25 Years of Compositionality
Issues in CADP: An Overview

Hubert Garavel and colleagues
Inria Grenoble – LIG
and Saarland University (part-time)

http://convecs.inria.fr
Outline

1. CADP in a nutshell
2. Compositionality issues:
   - 2.a. Types and data structures
   - 2.b. Concurrency I
   - 2.c. Concurrency II
3. Conclusion
1. CADP in a nutshell
CADP

- A modular toolbox for concurrent systems
- Research work at the crossroads between:
  - concurrency theory
  - formal methods
  - computer-aided verification
  - compiler construction
- A long-run effort:
  - development of CADP started in the mid 80s
  - initially: 2 tools
    - CAESAR: LOTOS → Petri nets with data → LTSs
    - ALDEBARAN: minimization and comparison of LTSs modulo bisimulations
  - today: 50 tools
Main features of CADP

- **Formal specification languages**
- **Verification techniques:**
  - Model checking (modal $\mu$-calculus)
  - Equivalence checking (bisimulations)
  - Visual checking (graph drawing)
  
  using
  - Reachability analysis
  - On-the-fly verification
  - Compositional verification
  - Distributed verification
  - Static analysis
- **Other features:**
  - Rapid prototyping
  - Step-by-step simulation
  - Test-case generation
  - Performance evaluation
MCL (Model Checking Language)

- Transition labels carry data values "SEND !2 !true !3.14"
- The MCL temporal logic handles these values
  - Base = alternation-free modal μ-calculus + fairness PDL-Δ operators to express cyclic behaviour
  - Action formulas: value extraction, value matching
  - Path formulas: if-then-else, case, let, for, while, etc.
  - State formulas: fixed points parameterized with typed variables, if-then-else, case, let, quantifiers over finite domains
- MCL supported by the EVALUATOR 4.0 model checker of CADP
LNT (LOTOS New Technology)

LOTOS NT:
- a process calculus disguised as an imperative language

Features:
- typed variables, explicit assignment, pattern matching
- symmetric sequential composition (≠ action prefix)
- usual control structures: if-then-else, case, while, for
- multiway rendezvous, choice, parallel composition

Implemented by translation to LOTOS
Languages connected to CADP

- SAM
- EB3
- WSDL-BPEL
- π-calculus
- SDL
- AADL
- BIP 1
- FSP
- LOTOS NT
- Fiacre
- CHP
- EXP
- LOTOS
- Open/Caesar
- systemC/TLM
2.a. Compositionality issues: Types and data structures
LOTOS abstract data types

type SimpleBoolean is

  sorts Bool

  opns false : → Bool
  true : → Bool
  not : Bool → Bool

  eqns
  not (false) = true;
  not (true) = false;

  endtype

- based on the ACT-ONE language
- initial algebra semantics: \( \Sigma_{\text{bool}} = \{\text{false, true}\} \)
LOTOS type imports

**Types can import other types**

- circular dependencies forbidden
- DAG-like dependencies allowed
- semantics: union of sorts, operations, and equations

```
type BasicBoolean is
  sorts Bool
  opns false : → Bool
  true : → Bool
endtype

+ type SimpleBoolean is BasicBoolean
  opns not : Bool → Bool
  eqns
    not (false) = true;
    not (true) = false;
endtype
```
**Issue #1: Algebra expansion**

\[
\Sigma_{\text{bool}} = \{\text{false}, \text{true}, \text{other}, \text{not (other)}\}
\]

- MyBoolean "corrupts" SimpleBoolean
  - and all types and processes based on SimpleBoolean

---

```
type SimpleBoolean is
  sorts Bool
  opns false : → Bool
  true : → Bool
  not : Bool → Bool
  eqns
    not (false) = true;
    not (true) = false;
endtype

type MyBoolean is SimpleBoolean
  opns other: → Bool
  eqns
    not (not (other)) = other;
endtype
```
Issue #2: Algebra collapse

These equations imply \( \text{true} = \text{false} \)

\[ \Sigma_{\text{Bool}} = \{ \omega \} \text{ where } \omega = \text{true} = \text{false} = \text{fun} (\text{true}) = \ldots \]

Again, MyBoolean "destroys" SimpleBoolean

and everything else based on SimpleBoolean
A way to avoid these issues

When implementing LOTOS in CADP:

