Compositional Verification of Concurrent Systems by Combining Bisimulations

Frédéric Lang, Radu Mateescu

Inria, LIG, Université Grenoble Alpes (Grenoble, France)

http://convecs.inria.fr

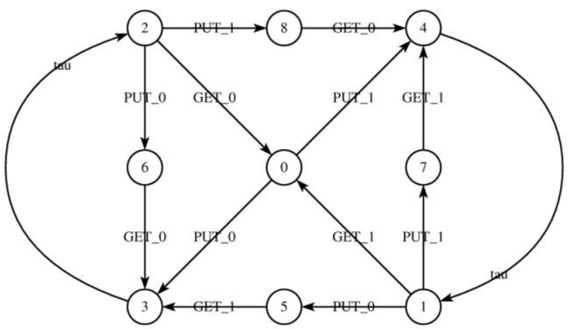
Franco Mazzanti

ISTI-CNR (Pisa, Italy)

http://fmt.isti.cnr.it

Motivation

- Explicit-state model checking of concurrent system
  - Asynchronous model $P_1||...||P_n$
  - LTS (Labelled Transition System) semantics
  - Action-based modal μ-calculus property φ



- Problem: state-space explosion
- Compositional verification can circumvent explosion
  - Apply to $P_1||...||P_n$ LTS reductions that preserve φ
  - Mateescu & Wijs (2014) define φ-preserving reductions: action hiding and quotient wrt. strong **or** divbranching bisimulation
  - Applied sucessfully to many case studies
- We refine the approach by combining both bisimulations

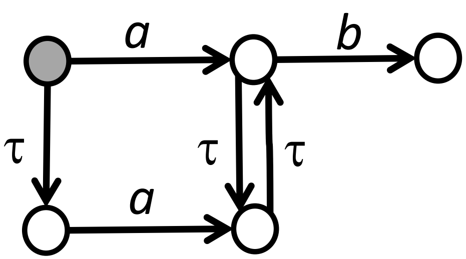
Outline

1. Background
2. The mono-bisimulation approach of Mateescu & Wijs
3. Our refined approach combining bisimulations
4. Applications and experimental results
5. Conclusion

1. Background

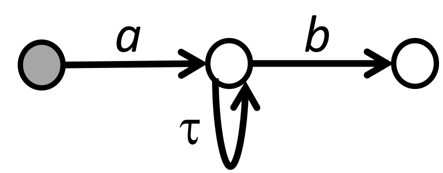
Divbranching bisimulation (van Glabbeek & Weijland, 1996)

- Short for divergence-preserving branching bisimulation
- Weaker than strong bisimulation:
  special treatment of invisible (τ) transitions
- Preserves choices of visible actions and infinite sequences of τ-transitions
- **Example**:



is divbranching bisimilar to:

$\sim_{db}$



- Like strong, divbranching is a congruence for ||
  $\Rightarrow$ reduction applicable compositionally

Compositional reduction

- Alternation between n-ary compositions/reductions ($rcomp_n$), until all processes are aggregated
- Many strategies are possible
  Example: $P_1 || P_2 || P_3$
  - $rcomp_3$ ($rcomp_1$ (P_1), $rcomp_1$ (P_2), $rcomp_1$ (P_3)))),
  - $rcomp_2$ ($rcomp_2$ ($rcomp_1$ (P_1), $rcomp_1$ (P_2))), $rcomp_1$ (P_3))), ...
- LTS constrain each others by synchronization
- Aim: maintain the "largest intermediate LTS size" small
- No optimal strategy available: heuristic is needed
- We use smart reduction (Crouzen & Lang, 2011)

The action-based modal mu-calculus L_μ (Kozen, 1983)

- Temporal logic interpreted over LTS
- Action formulas:

$$\alpha ::= a \mid \textbf{false} \mid \neg\alpha \mid \alpha_1 \vee \alpha_2$$

Notation: $[[\alpha]]$ set of actions satisfying α

- State formulas:

$$\varphi ::= \textbf{false} \mid \neg\varphi_0 \mid \langle\alpha\rangle\, \varphi_0 \mid \varphi_1 \vee \varphi_2 \mid \mu X.\ \varphi_0 \mid X$$

Notation: $P \models \varphi$ LTS P satisfies φ

- Derived operators: $\textbf{true} \mid [\alpha]\, \varphi_0 \mid \varphi_1 \wedge \varphi_2 \mid \nu X.\ \varphi_0$
- Subsumes (action-based) CTL, ACTL, PDL, PDL-Δ, etc.

