Debugging Process Algebra Specifications

Gwen Salaün

Grenoble INP, Inria, France

Lina Ye

Supélec, France

Context

- Designing and developing concurrent and distributed applications is a tedious and error-prone task
- Formal techniques and tools are of great help to specify and debug such systems and the corresponding models
- Here, we focus on value-passing process algebra as specification language and model checking as analysis technique
- Model checking: given a specification *S* and a property *P*, we check for a concrete input *I* whether *S* satisfies *P*:

$$LTS(S,I) \models P$$

Issues

- Building the set of validation examples (inputs) and debugging the system is a real burden for at least three reasons
  - Counterexamples provided by model checkers are the only feedback one may have, but their comprehension and analysis is often complicated
  - We do not know whether the set of validation examples covers all the possible execution scenarios described in the specification
  - The specification may contain errors (*e.g.*, ill-formed decisions, non-synchronizable actions, dead code), which are not necessarily found using model checking techniques

Contributions

- We propose to improve the quality of validation examples, and to debug process algebra specifications (LNT) through coverage analysis
- We define block, decision, and action coverage for specifications
- Collecting coverage information is achieved through probe insertion and follows a two-step methodology to reduce state space explosion problems
- We implemented these techniques as a tool built on top of the publicly available CADP verification toolbox
- We applied our tool to more than one hundred LNT specifications including six real-world systems

Outline

1. Overview of LNT
2. Coverage Analysis
3. Tool Support
4. Concluding Remarks

LOTOS NT

- LOTOS NT (LNT) is a value-passing process algebra with user-friendly syntax and operational semantics

- LNT is an imperative-like language where you can specify data types, functions (pattern matching and recursion), and processes

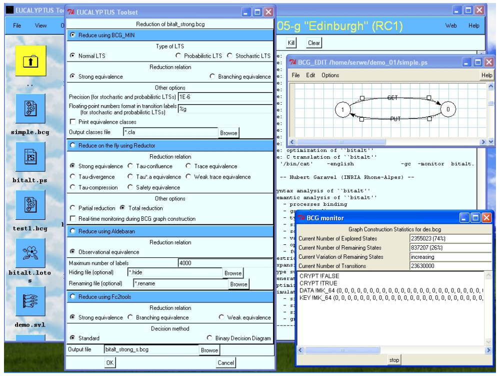
- Excerpt of the LNT process grammar:

B ::= **stop** | G (!E, ?X) **where** E' | **if** E **then** B1 **else** B2 **end if**
| x:=E | **hide** G **in** B **end hide** | **while** E **loop** B **end loop**
| **select** B1 **[]** … **[]** Bn **end select** | B1 **;** B2
| **par** G **in** B1 **||** … **||** Bn **end par** | **case** V **in** V1 -> B1 | … **end case**

- Verification using CADP through an automated translation to LOTOS

Construction and Analysis of Distributed Processes (CADP)

- Design of asynchronous systems
  - Concurrent processes
  - Message-passing communication
  - Nondeterministic behaviors



CADP (Inria/Convecs)

- Formal approach rooted in concurrency theory: process calculi, Labeled Transition Systems, bisimulations, branching temporal logics

- Many verification techniques: simulation, model and equivalence checking, compositional verification, test case generation, performance evaluation, etc.

- Numerous real-world applications: avionics, hardware design, cloud computing, bioinformatics, etc.