- Replace initial algebras with term rewrite systems
- Separate constructors from defined functions
- No equation between constructors
- Decreasing priorities between equations
- Constructors for sort S defined in the same type as S
- Equations for function F defined in the same type as F

When defining E-LOTOS and LOTOS NT:

- One step further: use a functional language
  (≈ ML without first-order, OPAL, etc.)
Impact on compositionality

"Fully flattened" semantics is insecure:
- Any local change may corrupt the global meaning
- Not acceptable from an engineering point of view
- Kind of "butterfly effect"

Solution: "frontiers" (inside, outside, interface)
- Defined things can be used everywhere
- but can only be modified at controlled locations

Many examples:
- Encapsulation: modules, classes, objects
- Monitors / rendezvous rather than shared variables
2.b. Compositionality issues: Concurrency I
Compositional model generation

\[ P_1 \parallel \ldots \parallel P_n = \Sigma \]

\[ P'_1 \parallel \ldots \parallel P'_n = \Sigma' \]

- only valid if \( \approx \) is a congruence with respect to \( \parallel \parallel \)
- can/should be applied recursively
Compositional LTS generation using CADP

- Parallel components are (explicit or implicit) LTSs
- This approach is heavily implemented in CADP
  - LTSs are generated from high-level languages
  - BCG_MIN: minimization of LTSs modulo strong or branching minimization
  - REDUCTOR: on-the-fly reduction of LTSs modulo 8 equivalence relations
  - EXP.OPEN: composition of LTSs using many parallel composition operators (+ hiding, renaming, cut)
Compositional IMC generation using CADP

Hermanns, LNCS 2428

Garavel-Hermanns, FME 2002

- Parallel components are IMCs (Interactive Markov Chains)
  - normal transitions + stochastic ("rate") transitions
- Parallel composition is similar to interleaving
  - implemented in the EXP.OPEN tool of CADP
- Minimization combines lumpability on Markov chains with strong/branching bisimulation on LTSs
  - implemented in the BCG_MIN tool of CADP
- Additional tools: steady-state / transient solvers
Smart reduction

use metrics that suggest a "good" composition order rather than leaving the decision to the user

1. Select a subset of the individual processes
2. Compose this subset in parallel, hiding the internal labels
3. Minimize the resulting parallel composition modulo some equivalence (congruence)

