Synchronous Closing of Timed SDL Systems for Model Checking

Natalia Sidorova   Martin Steffen

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

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

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?
- additional techniques:
  1. abstraction:
     (a) data abstraction: replace concrete domains by finite, abstract ones
     (b) control abstraction, i.e., add non-determinism
  2. system decomposition

Model checking in theory (and practice)

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

  $\Rightarrow$ add an environment *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
  - model checkers don't often work with abstract data

Specification Description Language (SDL)

- 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 time-outs
  - bells and whistles: graphical notation, structuring mechanisms, OO, . . .

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. three-valued timer abstraction
  3. no asynchronous communication with environment

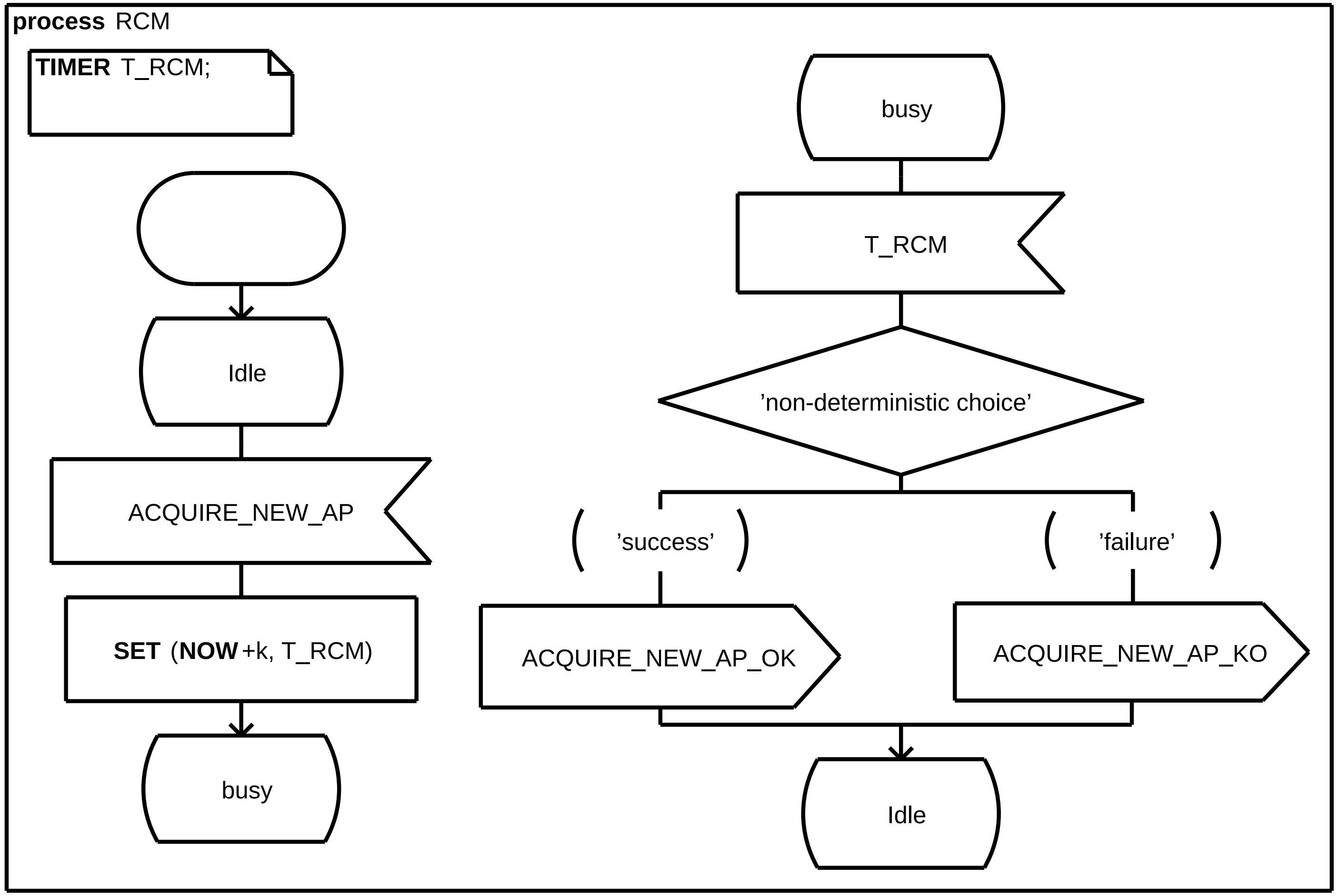
Goal

- yielding a closed system
- safe abstraction
- automatic transformation

Roadmap

1. (sketch of) syntax
2. SO-semantics of SDL
   (a) local and global rules
   (b) semantics of timers
3. eliminating external data via data-flow analysis
4. dealing with chaotic timers
5. synchronous instead of asynchronous environment $\Rightarrow$ eliminating external queues