Outline

1. Overview of LNT
2. Coverage Analysis
3. Tool Support
4. Concluding Remarks

Terminology

- We focus on three coverage criteria in this work: blocks, decisions, and actions
- A block is the largest sequence of instructions free of conditional, choice, and parallel constructs
- Given an LNT specification and a set of validation examples:
  - A block / action is covered if it is executed by at least one example
  - A decision is covered if both true and false outcomes are evaluated by one example (not necessarily the same one)
- Block / decision / action coverage is the percentage of the number of covered blocks / decisions / actions out of their total number

Probe Insertion (1/2)

- We instrument the LNT code with probes in order to collect structural coverage information

- Given an LNT specification and an example, we compile it into an LTS using CADP compilers

  => After probe insertion, we analyze the LTS for retrieving coverage information

- Block: we insert a probe at the end of each block

- Action: we insert a probe just after the target action (one different probe per action occurrence)

Probe Insertion (2/2)

- Decision: we equip the corresponding probe with this decision as its parameter

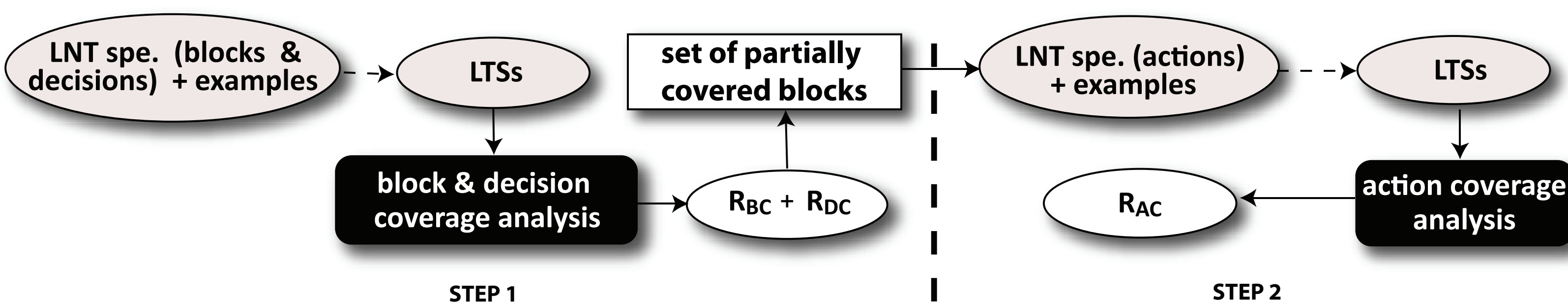
Types	Before Insertion	After Insertion
If	**if** E **then** B_1 **end if**	$\underline{P(!E)}$; **if** E **then** B_1 **end if**
Case	**case** V **in** $V_1 \rightarrow B_1 \mid V_2 \rightarrow B_2$ **end case**	**case** V **in** $V_1 \rightarrow \underline{P_1(!TRUE);}B_1$ $\mid V_2 \rightarrow \underline{P_1(!FALSE)}; \underline{P_2(!TRUE);}B_2$ **end case**
While	**while** E **loop** B_1 **end loop**	$\underline{P(!E);}$ **while** E **loop** B_1 **end loop**; $\underline{P(!E)}$

- Special attention must be taken when the LNT specification contains internal actions: critical blocks / decisions (see the paper for details)

- Behavior preservation: we proved that the original LNT specification is branching equivalent to the extended specification with probes hidden

Coverage Computing

- If we insert probes for all three coverage criteria, the corresponding LTSs would suffer from the state explosion problem
- To solve this, we first insert probes for blocks and decisions, and second we focus on actions (two-step analysis)
- A block whose entry is allowed is an executable block
- A block is partially covered if it is executable but not covered



Results Analysis

- Our approach returns the percentage of block / decision / action coverage + the set of uncovered blocks / decision outcomes / actions

