Compositional Verification in Action

Frédéric Lang

Inria Grenoble – LIG

Université Grenoble Alpes

http://convecs.inria.fr

joint work with

Hubert Garavel and **Laurent Mounier**

Introduction

- Goal: Formal verification of concurrent systems
  - *Action based models*
  - *Asynchronous concurrency*: interleaving & Hoare's rendezvous
  - *Enumerative techniques*: model checking, equivalence checking
- Generate a low-level model from a high-level description
- Compositional verification: "*divide and conquer*" approach to fight state explosion
  - Exploit the decomposition of the system into local processes
- This talk:
  - Basic compositional verification
  - Refined approach of Graf & Steffen (and Lüttgen)
  - Applications in the CADP toolbox

Six ingredients to verify a system (1-3)

1) Low-level model M

- State-transition formalism encoding the system's behaviour
- Examples: *labelled transition system*, *interactive Markov chain*

2) Parallel composition operator ||

- Returns the *composition* $M' = M_1 || \ldots || M_n$ of n *components*
- Complexity of M' = product of the complexities of $M_1, \ldots, M_n$

3) Equivalence relation $\approx \subseteq M \times M$

- Congruence for ||: $M_i \approx M_i' \Rightarrow M_1 || \ldots || M_n \approx M_1' || \ldots || M_n'$
- Examples: *strong bisimulation*, *branching bisimulation*, …

Six ingredients to verify a system (4-6)

4) Minimisation function min: $M \rightarrow M$

- Maps each model to an element of its equivalence class in $M/\approx$
- Minimizes some complexity criterion (e.g., state space size)
- $M_1 || \ldots || M_n \approx \min(M_1) || \ldots || \min(M_n)$

5) High-level language L

- Realistic systems cannot be described directly in M
- L also has concepts of *components* C and *parallel composition* $||$

6) Translation function $[[.]]: L \rightarrow M$

- Maps a system S into a low level model $[[S]]$
- Morphism for $||$: $[[C_1 || \ldots || C_n]] \approx [[C_1]] || \ldots || [[C_n]]$

Basic compositional verification

- Problem: generate a low level model for $S = C_1||\ldots||C_n$ where:
  - $[[S]]$ is excessively large (state explosion)
  - But $[[C_1]], \ldots, [[C_n]]$ are small enough to be generated
- Solution:
  Compute $\min([[C_1]])||\ldots||\min([[C_n]])$ instead of $[[S]]$
- Advocated in many research papers since end of the 80's
  - Functional verification setting: labelled transition systems
  - Performance evaluation setting: interactive Markov chains
- Efficiency is inversely proportional to the size of the largest intermediate model that is generated

This is more complex in practice…

- Problem: Some $[[C_i]]$ may be much larger than $[[S]]$
  - Cause: components are tightly synchronised and C_i's behaviour is constrained by other components
  - Examples: shared memories, hardware links, buses, …

- Solution: If S has a hierarchical structure, try different *strategies*
  - Compose / minimize different subsets of components

Compositional verification strategies

- Static strategies
  - min is applied to leaf components only, or
  - min is applied to every intermediate level in the hierarchy
- Dynamic strategies
  - Decide at each step which components to compose / minimize
  - Use heuristics (finding an optimal strategy is too complex)
- Example: *smart reduction* (Crouzen & Lang, 2011) based on metrics considering both:
  - The amount of synchronisations between components
  - The % of transitions that can be hidden after composition

The CADP verification toolbox (cadp.inria.fr)

- Continuously developed & maintained since the late 80's
- Provides all ingredients for compositional verification

	Tool	Description
M	BCG	Compact format for LTS and IMC
||	EXP.OPEN	Labelled transition systems synchronised using the parallel composition operators of various process calculi
≈	BCG_CMP	Comparison wrt. various equivalence relations
min	BCG_MIN	Minimisation wrt. various equivalence relations
L	LOTOS LNT	ISO/IEC standard 8807 (historic) Modern specification language combining features from process calculi, and imperative / functional languages
[[.]]	CAESAR.ADT CAESAR LNT2LOTOS	Compiler for the data part of LOTOS Compiler for the behaviour part of LOTOS Translator from LNT to LOTOS