Syntax: Example



Syntax

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

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

with guards g, signals s, processes P, channels c

Semantics (local)

- straightforward operational small-step semantics
  - interleaving semantics
  - top-level concurrency
  - channel queues between processes
- local process configuration:
  1. location/control state
  2. valuation of variables

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

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

Timers in SDL

- no real-time
- discrete-time semantics, as in [HP89] and 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])

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, i.e.,$t = 0$

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 except input, i.e.,

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

What's next

- goal:
  - abstract data from outside: chaotic data value $\mathbb{T}$
  - only synchronous external communication
- side-condition
  - verification with DTSpin model checker (tools):
    - there are no abstracted data
    - we cannot re-implement tick
  - keep it simple

The need for data-flow analysis

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

$\Rightarrow$ remove all variables (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.:

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

- 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 fbw−graph of the program
output: $\eta^\alpha_{pre}, \eta^\alpha_{post}$;

$\eta^\alpha(n) = \eta^\alpha_{init}(n)$;
$WL = \{n \mid \alpha_n = c?s(x), c \notin out(\bar{P})\}$;

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 $WL = \emptyset$;

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

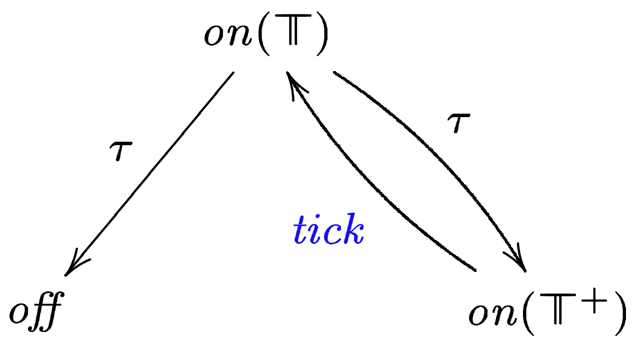
What about time?

- so far: we ignored timers
- 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. arbitrarily active
2. active, but not 0 (no time-out possible)
3. de-activated

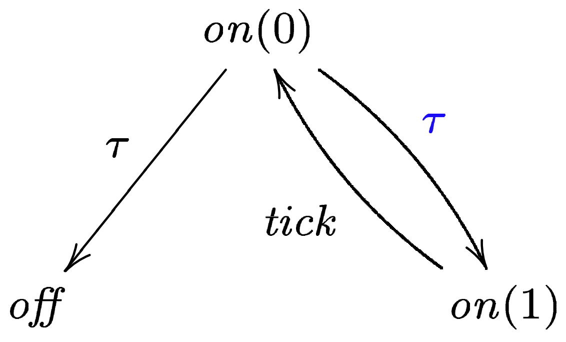


- arbitrary expiration time $\Rightarrow$ non-deterministic setting from $on(\mathbb{T})$ to $on(\mathbb{T}^+)$.

Transformation rules

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

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



- transformation yields a safe abstraction

Conditions on the environment

- closing environment is an abstraction of the rest of the system
- but: rest of the system is composed asynchronously

$\Rightarrow$ Question: when is it safe (no behavior lost) to replace asynchronous comm. with the environment by synchronous one.

$\Rightarrow$ environment process must be
  - input enabled
  - not reactive

- e.g., most abstract environment (“chaos”) is ok

Conditions on the environment (cont'd)

- $tick$-step only if all queues empty $\Rightarrow$ restrictions apply only per time slice

  A run is tick-separated =
  - it contains no zero-time cycle
  - for every time slice of the run holds:
    - no output action, or
    - no input except *input discard* and no output over two different channels.
- A process is tick-separated = all runs are tick-separated

Soundness result

Transformation of S into $S^\sharp$:

1. removing external data (using data-flow analysis)
2. making external communication synchronous

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

- i.e.,

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

where φ is an LTL-formula (which does not mention chaotically influenced variables)

Related work

- software testing
- VERISOFT, C, untimed [CGJ98]
- filtering [DP98] [Pas00]
- module checking:
  - checking open systems
  - e.g. MOCHA [AHM+98]

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/~dragan/DTSpin.h` 2000.

[HP89] Gerard Holzmann and Joanna Patti. Validating SDL specifi cations: an experiment. In Ed Brinksma, editor, *International Workshop on Protocol Specification, Testing and Verification IX (Twente, The Netherlands)*, pages 317–326. North-Holland, 1989. IFIP TC-6 International Workshop.

[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.