executable with standard SDL-semantics $\Rightarrow$ source code transformation.

Roadmap

1. (sketch) of syntax
2. SO-semantics of SDL
   (a) local rules
   (b) parallel composition
3. semantics of timers
4. closing the system via data-flow analysis
5. dealing with chaotic timers

Syntax: Example



Syntax

- guarded, labelled edges $l \longrightarrow_\alpha \hat{l}$ connecting locations
- actions α: (with guards g)
  - input: $?s(x)$
  - output: $g \triangleright P!s(e)$
  - assignment: $g \triangleright x := e$

Semantics (local)

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

⇒ labelled steps between configuations

Semantics cont'd (local rules)

$$\frac{l \longrightarrow_{?s(x)} \hat{l} \in \mathit{Edg}}{(l, \eta, (s, v) :: q) \rightarrow_{\tau} (\hat{l}, \eta[x \mapsto v], q)} \textsc{Input}$$

$$\frac{l \longrightarrow_{?s'(x)} \hat{l} \in \mathit{Edg} \Rightarrow s' \neq s}{(l, \eta, (s, \_) :: q) \rightarrow_{\tau} (l, \eta, q)} \textsc{Discard}$$

$$\frac{l \longrightarrow_{g \,\triangleright\, P'!(s,e)} \hat{l} \in \mathit{Edg} \qquad [\![g]\!]_{\eta} = \mathit{true} \qquad [\![e]\!]_{\eta} = v}{(l, \eta, q) \rightarrow_{P'!(s,v)} (\hat{l}, \eta, q)} \textsc{Output}$$

$$\frac{v \in D}{(l, \eta, q) \rightarrow_{P?(s,v)} (l, \eta, q :: (s, v))} \textsc{Receive}$$

$$\frac{l \longrightarrow_{g \,\triangleright\, x:=e} \hat{l} \in \mathit{Edg} \qquad [\![g]\!]_{\eta} = \mathit{true} \qquad [\![e]\!]_{\eta} = v}{(l, \eta, q) \rightarrow_{\tau} (\hat{l}, \eta[x \mapsto v], q)} \textsc{Assign}$$

Timers in SDL

- no real-time
- discrete-time semantics, as in the *DTSpin* ("discrete time *Spin*") model-checker [Bošnački and Dams, 1998, DTSpin2000, 2000]

⇒ time evolves by ticking down (active) timer variables

- timer: active or deactivated
- timeout possible: if active timer has reached 0
- modelled by time-out guards (cf. [Bošnački et al., 2000])

Syntax for timers

- 3 (guarded) actions involving timers

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

- Note: timeout is guarded by “timer-guard” g_t

Semantics cont'd (local timer rules)

$$\frac{l \longrightarrow_{g \,\triangleright\, set\ t:=e} \hat{l} \in Edg \qquad [\![g]\!]_\eta = true \qquad [\![e]\!]_\eta = v}{(l, \eta, q) \rightarrow_\tau (\hat{l}, \eta[t \mapsto on(v)], q)} \textsc{Set}$$

$$\frac{l \longrightarrow_{g \,\triangleright\, reset\ t} \hat{l} \in Edg \qquad [\![g]\!]_\eta = true}{(l, \eta, q) \rightarrow_\tau (\hat{l}, \eta[t \mapsto off], q)} \textsc{Reset}$$

$$\frac{l \longrightarrow_{g_t \,\triangleright\, reset\ t} \hat{l} \in Edg \qquad [\![t]\!]_\eta = on(0)}{(l, \eta, q) \rightarrow_\tau (\hat{l}, \eta[t \mapsto off], q)} \textsc{Timeout}$$

$$\frac{(l \longrightarrow_\alpha \hat{l} \in Edg \Rightarrow \alpha \neq g_t \triangleright reset\ t) \qquad [\![t]\!]_\eta = on(0)}{(l, \eta, q) \rightarrow_\tau (l, \eta[t \mapsto off], q)} \textsc{TDiscard}$$

Parallel composition

- standard product construction
- message passing using the labelled steps
- Note: Tick step = counting down active timers:
  - can be taken only when no other move possible

$\Rightarrow$ tick step has least priority!

$$\frac{\mathit{blocked}(l, \eta, q)}{(l, \eta, q) \rightarrow_{\mathit{tick}} (l, \eta[t \mapsto (t-1)], q)} \textsc{Tick}$$