The SVL language and compiler

- A unique feature of CADP (Garavel & Lang, 2001)
- Makes compositional verification easily accessible
- Can be seen as a process calculus extended with operations on low level models
  - Comparison and minimisation
  - Hiding and renaming of transition labels
  - Detection of deadlocks and livelocks
  - Static and dynamic strategies (including smart reduction)
- Automated translation to shell scripts

cadp.inria.fr/man/svl.html

cadp.inria.fr/man/svl-lang.html

Example of SVL script

```
% DEFAULT_PROCESS_FILE="SCENARIO.lnt"

"SCENARIO.bcg" = smart branching reduction of
    hide "GET_[AB]", "PUT_[AB]" in
        par
                SND_A, RCV_A → TFTP_A [PUT_A, GET_A, RCV_A, SND_A]
        ||      SND_B, RCV_B → TFTP_B [PUT_B, GET_B, RCV_B, SND_B]
        ||      SND_A, RCV_B → MEDIUM [SND_A, RCV_B]
        ||      SND_B, RCV_A → MEDIUM [SND_B, RCV_A]
        end par
    end hide;

"diagnostic.bcg" = deadlock of "SCENARIO.bcg"
```

Applications using CADP

- 11 CADP demos cadp.inria.fr/demos
  - 4 demos (5 to 20 components)
    direct generation fails but compositional verification succeeds
  - 7 demos (4 to 11 components)
    largest model is 1.7 to 24 × smaller than using direct generation

- 25 case-studies (out of 189) since 1991 [30 publications]
  including 3 in perf. evaluation cadp.inria.fr/case-studies
  - avionics/transport: 3
  - bioinformatics: 1
  - communication protocols: 9
  - distributed systems: 4
  - graphical user interfaces: 1
  - hardware design: 5
  - service-oriented computing: 2

The Graf & Steffen approach

- CAV'90 [154 citations], FACJ 1996 (with Lüttgen) [126 citations] + research reports
- Problem: Some $[[C_i]]$ may be much larger than $[[S]]$
  - But only a fraction of $[[C_i]]$ is actually permitted by its *environment* $C_1 || \ldots || C_{i-1} || C_{i+1} || \ldots || C_n$
- Solution: Express constraints on C_i as an *interface*
- In G&S's work, $||$ is CSP parallel composition with forced synchronisation on common actions

Graf & Steffen interfaces

- Set containing all traces allowed by the environment of some component C_i
- Concretely: the traces of a labelled transition system I
- The interface I may be provided by the user
  - It is not necessarily *exact*
  - If it has less traces than allowed by the environment, then I is *incorrect*
  - If it has more traces than allowed by the environment, then I might not express enough constraints $\Rightarrow$ performance problem
- Constraints represented by the interface are applied to C_i using a reduction operator (later called *semi-composition*)

