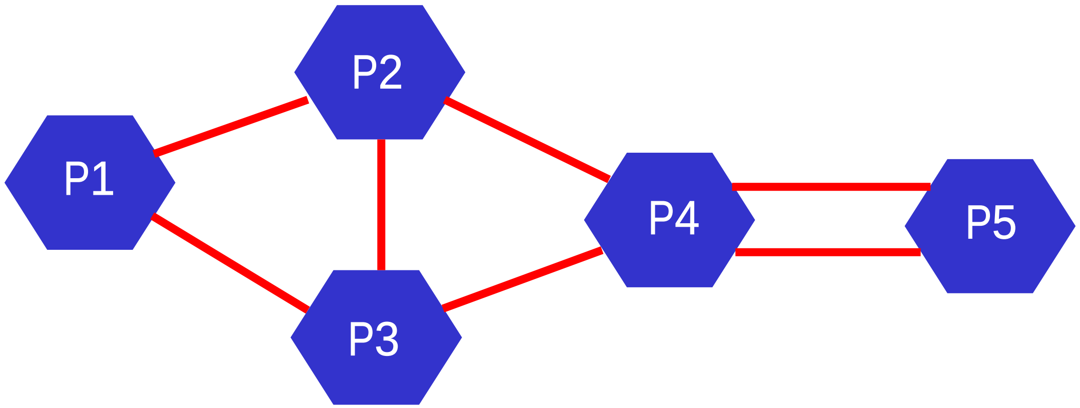
Recent developments and improvements of the CADP toolbox

Frédéric Lang

INRIA Rhône-Alpes / VASY

http://www.inrialpes.fr/vasy

Concurrent asynchronous systems



- several processes / tasks / activities / agents
- that execute in parallel
- at different speeds (no central clock)
- no need for a central, shared memory
- with unspecified communication delays / latencies

Origins of CADP

- Work initiated in 1986
- Latest stable version: CADP 2006 "Edinburgh"
- Developed and maintained by the VASY team of INRIA Grenoble
- Includes contributions from:
  - Holger Hermanns (performance evaluation tools)
  - INRIA Rennes ("tgv" tool)
  - Verimag ("aldebaran.old" tool)

Key concepts behind CADP

- CADP takes roots in concurrency theory
- Process algebra
  - Modular value-passing languages
  - Equivalences (Bisimulation)
  - Compositionality
- Explicit-state verification
  - As opposed to symbolic methods (BDDs, etc.)
  - Action-based models (Labelled Transition Systems)
  - Mu-calculus, temporal logics
  - Model checking

Main features of CADP

- Formal description using process algebras (LOTOS)
- C code generation, rapid prototyping
- Step by step simulation, random execution
- Enumerative ("explicit-state") verification:
  - exhaustive
  - partial
  - on the fly
  - compositional
  - parallel/distributed using clusters
- Various verification techniques:
  - visual checking (graph display)
  - model checking (modal mu-calculus)
  - equivalence checking (bisimulations)
- Performance evaluation
- Test generation

CADP acronym change

- Formerly (only 2 tools in 1989):

  CAESAR/ALDEBARAN Development Package

- Now (42 tools, 17 software components):

  Construction and Analysis of Distributed Processes

Main applications of CADP

- Industrial case-studies
  - hardware, software, telecom, embeded systems...
  - formal specification of critical systems and protocols
  - simulation, rapid prototyping, verification, testing
- Research
  - analysis of new systems/protocols
  - experimentation of new verification/testing algorithms
  - implementation of new modelling languages
- Education
  - concurrency, process algebras, bisimulations, model checking
  - robust tools for lab exercises and student projects

General enhancements

Computing platforms

- Support of recent C compilers:
  - Gcc 3.*, Gcc 4.*
  - Sun Studio 11
  - Intel ICC 9.*
- Support of recent Linux distributions:
  - Suse
  - Fedora Core
- Support of 64-bits processors (AMD, Intel Itanium, Sparc)
- Better support of Windows (2000, XP, Vista)
- Support of Mac OS X (PowerPC since 10.2 and Intel since 10.4)

Installation & support

- Enhanced installation tool (Installator)
- Enhanced self-checking tool (Tst)
- Enhanced licensing system:
  - multiple license files are allowed
  - automatic e-mails warnings before license expiration

Enhancements to LOTOS tools

The CAESAR.ADT tool

- One major improvement: data type iterators
- CAESAR.ADT 5.2 generates iterators for every finite type, including union types

