CADP Twenty Years After

Hubert Garavel

INRIA Grenoble Rhône-Alpes / VASY team

http://www.inrialpes.fr/vasy

Luca Aceto quoting Christos Papadimitriou:

"Successful exploratory theoretical research is bound to produce predominantly negative results"

In this talk, I will try to establish that:

"Successful application of concurrency theory may produce predominantly positive results".

About CADP…

- CADP is the oldest software program implementing concurrency theory results that is still used and enhanced

- Development started in 1986
- First tool demonstration 20 years ago (final review of European project "SEDOS", Toulouse, October 1987)

CADP today

- A comprehensive toolbox
  - 42 tools
  - 17 software libraries
- 4 computing platforms supported
  - Sparc/Solaris, PC/Linux, PC/Windows, MacOS X
- International dissemination
  - License agreements signed with 372 organizations
  - Licenses granted for 822 machines in 2006
  - 94 case-studies accomplished using CADP
  - 29 research tools connected to CADP
  - 28 university lectures based on CADP (since 2002)

Three main uses of CADP

- Design of critical systems:
  - academic and industrial case-studies
- Teaching concurrency theory:
  - practical feedback of process calculi, LTS, behavioural equivalences, μ-calculus, etc.
  - lab exercises
- Research in verification:
  - new tools developed using CADP libraries
  - new tools interfaced with CADP tools

Outline of the talk

1. A process calculus named LOTOS
2. Implementing process calculi efficiently
3. A modular architecture for explicit-state verification
4. Equivalence checking
5. Model checking
6. End-user interfaces
7. Towards better languages
8. Concluding remarks

1. A process calculus named LOTOS

25 years ago: the OSI project

- A huge project in the networking community:
  - replace old, proprietary protocols with new, standardized protocols (the OSI stack)
  - protocols are complex and involve concurrency
  - OSI approach: a standard comes with a formal description that will serve as a reference for all implementations

  ≠ *IETF approach: a proposed standard needs to be supported by two implementations*
- Different formalisms were competing:
  - Estelle: extended finite state machines
  - LOTOS: process calculus [ISO-1989]

The LOTOS project

- The LOTOS international standard (1983-1989)
  - process part: clever synthesis of CCS, CSP, and Circal
  - data part: abstract data types (the weakest point)
  - formally-defined syntax and semantics
  - large case-studies used to shape LOTOS features

- Key ideas behind LOTOS:
  - process calculi are useful to describe industrial systems
    $\Rightarrow$ they must evolve into computer languages
  - emphasis on software tools
  - critical mass (people, funding) required

Achievements and failures

- An ambitious research agenda for formal methods
- But technical issues:
  - LOTOS was cleaner and more expressive than its competitors, but harder to learn and to implement
  - LOTOS tools did not scale to middle- or large-size problems
  - Over-emphasis on refinement-based methodologies (disruptive, long, and costly for industry)
- And political issues too:
  - LOTOS did not become the unique modelling language:
    - competitors remained for some time: Estelle, SDL, RSL, etc.
    - other process algebras (ACP, CSP, CCS) continued their independent life
  - Formal methods had been oversold to industry and Europe

2. Implementing process calculi efficiently

Implementing LOTOS: a real challenge

- Goal: translate a LOTOS program into its LTS as defined by the formal semantics
- Sub-goal: the LTS should be as small as possible (up to strong equivalence)
- A hot topic research in the late 80's:
  - LOTOS was a very new kind of language
  - its process part was not "imperative" (SOS rules) and had "strange" features (n-party rendezvous choice over value domains, disabling operator)
  - its data part was "nasty" (ADTs, equational semantics, semi-termination issues)

The (former) mainstream approaches

- Semantics-driven implementations:
  - LTS obtained by "executing" the semantics of LOTOS
  - processes handled by a term rewrite applying SOS rules to derive successor states
  - data types passed to an equational or rewrite engine
  - LTS state = syntax tree derived from the LOTOS source program
- Major drawbacks:
  - memory intensive
  - slow, and possibly non-terminating (semi-decidability)
  - equality between states (i.e., loop detection) difficult
- Nowadays, these approaches are gone

The CADP approach

Principle 1: Deviate from the LOTOS standard when appropriate to restrict the problem to practical cases only