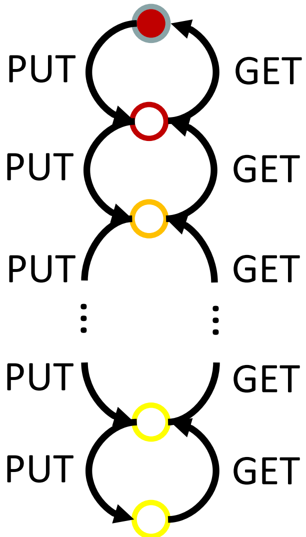
Graf & Steffen semi-composition

- Operator $\Pi_I(C_i)$ defined as the projection of $C_i \,||\, I$ onto C_i
  - state (x, y) of $C_i \,||\, I$ is mapped to x
  - transition $(x, y) \text{ –}a\text{–> } (x', y')$ of $C_i \,||\, I$ is mapped to $x \text{ –}a\text{–> } x'$ if a is an action of C_i, ignored otherwise
- Semi-composition has nice properties
  - $\Pi_I(C_i)$ is behaviourally included in and smaller than $[[C_i]]$
  - I can be reduced wrt. any relation that preserves language equivalence without modifying the final model
  - If I is correct then $[[C_1 || \ldots || C_n]] = [[C_1 || \ldots || \Pi_I(C_i) || \ldots || C_n]]$ i.e., $[[C_i]]$ can be replaced by $\Pi_I(C_i)$

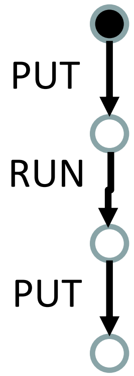
Detection of incorrect interfaces

- A key feature of the Graf & Steffen approach
- Fully automated mechanism
- *Undefinedness predicates* are put in $\Pi_I(C_i)$ to indicate which transitions have been cut off by I
- When recombining $\Pi_I(C_i)$ with its environment, predicates corresponding to impossible synchronisations are discharged
- I is correct if and only if all predicates are discharged in the result $[[C_1 || ... || \Pi_I(C_i) || ... || C_n]]$

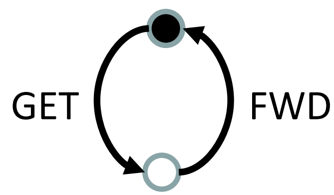
Example



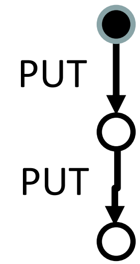
$[[C_1]]$ ||



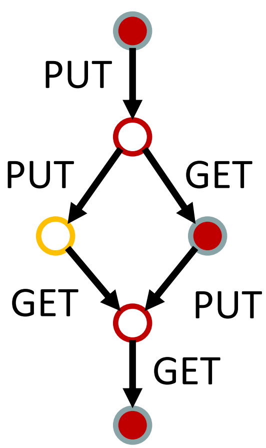
$[[C_2]]$ ||



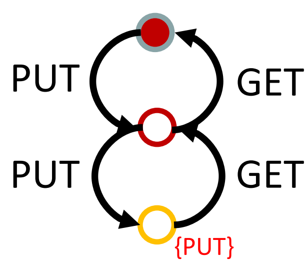
$[[C_3]]$



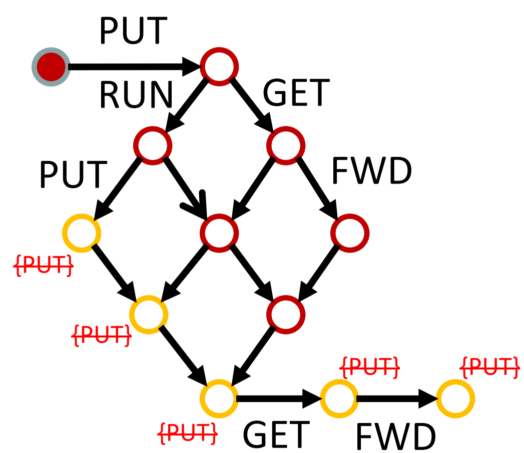
Interface I (for C_1)



$[[C_1 || I]]$



$\Pi_I(C_1)$



$[[\Pi_I(C_1) \;||\; C_2 \;||\; C_3]] = [[C_1 \;||\; C_2 \;||\; C_3]]$ ✓

Related approaches

- Following G&S, Cheung & Kramer (1993) and Valmari (2000) proposed alternative approaches, where C_i is replaced by $[[C_i \,||\, I]]$ instead of $\Pi_I(C_i)$

- But interfaces can be counter-productive in these approches:
  - $[[C_i \,||\, I]]$ can be much larger than $[[C_i]]$
  - Determinisation of the interface is (most often) necessary (potential blow up)

