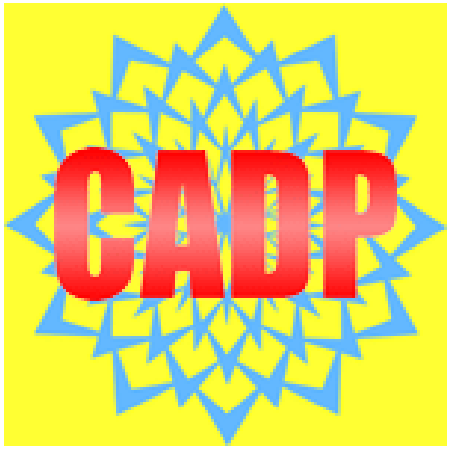
CADP

Construction and Analysis of Distributed Processes

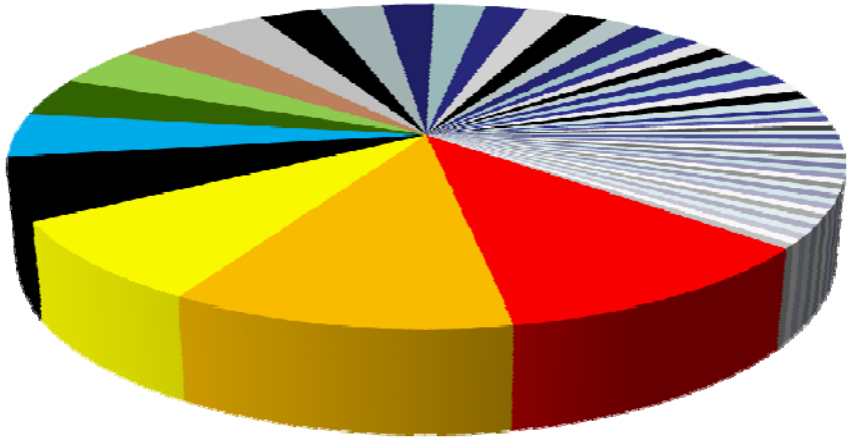
ETAPS Test-of-time Tool Award 2023

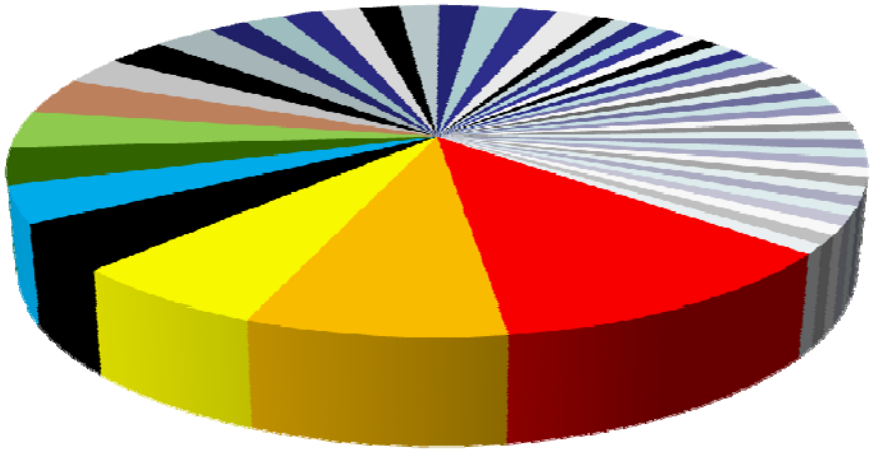
Hubert GARAVEL
Frédéric LANG
Radu MATEESCU
Wendelin SERWE



CADP Objectives

- Formal analysis of **concurrent systems**:
  design-error detection: safety, security, correctness
- Message-passing communication
- Wide usage in various (3rd party) research work:
  > 200 case studies > 100 research tools





Communication Protocols Hardware Design Formal Verification Web Services
Distributed Systems Embedded Systems Protocol Testing Embedded Systems