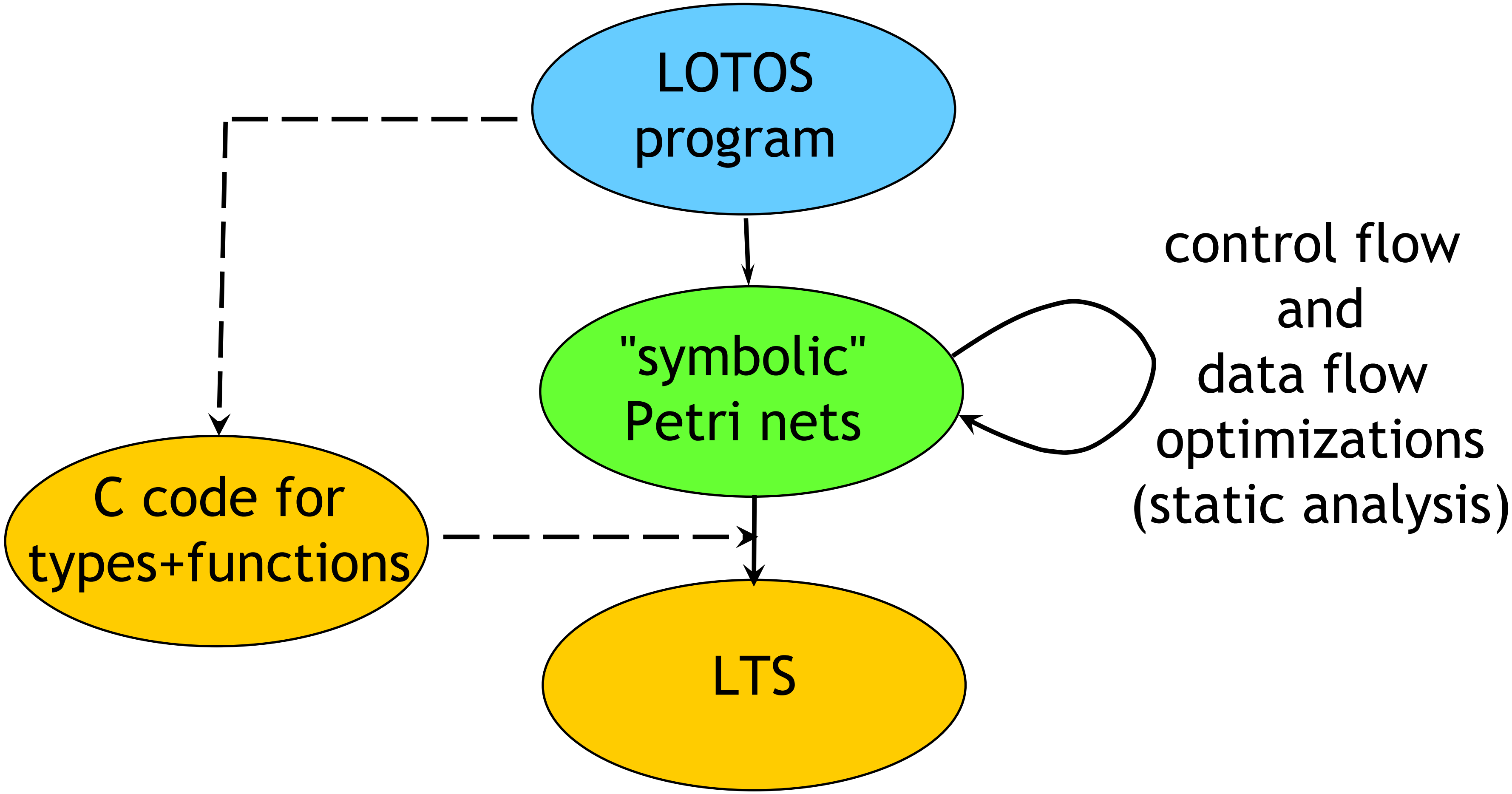
- Process part:
  - avoid infinite recursion through parallel composition (i.e., unbound process creation)
  - avoid recursions through [> or >>, which generate non-regular behaviours
- Data part:
  - distinguish between constructors/non-constructors
  - turn algebraic equations into rewrite rules
  - provide means to interface user-given C code

The CADP approach

Principle 2: Elegant semantics and efficient execution are two distinct issues

- Axioms (for data types) and SOS rules (for processes) are only good to define semantics concisely and to make proofs
- For efficient implementions, they are counter-productive (they don't pay enough attention to the underlying execution machinery)
- Instead, our goal was to:
  - build a LOTOS compiler, not an interpreter
  - use several translation steps, with intermediate models
  - do things at compile-time rather than run-time

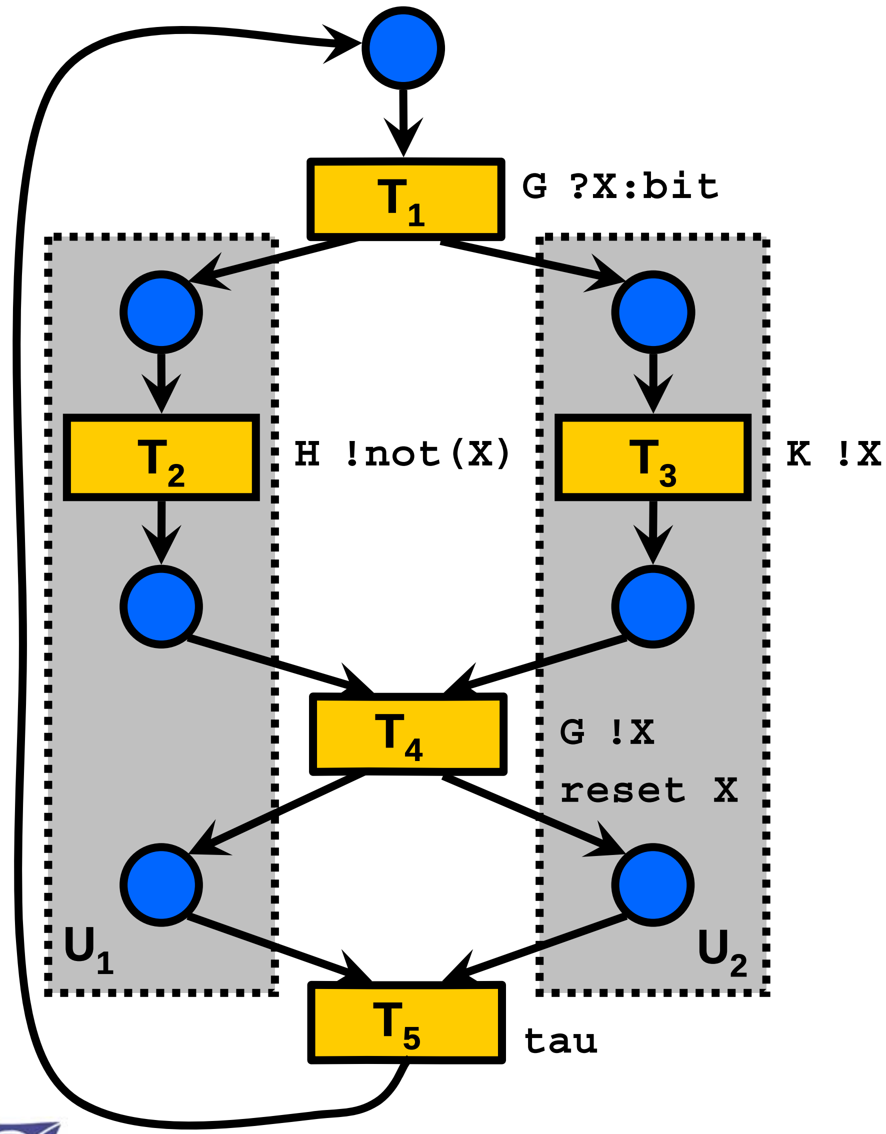
The CAESAR compiler (1989-now)