2. The mono-bisimulation approach of Mateescu & Wijs

The mono-bisimulation approach

- Find actions $a_1, \ldots, a_m$ and relation R among **divbranching** and **strong** bisimulations, such that φ can be verified on
  R **reduction of hide** $a_1, \ldots, a_m$ **in** $P_1 || \ldots || P_n$
  instead of
  $P_1 || \ldots || P_n$
- Procedure $H(\varphi)$ computes the largest set $a_1, \ldots, a_m$
  $H(\varphi) = \cap\, h(\alpha)$ $\quad$ $h(\alpha) =$ **if** $\tau \in [[\alpha]]$ **then** $[[\alpha]]$ **else** **all but** $[[\alpha]]$
  Example: $H(\mu X. \langle a \rangle\, \mathbf{true} \vee \langle \mathbf{true} \rangle\, X) =$ **all but** a
- A fragment $L_{\mu\text{-db}}$ of L_μ is defined such that:
  - R is **divbranching** if $\varphi \in L_{\mu\text{-db}}$
  - R is **strong** otherwise (less reduction)

The fragment $L_{\mu\text{-db}}$

Strong modalities $<\alpha>\ \varphi$ are replaced by weak modalities:

$\varphi ::= \textbf{false} \mid \neg\varphi_0 \mid \varphi_1 \vee \varphi_2 \mid \mu X.\ \varphi_0 \mid X$

$\mid <(\varphi_1?.\ \alpha_\tau)^*>\ \varphi_2$

there is a sequence of actions satisfying α_τ that traverses only states satisfying φ_1 and ends in a state satisfying φ_2

$\mid <(\varphi_1?.\ \alpha_\tau)^*.\ \varphi_1?.\ \alpha_a>\ \varphi_2$

there is a sequence of actions satisfying α_τ that traverses only states satisfying φ_1 and ends in a state satisfying $<\alpha_a>\ \varphi_2$

$\mid <\varphi_1?.\ \alpha_\tau>\ @$

there is an infinite sequence of actions satisfying α_τ that traverses only states satisfying φ_1

where $\tau \in [[\alpha_\tau]],\ \tau \notin [[\alpha_a]]$

Expressiveness of $L_{\mu\text{-db}}$

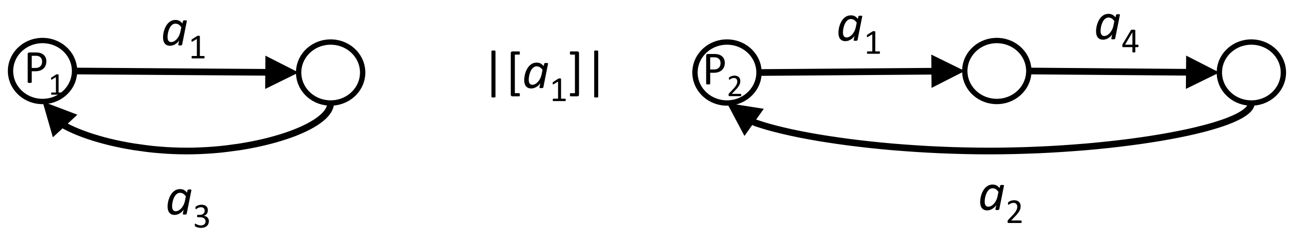
Translation to $L_{\mu\text{-db}}$ is possible for the following operators:

