New Developments around the μCRL Tool Set

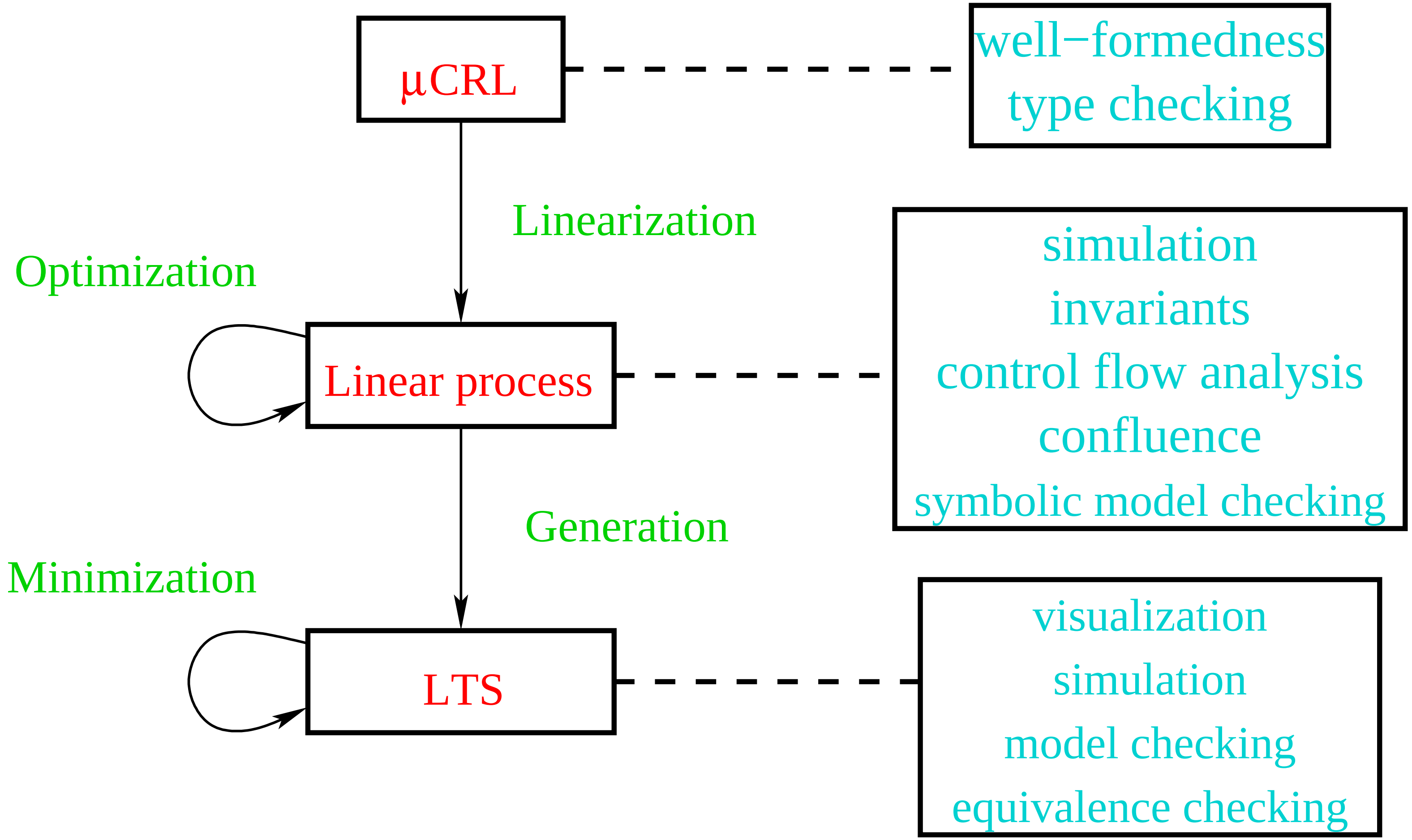
Stefan Blom, Jan Friso Groote
Izak van Langevelde, Bert Lisser
Jaco van de Pol

Centrum voor Wiskunde en Informatica
Specification and Analysis of Embedded Systems
Theme leader: Wan Fokkink
Amsterdam, The Netherlands