[Garavel-1989]

Our intermediate model

- Hierarchical Petri net
- Nested units featuring sequential processes
- Visible, tau-, and epsilon-transitions
- Typed variables with a defined scope
- Statements attached to transitions:
  - assignments
  - conditionals
  - iterations
  - variable resets



Which intermediate model?

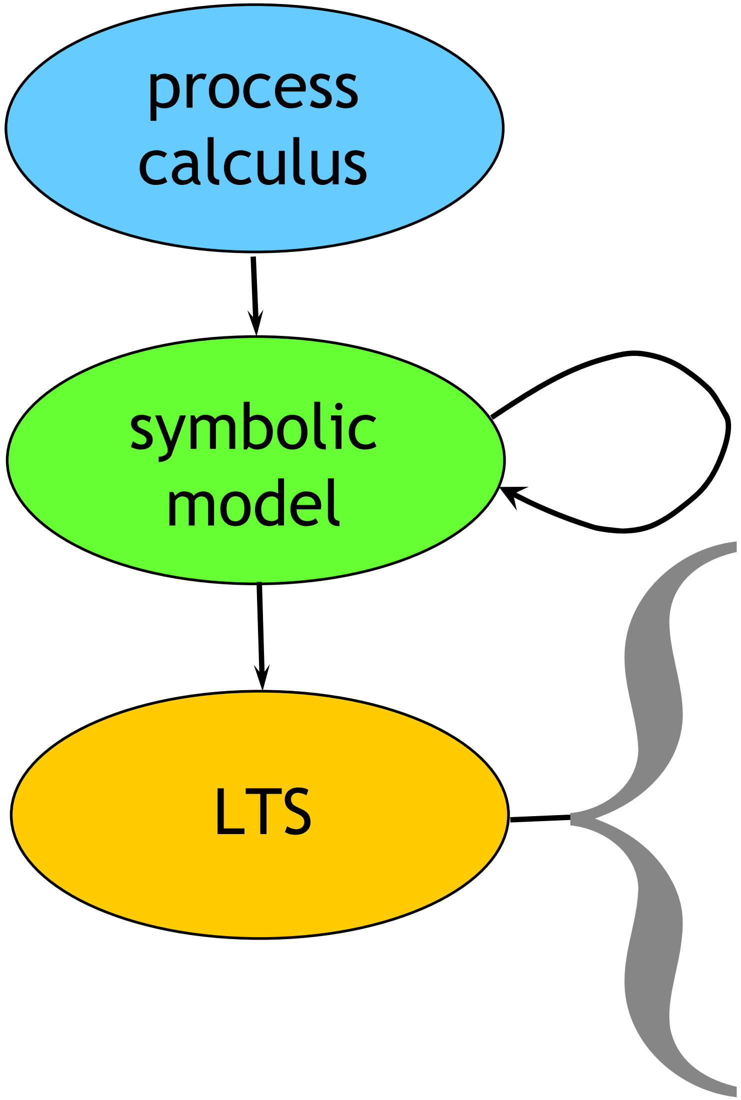
- There are plenty of possible intermediate models in our approach
- "Bad" intermediate models:
  - do not support data ("pure" Petri net)
  - do not support states/transitions (data structures only)
  - do not support concurrency, e.g. A||B ("flat" EFSM)
  - do not support nested processes, e.g. A.(B||C).D
- The CADP model for LOTOS was carefully designed
- "Enhanced" models exist:
  - XFSM [Karjoth-1992]: dynamic creation of processes
  - NTIF [Garavel-Lang-2002]: sequential code fragments

3. A modular architecture for explicit-state verification

A separation principle

- In many model checkers, state space generation is often intricated with verification
- CADP promotes a modular approach by separating clearly:
  - the generation of the LTS (produced by the LOTOS compiler)
  - the verification of the LTS (using visual, equivalence, or model checking)
- Semantic reasons:
  - concurrency theory promotes such an abstraction (this is a key reason behind the LTS model)
- Pragmatic reasons (lasting from the 80's):
  - compiling LOTOS was complex enough for a PhD thesis
  - other colleagues in Grenoble were already working on model checkers (Xesar) and equivalence checkers (Aldebaran)

A modular architecture with 3 levels



control and data flow optimizations

- simulation and code generation
- verification:
  - equivalence checking
  - model checking
  - visual checking
- testing
- performance evaluation, etc.

