Using LNT Formal Descriptions for Model-Based Diagnosis

Birgit Hofer[1], Radu Mateescu[2], Wendelin Serwe[2], and **Franz Wotawa**[1]
[1]TU Graz, Institute for Software Technology
[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG

wotawa@ist.tugraz.at

Context

- **Modeling is an issue** in model-based diagnosis
  - Modeling languages (strength, access, availability,...)
  - Modeling is not that easy

- **Observation**: There are other areas where modeling is needed, e.g., model-based testing or formal verification.

Objective

- ***Using models from testing/verification for diagnosis***

- **In particular**: Focus on LNT
- **Why?**
  - Availability of tools (CADP)
  - LNT is a follow-up of the ISO/IEC standard E-LOTOS (2001)

Rationale for the design of LNT

- Design challenges
  - Combine sequential and concurrent programming
  - Design a language for engineers, not theoreticians
- Same syntax for processes and functions
- Symmetric sequential composition (no action prefix)
- Ordinary variables
  - Write-many variables
  - Static analysis checks (variable initialization, no shared variables)
- Only tail recursion in processes
  - Non-tail recursion could be eliminated automatically
  - Arbitrary recursion in functions

Overview of LNT constructs

- LNT specification = set of modules
- Each module may contain:
  - types:
    - predefined: **bool**, **nat**, **int**, **real**, **char**, **string**
    - free constructors, including enumerations, records, unions
    - combinators: ranges, arrays, lists, sets, predicate subtypes
  - functions: either mathematical or procedural
    - predefined: arithmetical, logical, relational operators
    - generated automatically / handwritten by the user
  - channels: gate types, including **none** and **any**
  - processes: concurrent agents communicating via gates

Algorithmic constructs of LNT

- 70% of familiar-looking Ada-like constructs
  - **if-then-else** (with **elsif**), **case** with pattern matching
  - **while** … **loop**, **for** … **loop**, forever **loop** with **break**
  - functions with **return** statement
- Constructs from concurrency theory
  - nondeterministic assignment: X := **any** T **where** P (X)
  - nondeterministic choice: **select** … [] … [] … **end select**
  - parallel composition: **par** … ||… || … **end par**
  - hiding: **hide** … **end hide**
  - multiway rendezvous: G (O1, …, On)
- **Functions** and **processes** have many constructs in common

Dynamic semantics of LNT

- LNT functions:
  - state = memory store (mapping: variable → value)
  - LNT instructions: transitions between states (store updates)
- LNT processes:
  - Labelled Transition Systems
  - LTS state = <process term, memory store>
  - SOS rules define transitions between LTS states
  - static semantics restrictions
- Implementation of LNT in CADP:
  - LNT2LOTOS translator (funded by Bull)
  - reuse of the LOTOS compilers and verification tools of CADP