OVERVIEW

- Introduction
- Symbolic verification
  - Linear processes, Static Analysis
  - Confluence
  - Symbolic Model Checking
- Explicit state verification
  - Distributed implementation
  - On-the-fly via Open/Cæsar
  - Visualization
- Some Applications

μCRL Tool Set



μCRL = process algebra + abstract data types

μCRL inherits from abstract data types:

- sorts . . . . Nat, List, Bool
- function symbols . . . . and: Bool $\times$ Bool $\rightarrow$ Bool
- equations . . . . length(cons(x,l)) = succ(length(l))

μCRL inherits from ACP style process algebra :

- atomic actions with synchronization . . . . $read \mid write = comm$
- abstraction, encapsulation, renaming . . . . $\tau, \delta, \cdots$
- process operators . . . . $+, \cdot, \|$
- recursive process equations . . . . $X = a.c.X + b.X$

μCRL $= \cdots +$ integration

μCRL provides connections between data and processes:

atomic actions have data labels: .... $send(frame(x,y))$

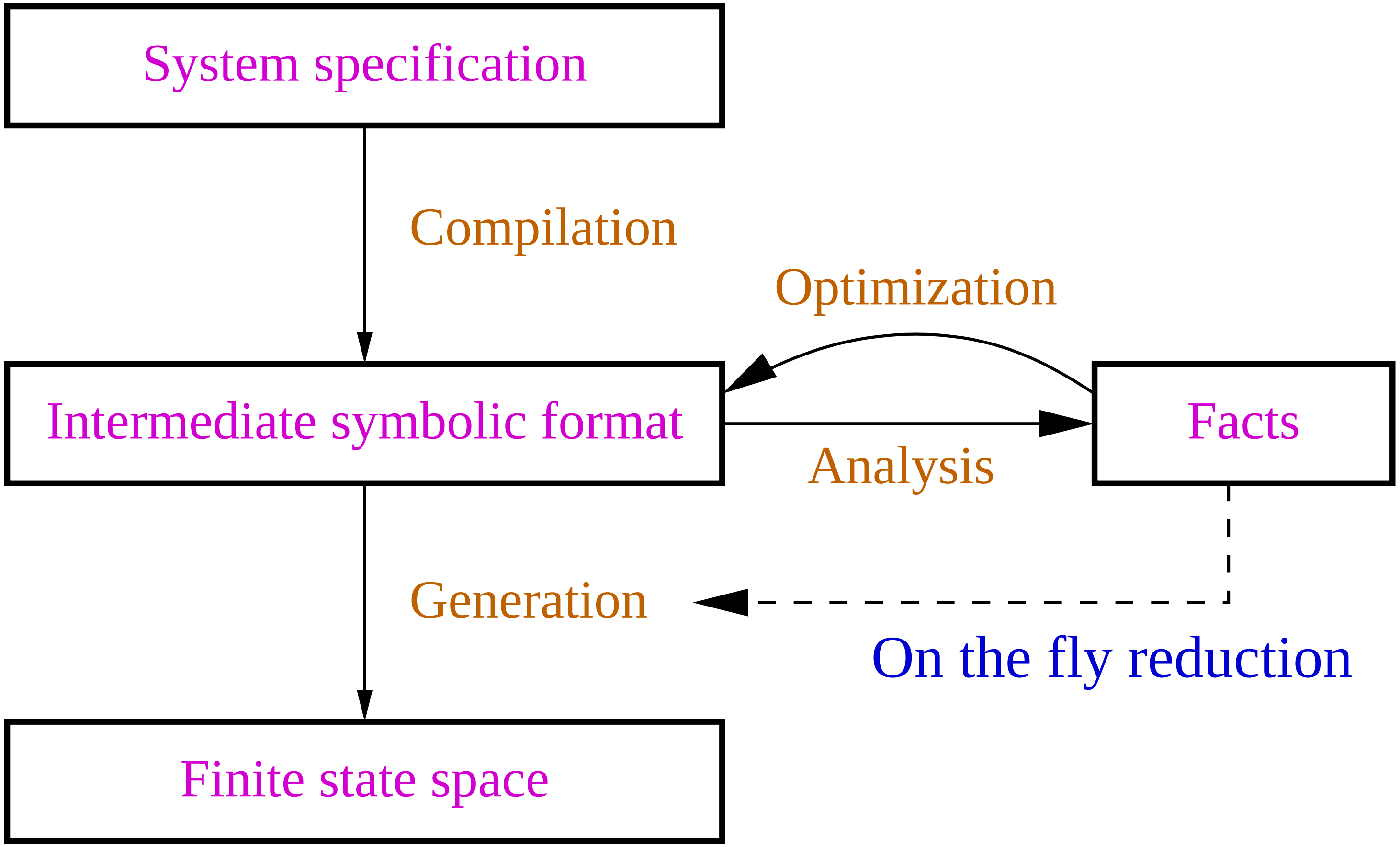
conditions on data: .... $finish \lhd empty(buffer) \rhd continue$

