Embedding Chaos

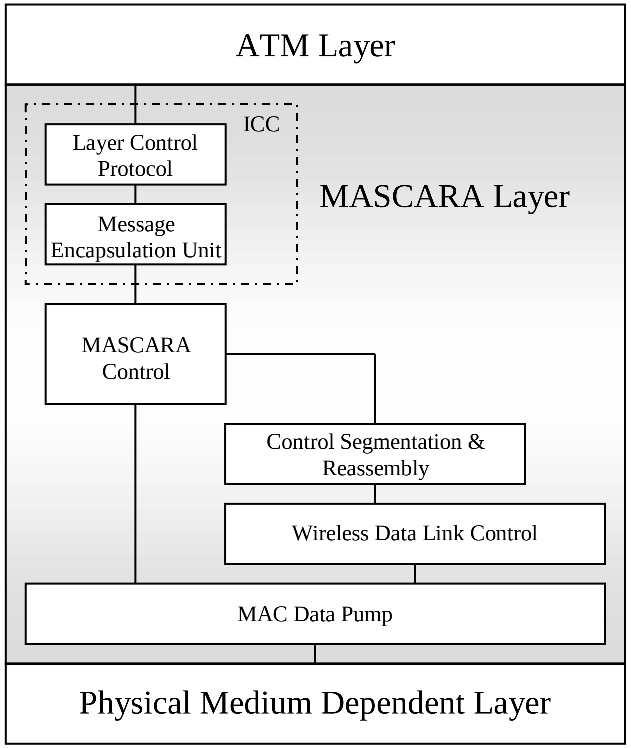
Natalia Sidorova   Martin Steffen

Dept. of Math. and Computer Science
Eindhoven University of Technology

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

Motivation & starting point

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



tf

Model checking

- pro: automatic ("push-button") verification method

$$p \models \varphi$$

- con:
  - state-space explosion
  - how to obtain the model from a piece of software?

Specification Description Language (SDL)

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

Model checking SDL

- various aggravations
  1. it's about software (data)
  2. it's about large 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

Model checking SDL in theory (and practice)

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

  $\Rightarrow$ add an enviroment *process* which
    - closes the sub-component
    - shows more behavior than the real environment

    $\Rightarrow$ *in extremis:* add chaos-process
  3. push the button ...
- in practice
  - components and interfaces might be large
  - closing is tedious
  - SDL-tools don't often work with abstract data

Model checking open SDL systems

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

"embedding chaos"

  3. three-valued timer abstraction

tf*rrr*

Goal

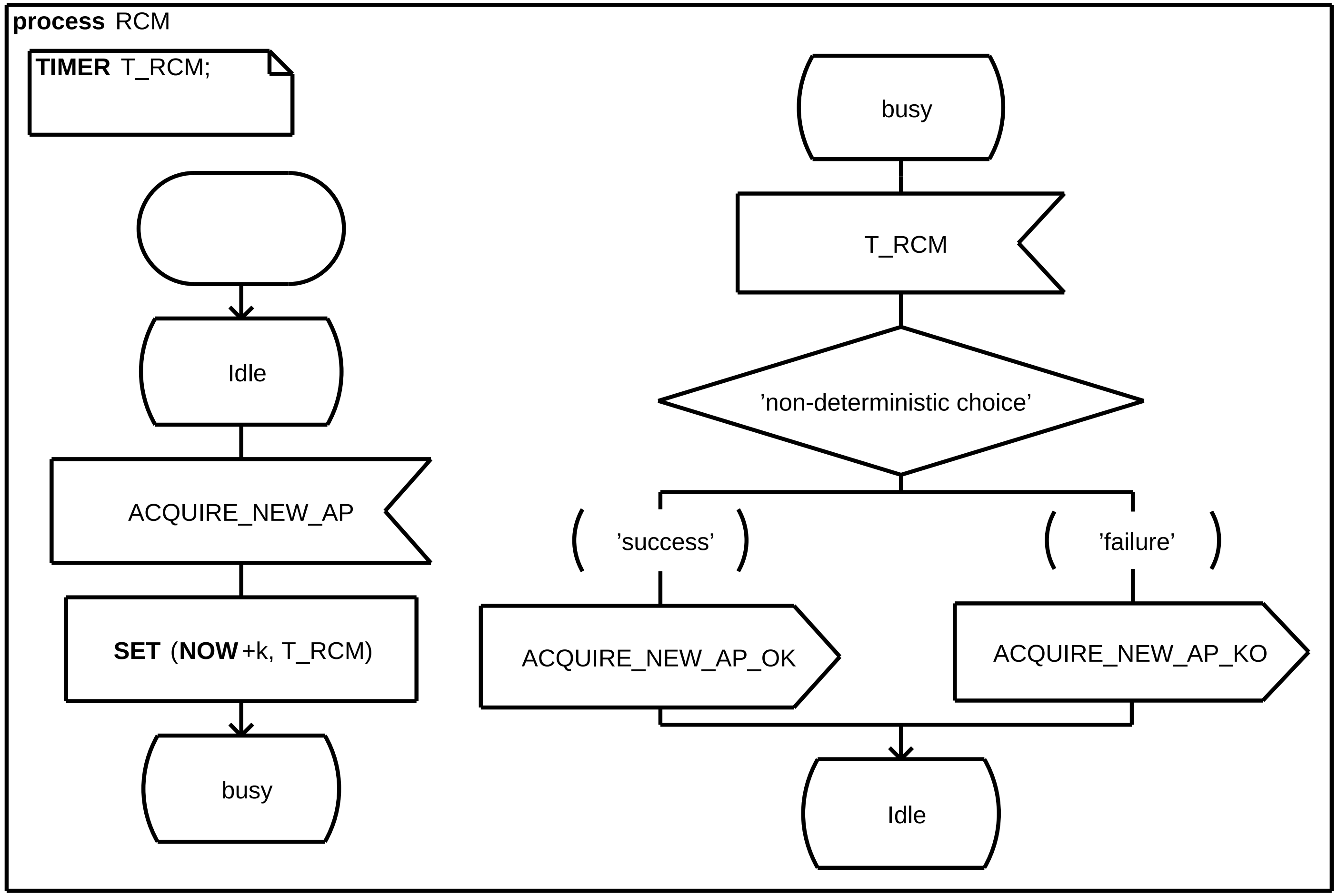
- automatic transformation
- yielding a closed system
- safe abstraction
- executable with standard SDL-semantics $\Rightarrow$ source code transformation.