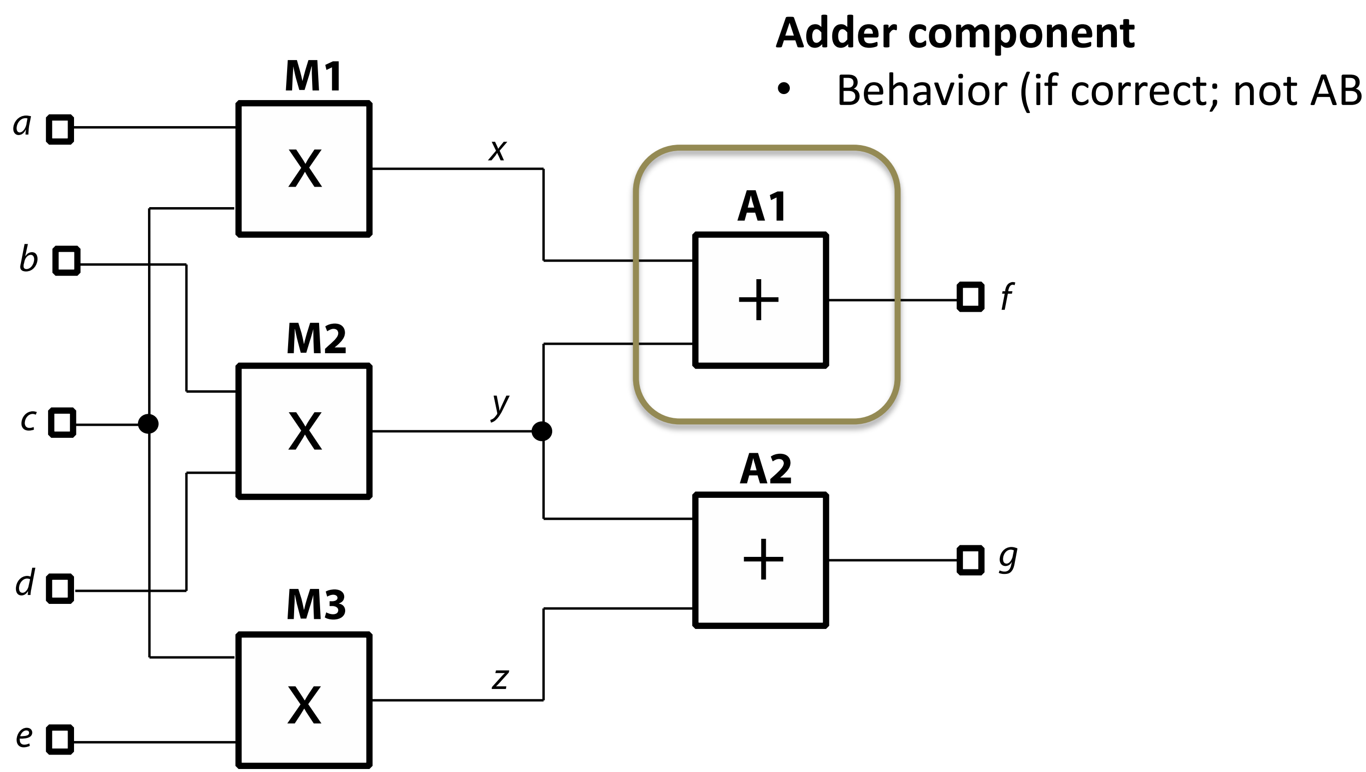
Impact of LNT so far

- 17 case studies done with LNT [21 publications]
  - avionics: 2
  - cloud computing: 3
  - distributed algorithms: 4
  - a hardware design: 4
  - human/computer interfaces: 2
  - other industrial systems: 2

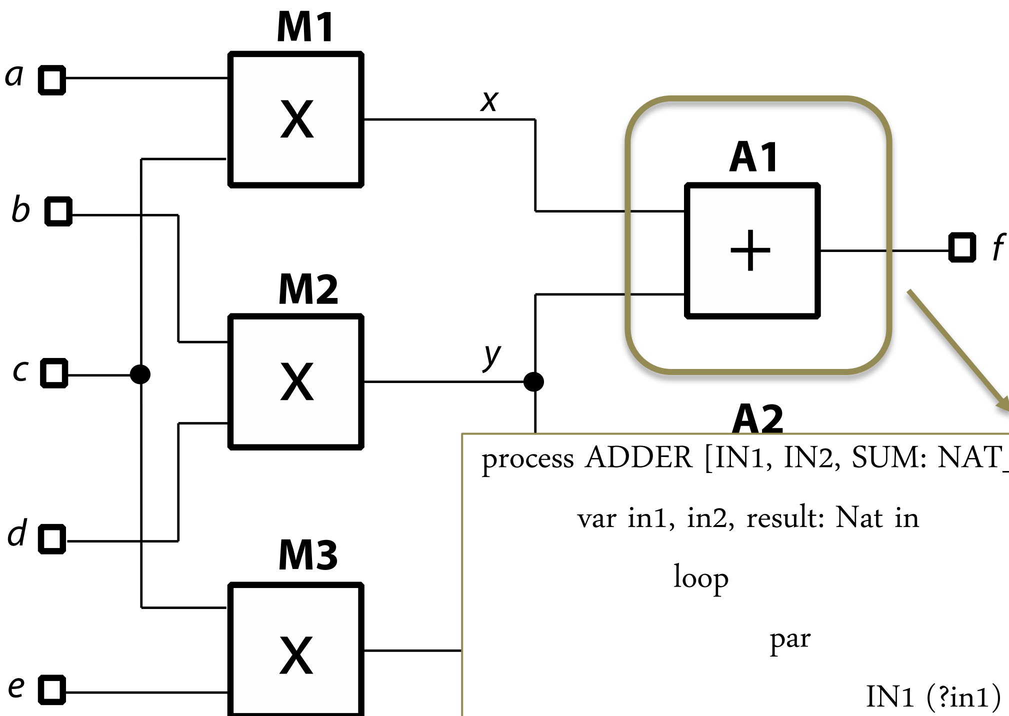
- 9 translators to LNT [11 publications]
  - AADL: 1 Toulouse-Sfax
  - applied π-calculus: 1 Grenoble
  - BPEL-WSDL: 2 MIT-Tsinghua, Bucharest-Grenoble
  - BPMN: 2 Nantes, Paris
  - DFT: 1 Twente
  - EB3: 1 Paris-Grenoble
  - GRL: 1 Grenoble

