Partial Model Checking using Networks of Labelled Transition Systems and Boolean Equation Systems

Frédéric Lang and Radu Mateescu

INRIA and LIG / CONVECS

http://convecs.inria.fr

Motivation

- **Model-checking**
  - **Network of (untimed) asynchronous communicating processes** $P_1 \,||\, \ldots \,||\, P_n$ (e.g., process algebra)
  - **Modal mu-calculus** formula ϕ
- **Explicit state techniques:** LTS (*Labeled Transition System*) exploration
- **Compositional verification**
  - *Divide-and-conquer* to palliate state explosion
  - Exploit the compositionality of parallel composition semantics
  - Tools for compositional verification are available in the **CADP** toolbox (http://cadp.inria.fr)

Compositional verification in CADP

Compositional LTS generation

[Graf-Steffen-90, Tai-Koppol-93, Cheung-Kramer-93, Krimm-Mounier-97, …]

- Generate a **reduced LTS** incrementally
  - Generate individual process LTSs
  - Alternate **composition** of a subset of the LTSs (product) with **hiding** and **reduction** modulo an equivalence relation (strong, branching, safety, trace, weak trace, …)
  - Possibly use interface constraints to restrict intermediate LTSs
- Then check ϕ on the reduced LTS

CADP tools for compositional verification

- Composition of LTSs: **EXP.OPEN**
  - **Rich language**: **parallel composition** (CCS, CSP, μCRL, LOTOS, E-LOTOS, LNT, etc., incl. *m* among *n* and synchronisation vectors) **+ generalized label hiding, renaming, and cutting**
  - **Internal representation**: **Network of LTSs** (≈ sync. vectors)
  - **C code generation** (initial state, successor function, …) for on-the-fly verification (OPEN/CAESAR implicit LTS)
- LTS generation with interface constraints: **PROJECTOR**
- LTS reduction: **BCG_MIN** and **REDUCTOR**
- Modal mu-calculus verification using a BES (*Boolean Equation System*): **EVALUATOR**
- Scripting and verification strategies: **SVL**

Alternative compositional approach

(not available in CADP)

Partial model checking [Andersen-95]

- Check formula ϕ on $P_1 \mid\mid \ldots \mid\mid P_n$ **incrementally**:
  1. Compute a formula $\phi \mathbin{//} P_1$ called **quotient** of ϕ by P_1
  2. Simplify $\phi \mathbin{//} P_1$ to reduce its size
  3. If n > 1 then check $\phi \mathbin{//} P_1$ on $P_2 \mid\mid \ldots \mid\mid P_n$ (back to step 1)
- **Andersen-95: Modal mu-calculus** and LTSs composed using **CCS parallel composition and restriction**
- Several extensions followed (state based, timed, synchronous, etc.) [Larsen-Peterson-Yi-95, Bodentien-et-al-99, Cassez-Laroussinie-00, Martinelli-03, Basu-Ramakrishnan-03, …]

This talk

- **Aim**: Implement partial model checking for **Networks of LTSs** efficiently
- **Contributions**
  - **Generalise quotienting** to Networks of LTSs
  - **Reformulate quotienting** as a synchronous product (another Network of LTSs) between a process LTS and an LTS representing the formula (**formula graph**)
  - **Reformulate formula simplification** as a combination of LTS reductions and partial evaluation of the formula graph using a BES
  - **Prototype implementation using CADP** and **case-study**

The modal mu-calculus

- **Syntax**: $\phi ::= \mathbf{ff} \mid \phi_1 \vee \phi_2 \mid \langle a \rangle\, \phi_0 \mid \mu X.\phi_0 \mid X$

  $\mid \quad \mathbf{tt} \mid \phi_1 \wedge \phi_2 \mid [a]\, \phi_0 \mid \nu X.\phi_0 \mid \neg\phi_0$

  + *Syntactic monotonicity*: even number of negations between a variable and its binder

- **Elimination of negations**

  $\neg\mathbf{ff} = \mathbf{tt}$ $\qquad \neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$

  $\neg\langle a \rangle\, \phi_0 = [a]\, \neg\phi_0$ $\qquad \neg\mu X.\phi_0 = \nu X.\neg\phi_0[\neg X/X]$ $\qquad$ ...