tfrrr

Roadmap

1. (sketch of) syntax
2. SO-semantics of SDL
   (a) local and global rules
   (b) semantics of timers
3. closing the system via data-flow analysis
4. 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 process configuration:
  1. location/control state
  2. valuation of variables
  3. content of input-queue

$\Rightarrow$ labelled steps between configuations, e.g.

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

Timers in SDL

- no real-time
- discrete-time semantics, as in the *DTSpin* (“discrete time *Spin*”) model-checker [BD98, DTS00]

$\Rightarrow$ 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. [BDHS00])

tfrrr

Syntax for timers

- 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	$g_t \triangleright reset\ t$	perform a timeout, thereby deactivate t

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

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!

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

tfrrr

What's next

- 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 $\Rightarrow$ replace it by τ-actions (in SDL: NONE-transitions)

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

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

closing the program

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

Data-flow analysis

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

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

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

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

What about time?

- so far: we ignored timers
- chaos $\Rightarrow$ also chaotic timed behaviour
- remember: time steps (ticks) have least priority!

$\Rightarrow$ new τ steps make ticks impossible!

$\Rightarrow$

chaos = at arbitrary points

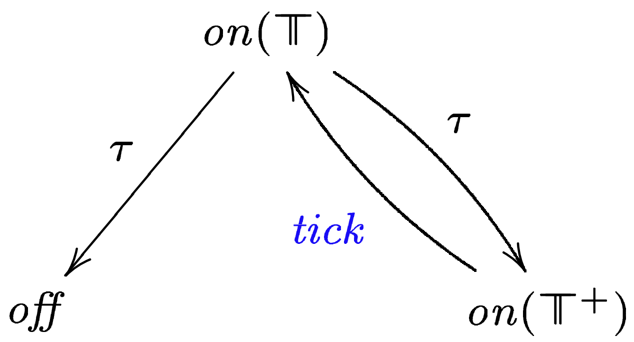
1. *sending* any possible value, and
2. refusing to send something (lest to get less ticks and thus less timeouts)

tfrrr

Timer abstraction

- three abstract values:

1. de-activated
2. arbitrarily active
3. active, but not 0 (no time-out possible)

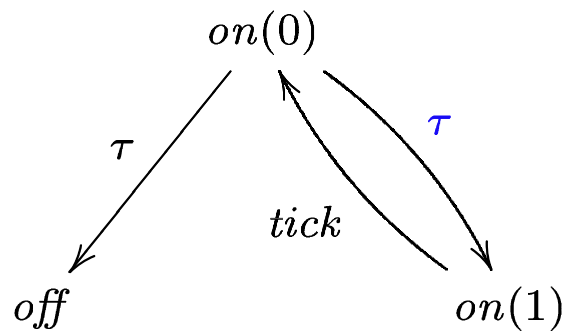


- arbitrary expiration time $\Rightarrow$ non-deterministic setting from $on(\mathbb{T})$ to $on(\mathbb{T}^+)$.
- embedding the timer: one additional timer t_P within each process