Basic idea

- Using d74 circuit as example



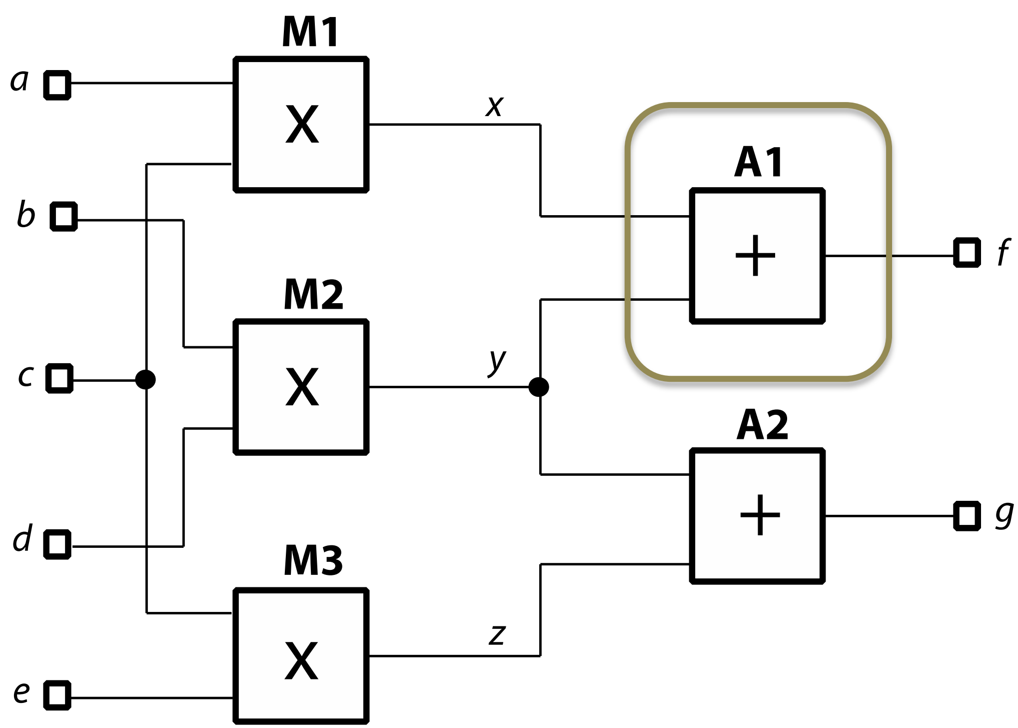
Correct behavior in LNT



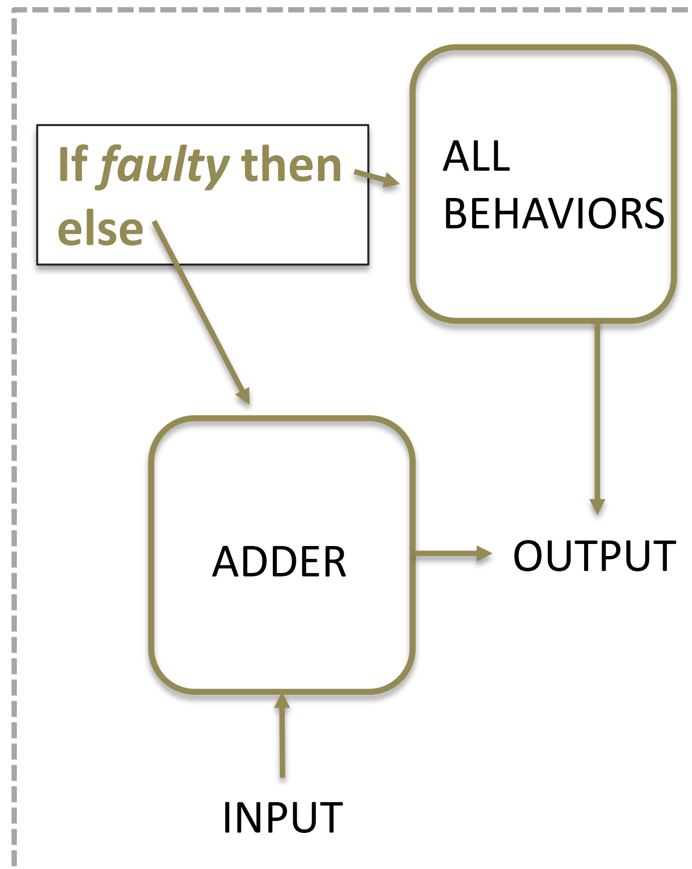

```
process ADDER [IN1, IN2, SUM: NAT_C] is
        var in1, in2, result: Nat in
                loop
                        par
                                IN1 (?in1)
                        || IN2 (?in2)
                        end par;
                        result := in1 + in2;
                        SUM (result)
                loop end loop
        end var
end process
```