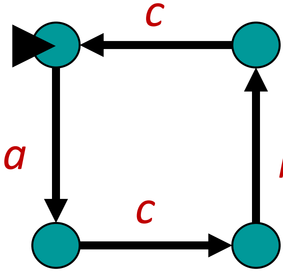
- **Alternation**
  - Maximum number of sign (μ or ν) switches between a variable and its binder
  - **Example formula of alternation 2**: $\mu X.\nu Y.(\langle a \rangle\, X \vee [b]\, Y)$

Networks of LTSs

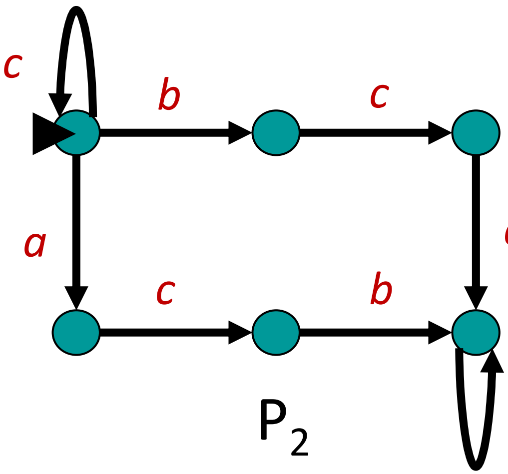
- Inspired by MEC and FC2
- Tuple $((P_1, ..., P_n), V)$ where:
  - $P_1, ..., P_n$ = LTSs (**of individual processes**)
  - V = set of **synchronization rules** $(a_1, ..., a_n) \rightarrow a_0$ where
    - each a_i ($i \in 1..n$) is either **a label** (action) or **the symbol** $\bullet$ (inaction)
    - a_0 is **a label** (resulting action)
- **Operational semantics:** LTS written **lts** $((P_1, ..., P_n), V)$
  - **State =** vector $(s_1, ..., s_n)$ of individual LTS states
  - $(s_1, ..., s_n) \xrightarrow{a_0} (s'_1, ..., s'_n)$ iff $(a_1, ..., a_n) \rightarrow a_0 \in V$, and
    - $s_i \xrightarrow{a_i} s'_i$ (for each $i \in 1..n$ s.t. $a_i \neq \bullet$), and
    - $s_i = s'_i$ (for each $i \in 1..n$ s.t. $a_i = \bullet$)

Example

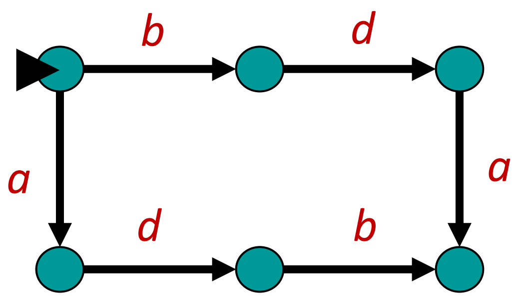
- $N = \left((P_1, P_2, P_3), \left\{ \begin{array}{lll} (a, a, \bullet) \rightarrow a, & (a, \bullet, a) \rightarrow a, & (b, b, b) \rightarrow b, \\ (c, c, \bullet) \rightarrow \tau, & (\bullet, \bullet, d) \rightarrow d & \end{array} \right\} \right)$