CADP in Practice

- Comprehensive software package:
  59 tools, 18 libraries, 630 pages of documentation
- Continuously improved since 1987
- Rolling release schedule (one per month)
- 6 supported 32/64-bit architectures
  (Linux, macOS, Solaris, Windows)
- Worldwide distribution
- https://cadp.inria.fr

Principal Modelling Language: LNT

- Features
  - Uniform usual imperative syntax
  - Heavy use of static analysis (semantics & warnings)
  - Strong typing
- Concepts: expressions, instructions, behaviours
- Convenient translation target
- Development language of most compilers in CADP
- Positive feedback from academia and industry

```
process FILTER [GET: option_channel,
                PUT: nat_channel] (b: Nat) is
  var opt: Option in
    loop
      GET (?opt) ;
      case opt var x: Nat in
        none -> null
      | some(x) where x > b -> PUT (x)
      end case
    end loop
  end var
end process
```

Other Modelling Languages

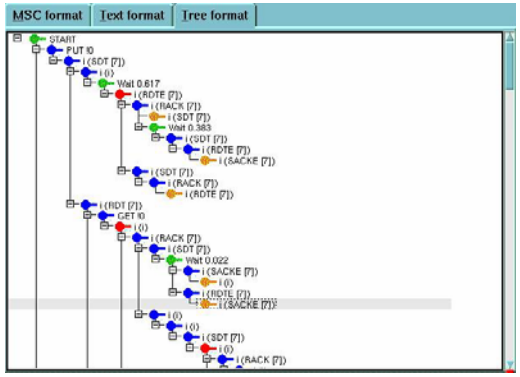
- LOTOS (ISO standard 8807)
  - Supported since the beginning of CADP
  - Current target of the LNT tool chain
- FSP (Finite State Processes)

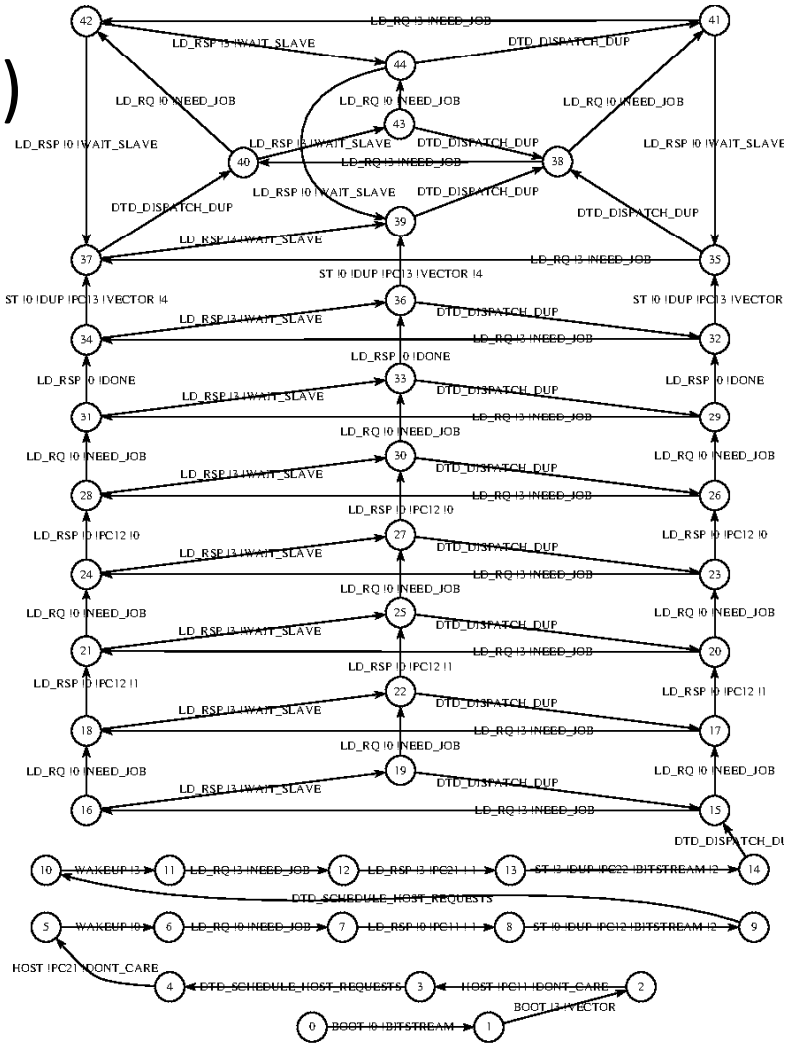
  Language of the LTSA tool
- EXP (networks of communicating automata)