Explicit LTS: the BCG format

Two practical issues arising in the early 90's

- Interoperability:
  - each bisimulation tool was equiped with its own LTS format

  $\Rightarrow$ a pivot format was needed to allow conversions
- Disk space limitations:
  - almost all LTS formats were textual (ASCII files)
  - large LTSs could not be stored on hard disk

  $\Rightarrow$ a compact format for LTS was needed

Design of BCG (Binary-Coded Graphs) [Garavel-1992]:

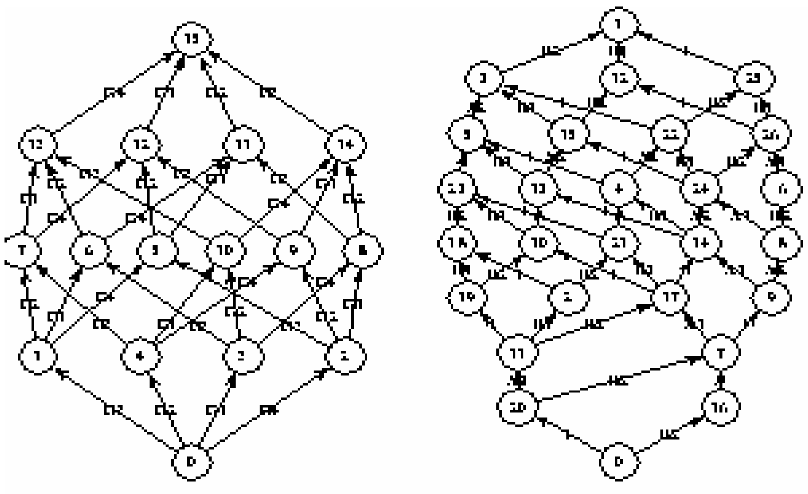
- binary file format for storing LTSs
- support for input/output streaming
- preservation of source-level information (types, functions...)
- specific compression techniques ($\approx$ 2 bytes per transition)

$\Rightarrow$ BCG + BZIP2 is a highly compact way to store a huge LTS

Visual checking

Since BCG is a binary format, the need for graph drawing tools was crucial:

- Development of BCG_DRAW and BCG_EDIT (1995)



- Connection of BCG to many other drawing tools: AUTOGRAPH, GML, GraphViZ, VCG, VISCOPE

Implicit LTS : Open/Caesar

Another practical issue arising in the early 90's

How to combine:

- a separation betwen LTS generation and LTS verification
- and the need for "on-the-fly" verification?

Both were needed, but seemed incompatible at first sight

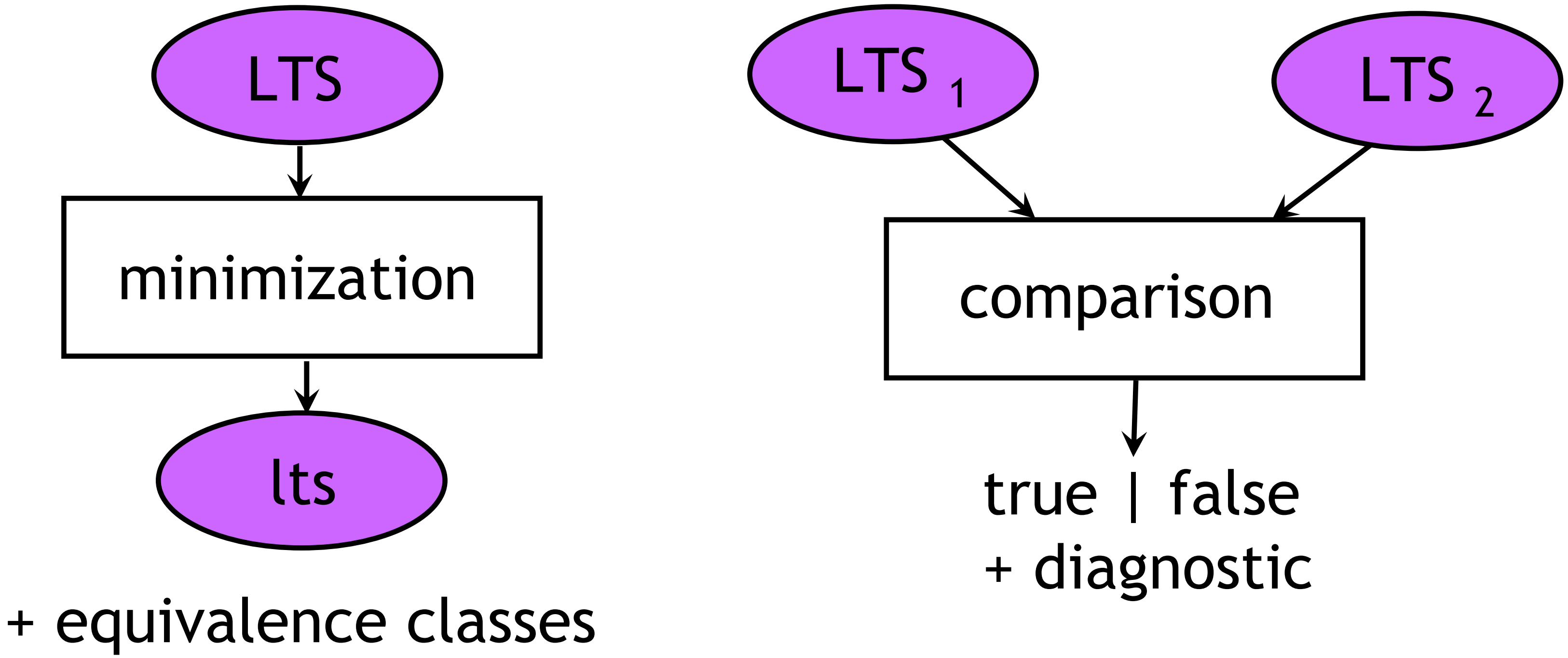
Solution: the Open/Caesar architecture [Garavel-1998]