Parallel composition (rules)

$$\frac{(\sigma_1, q_1) \rightarrow_{P!(s,v)} (\hat{\sigma}_1, \hat{q}_1) \qquad (\sigma_2, q_2) \rightarrow_{P?(s,v)} (\hat{\sigma}_2, \hat{q}_2) \qquad s \notin \mathit{Sig}_{ext}}{(\sigma_1, q_1) \times (\sigma_2, q_2) \rightarrow_{\tau} (\hat{\sigma}_1, \hat{q}_1) \times (\hat{\sigma}_2, \hat{q}_2)} \textsc{Comm}$$

$$\frac{(\sigma_1, q_1) \rightarrow_{\lambda} (\hat{\sigma}_1, \hat{q}_1) \qquad \lambda = \{\tau, P?(s,v), P!(s,v) \mid s \in \mathit{Sig}_{ext}\}}{(\sigma_1, q_1) \times (\sigma_2, q_2) \rightarrow_{\lambda} (\hat{\sigma}_1, \hat{q}_1) \times (\sigma_2, q_2)} \textsc{Interleave}$$

$$\frac{\mathit{blocked}(l, \eta, q)}{(l, \eta, q) \rightarrow_{\mathit{tick}} (l, \eta[t \mapsto (t-1)], q)} \textsc{Tick}$$

What's next

- status: semantics for open systems = chaotic signal-exchange with environment
- goal:
  - no external communication
  - abstract data from outside: chaotic data value $\mathbb{T}$
- side-condition
  - use official/implemented SDL-semantics (tools!):
    - ∗ there are no abstracted data in SDL
    - ∗ we cannot re-implement tick
  - keep it simple

The need for data-flow analysis

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

  remove external reception actions (= replace it by τ-actions[3])
- But: transformation may lead to less behavior

$\Rightarrow$ Unsound!

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

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

[3] In SDL: NONE-transitions

Closing the program

- two steps:

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

Data-flow analysis

- forward analysis
- control-flow (almost) directly given by SDL-automata
- modelling the abstract effect/transfer functions per action = node, e.g.:

$$
\begin{array}{rcll}
f(?s(x))\eta^\alpha & = & \begin{cases} \eta^\alpha[x \mapsto \top] & s \in Sig_{ext} \\ \eta^\alpha[x \mapsto \bigvee\{[\![e]\!]_{\eta^\alpha} | \alpha_{n'} = g \rhd P!s(e) \text{ for some node } n'] & \text{else} \end{cases} \\
f(g \rhd P!s(e))\eta^\alpha & = & \eta^\alpha \\
f(g \rhd x := e)\eta^\alpha & = & \eta^\alpha[x \mapsto [\![e]\!]_{\eta^\alpha}]
\end{array}
$$

- constraint solving: minimal solution for

$$\eta^\alpha_{post}(n) \geq f_n(\eta^\alpha_{pre}(n)) \tag{1.1}$$

$$\eta^\alpha_{pre}(n) \geq \bigvee\{\eta^\alpha_{post}(n') \mid (n', n) \text{ in flow relation}\} \tag{1.2}$$

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 $n' \in S$: $\eta^{\alpha}(n') := f(\eta^{\alpha}(n))$;
    $WL := WL \backslash n \cup S$;
until $WL = \emptyset$;

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

What about time?

- chaotic environment also means: chaotic timed behaviour
- so far: we ignored timers
- Remember: time steps (ticks) have least priority!

⇒ new τ steps make ticks impossible!

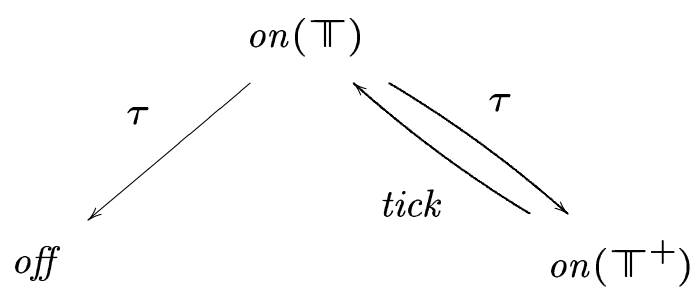
⇒ chaos = at arbitrary points

1. *sending* any possible value, **+**
2. refusing to send something (lest to get less ticks and thus less timeouts)