  Synchronization vectors as generic composition means
- Common features
  - Labelled Transition Systems semantics
  - Parallel composition, interleaving, rendezvous

Visual Checking / Simulation

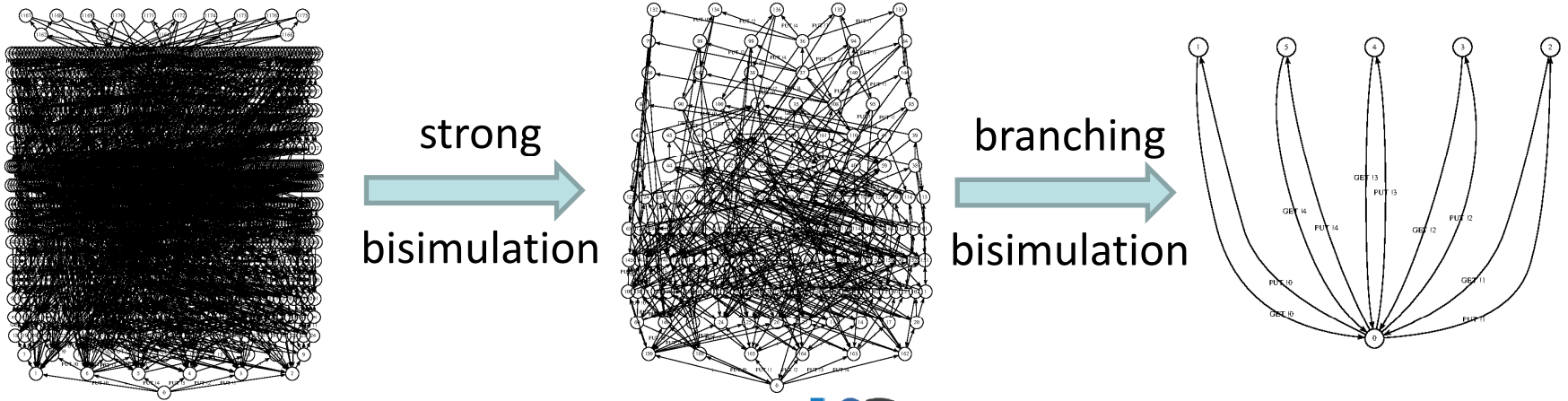
- Manipulation of explicit LTS
  - Compact storage (BCG format)
  - Visualization
- Step-by-step simulation
  - Backtracking
  - Save & load simulations
- Random simulation