choice over data: .... $\sum_{x:Nat} rd(x).wr(Suc(x))$

parameterized recursion: ....

$$X(prev:Nat) = \sum_{next:Nat} read(next).send(prev).X(next)$$

Outline of our Verification Process



Optimizations

Various optimizations are implemented

- Compiler techniques (control + data flow analysis)
  - replace unchanged variables by constants
  - remove variables that are not used
  - reset variables when temporarily not used
- Automated theorem prover based
  - invariant generation/checking
  - reachability analysis
  - Partial-order-like reduction based on
    * Confluence detection (static)
    * Confluence-based state space reduction (on-the-fly)

Linear process format

$$
\begin{aligned}
X(d:D) \quad = \quad & \sum_{e_1:E} \underline{c_1(d,e_1)} \Rightarrow \underline{a_1(d,e_1)}.X(\underline{g_1(d,e_1)}) \\
+ \quad & \cdots \\
+ \quad & \sum_{e_n:E} c_n(d,e_n) \Rightarrow a_n(d,e_n).X(g_n(d,e_n))
\end{aligned}
$$

- d is a vector of state variables
- e_i is the vector of local variables for summand i
- c_i is the enabling condition for summand i
- a_i is the (visible/invisible) actions for summand i
- g_i is the next-state function for summand i

$X(d) \xrightarrow{a} X(d')$ iff for some i,

$$\exists e_i.\ c_i(d,e_i) \wedge d' = g_i(d,e_i) \wedge a = a_i(d,e_i)$$

Example: linearization of lossy channel

$$K(a:Nat) = \sum_d in(a,d) \cdot \big(\tau \cdot loss + \tau \cdot out(a,d)\big) \cdot K(a)$$

$K(17)$ is linearized by introducing a program counter:

Example: linearization of lossy channel

$$K(a:Nat) = {}_0\sum_d in(a,d) \cdot_1 \big(\tau \cdot_2 loss + \tau \cdot_3 out(a,d)\big) \cdot_0 K(a)$$

$K(17)$ is linearized by introducing a program counter:

$$\begin{array}{llcll}
\textbf{proc} & K(a,x,pc) & = & \sum_d & pc=0 \Rightarrow in(a,d)\cdot K(a,d,1) \\
 & & + & & pc=1 \Rightarrow \tau \cdot K(a,x,2) \\
 & & + & & pc=1 \Rightarrow \tau \cdot K(a,x,3) \\
 & & + & & pc=2 \Rightarrow loss \cdot K(a,x,0) \\
 & & + & & pc=3 \Rightarrow out(a,x)\cdot K(a,x,0) \\
\textbf{init} & K(17,\perp,0) & & &
\end{array}$$

Parallel composition and hiding are defined directly on linear processes.
In practice, no problematic blow-up occurs.

Example: linearization of lossy channel

$$K(a : Nat) =_0 \sum_d in(a,d) \cdot_1 \big(\tau \cdot_2 loss + \tau \cdot_3 out(a,d)\big) \cdot_0 K(a)$$

The linear process can be optimized in various places:

$$\begin{array}{llcll}
\textbf{proc} & K(a,x,pc) & = & \sum_d & pc = 0 \Rightarrow in(a,d) \cdot K(a,d,1) \\
& & + & & pc = 1 \Rightarrow \tau \cdot K(a,x,2) \\
& & + & & pc = 1 \Rightarrow \tau \cdot K(a,x,3) \\
& & + & & pc = 2 \Rightarrow loss \cdot K(a,x,0) \\
& & + & & pc = 3 \Rightarrow out(a,x) \cdot K(a,x,0) \\
\textbf{init} & K(17,\perp,0) & & &
\end{array}$$