- Finiteness verification for types that need an iterator
- Introduction of "new-style" iterators
- Backward compatibility with "old-style" iterators
- Support for hand-written iterators (old- and new-style)

The CAESAR.BDD tool

- Reachability analysis for hierarchical Petri nets
- Based on the BDD package CUDD (F. Somenzi)
- Currently, two uses:
  - Improves efficiency of CAESAR's optimization E7 (elimination of dead transitions)
  - Determines information about concurrent processes required for static analysis

The CAESAR tool (1)

- Significant performance improvements:
  - reduced memory usage
  - maximal number of states increased from 2^24 to 2^32
  - support for label strings of arbitrary length
  - higher speed of the generated C code
- State space reduction using static analysis
  - local and global data flow analysis
  - resetting of locally dead variables
  - gains: several orders of magnitude

The CAESAR tool (2)

- Extension of the EXEC/CAESAR framework that connects LOTOS specifications to the "real-world"
- Feedback obtained after intensive use by Bull of EXEC/CAESAR (connection between LOTOS and CADENCE's Verilog simulator)
- Extended API:
  - new primitives for restarting the system
  - new primitives for coverage measurement
  - new primitives for logging events
- Automatic generation of "gate functions" (including overloaded gates)

Tools for on-the-fly verification

The OPEN/CAESAR libraries

- Two new libraries :
  - CAESAR_AREA_1: handling of memory chunks
  - CAESAR_MASK_1: hiding/renaming labels on-the-fly
- Improved hash functions in CAESAR_HASH library
- Many enhancements in CAESAR_TABLE_1
  - extended storage capacity
  - reduced memory usage
  - improved statistics display

The CAESAR_SOLVE library

- A generic solver for Boolean Equation Systems
- Built on top of Open/Caesar
- Generic encoding for Boolean Equations Systems of alternation 1, represented as boolean graphs
- Five algorithms for solving Boolean Equation Systems:
  - a general DFS algorithm
  - a general BFS algorithm
  - two memory-efficient DFS algorithms optimized for acyclic and conjunctive/disjunctive Boolean graphs
  - an optimized BFS algorithm dedicated to confluence
- Linear complexity in the size of the boolean graph
- Automatic diagnostic generation (fragments of LTSs)
  - Examples
  - Counter-examples

The EVALUATOR 3.5 tool

- A model checker for alternation-free μ-calculus extended with regular expressions over labels and sequences of actions
- The model checking problem is translated into the resolution of a Boolean Equation System (built on-the-fly)

- Entirely rewritten to use CAESAR_SOLVE_1
- Replaces the former model checker EVALUATOR 3.0 (CADP 2001) and its dedicated solver algorithm
- 3-10 times better in time and memory than Evaluator 3.0

The BISIMULATOR tool

- BISIMULATOR: A tool for checking equivalence on-the-fly
- Inputs:
  - an LTS S1 given implicitly (OPEN/CAESAR)
  - an LTS S2 given explicitly (BCG)
  - an equivalence relation chosen in a list of 7
  - a comparison mode (equal, contains, subset)
- Outputs:
  - a boolean verdict (true or false)
  - a diagnostic (DAG)
- Bisimulator is built on top of CAESAR_SOLVE_1

The REDUCTOR 5.0 tool

- REDUCTOR 5.0: A tool for on-the-fly minimization modulo various relations
- Inputs:
  - an LTS given implicitly (OPEN/CAESAR)
  - a relation chosen in a list of 9
  - optional: a list of hiding/renaming clauses for labels
- Outputs:
  - an explicit LTS (BCG)
  - optional: the set of equivalence classes

What happened to ALDEBARAN?

- Since 1998, the ALDEBARAN tool is no longer maintained by Verimag (25 bugs identified)
- Almost every feature of ALDEBARAN is also available in other recent CADP tools
- Since CADP 2006:
  - ALDEBARAN is replaced by a shell-script that invokes Bisimulator, Reductor, Bcg_Info, etc.
  - The old ALDEBARAN binary is kept for backward compatibility

Tools for compositional verification

Motivation

- Compositional generation: *"divide and conquer" to fight state explosion*
  - Partition the system into subsystems
  - Minimize each subsystem modulo a strong or weak bisimulation preserving the properties to verify
  - Recombine the subsystems to get a system equivalent to the initial one
- Refined compositional verification:
  - Tightly-coupled processes constrain each other
  - Separating them -> explosion
  - "Interfaces" used to model synchronization constraints