Repeat until all individual processes have been composed
### Smart reduction: Experimental results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Node</th>
<th>Root leaf</th>
<th>Smart (HM)</th>
<th>Smart (IM)</th>
<th>Smart (CM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABP 1</td>
<td>380</td>
<td>328</td>
<td>210</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>ABP 2</td>
<td>2,540</td>
<td>2,200</td>
<td>1,354</td>
<td>504</td>
<td>504</td>
</tr>
<tr>
<td>Cache</td>
<td>1,925</td>
<td>1,925</td>
<td>1,848</td>
<td>1,925</td>
<td>1,925</td>
</tr>
<tr>
<td>CFS</td>
<td>2,193,750</td>
<td>486,990</td>
<td>1,366,968</td>
<td>96,040,412</td>
<td>5,113,256</td>
</tr>
<tr>
<td>DES</td>
<td>22,544</td>
<td>3,508</td>
<td>3,508</td>
<td>14,205</td>
<td>14,205</td>
</tr>
<tr>
<td>DFT CAS</td>
<td>95,392</td>
<td>99,133</td>
<td>336</td>
<td>346</td>
<td>346</td>
</tr>
<tr>
<td>DFT HCPS</td>
<td>4,730</td>
<td>79,509</td>
<td>425</td>
<td>435</td>
<td>435</td>
</tr>
<tr>
<td>DFT IL</td>
<td>29,808</td>
<td>316</td>
<td>2,658</td>
<td>1,456</td>
<td>2,658</td>
</tr>
<tr>
<td>DFT MDCS</td>
<td>635,235</td>
<td>117,772</td>
<td>536</td>
<td>5,305</td>
<td>346</td>
</tr>
<tr>
<td>DFT NDPS</td>
<td>17,011</td>
<td>1,857</td>
<td>393</td>
<td>449</td>
<td>346</td>
</tr>
<tr>
<td>DLE 1</td>
<td>15,234</td>
<td>7,660</td>
<td>7,709</td>
<td>10,424</td>
<td>9,883</td>
</tr>
<tr>
<td>DLE 2</td>
<td>8,169</td>
<td>2,809</td>
<td>2,150</td>
<td>1,852</td>
<td>2,150</td>
</tr>
<tr>
<td>DLE 3</td>
<td>253,272</td>
<td>217,800</td>
<td>181,320</td>
<td>231,616</td>
<td>175,072</td>
</tr>
<tr>
<td>DLE 4</td>
<td>33,920</td>
<td>29,584</td>
<td>25,008</td>
<td>8,896</td>
<td>26,864</td>
</tr>
<tr>
<td>DLE 5</td>
<td>1,796,616</td>
<td>1,796,616</td>
<td>1,403,936</td>
<td>1,716,136</td>
<td>1,433,640</td>
</tr>
<tr>
<td>DLE 6</td>
<td>35,328</td>
<td>35,328</td>
<td>5,328</td>
<td>5,328</td>
<td>5,328</td>
</tr>
<tr>
<td>DLE 7</td>
<td>612,637</td>
<td>486,491</td>
<td>369,810</td>
<td>583,289</td>
<td>577,775</td>
</tr>
<tr>
<td>HAVI async</td>
<td>145,321</td>
<td>22,703</td>
<td>21,645</td>
<td>21,862</td>
<td>21,809</td>
</tr>
<tr>
<td>HAVI sync</td>
<td>19,339</td>
<td>5,021</td>
<td>4,743</td>
<td>4,743</td>
<td>4,743</td>
</tr>
<tr>
<td>NFP</td>
<td>199,728</td>
<td>1,986,768</td>
<td>104,960</td>
<td>89,696</td>
<td>89,696</td>
</tr>
<tr>
<td>ODP</td>
<td>158,318</td>
<td>158,318</td>
<td>87,936</td>
<td>39,841</td>
<td>87,936</td>
</tr>
<tr>
<td>RelRel 1</td>
<td>28,068</td>
<td>9,228</td>
<td>9,282</td>
<td>5,574</td>
<td>5,574</td>
</tr>
<tr>
<td>RelRel 2</td>
<td>11,610,235</td>
<td>5,341,821</td>
<td>5,341,821</td>
<td>5,341,821</td>
<td>5,341,821</td>
</tr>
<tr>
<td>SD 1</td>
<td>21,870</td>
<td>3,482</td>
<td>19,679</td>
<td>4,690</td>
<td>19,679</td>
</tr>
<tr>
<td>SD 2</td>
<td>6,561</td>
<td>11,997</td>
<td>3,624</td>
<td>2,297</td>
<td>3,192</td>
</tr>
<tr>
<td>SD 3</td>
<td>1,944</td>
<td>32,168</td>
<td>1,380</td>
<td>896</td>
<td>1,164</td>
</tr>
<tr>
<td>SD 4</td>
<td>633,130</td>
<td>1,208,592</td>
<td>975,872</td>
<td>789,886</td>
<td>975,872</td>
</tr>
<tr>
<td>TN</td>
<td>54,906,000</td>
<td>746,880</td>
<td>69,547,712</td>
<td>749,312</td>
<td>709,504</td>
</tr>
</tbody>
</table>
Interfaces and projections (1)

Sometimes splitting generates larger LTSs:

Because splitted processes constrained each other
Interfaces and projections (2)

semi-composition operator
Interfaces and projections

Interfaces $L$

- Finite-state automata (trace acceptors)
- Interfaces must be suggested by the user
- Warning messages if interfaces are too restrictive

Semi-composition operator $P_i \parallel \perp - L$

- Not a parallel composition!
- $P_i \parallel \perp - L$ has no more states than $P_i$
- Implemented by the PROJECTOR tool of CADP

A working approach to fight state explosion

Graf-Steffen, CAV 1990
Krimm-Mounier, TACAS 1997
Automatic generation of interfaces

- Computed for one process $P_1$ wrt to $P_2 \ldots P_n$
- Better reductions than using Krimm-Mounier-97
- Safety minimization and partial order reductions can be used:

Experimental results on large processes

- Philips' HAVi protocol: $365,923 \rightarrow 645$ states
- ODP trader: $1$ million states $\rightarrow 256$ states
- Cache coherency: $1$ million states $\rightarrow 60$ states

Lang, FORTE 2006
The SVL scripting language (1)

- Verification scenarios are complex and repetitive
- Many tools and techniques:
  - enumerative, on-the-fly, compositional, interfaces...
  - verification and performance evaluation
- Many files (and formats) to handle:
  - concurrent descriptions: LOTOS, LOTOS NT, EXP, FSP...
  - explicit and implicit LTSs, CTMCs, DTMCs, IMCs...
  - interfaces, logic formulas, probability vectors...