- A programming interface to separate language-dependent from language-independent aspects
- Many tools have been written above this interface: simulation, testing, verification, etc.
- Other languages than LOTOS have been connected to this interface
- An essential feature of CADP, often replicated in other papers/tools

4. Equivalence checking

Practical uses of equivalence relations

- Equivalences introduced by Milner in CCS
- In practice, not used at the process calculi level, but rather at the LTS level
- Two main usages:



Tools for equivalence checking

- Many equivalences: strong, branching, weak, safety, trace, etc.
- Many algorithms: explicit, implicit, symbolic (BDDs)
- Successive tools in CADP:
  - ALDEBARAN [Fernandez, Mounier, Kerbrat]
  - BCG_MIN [Garavel, Hermanns, Cherif, Bergamini]
  - BISIMULATOR [Mateescu, Bergamini]
  - REDUCTOR [Mateescu, Lang]
- Connection to other tools:
  - SCAN, AUTO, Fc2Tools, CWB, LTSMIN, …
  - many bisimulation tools (but only a few still maintained)

Compositional verification

- A "*divide and conquer*" approach to avoid state space explosion
- In an action-based setting: it relies on the fact that many equivalences are congruences for parallel composition
- Two variants:
  - "simple" compositional verification
    a.k.a. compositional reachability analysis
  - "refined" compositional verification with interfaces [Graf-Steffen-1990] [Krimm-Mounier-1997]