The EXP.OPEN 2.0 tool

- Complete rewrite of Exp.Open 1.0 (Mounier)
- Compositional verification of communicating LTSs connected using the operators of various languages
  - CCS, CSP, LOTOS, E-LOTOS, and mCRL parallel composition
  - Generalized hiding, renaming, and cut
  - Synchronization vectors (MEC, FC2)
- Several functionalities
  - On-the-fly state space exploration (using OPEN/CAESAR API)
  - Partial order reductions of the state space
  - Generation of FC2 networks and PEP Petri nets
  - Refined interface generation

The PROJECTOR 3.0 tool

- Follows PROJECTOR 1.0 (Krimm, 1997) and 2.0 (Pace, 2003)
- On-the-fly behavioural abstraction using interfaces
- Inputs:
  - an LTS S given on the fly
  - an interface I (LTS understood as a set of traces)
- Outputs:
  - an abstracted LTS obtained by removing all states and transitions of S that cannot be reached while following the traces in I
  - optionally: validity predicates to check interface correctness (to be checked by EXP.OPEN during later compositions)
- 3 times better in time and memory than PROJECTOR 2.0 on average (up to 36 times better in time on some examples)

The BCG_GRAPH tool

- BCG_GRAPH generates particular forms of graphs useful to compositional verification

1. Chaos automata over a set of labels L
2. FIFO buffers of length N over a set of labels L
3. Bag automata of length N over a set a labels L

The SVL tool

- SVL: script language for compositional verification
- Main enhancements since 2001:
  - Support for the new on-the-fly tools (Bisimulator, Reductor, etc.)
  - Support for EXP.OPEN 2.0, Projector 3.0, Bcg_Graph
  - Support of partial order reductions
- Other enhancements:
  - Improved error and warning messages
  - Improved management of intermediate files
  - Support for script parameters, shell variables, lists of labels