Equivalence Checking

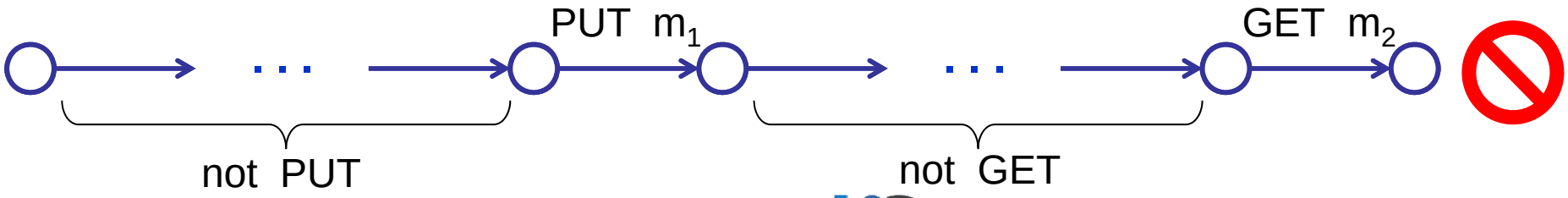
- LTS *comparison* and reduction/*minimization*
- Various relations and preorders, including
  - strong bisimulation
  - (divergence-preserving) branching bisimulation
  - (weak) trace equivalence
  - probabilistic/stochastic variants
- Basis for compositional techniques



Model Checking

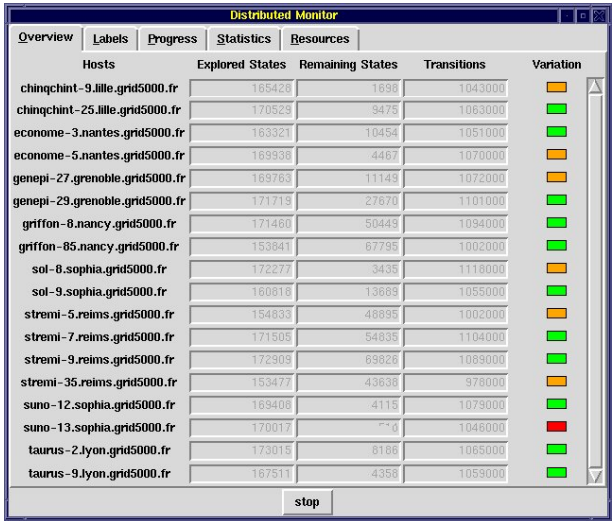
- XTL (on LTS in the BCG format): functional language for graph traversals
- MCL (on-the-fly)
  - regular alternation free μ-calculus
  - macro libraries for CTL, ACTL, ...
  - infinite looping operator
  - regular expressions on action sequences
  - action predicates with data-handling constructs
  - probabilistic extensions

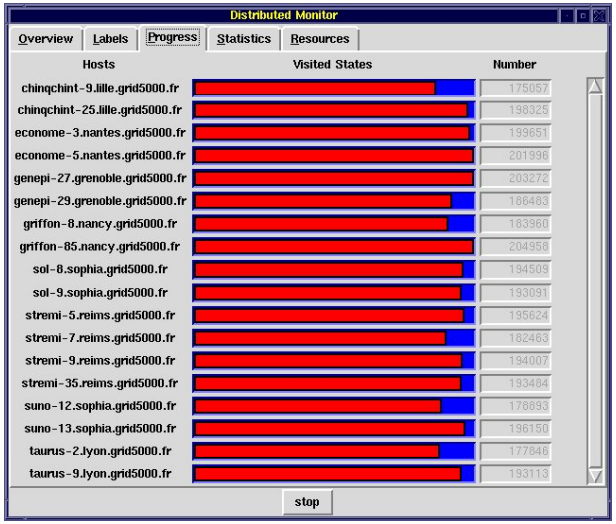
```
NEVER (
   (not { PUT ?any })* .
   { PUT ?m1:Nat } .
   (not { GET ?any })* .
   { GET ?m2:Nat
      where m1 <> m2 }
)
```



Distributed Tools

- Distributed state space generation
  - Use memory of up to hundreds of computers
  - Good speed-up
- Manipulation of distributed state spaces
- Distributed solving of Boolean equation systems (distributed model- and equivalence checking)