Example: linearization of lossy channel

$$K(a : Nat) =_0 \sum_d in(a,d) \cdot_1 \big(\tau \cdot_2 loss + \tau \cdot_3 out(a,d)\big) \cdot_0 K(a)$$

The optimized linear process will be:

$$
\begin{array}{llcll}
\textbf{proc} & K(x, pc) & = & \sum_d & pc = 0 \Rightarrow in(17, d) \cdot K(d, 1) \\
 & & + & & pc = 1 \Rightarrow \tau \cdot K(\perp, 2) \\
 & & + & & pc = 1 \Rightarrow \tau \cdot K(x, 3) \\
 & & + & & pc = 2 \Rightarrow loss \cdot K(\perp, 0) \\
 & & + & & pc = 3 \Rightarrow out(17, x) \cdot K(\perp, 0) \\
\textbf{init} & K(\perp, 0) & & &
\end{array}
$$

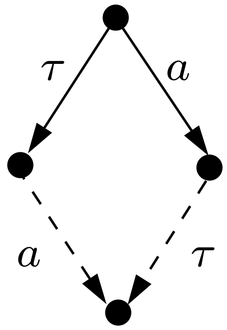
Correctness of static analysis tools

- most optimization tools yield state mappings on LPOs
- state mappings on LPOs yield functional bisimulations on LTSs
- invariants can be used to verify state mappings
- state mappings preserve invariants (in two directions)
- the Focus and Cones method provides matching criteria to prove that two linear processes are branching bisimilar
- LPO meta-theory has been completely verified in PVS
- `mcrl2pvs`: individual specifications can be translated to PVS automatically, and verified by interactive theorem proving

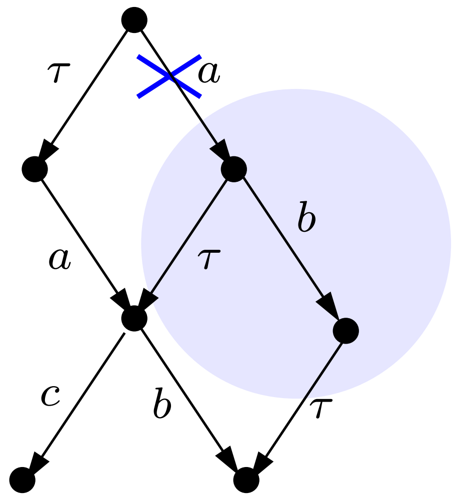
State Space Reduction by Confluence

An LTS can be reduced, by exploiting confluence properties.

strong commutation:



state space reduction:



We will study subsets $\xrightarrow{\textcircled{\tau}} \subseteq \xrightarrow{\tau}$.

Confluence Notions

$\xrightarrow{\tau} \subseteq \xrightarrow{\tau}$ is step/reduce confluent in an LTS iff:

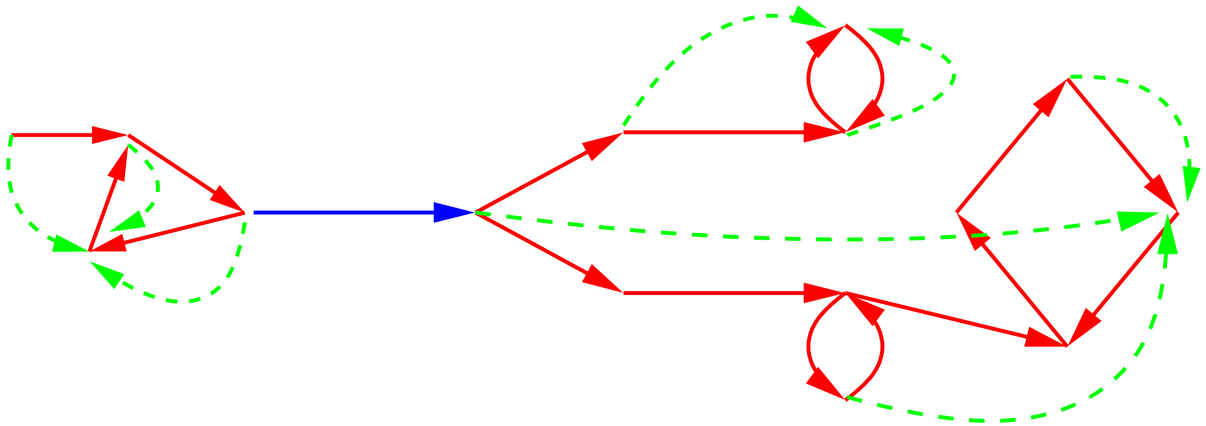
$$\begin{tikzcd} \cdot \arrow[r, "\tau"] \arrow[d, "a"'] \arrow[dr, phantom, "SC"] & \cdot \arrow[d, Rightarrow, dashed, "a"] \\ \cdot \arrow[r, dashed, "\tau"'] & \cdot \end{tikzcd}$$