Transformation rules

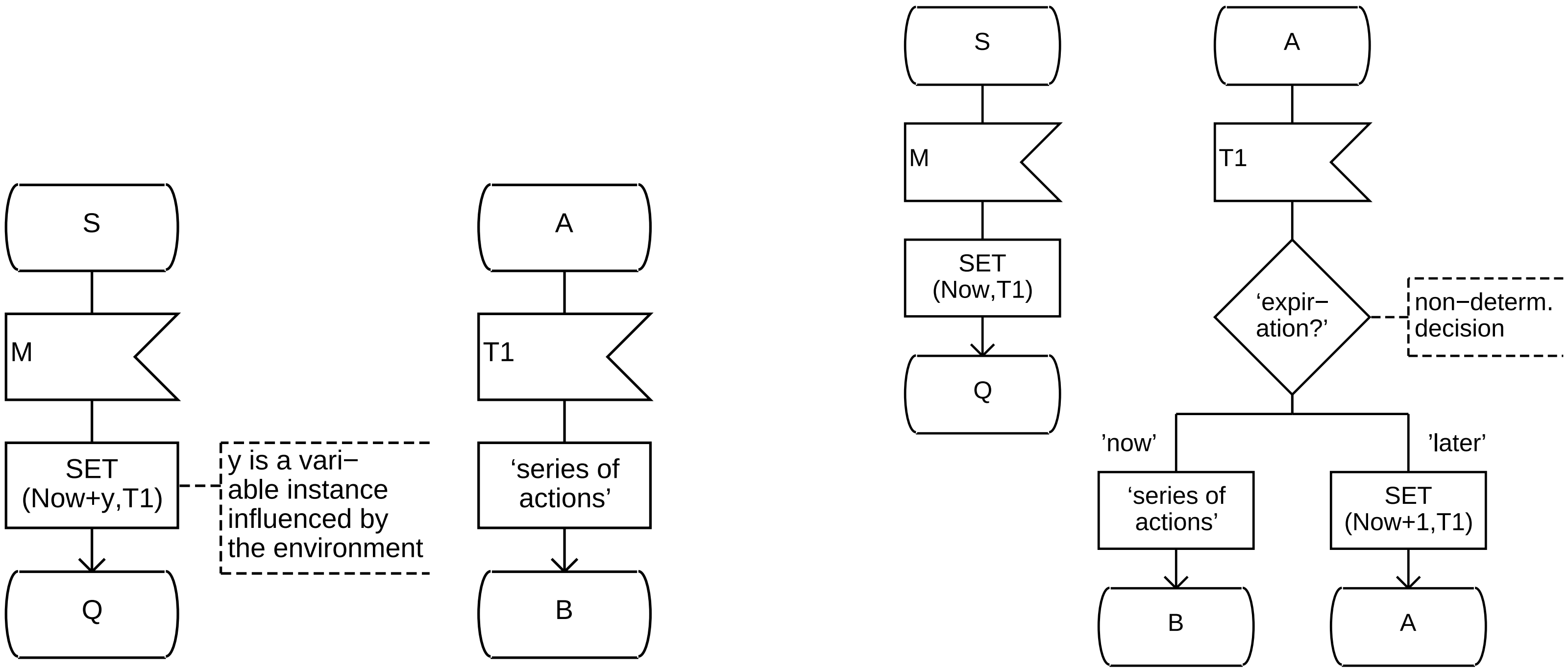
- using result of the flow analysis
- inference rule(s) for each syntax construct
- e.g.,

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



tfrrr

Transformation rule: in SDL



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

Proof:

- transformed system and original in simulation relation

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

Related work

- software testing
- VERISOFT, C, untimed [CGJ98]
- filtering = “refined” chaos, but external [DP98] [Pas00]
- module checking:
  - checking open systems
  - e.g. MOCHA [AHM^{+}98]

Future work

- implementation
- embedding "refined" chaos
  - specified properties by LTL
  - arbitrarily chaotic timer exporation $\Rightarrow$ calculated by data-flow analysis

References

[AHM+98] Rajeev Alur, Thomas A. Henzinger, F.Y.C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. Mocha: Modularity in model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Proceedings of CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer-Verlag, 1998.

[BD98] Dragan Bošnački and Dennis Dams. Integrating real time into Spin: A prototype implementation. In S. Budkowski, A. Cavalli, and E. Najm, editors, *Proceedings of Formal Description Techniques and Protocol Specification, Testing, and Verification (FORTE/PSTV'98)*. Kluwer Academic Publishers, 1998.

[BDHS00] Dragan Bošnački, Dennis Dams, Leszek Holenderski, and Natalia Sidorova. Verifying SDL in Spin. In S. Graf and M. Schwartzbach, editors, *TACAS 2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[CGJ98] C. Colby, P. Godefroid, and L. J. Jagadeesan. Automatically closing of open reactive systems. In *Proceedings of 1998 ACM SIGPLAN Conference on*

Programming Language Design and Implementation. ACM Press, 1998.

[DP98] M. B. Dwyer and C. S. Pasareanu. 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, 1998.

[DTS00] Discrete-time Spin. `http://win.tue.nl/~draga` 2000.

[Pas00] Corina S. Pasareanu. DEAO kernel: Environment modeling using LTL assumptions. Technical Report SASA-ARC-IC-2000-196, NASA Ames, 2000.

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