The Krimm & Mounier approach (1/2)

- Krimm & Mounier, TACAS'97
- 1st complete implementation of the G&S approach
- Generalisation to LOTOS hiding and parallel composition
  - operator $|[g_1, \ldots, g_n]|$ (forced synchronisation on gates $g_1, \ldots, g_n$)
  - Enables common yet non-synchronised actions
    e.g., $C_1|[]|C_2$ where C_1 and C_2 propose the same action
  - Enables nondeterministic synchronisation
    e.g., $(C_1 \;|[]|\; C_2) \;|[g]|\; C_3$ where g proposed by C_1, C_2, and C_3
  - Non-associative: $(C_1|[g]|C_2)|[g']|C_3 \neq C_1|[g]|(C_2|[g']|C_3)$ if $g \neq g'$

The Krimm & Mounier approach (2/2)

- $\Pi_I(C_i)$ is generalised to an operator with four arguments
  - A component C_i
  - An interface I
  - A list of gates $g_1, \ldots, g_n$ on which C_i and I must synchronise
  - A Boolean stating whether the interface is surely correct or not
- Useful properties of $\Pi_I(C_i)$ still hold
- Undefinedness predicates are encoded as *fail transitions*: s –fail(a)$\rightarrow$ s if the interface has cut off a in s
- Parallel composition is modified to handle fail transitions

CADP tools for G&S interfaces

- PROJECTOR: On-the-fly semi-composition
  - Generalisation to LOTOS parallel composition and hiding
  - Initially a prototype developed by Krimm & Mounier
  - Entirely rewritten and integrated in CADP (now in version 3.1)
  - *I* is a labelled transition system in the BCG format (explicit)
  - C_i may be expressed in any language connected to the Open/Cæsar API: BCG, LOTOS, LNT, EXP, etc.
- EXP.OPEN: Parallel compo. with undefinedness predicates
- SVL (**abstraction** operator)
  - Example:
    user abstraction "itf.bcg" **sync** SND_A, RCV_A **of** TFTP_A

Interface Synthesis (1/2)

- In $S = C_1 || \ldots || C_n$, how can an interface be computed automatically for some $[[C_i]]$ too large to be generated?
- Practical considerations must be taken into account
  - Used operators are more general than CSP $||$
  - Computing the exact interface may be intractable
- Krimm & Mounier, TACAS'97
  - Automatic interface computation for a given component, given a (flat or hierarchical) component of its environment
  - Based on algebraic rules defined in the framework of LOTOS

Interface Synthesis (2/2)

- Lang, FORTE'06: generalisation of K&M to networks of communicating automata
  - Compute a correct interface from a (user-given) subset of context components by analysing synchronisations
  - Components are not necessarily connected in a PA expression
  - Applicable to other languages than LOTOS
  - Less permissive interfaces are generated when components synchronise nondeterministically
  - Implementation in EXP.OPEN and SVL

Applications using CADP

- 4 CADP demos cadp.inria.fr/demos
  - From 3 to 60 components
  - Direct generation and compositional verification without interfaces fail
  - With semi-composition, largest intermediate model has up to 700,000 states

- 8 case-studies [8 publications]

  mostly industrial examples: Bull, HP, Tiempo, Scalagent

  - avionics/transport: 1
  - cloud computing: 1
  - communication protocols: 2
  - hardware design: 4

Conclusion

- Compositional verification is effective vs. state explosion (many case studies since 30 years)
- Major breakthrough in the 90's: Graf & Steffen
  - Interfaces inspired other (inferior) approaches
  - Semi-composition is not well understood: cited, rarely explained
- CADP exploits the G&S approach
  - Generalisation to LOTOS and LNT, full implementation
  - Application to several case-studies, with impressive results: *Asynchronous circuit (660 concurrent processes) verified in a few hours by a novice industry engineer*