Garavel-Lang, FORTE 2001
The SVL scripting language (2)

- Many operations to perform:
  - LTS/IMC generation and projection
  - Label hiding, renaming, cut
  - Minimization and comparison modulo equivalences
  - Model checking, deadlock and livelock detection

- SVL:
  - a language to specify scenarios (+ Unix shell)
  - a compiler to execute them
  - provides a unified view of CADP tools
  - implement expert verification strategies
2.c. Compositionality issues: Concurrency II
Difference between parts I and II

- **Part I**
  - Only **equivalences** are considered
  - State space reduction must preserve an equivalence
  - Goal: generate a reduced/minimal state space

- **Part II**
  - A set of **logical formula** $\{\varphi_1, \varphi_2, ..., \varphi_n\}$ is considered
  - State space reduction must preserve the truth values of these formulas
  - Goal: evaluate these formulas on a reduced/minimal state space
Decomposition wrt the formula set

\[ \Sigma \models \varphi_1, \varphi_2, \ldots, \varphi_n \]

The unique \( \Sigma \) is replaced by several state spaces \( \Sigma_i \)
Each \( \Sigma_i \) is specialized/reduced wrt a given formula \( \varphi_i \)

\[ \Sigma_1 \models \varphi_1 \quad \Sigma_2 \models \varphi_2 \quad \ldots \quad \Sigma_n \models \varphi_n \]
Approach 1: strong equivalence

Mateescu-Wijs, SPIN 2011

- $\varphi_1$, $\varphi_2$, ..., $\varphi_n$ are written in modal $\mu$-calculus
- For each $\varphi_i$ one computes a set of actions $A_i$ such that: $\Sigma \models \varphi_i \iff (\text{hide } A_i \text{ in } \Sigma) \models \varphi_i$
- Basically, $A_i$ gathers actions not occurring in $\varphi_i$
- $A_i$ should be as large as possible (maximal hiding) to enable the greatest possible reduction
- $(\text{hide } A_i \text{ in } \Sigma)$ is reduced wrt strong bisimulation before evaluating $\varphi_i$ (global model checking) or on-the-fly while evaluating $\varphi_i$ (local model checking)
Approach 2: diverg. branching equiv.

- $\varphi_1, \varphi_2, ..., \varphi_n$ are written in a subset of the modal $\mu$-calculus compatible with divergence-sensitive branching bisimulation.

- For each $\varphi_i$ one computes a set of actions $A_i$ such that: $\Sigma \models \varphi_i \iff (\text{hide } A_i \text{ in } \Sigma) \models \varphi_i$.

- $(\text{hide } A_i \text{ in } \Sigma)$ is reduced with divergence-sensitive branching bisimulation (enabling greater reductions than using strong bisimulation) or $\tau$-confluence reduction (done on the fly).

Mateescu-Wijs, SPIN 2011
Experimental results using CADP

- Using strong bisimulation
  - alternating bit (12 M states, 46 M transitions): speedup $\times 4$, memory / 2
  - token ring (53 M states, 214 M transitions): speedup $\times 2.8$, memory / 2.5

- Using divergence-sensitive branching bisimulation
  - Philips BRP (12 M states, 14 M transitions): memory / 1.6

- Using $\tau$-confluence reduction
  - Erathosthene sieve: speedup $\times 10$
Partial model checking  [Andersen, LICS 95]

\[ P_1 \parallel \cdots \parallel P_n \models \varphi \]

\[ (to~be~applied~recursively) \]
Three issues with partial model checking

- The left-hand side should decrease a lot
  - $P_2 \parallel \ldots \parallel P_n$ should be much smaller than $P_1 \parallel \ldots \parallel P_n$
  - Not necessarily the case if $P_1$ constrains the others