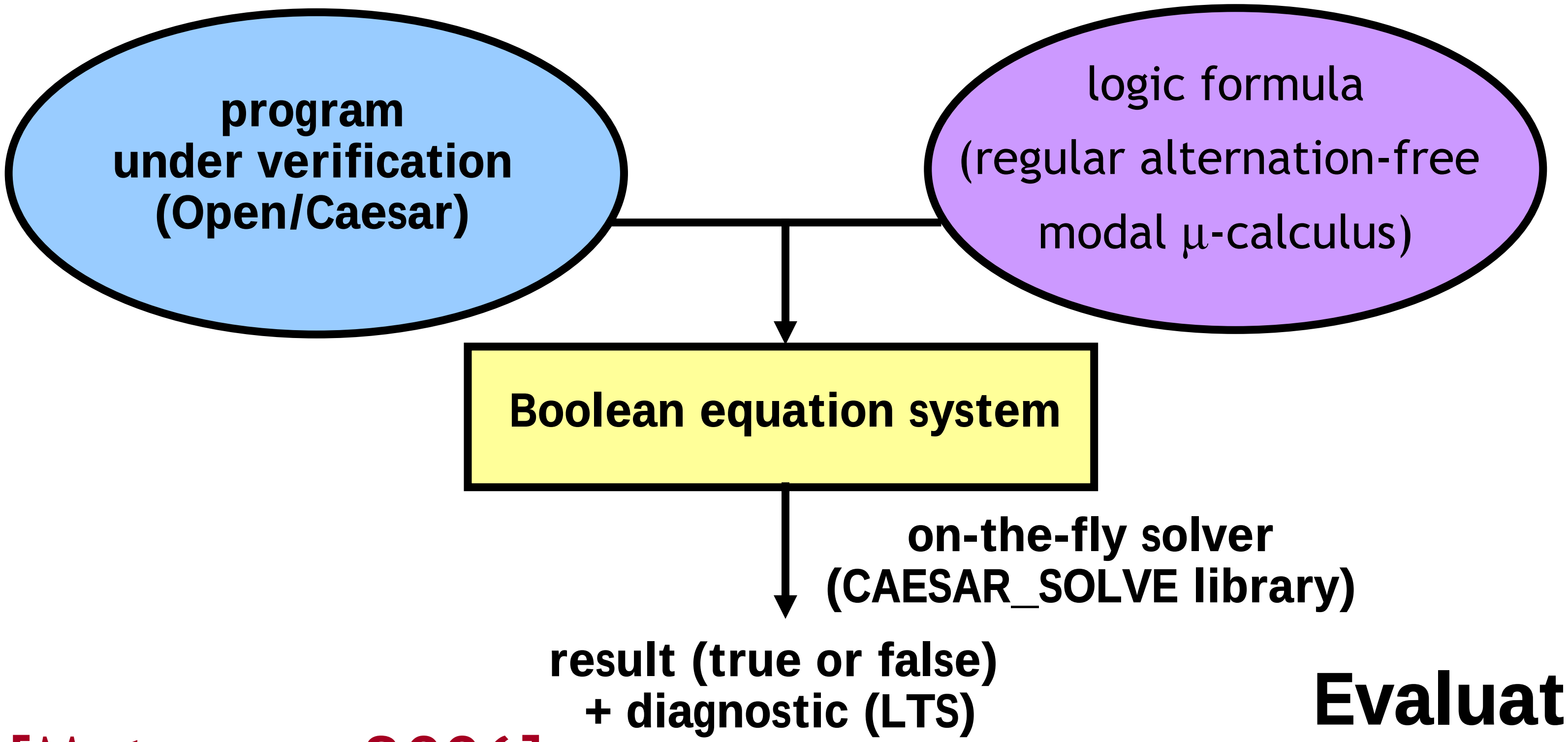
Compositional verification

- Fully supported in CADP:
  - Exp.Open 2.0 [Lang, Garavel]
  - Projector 2.0 [Pace, Ondet, Descoubes, Lang]
  - SVL [Garavel-Lang-2001] [Lang-2002]
- A practical way to verify large systems:
  so far, up to 70 concurrent processes ≈ 9.10^{64} states
  [Tronel-Lang-Garavel-2003]
- Compositional verification is a very strong reason to prefer process calculi (message passing) rather than communicating state machines (shared variables)

5. Model checking

"Standard" model-checking

- LTS model => "action-based" properties
- The information is in the transition labels (rather than in the states)



[Mateescu-2006]

Evaluator 3.5 model checker

Need for an extended LTS model

- In the standard LTS model:
  - transition labels are actions belonging to an alphabet
- In practice, labels contain typed data
  - Exemple: "SEND !23 !true"
- One often needs to handle these data
  - SEND !X !true where F (X) < 15

$\Rightarrow$ Extended LTS model:

- it handles structured labels
- it exports the user-defined types/functions present in the source program
- this model is supported by the BCG format

Need for value-passing logic formulas

- Examples of value-passing properties:
  - On every execution path, the value of x in all occurrences of "SEND !x" is strictly increasing
  - For each x, between all successive occurrences of "OPEN !x" and "CLOSE !x" actions, there may not be an "OPEN !y" action (critical section)
- First approach to define such a logic: the RICO logic [Garavel-1989]

2^{nd} approach: XTL

- For explicit LTSs (encoded in the BCG format)
- XTL (*eXtended Temporal Logic*): [Mateescu-Garavel-1998]
  - a functional framework for implementing model checkers
  - usual branching-time logics (CTL, HML...) can be expressed in XTL
  - value-passing extensions of these logics can also be described

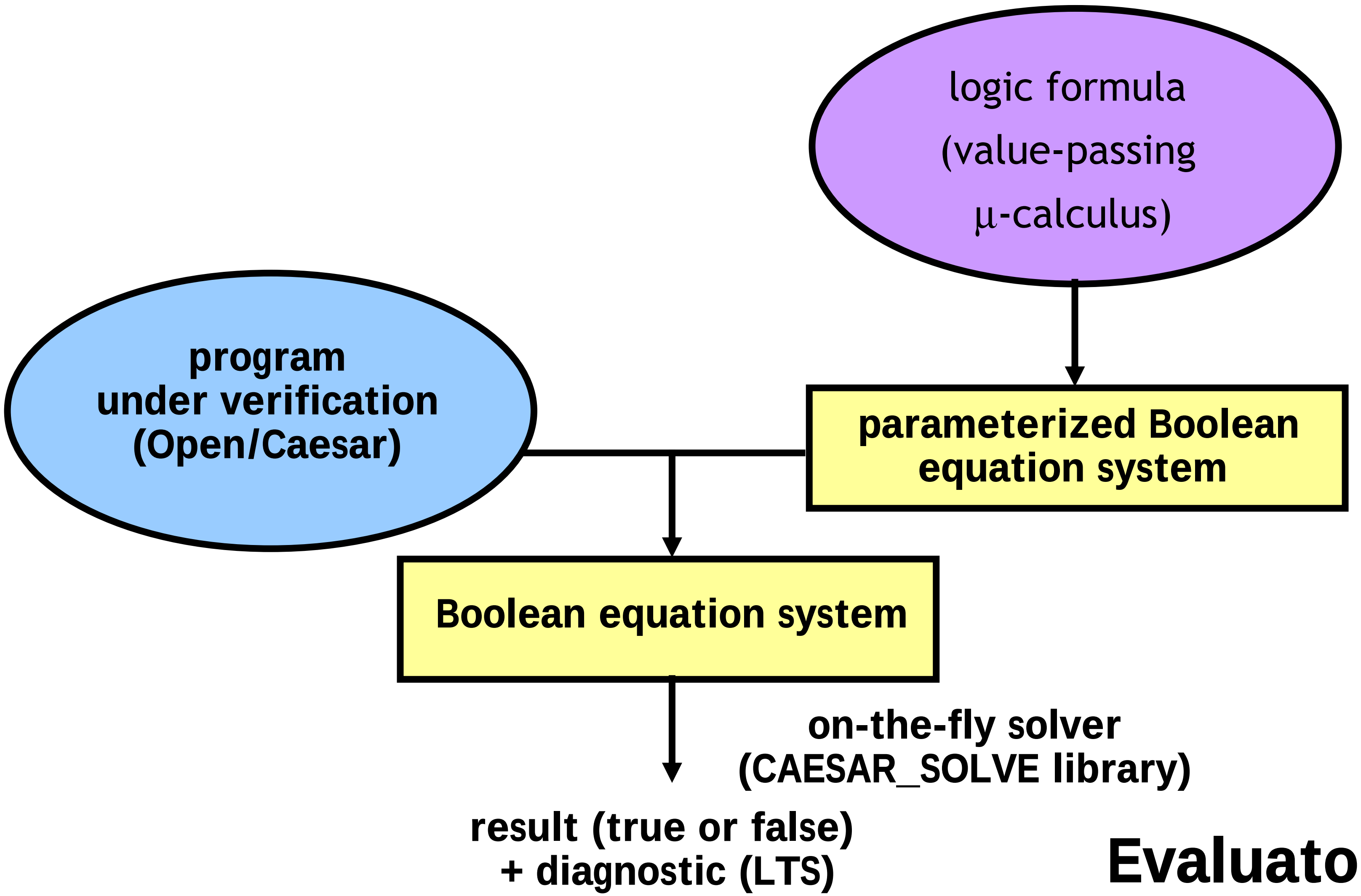
3^{rd} approach: EVALUATOR 4.0

- For implicit LTSs (explored on-the-fly using OpenCaesar)