Basic idea (cont.)

Have to introduce means for stating that a component is not working as expected!



Use a wrapper!



Wrapper in LNT

```
process ADDER_WRAP_ND [IN1, IN2, SUM: NAT_C]
(faulty: Bool) is
        if faulty then
                loop
                        par
                                IN1 (?any Nat)
                        || IN2 (?any Nat)
                        end par;
                        SUM (?any Nat)
                end loop
        else
                ADDER [IN1, IN2, SUM]
        end if
end process
```

Faulty behavior

Correct behavior

System modeling with Wrappers

```
process MAIN [IN1, IN2, IN3, IN4, IN5, OUT1, OUT2: NAT_C]
(f1, f2, f3, f4, f5: Bool, i1, i2, i3, i4, i5: Nat) is
    hide C1, C2, C3: NAT_C in
        par
            IN1, IN2, IN3, IN4, IN5 ->
                IN1 (i1); IN2 (i2); IN3 (i3);
                IN4 (i4); IN5 (i5); stop
            || IN1, IN3, C1 -> (* M1 *)
                MULTI_WRAP [IN1, IN3, C1] (f1)
            || IN2, IN4, C2 -> (* M2 *)
                MULTI_WRAP [IN2, IN4, C2] (f2)
            || IN3, IN5, C3 -> (* M3 *)
                MULTI_WRAP [IN3, IN5, C3] (f3)
            || C1, C2 -> (* A1 *)
                ADDER_WRAP [C1, C2, OUT1] (f4)
            || C2, C3 -> (* A2 *)
                ADDER_WRAP [C2, C3, OUT2] (f5)
        end par
    end hide
end process
```

Bringing it all together

Consistency-based diagnosis in CADP:

1. model the system structure SD and the behavior of individual components COMP in LNT using wrappers
2. instantiate the system, specifying a component C as faulty (via the corresponding parameter) if and only if C belongs to Δ
3. represent the observations OBS as temporal formulas (in MCL [16]) or sequences of events (i.e., a particular kind of LTS), and
4. determine the presence of observations in the considered system configuration using on-the-fly verification techniques, e.g., model checking (with EVALUATOR) or checking inclusion modulo equivalence relations (with BISIMULATOR).

Bringing it all together

- Input sequence:

  "IN1 !2"

  "IN2 !3"

  "IN3 !3"

  "IN4 !2"

  "IN5 !2"

  "OUT1 !10"

  "OUT2 !12"

- Checking for diagnosis {**M2;M3**} :