- PDL-Δ: $\langle\alpha_\tau{}^*\rangle\ \varphi_0$ $\langle\alpha_\tau{}^*.\alpha_a\rangle\ \varphi$ $\langle\alpha_\tau\rangle\ @$
- ACTL: $\mathbf{A}\ (\varphi_1\ {}_{\alpha 1}\mathbf{U}\ \varphi_2)$ $\mathbf{A}\ (\varphi_1\ {}_{\alpha 1}\mathbf{U}_{\alpha 2}\ \varphi_2)$ $\mathbf{AG}_{\alpha 0}\ (\varphi_0)$
  $\mathbf{E}\ (\varphi_1\ {}_{\alpha 1}\mathbf{U}\ \varphi_2)$ $\mathbf{E}\ (\varphi_1\ {}_{\alpha 1}\mathbf{U}_{\alpha 2}\ \varphi_2)$ $\mathbf{EF}_{\alpha 0}\ (\varphi_0)$
  (μ-ACTL\X is slightly less expressive than $L_{\mu\text{-db}}$)
- CTL: $\mathbf{A}\ (\varphi_1\ \mathbf{U}\ \varphi_2)$ $\mathbf{A}\ (\varphi_1\ \mathbf{W}\ \varphi_2)$ $\mathbf{AG}\ (\varphi_0)$ $\mathbf{AF}\ (\varphi_0)$
  $\mathbf{E}\ (\varphi_1\ \mathbf{U}\ \varphi_2)$ $\mathbf{E}\ (\varphi_1\ \mathbf{W}\ \varphi_2)$ $\mathbf{EF}\ (\varphi_0)$ $\mathbf{EG}\ (\varphi_0)$
  $\mathbf{A}\ (([\alpha_a]\ \varphi_1)\ \mathbf{U}\ \varphi_2)$ $\mathbf{A}\ (([\alpha_a]\ \varphi_1)\ \mathbf{W}\ \varphi_2)$
  $\mathbf{AG}\ (\varphi_1 \vee [\alpha_a]\ \varphi_2)$ $\mathbf{EF}\ (\varphi_1 \wedge \langle\alpha_a\rangle\ \varphi_2)$

  New result

where $\varphi_0, \varphi_1, \varphi_2 \in L_{\mu\text{-db}},\ \tau \in [[\alpha_\tau]],\ \tau \notin [[\alpha_a]]$

Compositional verification example



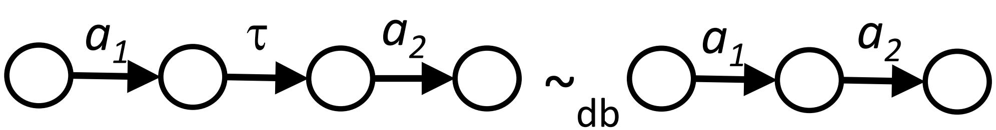
- φ_1 = **<true*.a_1> true** $\in L_{\mu\text{-db}}$
  smart divbranching reduction of
  hide all but a_1 **in** $(P_1 \,|[a_1]|\, P_2)$ **|=** φ_1
  $\Rightarrow$ Largest LTS: **3 states / 3 transitions** (P_2)

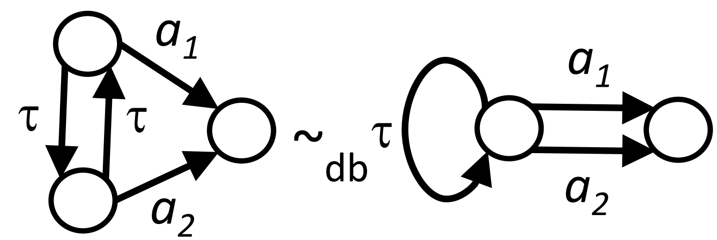
- φ_2 = **[true*.a_1.a_2] false** $\notin L_{\mu\text{-db}}$
  smart strong reduction of
  hide all but a_1, a_2 **in** $(P_1 \,|[a_1]|\, P_2)$ **|=** φ_2
  $\Rightarrow$ Largest LTS: **6 states / 8 transitions**

3. Our refined approach combining bisimulations

Principles

- Formulas may combine strong and weak modalities
  Examples: [**true***.a_1.a_2] **false**   <**true***> (<a_1> **true** $\wedge$ <a_2> **true**)
- Such formulas are not preserved by divbranching





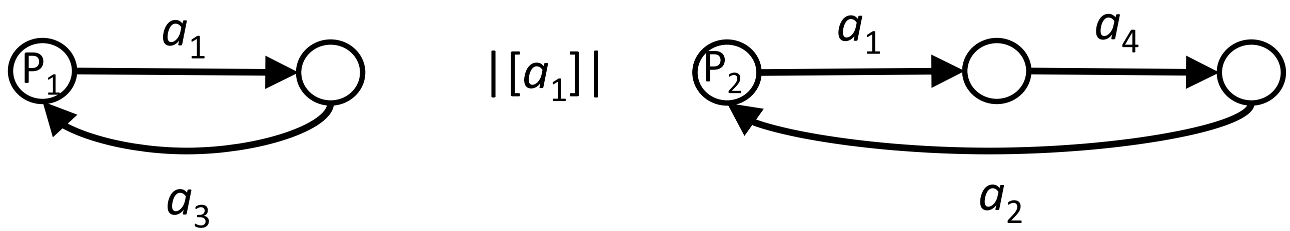
- Theorem: If no action of some P_i is matched by a strong modality then P_i can be reduced for divbranching
- We write $\varphi \in L_{\mu\text{-str}}(A_s)$ and call A_s the set of strong actions if all strong modalities of φ satisfy $[[\alpha]] \subseteq A_s$