$$\begin{tikzcd} \cdot \arrow[r, "\tau"] \arrow[d, "a"'] \arrow[dr, phantom, "RC"] & \cdot \arrow[d, Rightarrow, dashed, "a"] \\ \cdot \arrow[r, dashed, two heads, "\tau"'] & \cdot \end{tikzcd}$$

Note: $SC \Rightarrow RC$

Reduction based on Confluence Information

A representation map replaces each state by its representative, which must be unique in the final strongly connected components.



$\longrightarrow$ is a visible step, $\longrightarrow$ are $\xrightarrow{\tau}$ steps.

Representation maps can be computed on-the-fly by an adaptation of Tarjan's algorithm.

Theorem: if $\xrightarrow{\tau}$ is RC and ϕ is a representation map, then $L \mathrel{\underline{\leftrightarrow}}_b L_\phi$.

Confluence detection on LPO

- Mark all τ-summands that commute with all other summands.
- Invariants can be used to prove commutation.
- $\xrightarrow{\textcircled{\tau}}$:= the transitions generated from marked τ-summands.
- Then $\xrightarrow{\textcircled{\tau}}$ is an SC, and hence RC, subset of $\xrightarrow{\tau}$, so it can be used for on-the-fly reduction.
- Confluence marking is preserved by state mappings
- All meta-theory on confluence has been verified in PVS.

Confluence Formula Generation

$$\sum_{e_a} c_a(d,e_a) \Rightarrow a(d,e_a).X(g_a(d,e_a))$$
$$\sum_{e_\tau} c_\tau(d,e_\tau) \Rightarrow \tau.X(g_\tau(d,e_\tau))$$

The commutation formula for this (a,τ)-pair is:

$$\begin{aligned}
\forall d, e_a, e_\tau.\ c_a(d,e_a) \wedge c_\tau(d,e_\tau) \longrightarrow \\
& c_\tau(g_a(d,e_a),e_\tau) \\
\wedge \quad & c_a(g_\tau(d,e_\tau),e_\tau) \\
\wedge \quad & a(d,e_a) = a(g_\tau(d,e_\tau),e_a) \\
\wedge \quad & g_a(g_\tau(d,e_\tau),e_a) = g_\tau(g_a(d,e_a),e_\tau)
\end{aligned}$$