P_1

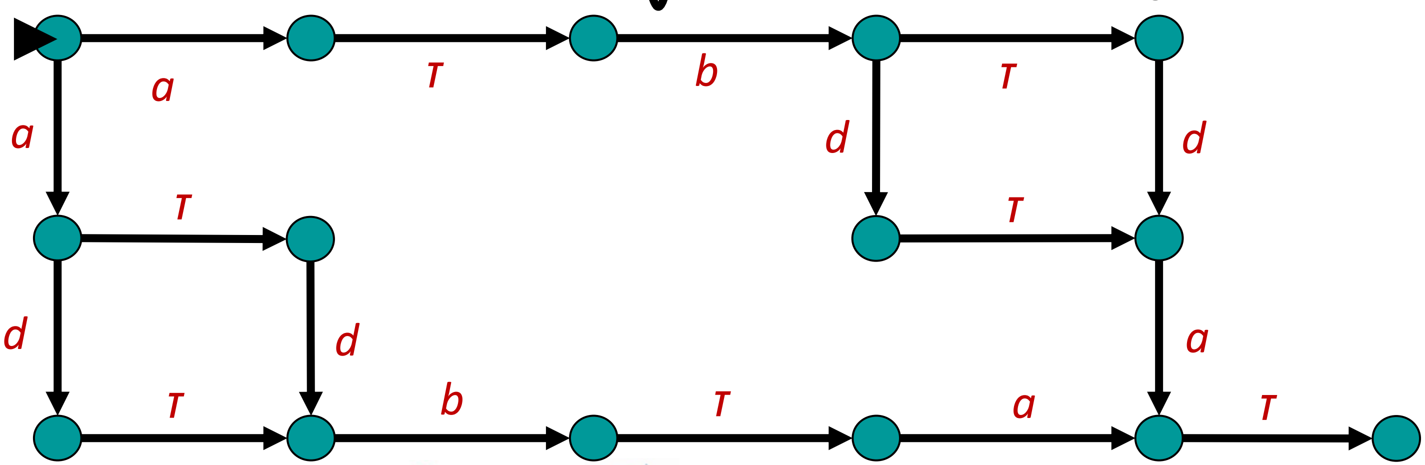


P_2



P_3

- **lts (N)** =



Network compositionality

- Given a network $N = ((P_1, \ldots, P_n), V)$ and $i \in 1..n$ one can automatically build
  - a network $N_{\backslash i}$ consisting of the composition of all P_j but P_i and
  - a new set of rules V'

  such that $lts\ (N) = lts\ ((P_i, lts\ (N_{\backslash i})), V')$

  (generalisable to any subset $I \subseteq 1..n$)

- Standard equivalence relations are **congruences** for networks (strong, observational, branching, safety, trace, weak trace, ...), provided hidden labels are neither renamed, nor synchronised, nor cut

Example

- $N = \left((P_1, P_2, P_3), \left\{ \begin{array}{lll} (a, a, \bullet) \rightarrow a, & (a, \bullet, a) \rightarrow a, & (b, b, b) \rightarrow b, \\ (c, c, \bullet) \rightarrow \tau, & (\bullet, \bullet, d) \rightarrow d & \end{array} \right\} \right)$

- $N_{\backslash 3} = \left((P_1, P_2), \left\{ \begin{array}{ll} (a, a) \rightarrow a, & (a, \bullet) \rightarrow \alpha_a, \\ (b, b) \rightarrow \alpha_b, & (c, c) \rightarrow \tau \end{array} \right\} \right)$

- $V' = \left\{ \begin{array}{ll} (a, \bullet) \rightarrow a, & (\alpha_a, a) \rightarrow a, \\ (\alpha_b, b) \rightarrow b, & (\bullet, d) \rightarrow d \end{array} \right\}$

α_a, α_b = new intermediate labels (glue)

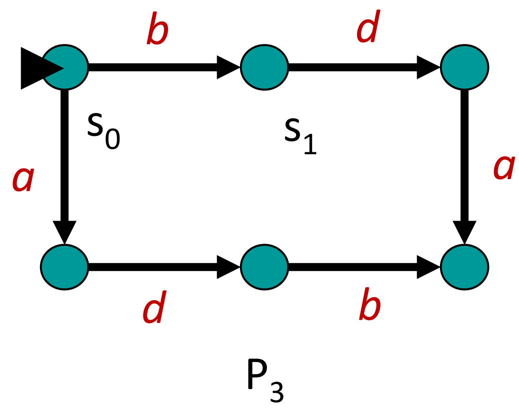
Quotienting for networks

- Given **$N = ((P_1, \ldots, P_n), V)$** and **$i \in 1..n$**, the **quotient of ϕ by P_i** is written **$\phi \mathbin{//} P_i$**
- **ϕ** is **true** on **N** iff **$\phi \mathbin{//} P_i$** is **true** on **$N_{\backslash i}$**
- Quotient introduces new variables of the form **X_s** where **X** is a variable of **ϕ** and **s** a state of **P_i** (product)

  Intuitively: **X** is **true** on **N** iff **X_s** is **true** on **$N_{\backslash i}$**, when **P_i** is in state **s**
- Quotient progressively eliminates modalities
- (technical details in paper)