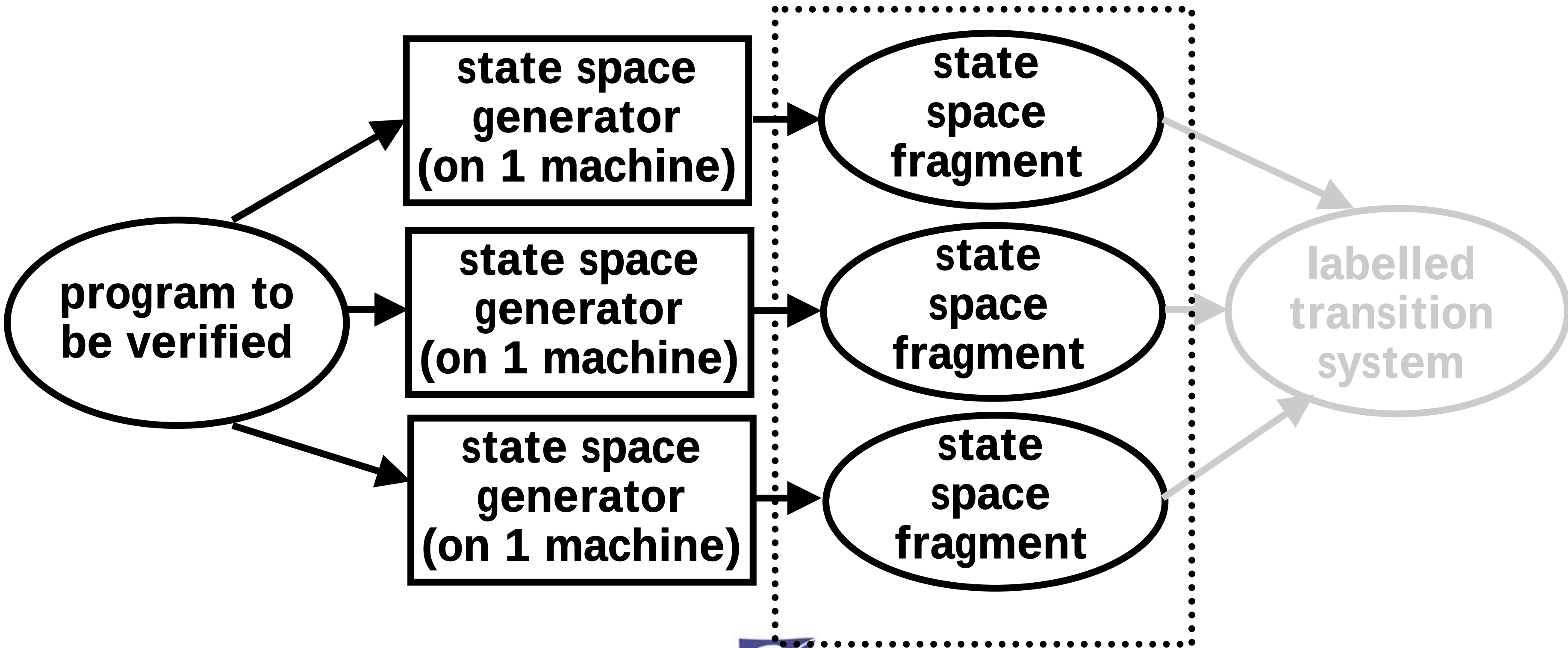
Tools for distributed verification

Motivation

- Verification on a single computer (PC or workstation) suffers from limitations:
  - memory size (3-4 Gbytes max. on 32-bits)
  - CPU time
- Idea:
  - using several machines (network of workstations, clusters of PCs)
  - distributed algorithms
- Tool support
  - Distributor: distributed state space generator
  - Bcg_Merge: merger of distributed state spaces

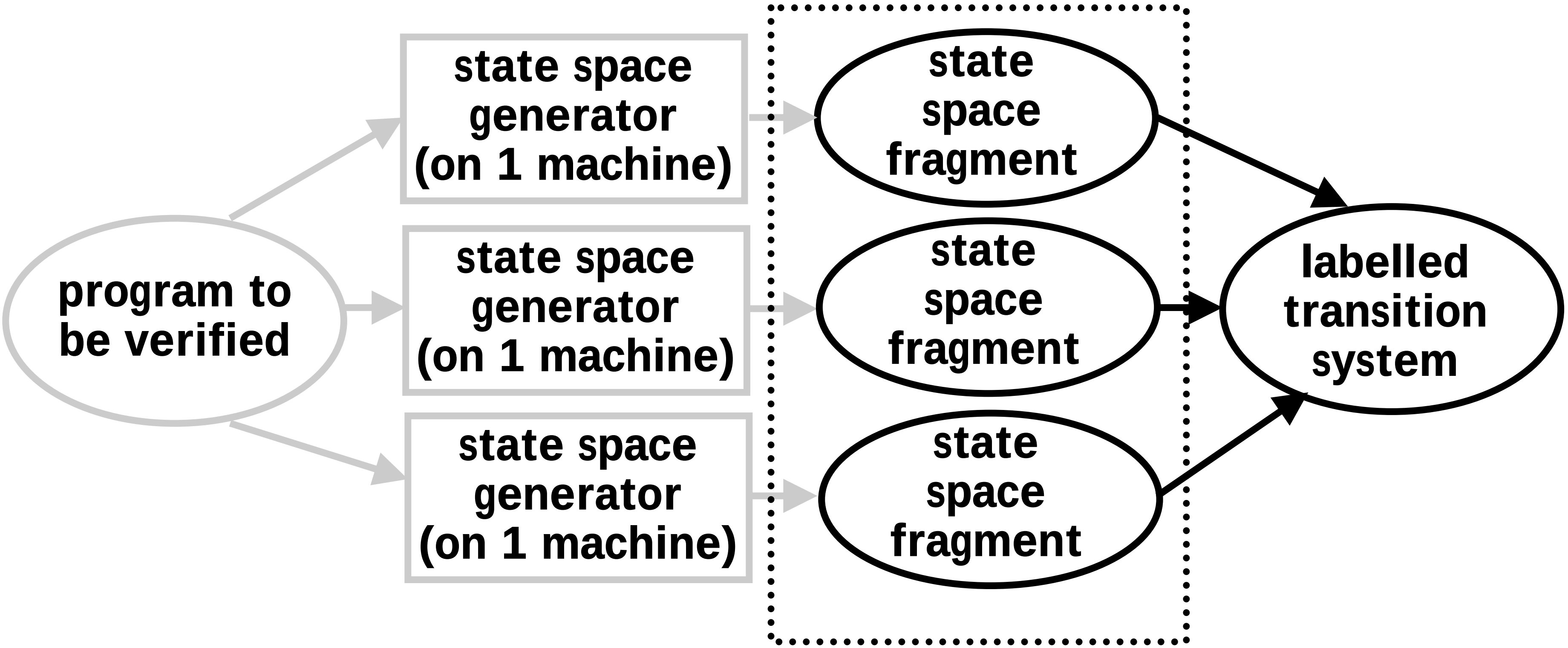
The DISTRIBUTOR tool

- distributed state space generation using a cluster or a grid
- allows tau-compression and tau-confluence reductions preserving branching bisimulation