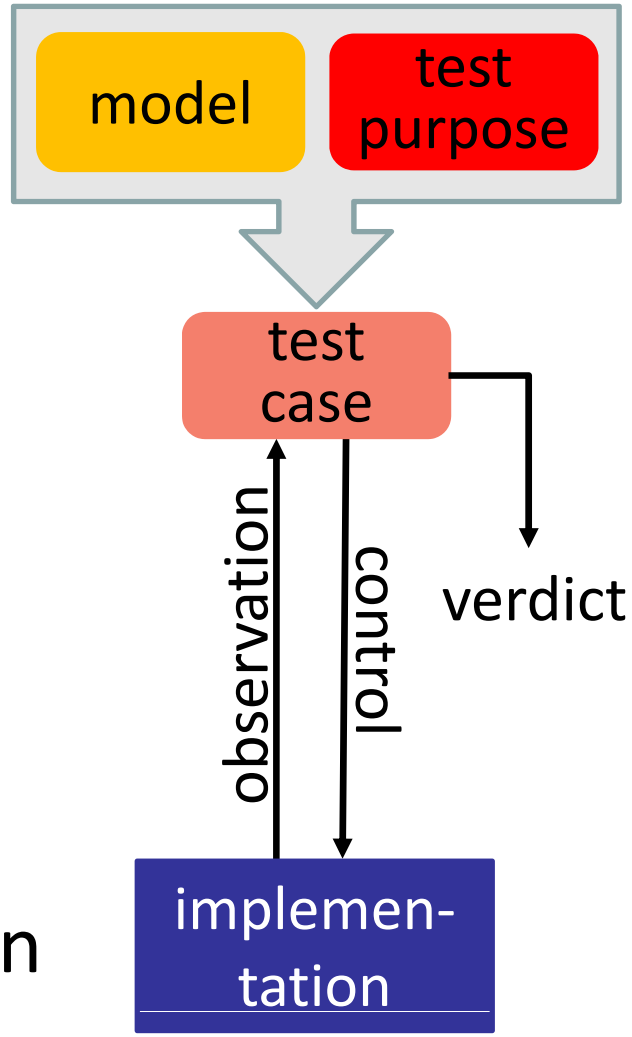


Compositional Verification

- Divide and conquer principle:
  generate, reduce, and compose components hierarchically
- Heuristics to select composition order
- Semi-composition using interfaces
- Property-dependent reductions
- Success stories
  - [ASYNC18] 146 LNT processes, 660 concurrent units, semi-composition, intermediate LTSs below 116 Mstates
  - [RERS2019] up to 70 automata (each with up to 153 states), new sharp bisimulation, property dependent reduction

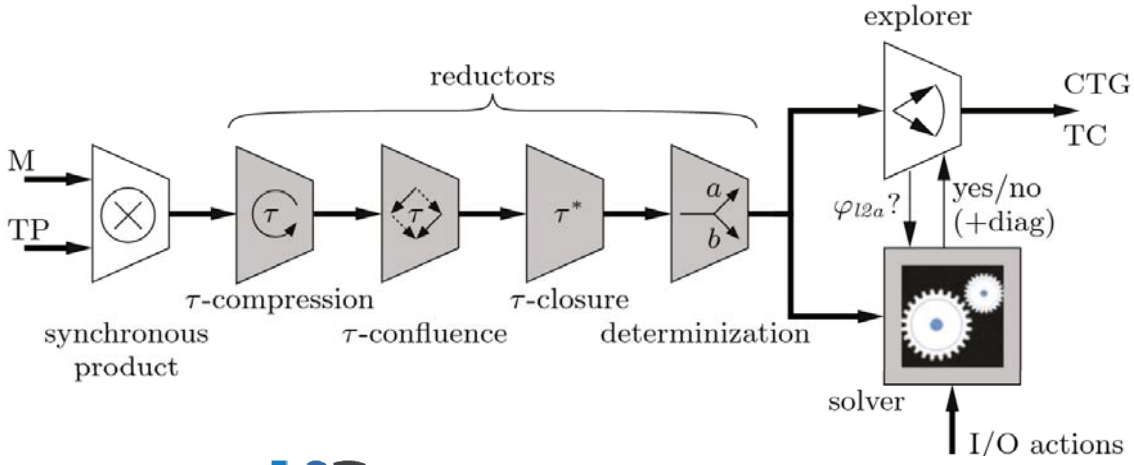
Conformance Testing

- Model-based testing:
  model as *test oracle*
- **ioco** conformance relation
- Test purpose to guide on-the-fly test case generation
- Coverage guarantees
- EXEC/CÆSAR framework
  simultaneous execution of
  the model and an implementation