Example

- $N = \left((P_1, P_2, P_3), \left\{ \begin{array}{lll} (a, a, \bullet) \rightarrow a, & (a, \bullet, a) \rightarrow a, & (b, b, b) \rightarrow b, \\ (c, c, \bullet) \rightarrow \tau, & (\bullet, \bullet, d) \rightarrow d & \end{array} \right\} \right)$



- $\phi = \mu X. \langle a \rangle\, \mathbf{tt} \vee \langle b \rangle\, X$
  (a sequence of *b* leads to an *a*)
- $\phi \,//\, P_3 =$

  $\mu X_{s0}. \langle a \rangle\, \mathbf{tt} \vee \langle \alpha_a \rangle\, \mathbf{tt} \vee \langle \alpha_b \rangle\, \mu X_{s1}. \langle a \rangle\, \mathbf{tt} \vee \mathbf{ff}$

  (to be checked on $N_{\backslash 3}$)

Implementing the quotient

- Formulas are potentially very large
- Trees and pointers should be avoided
  - Waste of memory
  - Slow computation
- We use the **similarity between quotienting and synchronous product**:
  - Turn the formula to **disjunctive form**
  - **Encode it as an LTS**
  - Implement **quotienting as a product** using a network of LTS

LTS encoding of the formula

- **Assumption**: formula ϕ is in disjunctive form (with negations)
- LTS written **enc (ϕ)** and called **formula graph**
  - **State**: a subformula of ϕ
  - **Label**: a mu-calculus operator
- **Transition relation**

$X \xrightarrow{\vee} \phi[X]$ $\qquad$ $\neg\phi_0 \xrightarrow{\neg} \phi_0$

$\langle a \rangle\, \phi_0 \xrightarrow{\langle a \rangle} \phi_0$ $\qquad$ $\mu X.\, \phi_0 \xrightarrow{\mu} \phi_0$

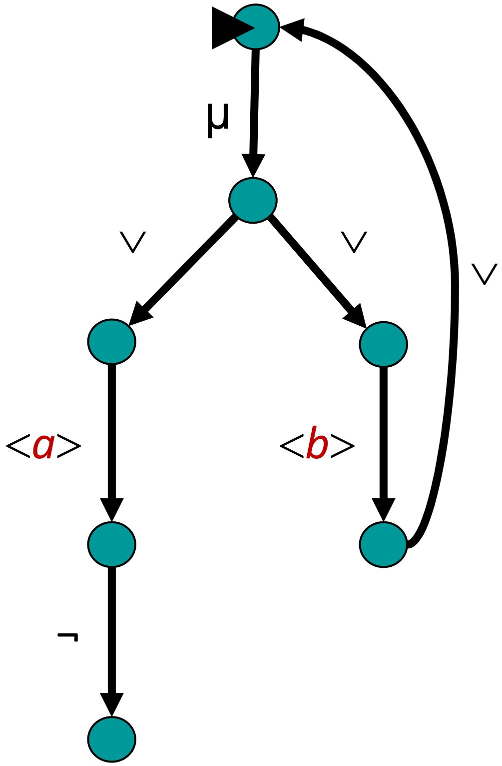
$\phi_1 \vee \phi_2 \xrightarrow{\vee} \phi_1$ $\qquad$ $\phi_1 \vee \phi_2 \xrightarrow{\vee} \phi_2$

(**ff** is a deadlock state : empty disjunction)

Example

- **Formula**: μX.<*a*> ¬**ff** ∨ <*b*> X

- **Formula graph**:



Quotienting using a network

- **Individual processes**: **enc (ϕ)** and $\mathbf{P_i}$
- **Synchronisation rules**:

  synchronise modalities on actions to which $\mathbf{P_i}$ contributes actively

$\{ (\neg, \bullet) \rightarrow \neg, \quad (\vee, \bullet) \rightarrow \vee, \quad (\mu, \bullet) \rightarrow \mu \} \;\cup$

$\{ (\langle a_0 \rangle, \bullet) \rightarrow \langle a_0 \rangle \quad | \; (a_1, \ldots, a_n) \rightarrow a_0 \in V \wedge a_i = \bullet \} \cup$

$\{ (\langle a_0 \rangle, a_i) \rightarrow \langle \alpha \rangle \quad | \; (a_1, \ldots, a_n) \rightarrow a_0 \in V \wedge a_i \neq \bullet \wedge (\exists j \in 1..n \setminus \{i\}) \; a_j \neq \bullet \} \cup$