  Examples: [**true***.a_1.a_2] **false** $\in L_{\mu\text{-str}}(\{a_2\})$ $L_{\mu\text{-db}} = L_{\mu\text{-str}}(\emptyset)$

New verification strategy

- Partitioning the set of processes
  - $\mathscr{P}_s$: processes among $P_1, \ldots, P_n$ containing strong actions
  - $\mathscr{P}_w = \{P_1, \ldots, P_n\} \setminus \mathscr{P}_s$: processes not containing strong actions
- Refactoring $P_1 || \ldots || P_n$ into $(||_{P_s \in \mathscr{P}_s} P_s) \; || \; (||_{P_w \in \mathscr{P}_w} P_w)$
- Reducing the sets of processes compositionally according to theorem:
  - Q = **smart divbranching reduction of** $(||_{P_w \in \mathscr{P}_w} P_w)$
  - Q' = **smart strong reduction of** $(||_{P_s \in \mathscr{P}_s} P_s) \; || \; Q$
- Finally checking **hide** H(φ) **in** Q' **|=** φ

Example



- φ_2 = **[true*.a_1.a_2] false** $\in L_{\mu\text{-str}}(\{a_2\})$
smart strong reduction of hide all but a_1, a_2 in
((**smart divbranching reduction of** -- $a_2 \notin P_1$
hide all but a_1 in P_1)
|[a_1]|
(**smart strong reduction of** -- $a_2 \in P_2$
hide all but a_1, a_2 in P_2)) **|=** φ_2
- $\Rightarrow$ Largest LTS: **3 states / 3 transitions**
instead of 6 states / 8 transitions

Extracting A_s from the formula

- Problem: Given $\varphi \in L_\mu$, how to infer A_s s.t. $\varphi \in L_{\mu\text{-str}}(A_s)$?
- Hard for arbitrary low-level L_μ formula
  - Need to prove that a strong modality can be turned to weak one
  - Analogy: prove that binary code implements function correctly
- Easier for higher-level logics (CTL, ACTL, PDL, PDL-Δ): Use knowledge of $L_{\mu\text{-db}}$ expressiveness (patterns)
  Example: **A** (([a] **false**) **U** <b> **true**) $\in L_{\mu\text{-str}}(\{b\})$
  because **A** (([α_a] φ_1) **U** φ_2) $\in L_{\mu\text{-db}}$ and <b> **true** $\in L_{\mu\text{-str}}(\{b\})$
- A_s can be safely over-approximated, but smaller is better
- Automatic extraction of minimal A_s faces issues

Issues with extracting a minimal A_s

- Issue 1: It requires semantic reasoning
  - **Example**: in **AG** ($\langle a\rangle$ **true** $\Rightarrow [a]\ \varphi_0$), *a* seems to be strong
    In fact it is not as this formula is equivalent to **AG** ($[a]\ \varphi_0$)
  - L_μ satisfiability checking (EXPTIME) might be necessary
- Issue 2: minimal A_s is not unique
  - **Example**: $\varphi = \langle$**true**$^*\rangle(\langle a_1\rangle$ **true** $\wedge \langle a_2\rangle$ **true**$) \notin L_{\mu\text{-str}}(\emptyset)$
    $\varphi \equiv \langle(\langle a_1\rangle$ **true**?.**true**$)^*.\langle a_1\rangle$ **true**?.$a_2\rangle$ **true** $\in L_{\mu\text{-str}}(\{a_1\})$
    $\varphi \equiv \langle(\langle a_2\rangle$ **true**?.**true**$)^*.\langle a_2\rangle$ **true**?.$a_1\rangle$ **true** $\in L_{\mu\text{-str}}(\{a_2\})$
  - a_1 and a_2 can be weak actions but not both simultaneously
  - Choosing one or the other may impact performance
- In general: rely on expertise, side proof needed