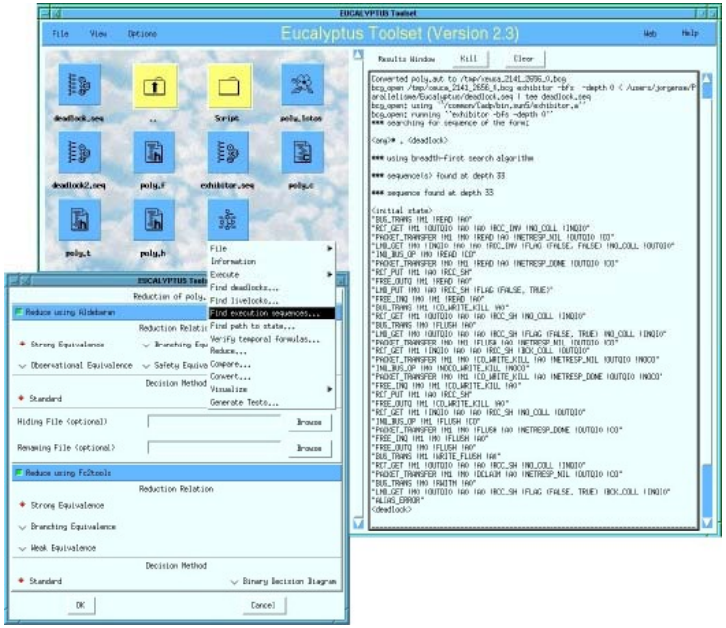


Software Libraries

- Manipulation of binary LTS format
- Auxiliary data structures:
  hash table, bitmap, cache, stack, hiding/renaming, ...
- Common algorithms: BES solving
- Extensive documentation
- Example: components of a test generation tool
  - grey: reused
  - white: new



User Interfaces



- Convenient access to tools
- Eucalyptus GUI
  - Contextual menus
  - Well-chosen default values
- SVL (Script Verification Language)
  - Integration of verification, properties, and shell commands
  - Automatic advanced verification heuristics
  - Support for compositional techniques

Quality Control and Support

- Code review before integration
- Nightly test of demo examples
- Large collections of models and properties
  - CONTRIBUTOR tool for automatic gathering of examples
  - Example provider for benchmarks and model repositories (VLTS, VLSAT, MARS)
  - Benchmark provider for tool competitions (MCC, SAT competition, SMT-COMP, Model Counting)
- Support tools
  - INSTALLATOR: graphical installer
  - TST: diagnostics of installation problems
  - UPC: upgrade specifications following language changes

Conclusion: Salient CADP Features

- Sophisticated *rich modelling languages* with explicit *parallelism* and *general* data types
- Action-based branching-time logics
- *Model checking* and *equivalence checking*
  - *On-the-fly* algorithms
  - *Distributed* tools
  - *Compositional* approaches
- Smooth *combination of all techniques*

"Concurrency theory in practically usable tools"

More Information about CADP

- Website: https://cadp.inria.fr
  demo examples, documentation, current status, ...
- User forum: https://cadp.forumotion.com
- Overview [Garavel-Lang-Mateescu-Serwe-13]
- Awards
  - 9 gold medals at RERS competitions (2019 & 2020)
  - Innovation award (French Académie des sciences, 2021)
  - ETAPS Test-of-time Tool (2023)
- To obtain a free academic license
  https://cadp.inria.fr/registration