$\{ (\langle a_0 \rangle, a_i) \rightarrow \vee \quad | \; (a_1, \ldots, a_n) \rightarrow a_0 \in V \wedge a_i \neq \bullet \wedge (\forall j \in 1..n \setminus \{i\}) \; a_j = \bullet \}$

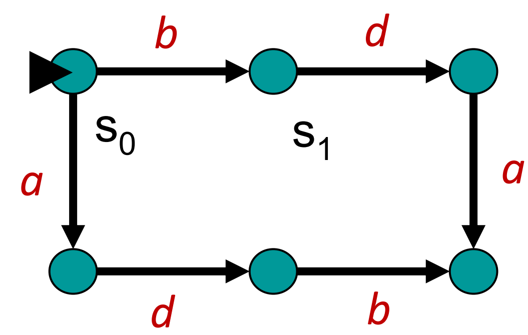
The glue α associated to $a_1, \ldots, a_n \rightarrow a_0$

- The LTS of this network encodes the formula graph of the quotient

Example (1/2)

- $N = \left((P_1, P_2, P_3), \left\{ \begin{array}{lll} (a, a, \bullet) \to a, & (a, \bullet, a) \to a, & (b, b, b) \to b, \\ (c, c, \bullet) \to \tau, & (\bullet, \bullet, d) \to d & \end{array} \right\} \right)$