  % I1=2; I2=3; I3=3; I4=2; I5=2

  branching comparison

  ~~"obs.seq" <= "MAIN(false,false,false,~~

  ~~false,false,$I1,$I2,$I3,$I4,$I5)" ;~~

  branching comparison

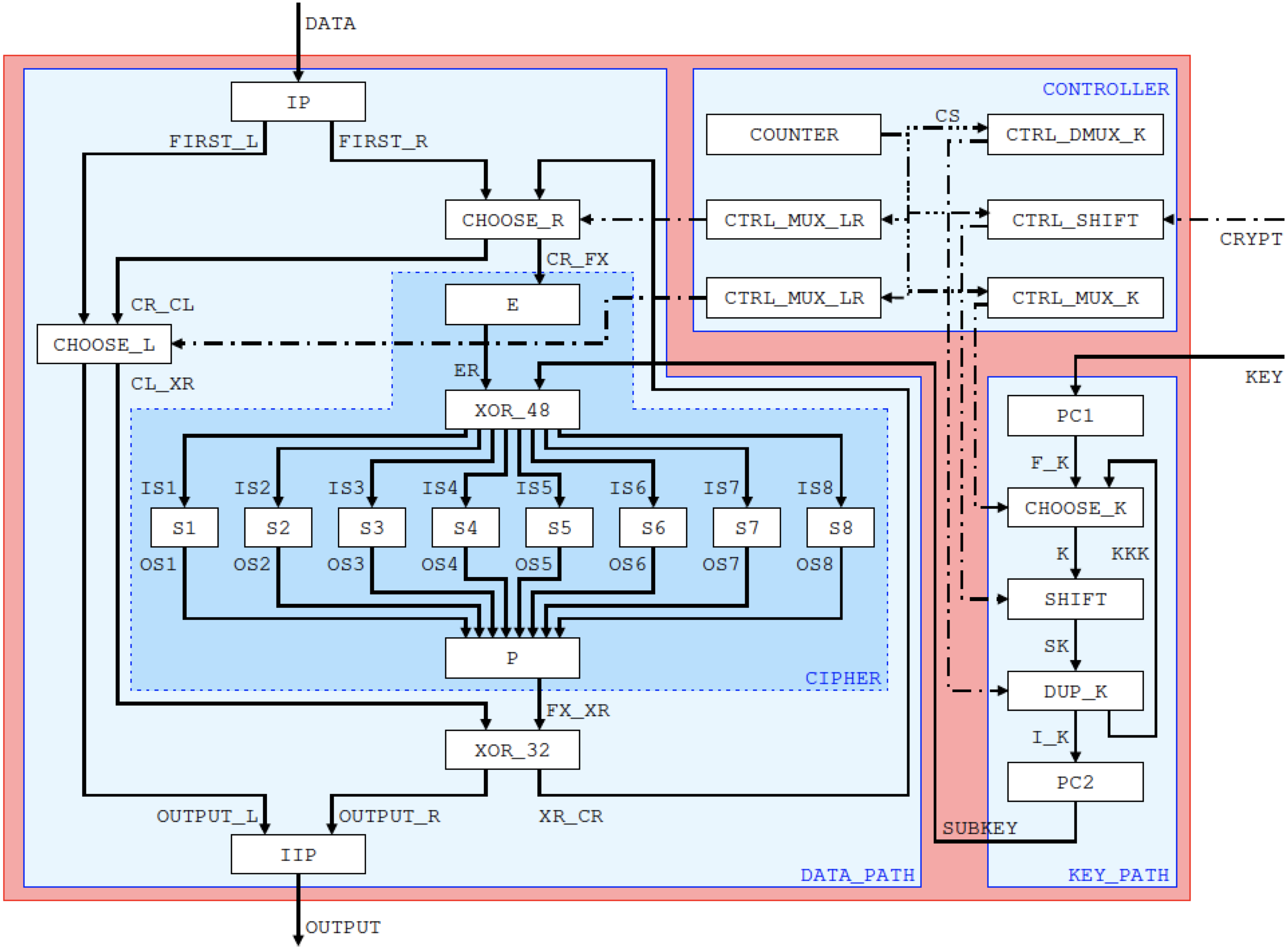
  "obs.seq" <= "MAIN(false,true,true,

  false,false,$I1,$I2,$I3,$I4,$I5)" ;

Diagnosis using LNT

- Use wrappers for components
- Set health status of component such that the system behaves like expected
- Diagnosis = search for health assignments (like always)

Case study DES



Case study DES (cont.)

- Introduce fault in one of the S-boxes
- Use a simplified calculation scheme (only one iteration)

- Use script for diagnosis including minimization steps from CADP
- S-Box was always correctly identified as being faulty
- Whole diagnosis took 11 minute on a Intel Core i5 M560 CPU at 2.67 Ghz and 8 MB of RAM.

Case study DES (cont.)

- Testing for correctness took only seconds

- Why?
  - Huge state space of the corresponding LTS used.

Conclusions

- Are able to use LNT models for diagnosis
- Make use of wrapper components introducing the health state
- Diagnosis feasible for smaller models
- Make use of LNT models (almost) directly
- Rich set of tools and models available

Conclusions

- Able to introduce fault models as well
- Models of behavior including time

- But there is a need to improve diagnosis computation


Thank you for your attention!
QUESTIONS?