- The right-hand side should not increase too much
  - Quotienting removes modalities, but adds variables
  - Quotiented formulas $\varphi \parallel P_1$ can become very large
  - Simplifications must be applied after quotienting

- It requires a complex software machinery
  - Only a few implementations available
Partial model checking using CADP

- Asynchronous, action-based setting
- Concurrent processes $P_1 \parallel \ldots \parallel P_n$:
  - Networks of LTSs (i.e., the EXP format of CADP)
  - Based on "synchronization vectors" + hiding, renaming
  - Supports the binary and n-ary parallel operators of CCS, CSP, LOTOS, LOTOS NT, etc.
- Formulas $\varphi$:
  - Alternation-free modal $\mu$-calculus
  - + fairness operators of alternation 2

Lang-Mateescu, TACAS 2012
Quotienting revisited

- Formula $\varphi$ is encoded as an LTS (formula graph)
  - LTSs are represented using the BCG format of CADP

\[
\mu X^0 . (<a> \text{ true}) \lor (<b> X^0)
\]

- Quotient $\varphi // \text{P}_1$ is reformulated as a synchronous product of 2 LTSs (the formula graph of $\varphi$ and $\text{P}_1$)
  - Product can be expressed in the EXP format of CADP
  - It is computed using the EXP.OPEN tool of CADP
Post-quotienting simplifications

- Elimination of double negations
- Elimination of useless $\mu$–transitions
  - sufficient conditions are used
- Elimination of $\lor$-transitions
  - hiding and reduction modulo $\tau^*.a$ equivalence
- Sharing of identical sub-formulas
  - tagging $\mu$-transitions $\rightarrow$ strong bisimulation reduction
- Partial evaluation of states
  - detection and propagation of constant sub-formulas
  - using the CADP solver for Boolean Equation Systems
Experimental results: SCSI-2 benchmark

<table>
<thead>
<tr>
<th>Number of disks</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product LTS size (states)</td>
<td>56,168</td>
<td>1,384,021</td>
<td>32,003,282</td>
<td>708,174,559</td>
</tr>
<tr>
<td>Product LTS size (transitions)</td>
<td>154,748</td>
<td>4,499,237</td>
<td>119,691,662</td>
<td>2,992,012,087</td>
</tr>
<tr>
<td>Generation time (seconds)</td>
<td>1</td>
<td>17</td>
<td>884</td>
<td>31,193</td>
</tr>
<tr>
<td>Memory peak (MB)</td>
<td>66</td>
<td>66</td>
<td>680</td>
<td>17,594</td>
</tr>
<tr>
<td>On-the-fly model checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verification time (seconds)</td>
<td>1</td>
<td>17</td>
<td>1,273</td>
<td>47,532</td>
</tr>
<tr>
<td>Memory peak (MB)</td>
<td>66</td>
<td>95</td>
<td>1,705</td>
<td>39,236</td>
</tr>
<tr>
<td>Partial model checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verification time (seconds)</td>
<td>19</td>
<td>61</td>
<td>759</td>
<td>24,276</td>
</tr>
<tr>
<td>Memory peak (MB)</td>
<td>66</td>
<td>66</td>
<td>1,007</td>
<td>16,239</td>
</tr>
<tr>
<td>Largest formula graph (states)</td>
<td>22,171</td>
<td>253,723</td>
<td>2,773,147</td>
<td>29,367,067</td>
</tr>
<tr>
<td>Largest formula graph (transitions)</td>
<td>198,467</td>
<td>3,023,449</td>
<td>45,639,547</td>
<td>710,452,069</td>
</tr>
</tbody>
</table>
### Experimental results: TFTP benchmark