- $\phi = \mu X. \langle a \rangle\, \mathbf{tt} \vee \langle b \rangle\, X$



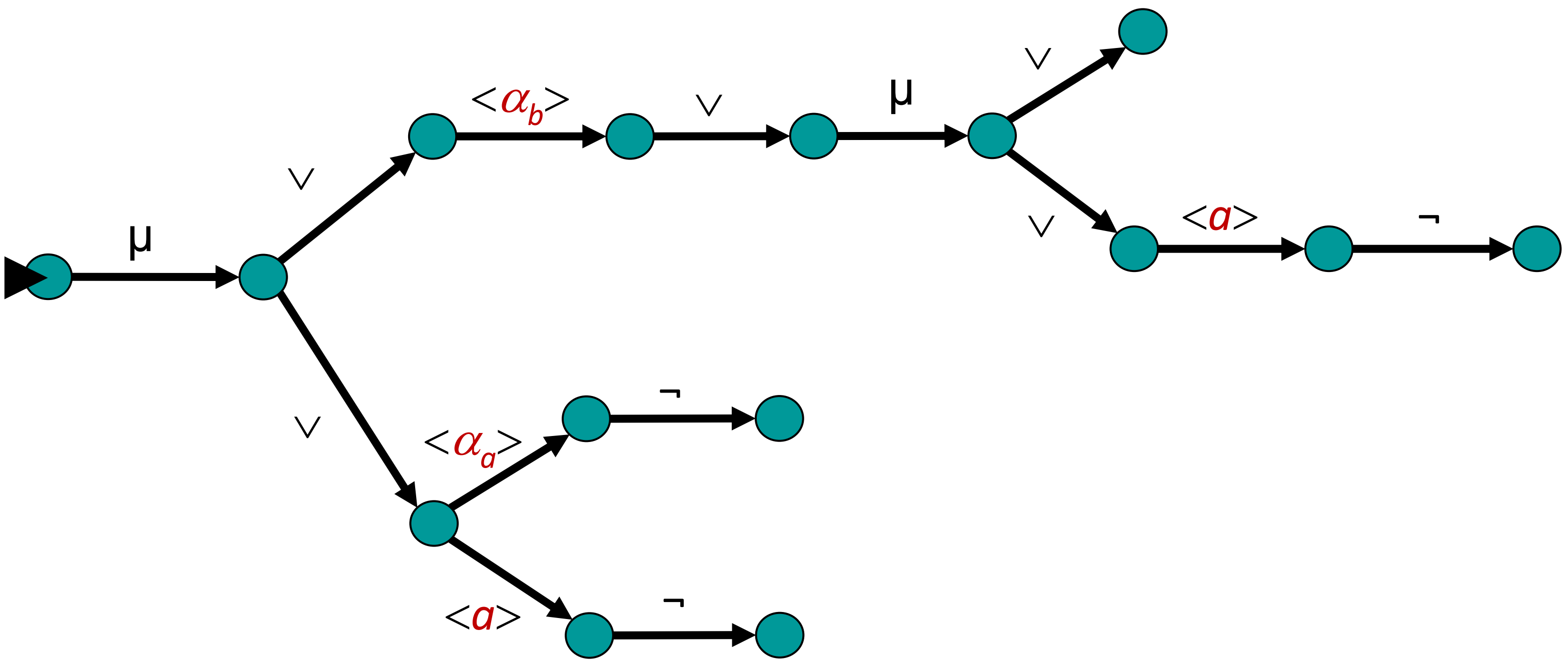
P3

- **ϕ // P_3**
  is implemented by the network
  (enc (ϕ), P_3), with synchronisation rules

$$\left\{ \begin{array}{lll} (\neg, \bullet) \to \neg, & (\vee, \bullet) \to \vee, & (\mu, \bullet) \to \mu, \\ (\langle a \rangle, \bullet) \to \langle a \rangle, & (\langle a \rangle, a) \to \langle \alpha_a \rangle, & (\langle b \rangle, b) \to \langle \alpha_b \rangle \end{array} \right\}$$

Example (2/2)

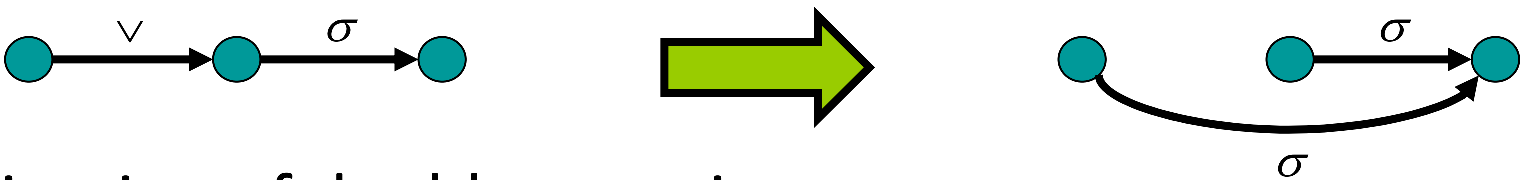
- Resulting formula graph:



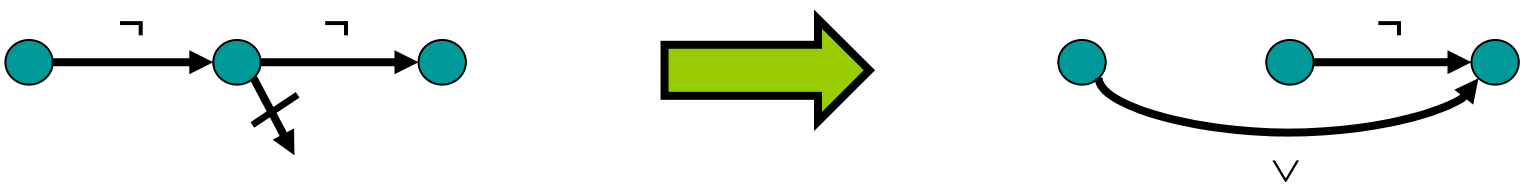
(encodes $\mu X_{s0}.<a>\ \mathbf{tt} \vee <\alpha_a>\ \mathbf{tt} \vee <\alpha_b>\ \mu X_{s1}.<a>\ \mathbf{tt} \vee \mathbf{ff}$)

Formula simplification (1/2)

- Applied directly to formula graphs
- Elimination of ∨-transitions (hiding and reduction modulo ***τ*.a* equivalence**)