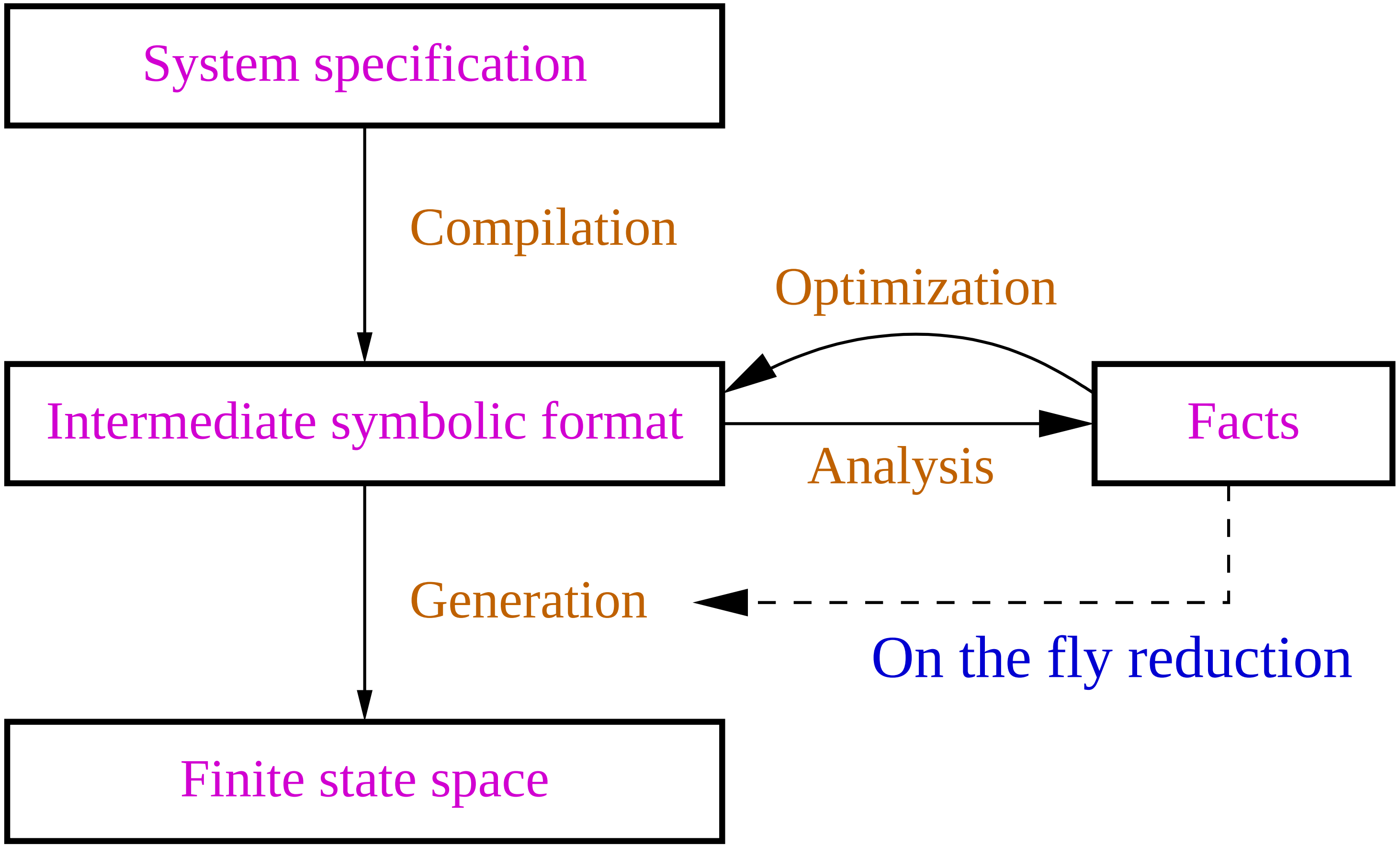
Special-purpose theorem prover

- The μCRL toolset comes with a special-purpose automated theorem prover.
- It handles q.f.f. Boolean formulas over an abstract data type.
- It is based on EQ-BDDs, an extension of BDDs with equations and function symbols (Groote, vdP).
- Other applications are:
  - inductive invariant checking
  - removal of "dead" summands
  - enhance static analysis tools
  - Future: check user provided state mappings

Very Recent Developments

- Symbolic Model Checking on LPO [Groote, Willemse]
  - handles regular μ-calculus with data and quantifiers
  - applies directly to LPOs (possibly infinite state spaces)
  - transformed to Boolean equation systems with data parameters [Groote, Mateescu]
  - solved by equational binary decision diagrams
- Abstract interpretation of LPO [Valero, JvdP]
  - based on abstraction of data domains.
  - results in a Modal LPO, containing may/must transitions.
  - yields under/over approximations, using 3-valued logic.
- Symmetry Reduction [van Langevelde]