Timer abstraction

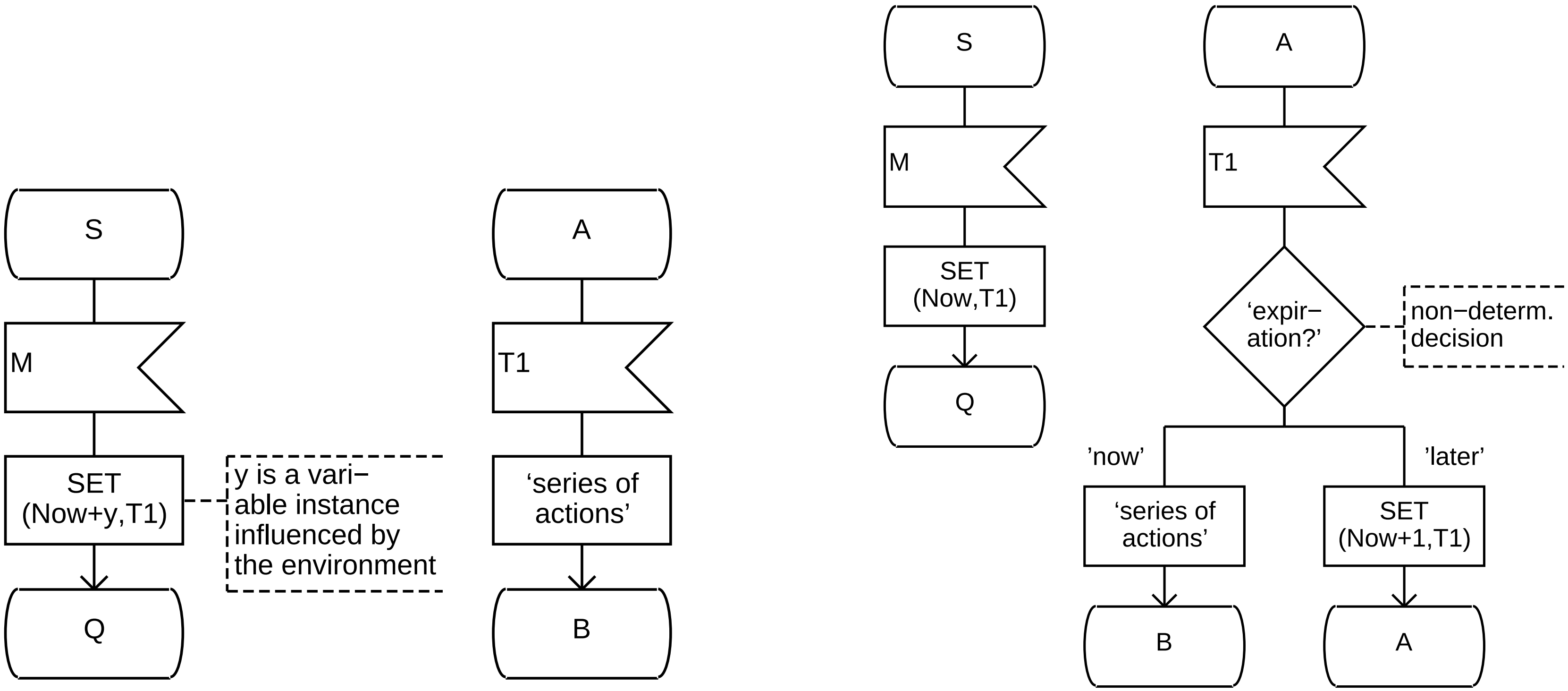
- Three abstract values:
  - de-activated
  - arbitrarily active
  - active, but not 0 (no time-out possible)

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



- embedding the timer: one additional timer t_P per process

Transformation of timers (in SDL)



Transformation rules

- e.g.: input[4].

$$\frac{l \longrightarrow_{?s(x)} \hat{l} \in Edg^{\top} \qquad s \in Sig_{ext}}{l \longrightarrow_{g^\sharp \,\triangleright\, skip} \hat{l} \in Edg^\sharp} \text{ T-INPUT}_2$$

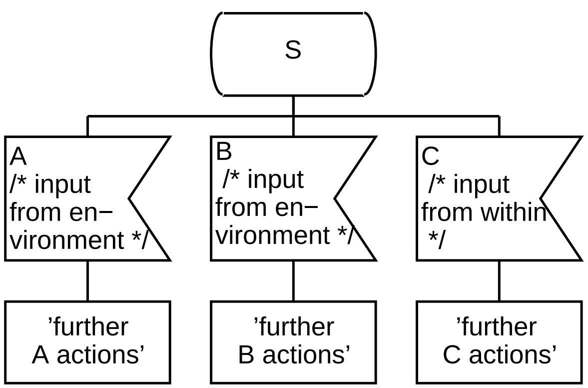
$$\frac{}{l \longrightarrow_{g_{t_P} \,\triangleright\, set\ t_P := 1} l \in Edg^\sharp} \text{ T-NOINPUT}$$