- New concepts:
  - Value-passing μ-calculus = modal μ-calculus with typed variables, if-then-else, case, … statements
  - Parameterized Boolean equation systems
  - [Mateescu-1998a] [Mateescu-1998-b]

- Implementation:
  - Evaluator 4 model checker (to be released soon)

Architecture of EVALUATOR 4.0



Evaluator 4.0 model checker

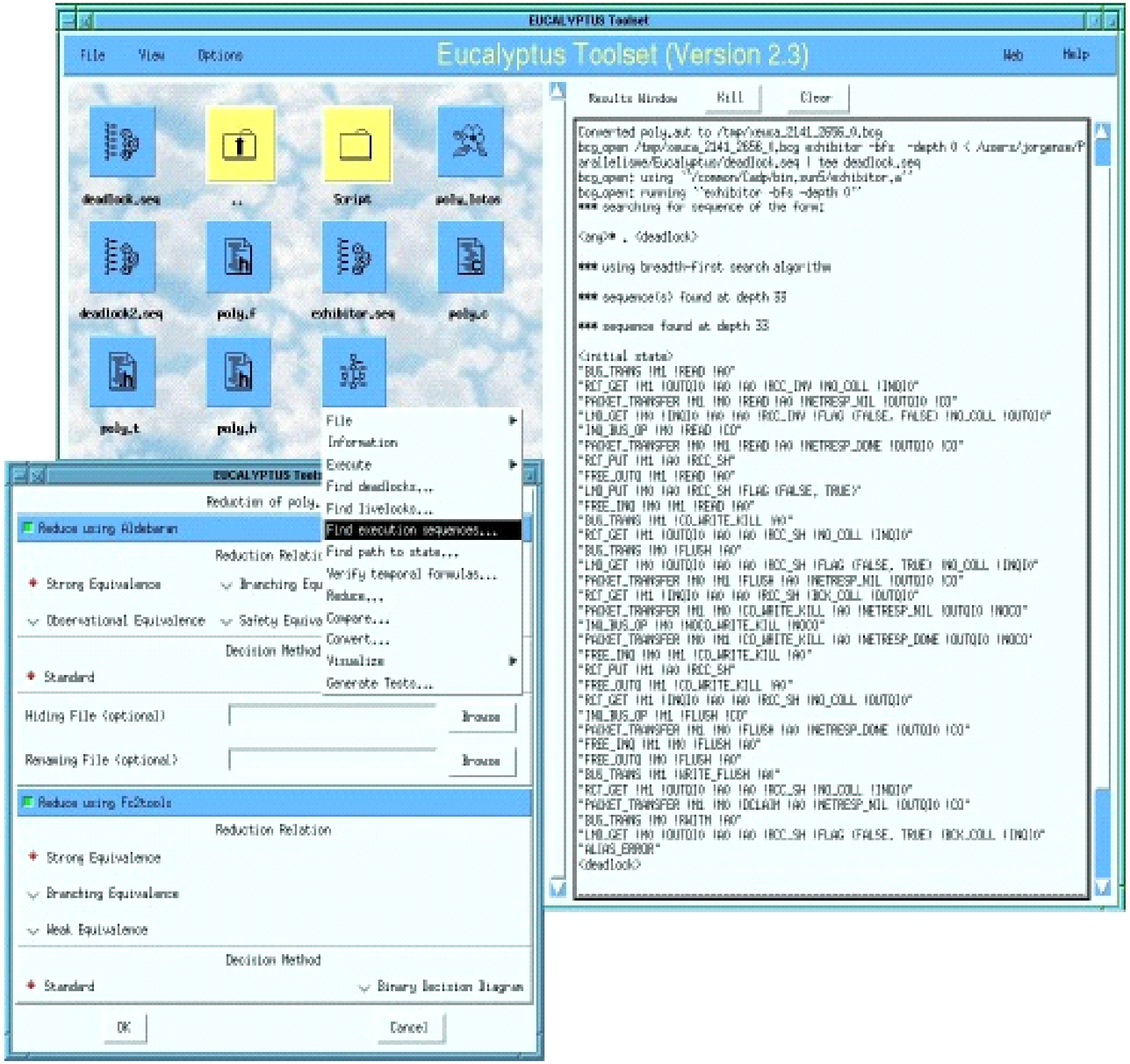
6. End-user interfaces

A key feature for industrial use

- Early verification tools only had simple command-line interfaces, e.g.
  - ad hoc command interpreters (QUASAR, CWB)
  - LISP or Tcl/Tk commands (Meije, FcTools)
- More elaborate interfaces have been developed for CADP
- Two lines of work:
  - a graphical user interface (EUCALYPTUS)
  - a scripting language for verification (SVL)