Outline of our Verification Process



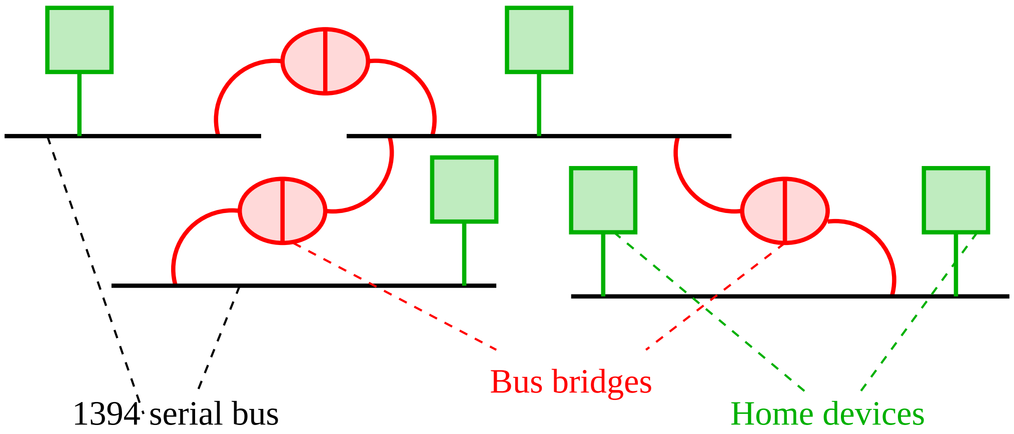
State Space Generation and Analysis

(this is only possible for finite state spaces)

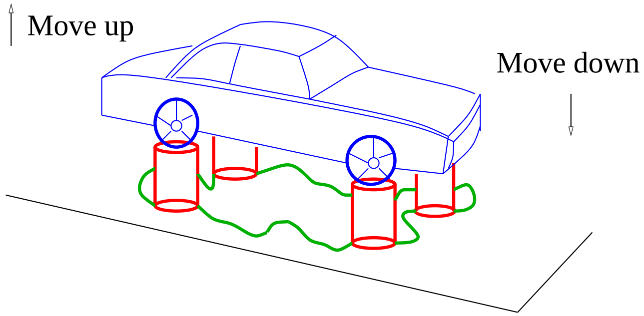
- Explicit LTS Generation from a linear process
  (narrowing-like technique to solve $\sum$ over infinite domains)
- Distributed implementations [Blom, Orzan]
  - state space generation (in files S_i, T_{ij})
  - strong bisimulation minimization
  - branching bisimulation minimization
- Open/Cæsar interface is implemented.
  - on-the-fly analysis of μCRL specs by CADP toolset
  - model checking, equivalence checking, visualization ...
- Visualization of state space of $> 10^6$ nodes [Groote, van Ham]

Protocols and Distributed Algorithms

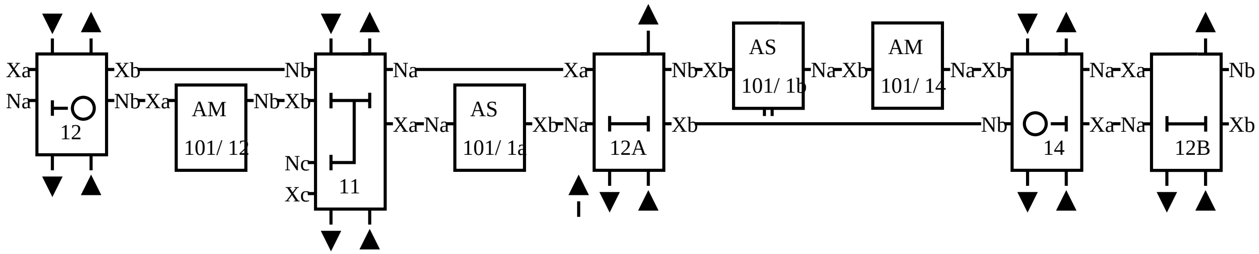
- Sliding Window Protocol
- Leader Election Protocol [Dolev,Klaw,Rodeh]
- Cache Coherence Protocol for Java Distributed Memory Model
- Failure recovery algorithms for Telecom [Arts, Benac Earle]
- IEEE 1394.1 Firewire Busbridges Standardization



