Formal Modeling and Verification of Concurrent Systems using CADP

Hubert Garavel, Frédéric Lang, Radu Mateescu, Gwen Salaün, and Wendelin Serwe

http://convecs.inria.fr
Motivation

- Asynchronous concurrency becomes ubiquitous

Increasingly complex safety-critical asynchronous systems

\[\text{formal verification required more than ever in the design process}\]

Scientific challenges:
- Facilitate acceptance of formal verification in industry
- Augment verification power to handle complex systems
Asynchronous Concurrent Systems

Characteristics:
- Set of concurrent agents
- No global clock
- Nondeterminism

Multiple application domains:
- Safety-critical systems
- Multiprocessor architectures
- Communication protocols
- . . .
CADP

Construction and Analysis of Distributed Processes

- A modular toolbox for asynchronous systems
- 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: only 2 tools (Caesar and Aldébaran)
  - last stable version: CADP 2006
  - today: nearly 50 tools in CADP 2013 (close to stable)
CADP: Main Features

**Specification languages**
- Formal semantics
- Based on process calculi
- User-friendly syntax

**Verification paradigms**
- Model checking
  - (extended modal μ-calculus)
- Equivalence checking
  - (bisimulations)
- Visual checking
  - (graph drawing)

**Verification techniques**
- Reachability analysis
- On-the-fly verification
- Compositional verification
- Distributed verification
- Static analysis

**Other features**
- Step-by-step simulation
- Rapid prototyping
- Test-case generation
- Performance evaluation
CADP w.r.t. other Model Checkers

- Parallel programs (rather than sequential programs)
- Message passing (rather than shared memory)
- Languages with a formal semantics (process calculi)
- Dynamic data structures (records, lists, trees…)
- Explicit-state (rather than symbolic)
- Action-based (rather than state-based)
- Branching-time logic (rather than linear-time logic)
Application Domains

Case-studies handled with CADP cover the following domains:

avionics, bioinformatics, business processes, cognitive systems, communication protocols, component-based systems, constraint programming, control systems, coordination architectures, critical infrastructures, cryptography, database protocols, distributed algorithms, distributed systems, e-commerce, e-democracy, embedded software, grid services, hardware design, hardware/software co-design, healthcare, human-computer interaction, industrial manufacturing systems, middleware, mobile agents, model-driven engineering, networks, object-oriented languages, performance evaluation, planning, radiotherapy equipments, real-time systems, security, sensor networks, service-oriented computing, software adaptation, software architectures, stochastic systems, systems on chip, telephony, transport safety, Web services

list of (published) case studies: http://cadp.inria.fr/case-studies
Languages Supported by CADP

- SAM
- EB3
- WSDL-BPEL
- π-calculus
- SDL
- AADL
- BIP 1
- FSP
- LNT
- Fiacre
- CHP
- EXP
- LOTOS
- SystemC TLM
- Open/Cæsar
Short History of LOTOS NT & LNT


2000: release of TRAIAN
- data part of LOTOS NT into C
- since then, compiler development at VASY based on TRAIAN: SVL, Exp.Open 2.0, Evaluator 3.0, NTIF, chp2lotos, Int2lotos, ...

- use of LOTOS NT to model critical parts of Bull's high-end servers
- funding for the development of a LOTOS NT to LOTOS translator

2006, 2008: release of Int2lotos (data part, then full LOTOS NT)

2010: integration into CADP (release of Int.open)

2011: renaming of LOTOS NT to LNT
Glimpse of LNT
(dynamic reconfiguration protocol for cloud applications)

process MAIN [INBUS, OUTBUS:Messages] is
par INBUS, OUTBUS in
par
Agent [INBUS, OUTBUS] (a1@s1)
||
Agent [INBUS, OUTBUS] (a2@s1)
||
Configurator [INBUS, OUTBUS]
({ a1@s1, a2@s1 })
end par
||
Bus [INBUS, OUTBUS]
end par
end process

process Agent [SEND, RECV:Messages]
(A:Addr) is
var S:State, R:AddrSet, A1, A2, A3:Addr in
S := DEAD; R := {} of AddrSet;
loop
case S in
ACTIVE ->
select
RECV (A, cfaddr, BIND, ?A2, d)
where (A ne A2) and (R eq {});
SEND (cfaddr, A, ACK, d, d);
R := insert (A2, R)
[
RECV (A, cfaddr, PASSIVATE, d, d);
.
end case
end loop
end var
end process

Labelled Transition System

**LNT.OPEN tool**

Dynamic reconfiguration protocol:
- Exhaustive simulation of a small configuration (2 agents, 2 sites)
- LTS
  - 181 states,
  - 235 transitions
Explicit LTS Representation: BCG Format

Text-based formats are not satisfactory to store large LTSs in computer files
- disk space consuming (Gbytes)
- slow (read/write operations are costly)

BCG (Binary-Coded Graphs):
- a compact file format for storing LTSs
- a set of APIs
- a set of software libraries
- a set of tools (binary programs and scripts)
Implicit LTS Representation: OPEN/CÆSAR

LOTOS → caesar.open
LOTOS NT → Int.open
FSP → fsp.open
LTS → bcg_open
communicating LTSs → exp.open...