EUCALYPTUS graphical-user interface

- Version 1 (1994)
- Version 2 (1996-now)
- Main features:
  - file types
  - user-friendly contextual menus
  - support all the CADP tools



SVL (*Script Verification Language*)

- Scripting language for verification scenarios [Garavel-Lang-2001] [Lang-2002]
- Special constructs for:
  - equivalence checking
  - model checking
  - compositional verification
- "Semantics-aware"

```
"F.exp" = leaf branching reduction of
  hide G in
    (
    "spec.lotos":P1 [A, B, G]
    |[G]|
    "spec.lotos":P2 [C, G]
    ) ;
"D.seq" = deadlock of  "F.exp";
"L.seq" = livelock of "F.exp";
```

an SVL script

7. Towards better languages

Enhancements to LOTOS

- 1993-2001: Standardization project at ISO to enhance E-LOTOS
- Initial goal: a simple revision of LOTOS
- Final result: E-LOTOS [ISO-2001]
  - complete rewrite of LOTOS
  - abstract data types replaced by functional types
  - process operators replaced by equivalent functional / imperative constructs
  - new features: time, exceptions, modules

E-LOTOS: A balanced result

- Positive aspects of E-LOTOS:
  - better than LOTOS in most respects
  - simpler syntax (away from the "algebraic" mania)
  - formal semantics (timed LTS, SOS rules)
  - industrial users seem to prefer E-LOTOS to LOTOS
- Negative aspects of E-LOTOS:
  - semantics too complex, irregular at places
  - lack of funding for E-LOTOS (perhaps because LOTOS was oversold)
  - never implemented entirely

On-going work at VASY

- LOTOS NT:
  - a reasonable (untimed) subset of E-LOTOS
- TRAIAN (1996-now):
  - a LOTOS NT → C compiler
  - so far, only LOTOS NT data types are compiled
  - intensively used to build VASY compilers
- LNT2LOTOS (2005-now):
  - a LOTOS NT → LOTOS translator
  - data types translation finished
  - process translation being implemented
  - already used successfully by Bull

8. Concluding remarks

Applied concurrency theory

- CADP is based on concurrency theory results

- Yet, its development was driven by practical challenges:
  - industrial needs observed in real-life case-studies
  - limited computing resources (memory, disk space, CPU time)
  - limited human resources (manpower, project schedules…)
  - software engineering guidelines (interfaces for work division)

Innovation brought by CADP

- Innovation can arise from practical constraints:
  - intermediate models for compiling process calculi efficiently
  - static analysis for state space reduction
  - separation of state space generation and verification
  - compression techniques for storing LTSs to disk
  - value-passing μ-calculus
  - parameterized Boolean Equation Systems
  - end-user interfaces for verification
  - enhanced languages acceptable by industry
  - etc.

Dissemination of CADP ideas

- CADP is influential in the academic community:
  - From the beginning, we made the right assumptions and design choices
  - Many case-studies and prototypes done using CADP
  - Recent toolboxes using explicit-state verification replicate the same architecture as CADP
- Industrial dissemination is in progress:
  - CADP is being used for hardware design
  - MULTIVAL project on multiprocessor architectures (Bull, CEA/Leti, INRIA, ST Microelectronics)

A few references (1/3)

- [Garavel-1989]
  Compilation et vérification de programmes LOTOS. PhD thesis, Univ. Grenoble
- [Garavel-1992]
  Binary-Coded Graphs. Technical Report, Grenoble
- [Garavel-1998]
  OPEN/CAESAR: An Open Software Architecture for Verification, Simulation, and Testing. *Proc. TACAS'98, LNCS 1384*
- [Garavel-Lang-2001]
  SVL: A Scripting Language for Compositional Verification. *Proc. FORTE'2001, Kluwer Academic Publishers*
- [Garavel-Lang-2002]
  NTIF: A General Symbolic Model for Communicating Sequential Processes with Data. *Proc. FORTE'2002, LNCS 2529*
- [Garavel-Lang-Mateescu-Serwe-2007]
  CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. Proc CAV'2007, LNCS 4590

A few references (2/3)

- [Graf-Steffen-1990]
  Compositional Minimization of Finite State Systems. *Proc. CAV'90, LNCS 531*
- [ISO-1989]
  LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. ISO/IEC International Standard 8807:1989
- [Karjoth-1992]
  Implementing LOTOS Specifications by Communicating State Machines. Proc. CONCUR '92, LNCS 630
- [ISO-2001]
  Enhancements to LOTOS (E-LOTOS). ISO/IEC International Standard 15437:2001
- [Krimm-Mounier-1997]
  Compositional State Space Generation from LOTOS Programs. *Proc. TACAS'97, LNCS 1217*
- [Lang-2002]
  Compositional Verification using SVL Scripts. *Proc. TACAS'2002, LNCS 2280*

A few references (3/3)

- [Mateescu-1998-a]
  Vérification des propriétés temporelles des programmes parallèles. PhD thesis, INPG, Grenoble
- [Mateescu-1998-b]
  Local Model-Checking of an Alternation-Free Value-Based Modal Mu-Calculus. Proc. VMCAI'98
- [Mateescu-Garavel-1998]
  XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. *Proc. STTT'98 workshop (BRICS)*
- [Mateescu-2006]
  CAESAR_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-Free Boolean Equation Systems. *Springer Journal STTT, 8(1)*
- [Tronel-Lang-Garavel-2003]
  Compositional Verification Using CADP of the ScalAgent Deployment Protocol for Software Components. *Proc. FMOODS'2003, LNCS 2884*