- Elimination of double negations



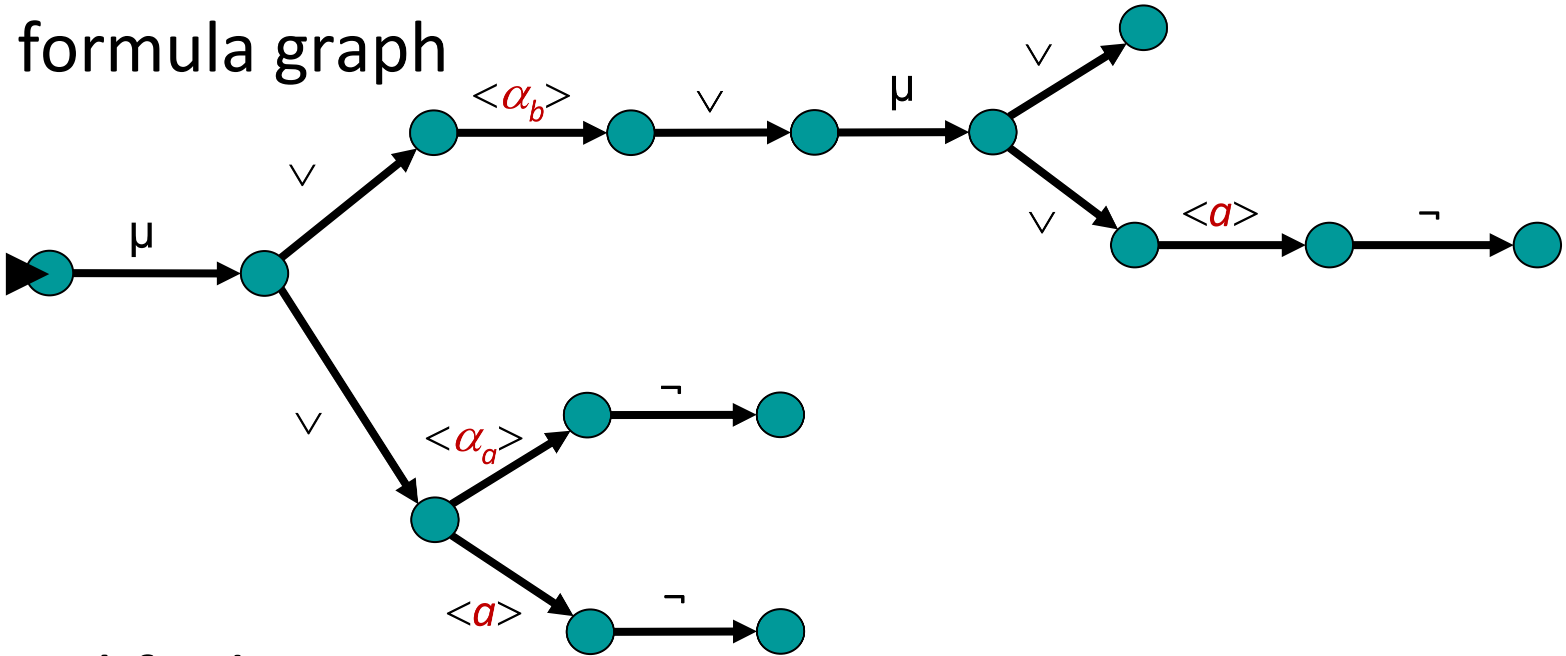
- Elimination of useless μ–transitions (sufficient conditions)

Formula simplification (2/2)

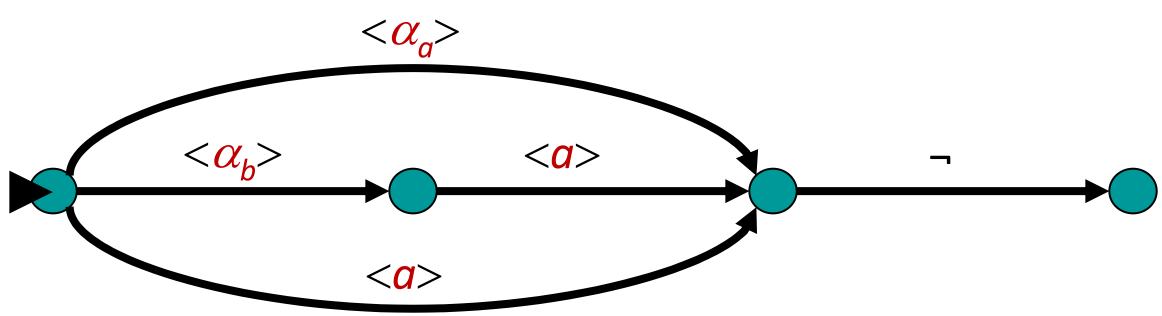
- Partial evaluation of states
  - Identify states that denote constant sub-formulas (e.g. $\mu X.\langle a \rangle \langle b \rangle X \vee \langle b \rangle \mathbf{ff} = \mathbf{ff}$) using a BES
  - Simplify the formula graph accordingly (constant propagation)
  - BES evaluates every formula graph without modalities to a constant
- Sharing of identical sub-formulas
  - **By strong bisimulation reduction** (requires tagging μ transitions with block numbers)
  - Implements similar reductions found in [Andersen-95, Basu-Ramakrishnan-03]