The BCG_MERGE tool

- merges a distributed state space produced by DISTRIBUTOR into a labelled transition system



Tools for performance evaluation

Motivation

- Using the same models for
  - functional verification
  - performance evaluation

- 4 tools dedicated to performance evaluation:
  - Bcg_Min
  - Bcg_Steady
  - Bcg_Transient
  - Determinator

The BCG_MIN tool

- In addition to standard LTSs, Bcg_Min can also minimize Markov models:
  - probabilistic LTSs "**prob** *p*" transitions
  - stochastic LTSs "**rate** λ" transitions
  - mixed models "*label* ; **prob** *p*" or "*label* ; **rate** λ" transitions

- For such models, bisimulation is connected to the concept of *lumpability*

The BCG_STEADY tool

- Numerical solver for Markov chains
- Steady state analysis (equilibrium)
- Input:
  - Markov chain (BCG graph with "action; rate r" labels)
  - no deadlock allowed
- Output:
  - steady-state probabilities and throughputs on the long run
  - numerical data usable by Excel, Gnuplot...
- Method:
  - BCG graph converted into a sparse matrix
  - computation of a probabilistic vector solution
  - iterative algorithm using Gauss-Seidel [Stewart94]

$$\pi_i^{(k+1)} = -\frac{1}{a_{i,i}} \left(\sum_{j<i} \pi_j^{(k+1)} a_{i,j} + \sum_{j>i} \pi_j^{(k)} a_{i,j} \right)$$

The BCG_TRANSIENT tool

- Numerical solver for Markov chains
- Transient analysis
- Inputs:
  - BCG graph with "*action;* **rate** *r*" labels
  - deadlocks permitted
  - list of time instants
- Outputs:
  - transient probabilities and throughputs at the time instants
  - numerical data usable by Excel, Gnuplot...
- Method:
  - BCG graph converted into a sparse matrix
  - uniformisation method to compute Poisson probabilities
  - Fox-Glynn algorithm [Stewart94]

$$\underline{\tilde{\pi}}(t) = \sum_{n=0}^{k_{ss}} \psi(\lambda t; n)\underline{\hat{\pi}}(n) + \left(\sum_{n=k_{ss}+1}^{k_{\varepsilon}} \psi(\lambda t; n) \right) \underline{\hat{\pi}}(k_{ss}) \quad \textit{with} \quad \psi(\lambda t; 0) = e^{-\lambda t}$$

$$\textit{and} \quad \psi(\lambda t; n+1) = \psi(\lambda t; n) \frac{\lambda t}{n+1}, n \in \mathrm{N}$$

The DETERMINATOR tool

- Extracts a Markov chain from a stochastic LTS
- Checks a sufficient condition for determinism ("well-formed" Markov chain)
- Works on-the-fly (the stochastic LTS is given implicitly)
- Speeds up performance computations
- Used in two case-studies:
  - Life cycle analysis for the gyroscopes of Hubble space telescope
  - Performance evaluation for the SCSI-2 bus arbitration protocol

Tools for testing

The SEQ.OPEN tool

- Trace-based verification of industrial systems
  - Black-box assumption: only I/O events available
  - View traces as *implicit* LTSs
- Generic encoding of execution traces
  - Execution monitoring → event traces (logs)
  - Store trace files on disk
  - Text files using the SEQUENCE format of CADP (one event per line)
- Support for on-the-fly trace exploration
  - SEQ.OPEN tool: connection from SEQUENCE to OPEN/CAESAR API
  - Memory reduction using disk cache techniques
- Applications : Bull's Multiprocessor Systems
  - Random simulation → large traces (1,000,000 events)
  - Coverage analysis (traces w.r.t. specification)

Conclusion

Conclusion

- A verification toolbox for asynchronous systems
- A modular, extensible architecture (APIs)
- Eight platforms supported
  - Five 32-bits platforms and three 64-bits platforms
- International dissemination
  - license agreements signed with 407 organizations
  - in 2008: licenses granted to 562 machines
- Many applications
  - 104 case-studies accomplished using CADP
  - 32 research tools connected to CADP
  - 17 university lectures based on CADP

More information...

http://www.inrialpes.fr/vasy/cadp

and

http://cadp.forumotion.com