Is CADP an Applicable Formal Method?

Hubert Garavel
Radu Mateescu
Frédéric Lang
Wendelin Serwe

CADP

- *Construction and Analysis of Distributed Processes*
- Comprehensive toolbox
- Rooted in **concurrency theory**
- Various Verification approaches & techniques
- Complete design cycle of asynchronous systems:
  specification, interactive simulation, rapid prototyping, verification, testing, performance evaluation
- Continuously improved since 1990
- Distributed worldwide
- http://cadp.inria.fr

Applicability

How can the approach be applied in practice?

- **Students** learning concurrency theory
  - **instantiation of theoretical concepts** (process, automata, synchronization, ...)
  - list of lectures: http://cadp.inria.fr/training
- **Scientists/Engineers** building complex systems
  - assistance in all main design phases
  - most frequently: **formal modelling and verification**
  - but also: performance evaluation, conformance testing, and rapid prototyping
  - list of case studies: http://cadp.inria.fr/case-studies
  - list of tools: http://cadp.inria.fr/software

Automation

Which tool support is proposed?
If abstraction is needed, how is it automated?

- Completely automatic simulation tools
- Need for experts to devise verification strategies
  - on-the-fly techniques
  - compositional techniques
  - SVL (*Script Verification Language*)
- Modelling languages with rich data types
  - ease the step from informal specifications to models
  - convenient targets for domain specific languages

Automation

Translation from SystemVerilog to LNT

```
-- main SV module
module address_decoder (
   ch_bit.in add_in,
   ch_data_t.in d_in,
   ch_data_t.out d_out0,
   ch_data_t.out d_out1
);
   always begin
      bit address;
      data_t data;
      fork
         add_in.BeginRead(address);
         d_in.BeginRead(data);
      join
      case (address)
         1'b0: d_out0.Write(data);
         1'b1: d_out1.Write(data);
      end case
      fork
         add_in.EndRead();
         d_in.EndRead();
      join
   end
end module
```

```
-- main LNT process
process main[
   add_in : ch_bit,
   d_in,
   d_out0,
   d_out1 : ch_data_t]
 is
   loop  var
      address : bit,
      data : data_t in
         par
            add_in(?address)
         || d_in(?data)
         end par;
         case address in
            0 -> d_out0(data); d_out0
          | 1 -> d_out1(data); d_out1
         end case;
         par
            add_in
         || d_in
         end par
   end var end loop
end process
```

Integration

What are the benefits of integrating several approaches?

- Tools and libraries for various abstraction levels
- Documented interfaces
- OPEN/CÆSAR architecture separating
  - language-dependent and
  - language-independent aspects

vides transitions between otherwise opaque and monolithic states. For example, the OPEN/CÆSAR interface [1] has been underlying the success of the CADP toolkit [2].

- Reuse of existing C-code (mostly data handling)
- Ease development of new tools and prototypes

Scalability

How can the approach be applied at scale?

- Optimised to reduce memory before run time
- Distributed tools
- Main asset:
  Compositional techniques

The advantage of using compositional construction in terms of space and time is apparent. Stepwise minimization keeps the size of state spaces low. This, in turns, reduces the duration of the minimization time in the next step, and so on, thus saving significant amount of time.

- Gold medals in parallel tracks of RERS challenges

Transfer

How is teaching or training to be organized?

- Towards a flat learning curve
- Goal: **autonomous users**
  analyzing confidential systems in-house
- User-friendly languages with familiar syntax
  LNT: modelling asynchronous systems
  MCL: model checking language
- Comprehensive documentation

```
< true* . {Step ... ?F:NatSet
            where is_in(tid,F)} >
 < for tid:Nat from 0 to MAX_ID do
     if is_in(tid, F) then
       true* . {Step !tid ...}
     end if
   end for
 > @
```

Usefulness

Is the approach effective?

- > 200 case studies & > 100 connected tools
- Early error detection

In October 2014, STMicroelectronics architects detected a limitation in the IP implementation of the CCI. This limitation manifests in a subset of the counterexamples for the data integrity property we verified 20 months before. Pre-

- Leveraging modelling effort over several activities

all the testing activity would be completely automated. The time spent in specifying the BULL's CC_NUMA architecture, formalizing test purposes and generating the test cases with TGV is completely paid by the better correctness and the confidence to put in the implementation. This approach permitted to detect 5 bugs concerning principally the address collision, and

- Counterexample generation

Ease of Use

How is ease of use achieved? Is the approach effective?

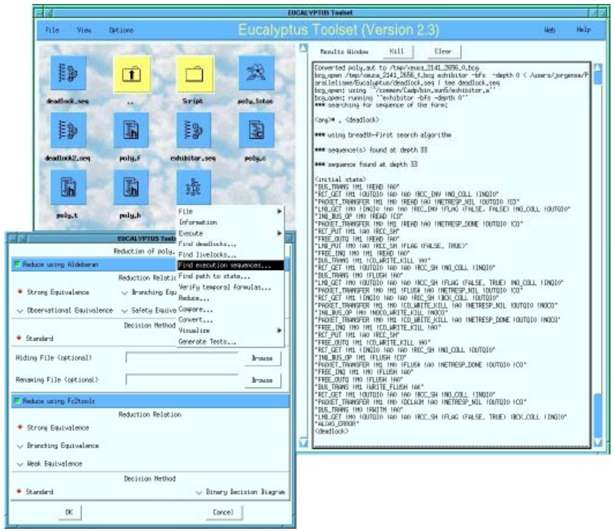
- From mathematics to concrete computer science: **flat learning curve & intuitive syntax**

Contributions. We illustrate several advantages of modeling and analyzing the DTD using LNT, a new formal language based on process algebra and functional programming. First, although modeling the DTD in a classical formal specification language, such as LOTOS [6], is theoretically possible, using LNT made the development of a formal model practically feasible. In particular, features such as predefined array data-types, loops, and modifiable variables helped to obtain a model easily understandable by hardware architects. Second, the automatic analysis capabilities offered by CADP (e.g. step-by-step simulation, model checking, co-simulation) enabled to uncover a problem in the borderline use case with both heavy application

- SVL (*Script Verification Language*)

To enable mechanized interaction, CADP provides a scripting language, SVL, which is particularly convenient to experiment with different strategies to alternate construction and minimization steps. Note that due to the considerations in

- Graphical user interface
- Carefully selected default options



Evaluation

Why will the approach be useful
for a wide range of critical applications?

- Numerous case-studies with critical systems
  http://cadp.inria.fr/case-studies
- Generic theoretical concepts
- Modular architecture and interfaces
- *Promotion of formal methods* by contributions to challenges, contests, and model repositories

Model Checking Contest

Models for Formal Analysis of Real Systems (MARS)

Conclusion

- **Software primacy**
- **Stability**
  - backward compatibility
  - no systematic inclusion of prototype tools

tation of similar tools could likely yield worse performance. We exploit CADP [10] since it is a popular toolbox maintained, regularly improved, and used in many industrial projects, as a verification framework. Another important advantage of using CADP is that, when a property does not hold, the model

- **Regular testing**
  collection of models, formulas, scripts ...
- **Documentation**
  - manual pages for all tools
  - demo examples
  - user community (web, FAQ, forum)

that usability may not be a strong barrier for formal tools' adoption. Main barriers are the limited support for development functionalities, such as traceability, and other process-integration features. We share our evaluation sheets [56],