Example

The formula graph



is simplified into



Prototype implementation using CADP

- Restricted to **alternation-free** modal mu-calculus
- Reuse existing tools (less than 2000 new lines of code)
  - Minor extensions to **EXP.OPEN** and **EVALUATOR**
  - **BCG_LABELS**/**REDUCTOR** implement elimination of $\vee$-transitions
  - **BCG_MIN** implements sharing of identical sub-formulas
  - **New prototype tool** (C, 1000 lines): other simplification rules; uses the **CÆSAR_SOLVE** library for solving alternation-free BES
  - **New script** (Bourne shell, 300 lines): invocation of tools

Case study

- Application to a **case-study in avionics**: communication protocol based on TFTP/UDP [Garavel-Thivolle-09]
- Two instances of TFTP connected via UDP using a FIFO buffer
- Five scenarios, depending whether each instance can read and/or write a file
- 28 alternation-free mu-calculus properties checked
- Comparison between memory peaks: on-the-fly (**EVALUATOR**) vs partial model checking

1

Results

Prop	Scenario A 1,963 ks fly	pmc	Scenario B 867 ks fly	pmc	Scenario C 35,024 ks fly	pmc	Scenario D 40,856 ks fly	pmc	Scenario E 19,436 ks fly	pmc
$A01$	199	6	89	6	2,947	24	3,351	27	1,530	23
$A02$	207	6	93	6	3,156	25	3,631	28	1,612	10
$A03$	182	6	80	6	2,737	6	3,162	6	1,386	6
$A04$	199	6	89	6	2,947	6	3,351	29	1,530	7
$A05$	10	6	7	6	7	6	7	6	10	10
$A06$	187	6	85	6	2,808	6	3,249	7	1,428	6
$A07$	187	6	85	6	2,808	6	3,249	6	1,428	6
$A08$	186	6	80	6	2,745	6	3,170	6	1,390	6
$A09a$							3,290	28	1,488	6
$A09b$					2,955	6				
$A10$					3,354	6			1,674	6
$A11$					3,206	6	4,444	7	1,711	6
$A12$					620	⋆	133	⋆	101	⋆
$A13$							4,499	⋆	2,094	⋆
$A14$	267	6			3,988	23			2,107	15
$A15$			118	15	521	⋆	156	⋆	1,524	59
$A16$									186	8
$A17$					667	⋆	569	2,702		
$A18$			85	6	476	11	255	6	1,391	6
$A19$			207	6	6,352	90	8,753	13	3,104	55
$A20$	31	9			837	21			261	25
$A21$	374	6			4,958	25			2,817	25
$A22$			35	7			427	1,271	191	650
$A23$			170	6			6,909	9	3,039	40
$A24$	41	9			427	1,786				
$A25$	391	6			5,480	40				
$A26$	195	6			2,857	15			1,477	10
$A27$	228	6			3,534	6			1,871	6
$A28$			102	6	3,654	22	4,032	6	1,821	6

« = explosion

Best ratio = 767

Conclusions

- **Generalization of partial model checking to networks**: enables application to various models (CCS, CSP, mCRL, LOTOS, *m* among *n*, synchronization vectors, ...)
- Original **graph encoding of the formula** (no need to decompile)
- **Lightweight (prototype) implementation** for alternation-free formulas
- Case study shows that **memory peak may be reduced by several orders of magnitude**
- Compositional LTS generation and partial model checking are complementary

Future work

- **Improve** the simplification strategy (e.g., order of rule applications)
- Generate a **verification diagnostic**
- Combine with **other compositional techniques**: interface constraints, tau-confluence, ...
- Consider logic with data
- Extend implementation to some **mu-calculus formulas of alternation 2** (e.g., infinite repetition of regular sequences *a**.*b*)
- Apply to **equivalence checking**, using characteristic formulas (alternation 2)