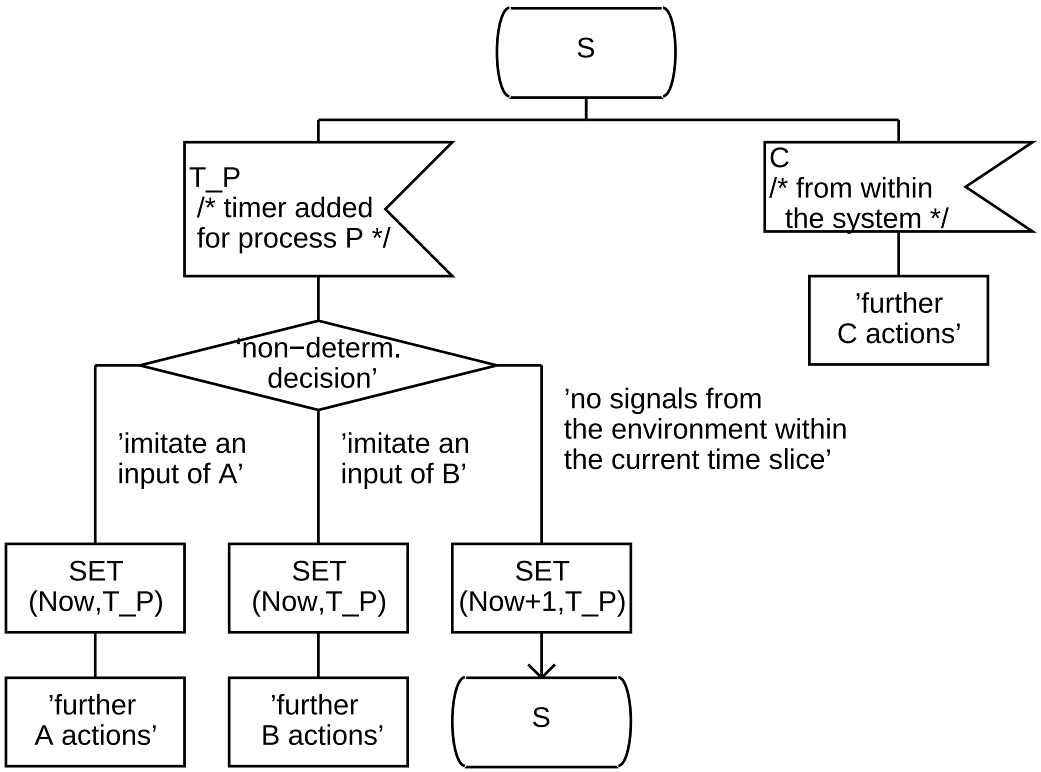
- e.g.: assignement

$$\frac{l \longrightarrow_{g \,\triangleright\, x:=e} \hat{l} \in Edg^{\top} \qquad [\![e]\!]_{\eta_l^\alpha} = \top \qquad g^\sharp = [\![g]\!]_{\eta_l^\alpha}}{l \longrightarrow_{g^\sharp \,\triangleright\, skip} \hat{l} \in Edg^\sharp} \text{ T-ASSIGN}_2$$

[4] T-INPUT$_1$ and T-ASSIGN$_1$ do “nothing”

Transformation of inputs (in SDL)





Transformation

$$\frac{l \longrightarrow_{g \,\triangleright\, x:=e} \hat{l} \in Edg^{\mathbb{T}} \qquad [\![e]\!]_{\eta_l^\alpha} \neq \top \qquad g^\sharp = [\![g]\!]_{\eta_l^\alpha}}{l \longrightarrow_{g^\sharp \,\triangleright\, x:=e} \hat{l} \in Edg^\sharp} \ \text{T-ASSIGN}_1$$

$$\frac{l \longrightarrow_{g \,\triangleright\, x:=e} \hat{l} \in Edg^{\mathbb{T}} \qquad [\![e]\!]_{\eta_l^\alpha} = \top \qquad g^\sharp = [\![g]\!]_{\eta_l^\alpha}}{l \longrightarrow_{g^\sharp \,\triangleright\, skip} \hat{l} \in Edg^\sharp} \ \text{T-ASSIGN}_2$$