<table>
<thead>
<tr>
<th>Prop</th>
<th>Scenario A 1,963 ks</th>
<th>Scenario B 867 ks</th>
<th>Scenario C 35,024 ks</th>
<th>Scenario D 40,856 ks</th>
<th>Scenario E 19,436 ks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fly</td>
<td>pmc</td>
<td>fly</td>
<td>pmc</td>
<td>fly</td>
</tr>
<tr>
<td>A01</td>
<td>199</td>
<td>6</td>
<td>89</td>
<td>6</td>
<td>2,947</td>
</tr>
<tr>
<td>A02</td>
<td>207</td>
<td>6</td>
<td>93</td>
<td>6</td>
<td>3,156</td>
</tr>
<tr>
<td>A03</td>
<td>182</td>
<td>6</td>
<td>80</td>
<td>6</td>
<td>2,737</td>
</tr>
<tr>
<td>A04</td>
<td>199</td>
<td>6</td>
<td>89</td>
<td>6</td>
<td>2,947</td>
</tr>
<tr>
<td>A05</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>A06</td>
<td>187</td>
<td>6</td>
<td>85</td>
<td>6</td>
<td>2,808</td>
</tr>
<tr>
<td>A07</td>
<td>187</td>
<td>6</td>
<td>85</td>
<td>6</td>
<td>2,808</td>
</tr>
<tr>
<td>A08</td>
<td>186</td>
<td>6</td>
<td>80</td>
<td>6</td>
<td>2,745</td>
</tr>
<tr>
<td>A09a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,290</td>
</tr>
<tr>
<td>A09b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,955</td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,354</td>
</tr>
<tr>
<td>A11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,206</td>
</tr>
<tr>
<td>A12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>620*</td>
</tr>
<tr>
<td>A13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,499*</td>
</tr>
<tr>
<td>A14</td>
<td>267</td>
<td>6</td>
<td></td>
<td></td>
<td>3,988</td>
</tr>
<tr>
<td>A15</td>
<td>118</td>
<td>15</td>
<td>521</td>
<td>15</td>
<td>156</td>
</tr>
<tr>
<td>A16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>186</td>
</tr>
<tr>
<td>A17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>667*</td>
</tr>
<tr>
<td>A18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,52</td>
</tr>
<tr>
<td>A19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>A20</td>
<td>31</td>
<td>9</td>
<td></td>
<td></td>
<td>837</td>
</tr>
<tr>
<td>A21</td>
<td>374</td>
<td>6</td>
<td></td>
<td></td>
<td>4,958</td>
</tr>
<tr>
<td>A22</td>
<td>35</td>
<td>6</td>
<td></td>
<td></td>
<td>427</td>
</tr>
<tr>
<td>A23</td>
<td>170</td>
<td>6</td>
<td></td>
<td></td>
<td>6,909</td>
</tr>
<tr>
<td>A24</td>
<td>41</td>
<td>9</td>
<td></td>
<td></td>
<td>427</td>
</tr>
<tr>
<td>A25</td>
<td>391</td>
<td>6</td>
<td></td>
<td></td>
<td>5,480</td>
</tr>
<tr>
<td>A26</td>
<td>195</td>
<td>6</td>
<td></td>
<td></td>
<td>2,857</td>
</tr>
<tr>
<td>A27</td>
<td>228</td>
<td>6</td>
<td></td>
<td></td>
<td>3,534</td>
</tr>
<tr>
<td>A28</td>
<td>102</td>
<td>6</td>
<td></td>
<td></td>
<td>3,654</td>
</tr>
</tbody>
</table>
3. Conclusion
Conclusion

- Compositionality is essential
  - modular design
  - formal verification
  - performance analysis
  - divide-and-conquer to fight state explosion

- Compositionality has multiple facets
  - data vs behaviour
  - action-based vs state-based
  - logics vs equivalences
Compositionality is demanding — it requires:

- Suitable low-level semantic **models**
  \[ \Rightarrow \text{LTSs, IMCs, etc.} \]

- Well-chosen behavioural **equivalences**
  \[ \Rightarrow \text{bisimulations: strong, branching, divergence-preserving, lumpability on Markov chains} \]

- Well-chosen **logics**
  \[ \Rightarrow \text{mu-calculus, temporal logics} \]
  \[ \Rightarrow \text{adequation results relating logics and equivalences} \]

- Concurrent **languages** with a proper semantics
  \[ \Rightarrow \text{process calculi and their modern variants (such as LNT)} \]
  \[ \Rightarrow \text{congruence results relating parallel composition and equivalences} \]
Compositionality and CADP

CADP:
- A modular toolbox implementing concurrency theory
- Used for teaching, research, and industrial problems
- Free for academics

Compositionality underlies CADP architecture:
- Many compositional approaches implemented
- Combinations of existing and new CADP components
- Mostly in an action-based setting

Our wish: Compositionality made easy using CADP