Open/Cæsar API

SystemC/TLM

Libraries:

Open/Cæsar

Features:
- LTS generation
- Interactive simulation
- Random execution
- On the fly verification
- Partial verification
- Test generation

After a command for moving an agent has been sent, the agent cannot receive any application event until it completes its migration.

\[
\begin{align*}
&\text{true}^*.
\{ \text{INBUS } ?A1:\text{Nat } ?S1:\text{Nat } \text{any } \text{any } !"\text{MOVE}" ?A2:\text{Nat } ?S2:\text{Nat } \ldots \} . \\
&(\text{not } \{ \text{INBUS } \text{any } \text{any } !A2 !S2 !"\text{ACK}" \ldots \})^* . \\
&\{ \text{OUTBUS } !A1 !S1 \text{any } \text{any } !"\text{SERVICE}" \ldots \} \\
&\text{false}
\end{align*}
\]
On-the-fly Verification for MCL (Evaluator 4.0)

- **LNT specification**
  - compilation
  - implicit LTS
  - Caesar_Solve
  - Open/Caesar environment

- **translation**
  - parameterized HMLR
  - encoding
  - parameterized BES
  - instantiation & resolution

- **optimisation**

- **MCL formula**

- **verdict & diagnostic**

- **On-the-fly activities**
Massively Parallel LTS Manipulation

Distributed verification: exploit the computing resources (memory) of distributed computing infrastructures

Multi-cluster experiments on Grid’5000 using 512 processes
User Interface: Graphics vs Scripts

1. Graphical User-Interface
   - EUCALYPTUS

2. Scripting Language
   - SVL

3. CADP Command-line Tools

4. CADP Code Libraries and APIs
Interactive vs Batch Mode

**Eucalyptus interface**

**SVL script**

"SPEC.bcg" = generation of "SPEC.Int";

"SPEC_r.bcg" = strong reduction of "SPEC.bcg";

% for N in 1 2 3 4 5 6 7 8 9 10
% do
    verify "prop$N.mcl" in "SPEC_strong.bcg";
% done
Bringing Functional Verification and Performance Evaluation Closer

Extended Markovian model = 
LTS (Labelled Transition System) 
+ 
probabilistic transitions ("prob 0.8") 
+ 
stochastic transitions ("rate 3.1")

- Transient and steady-state analysis 
  (BCG_TRANSIENT, BCG_STEADY)  
- On-the-fly simulation (CUNCTATOR)
Further Features of CADP

- Equivalence checking (BCG_MIN and BISIMULATOR)
- Compositional verification (EXP.OPEN and SVL)
- Cosimulation and rapid prototyping (EXEC/CAESAR)
- Test generation (TGV)
- XTL query language on BCG graphs
Some Figures about CADP

Wide dissemination
- In 2012, the 10000\textsuperscript{th} license was granted
- $\geq$ 441 academic license contracts
- $\geq$ 155 published case studies using CADP since 1990 ([http://cadp.inria.fr/case-studies](http://cadp.inria.fr/case-studies))
- $\geq$ 72 third-party tools connected to CADP since 1996 ([http://cadp.inria.fr/software](http://cadp.inria.fr/software))
- $\geq$ 230 users and $\geq$ 1400 messages in the CADP forum since 2007 ([http://cadp.inria.fr/forum.html](http://cadp.inria.fr/forum.html))

Various supported architectures
- processors: Itanium, PowerPC, Sparc, x86, x64
- operating systems: Linux, MacOS X, Solaris, Windows
- C compilers: gcc3, gcc4, Intel, Sun
Some Ongoing Research Projects

**Orange Labs [OpenCloudware]**
- Formal modeling and analysis of cloud computing protocols
  (deployment, reconfiguration, …)

**STMicroelectronics**
- Specification, testing and verification of cache coherency protocols

**Crouzet Automatismes [Bluesky]**
- Formal modeling and validation of PLC networks (GALS)

**EDF and ATOS [Connexion]**
- Formal specification and analysis of HCI for nuclear plants
A Promising Future

Ubiquitous concurrency
- Hardware: many-core CPUs, clusters, grids, clouds
- Software: concurrency required to exploit new hardware

Impact on verification
- Formal analysis of concurrent systems
- Massively parallel verification

Industry awareness
- Increasing need for hardware and software reliability
- Models (even non-formal) become standard practice

"Applied concurrency" starts being effective
For More Information ...

CADP Web site:
http://cadp.inria.fr

CADP forum:
http://cadp.inria.fr/forum.html
http://cadp.forumotion.com

CADP on-line manual pages:
http://cadp.inria.fr/man