$$\frac{l \longrightarrow_{?s(x)} \hat{l} \in Edg^{\mathbb{T}} \qquad s \notin Sig_{ext}}{l \longrightarrow_{?s(x)} \hat{l} \in Edg^\sharp} \ \text{T-INPUT}_1$$

$$\frac{l \longrightarrow_{?s(x)} \hat{l} \in Edg^{\mathbb{T}} \qquad s \in Sig_{ext}}{l \longrightarrow_{g_{t_P} \,\triangleright\, reset\ t_P} \longrightarrow_{set\ t_P:=0} \hat{l} \in Edg^\sharp} \ \text{T-INPUT}_2$$

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

$$\frac{l \longrightarrow_{g \,\triangleright\, P'!(s,e)} \hat{l} \in Edg^{\mathbb{T}} \qquad s \notin Sig_{ext} \qquad g^\sharp = [\![g]\!]_{\eta_l^\alpha}}{l \longrightarrow_{g^\sharp \,\triangleright\, P'!(s,e)} \hat{l} \in Edg^\sharp} \ \text{T-OUTPUT}_1$$

$$\frac{l \longrightarrow_{g \,\triangleright\, P'!(s,e)} \hat{l} \in Edg^{\top} \qquad s \in Sig_{ext} \qquad g^{\sharp} = [\![g]\!]_{\eta_l^{\alpha}}}{l \longrightarrow_{g^{\sharp} \,\triangleright\, skip} \hat{l} \in Edg^{\sharp}}\ \text{T-OUTPUT}_2$$

$$\frac{l \longrightarrow_{g \,\triangleright\, set\ t:=e} \hat{l} \in Edg^{\top} \qquad g^{\sharp} = [\![g]\!]_{\eta_l^{\alpha}} \qquad [\![e]\!]_{\eta_l^{\alpha}} \neq \top}{l \longrightarrow_{g^{\sharp} \,\triangleright\, set\ t:=e} \hat{l} \in Edg^{\sharp}}\ \text{T-SET}_1$$

$$\frac{l \longrightarrow_{g \,\triangleright\, set\ t:=e} \hat{l} \in Edg^{\top} \qquad g^{\sharp} = [\![g]\!]_{\eta_l^{\alpha}} \qquad [\![e]\!]_{\eta_l^{\alpha}} = \top}{l \longrightarrow_{g^{\sharp} \,\triangleright\, set\ t:=0} \hat{l} \in Edg^{\sharp}}\ \text{T-SET}_2$$

$$\frac{l \longrightarrow_{g \,\triangleright\, reset\ t} \hat{l} \in Edg^{\top} \qquad g^{\sharp} = [\![g]\!]_{\eta_l^{\alpha}}}{l \longrightarrow_{g^{\sharp} \,\triangleright\, reset\ t} \hat{l} \in Edg^{\sharp}}\ \text{T-RESET}$$

$$\frac{l \longrightarrow_{g_t \,\triangleright\, reset\ t} \hat{l} \in Edg^{\top} \qquad g_t^{\sharp} = [\![g_t]\!]_{\eta_l^{\alpha}}}{l \longrightarrow_{g_t^{\sharp} \,\triangleright\, reset\ t} \hat{l} \in Edg^{\sharp}}\ \text{T-TIMEOUT}$$

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

Soundness result

Theorem. *The transformed system is closed and a safe abstraction of the original one*

- safe abstraction, i.e.,

$$\boxed{\text{if } S^\sharp \models \varphi \text{ then } S \models \varphi},$$

  where φ is an LTL-formula[5]

Proof:

- transformed system and original in simulation relation

$\Rightarrow$ $S^\sharp$ shows more behavior than S, i.e., it has more traces.

[5] which does not mention chaotically influenced variables.

Related work

- software testing
- e.g. [Colby et al., 1998] VERISOFT, C, untimed
- [Dwyer and Pasareanu, 1998]: filtering = “refined” chaos, but external
- Module checking:
  - checking open systems
  - e.g. MOCHA [Alur et al., 1998]

Future work

- implementation
- “refined” chaos
  - specified properties by LTL
  - arbitrarly chaotic timer exporation $\Rightarrow$ calulated by data-flow analysis

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.

[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.

[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. To appear in the Proceedings of the 8th International Static Analysis Symposium (SAS'01), Paris, 2001.

[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 399–416. Springer-Verlag.

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