4. Applications and experimental results

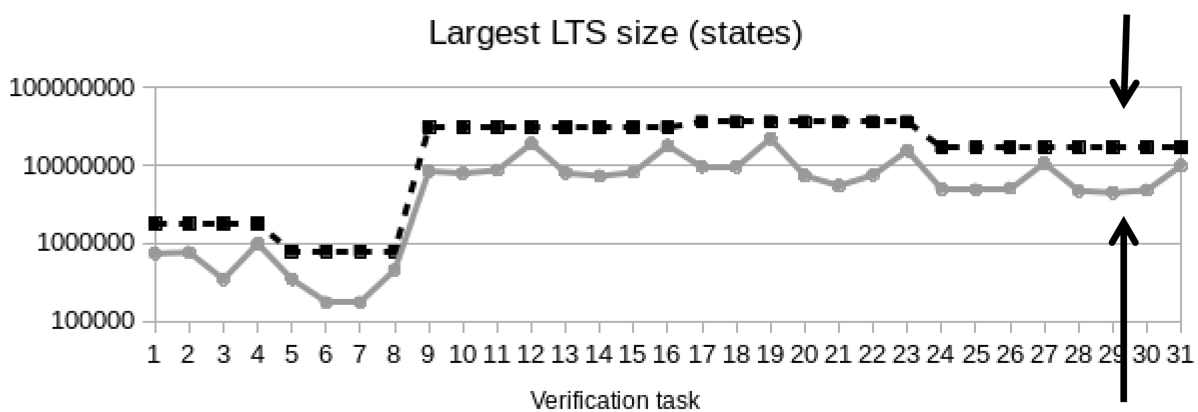
Implementation

- Approach implemented using CADP toolbox (cadp.inria.fr)
  - Formal verification of asynchronous concurrent systems
  - Toolbox developed since the late 80's (≈ 70 tools and libraries)
- Several software components used in this work:
  - LNT.OPEN/GENERATOR: compiling LNT processes to LTS
  - EXP.OPEN 2/GENERATOR: composing LTS in parallel
  - BCG_MIN 2: minimizing LTS for strong and divbranching
  - BCG_OPEN/EVALUATOR 4: model checking MCL temporal logic (regular alternation-free modal mu-calculus with data)
  - SVL: scripting, smart compositional verification heuristic
- Successful application to several examples

TFTP (Trivial File Transfer Protocol)

- Avionics case study (Garavel&Thivolle, 2009)
- 31 verification tasks involve properties that contain both weak and strong modalities
- Comparison with the mono-bisimulation approach
- Result: largest LTS up to 7 times smaller

mono-bisimulation



combined bisimulations

Gains in CPU time and memory peak are similar

RERS (Rigorous Evaluation of Reactive Systems)

- Verification competition http://rers-challenge.org
- RERS 2018 “parallel CTL” benchmark
  - 3 concurrent models (101...103) with 9 to 34 parallel processes
  - 9 properties – 3 per model (21..23)
- 7 properties combine weak and strong modalities
- Mono-bisimulation: explosion for 5 properties
- Combined bisimulations approach is successful
  - 4 properties from 5 to 10 min. and from 22 to 101 MB
  - 1 property: 42 min. and 1.6 GB

RERS - CTL example (103#23)

- **AG** (<A34> **true** ⇒ [A34] **A** ([A68] **false W** <A59> **true**)) checked on a composition of 34 processes (70 actions)
  - All but A34 , A59, A68 can be hidden (67 actions)
  - A34, A68 are weak, formula belongs to $L_{\mu\text{-str}}$({A59})
- Mono-bisimulation (strong) does not prevent explosion
  - Stopped after several hours
  - Largest LTS: ≥ **4.5 Giga states / 36 Giga transitions**
- Combining bisimulations is successful
  - Strong action in 7 proc. ⇒ 27 proc. reduced for divbranching
  - Result **true** after **< 10 min CPU**, using **35 MB** memory
  - Largest LTS: **122,292 states / 888,156 transitions**

5. Conclusion

- Improvement of property-preserving LTS reductions
  - New strategy combining bisimulations applicable to properties not preserved by divbranching bisimulation
  - Based on property analysis, classifying actions as weak or strong
  - Big LTS reductions wrt. mono-bisimulation
  - Proofs and examples available at doi.org/10.5281/zenodo.2634148
- Future work:
  - Automate A_s computation or automatically check user-given A_s
  - Automate composition refactoring $(||_{P_s \in \mathcal{P}_s} P_s) \;||\; (||_{P_w \in \mathcal{P}_w} P_w)$
  - Approach further refined $\Rightarrow$ gold medals won at RERS 2019