Embedded Systems



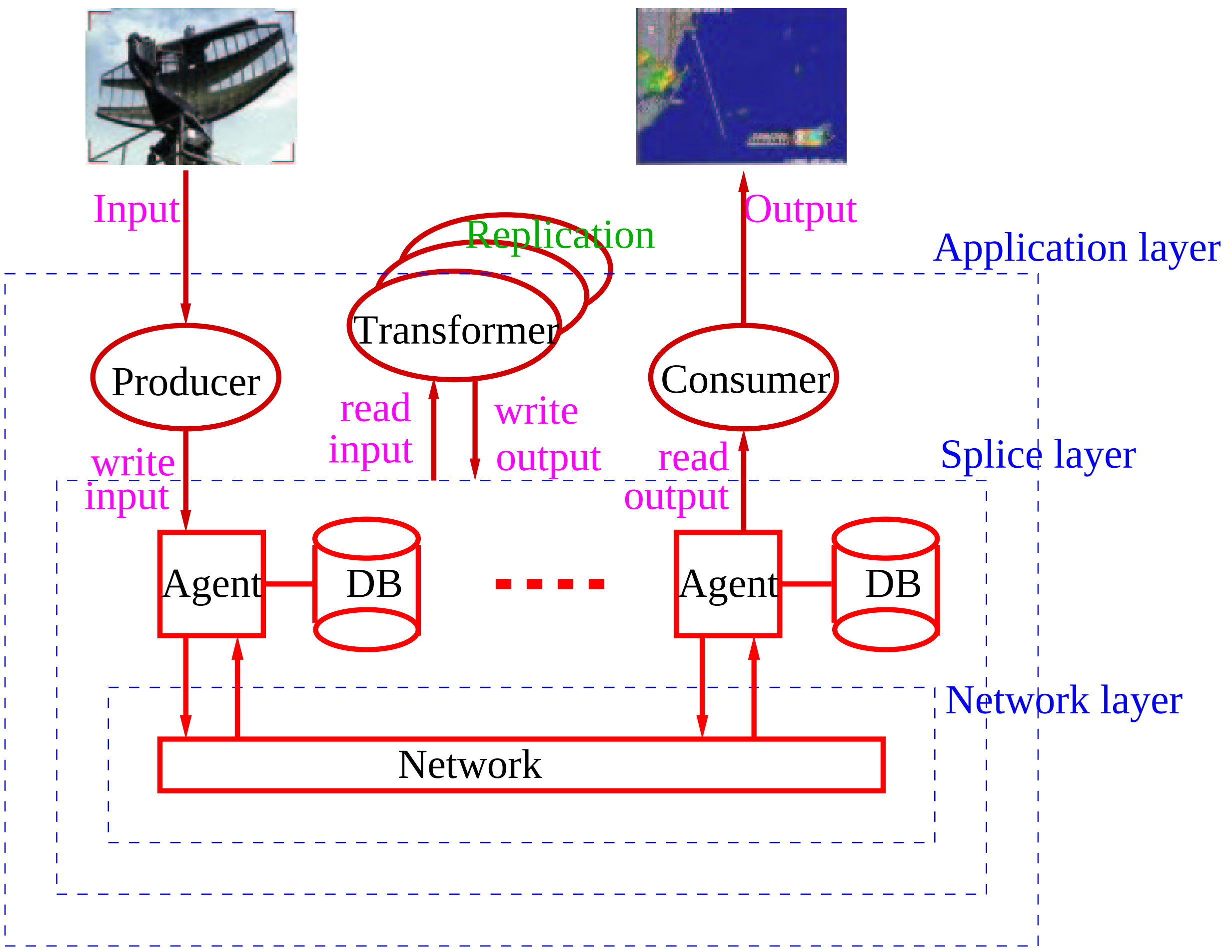
- Truck lift controllers built by Add-controls
- In-flight Data-acquisition Unit for Lynx helicopter [RNLN, NLR]
- Avionics Control Systems [Moscow State Univ., RedLab Ltd.]
- Safety of railroad tracks (Euris specifications)



Shared Dataspace Architectures

- JavaSpaces^{tm} Distributed Applications [Sun Microsystems]
  - read/write/take on a global shared object space
  - transactions, notification events, resource leasing
  - dining philosophers, termination detection, parallel summation
- Splice Coordination Architecture [Thales]
  - Real-time distributed databases with replicated data
  - Publish/subscribe mechanism for loosely coupled components
  - Verification question: transparent replication of software components

Replication in Splice



OVERVIEW

- Introduction
- Symbolic verification
  - Static Analysis yields state mappings
  - Confluence for state space reduction
  - Symbolic Model Checking
- Explicit state verification
  - Distributed implementation
  - On-the-fly via Open/Cæsar
  - Visualization
- Some Applications

Conclusion

- LPO format contributes to modularity of the tool set
- Methodological integration of symbolic, on-the-fly and explicit state analysis
- Combination of interactive (PVS) and automated theorem proving (EQ-BDDs), symbolic and explicit state model checking.
- Meta-theory is completely verified in PVS
- In principle, an individual verification in the tool set could be mapped onto PVS, for a "second opinion"