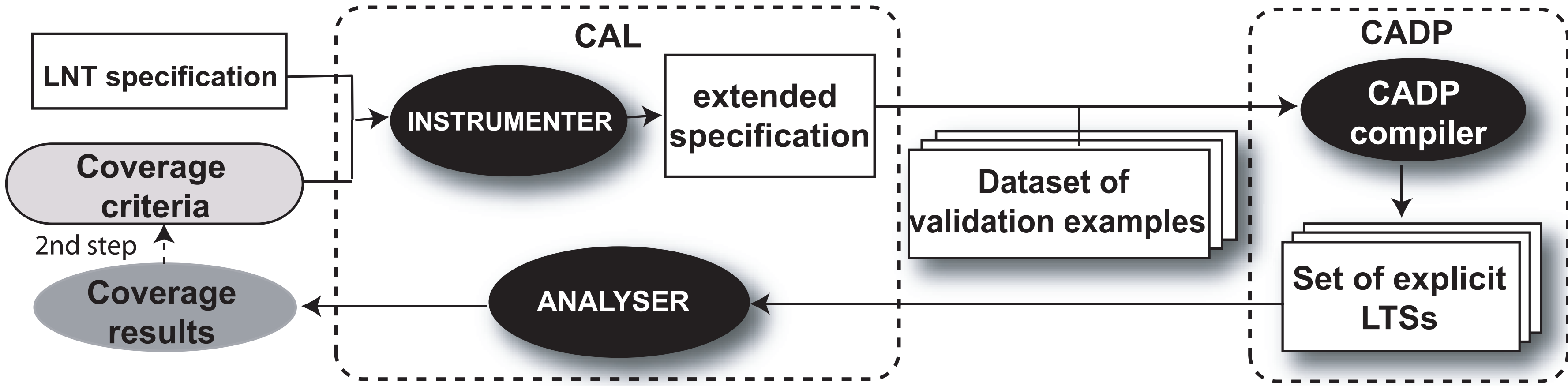
- Two reasons can explain why coverage percentages are lower than 100%
  - lack of validation examples
  - defects contained in the corresponding LNT specification

- The following types of errors may be the source of the uncovered parts
  - Ill-formed decisions
  - Unnecessary decision
  - Non-synchronizable actions
  - Dead code

Outline

1. Overview of LNT
2. Coverage Analysis
3. Tool Support
4. Concluding Remarks

Implementation



Experiments (1/2)

Case Study	Designer	Description
DirectCache	STMicroelectronics	deals with cache coherence in multiprocessor systems by using a common directory.
AgtReconfig	Inria	provides an agent-based mechanism allowing distributed applications to be reconfigured at run-time [8].
DisCache	STMicroelectronics	ensures data consistency in multiprocessor shared memory systems that allow multiple copies of a datum [1].
SelfConfig	Inria, Orange labs	automates the configuration of a cloud application that is distributed on more than one virtual machine without requiring any centralized server [9, 24, 10].
ReConfig	Inria, Orange labs	reconfigures a running system composed of a set of interconnected components, where multiple failures occurring at reconfiguration time are tolerated [5].
Synchro	Inria	realizes the multiway rendezvous of LNT, where all parallel processes are organized in a hierarchical structure [11].

Experiments (2/2)

	DirectCache	AgtReconfig	DisCache	SelfConfig	ReConfig	Synchro1	Synchro2
N_L	196	785	981	1635	3700	486	480
N_{VE}	5	4	6	60	200	18	30
N_B	12	31	33	31	90	66	66
BC	83.3%	67.7%	93.9%	83.8%	97.8%	62.1%	100%
N_D	12	27	23	23	89	50	50
DC	83.3%	74.1%	91.3%	73.9%	92.1%	60%	100%
N_A	9	50	33	32	53	72	72
AC	100%	64%	100%	93.8%	96.2%	68.1%	100%

- Helped to improve the quality of the validation examples (Synchro2) and several kinds of errors (*e.g.*, several ill-formed decisions in ReConfig)
- Naive approach vs. two-step methodology: up to half the number of probes, and reduction from 30 to 60% in terms of states/transitions

Outline

1. Overview of LNT
2. Coverage Analysis
3. Tool Support
4. Concluding Remarks

Concluding Remarks

- We propose a tool-supported approach for debugging value-passing process algebra, which relies on coverage analysis and probe insertion

- The obtained results can be considered as accurate guides to either complete validation examples or correct errors in the given specification

- It is worth pointing out that our approach could be applied to other value-passing process algebra such as CSP/FDR2 or mCRL2

- Main perspective: extension to other criteria for coverage analysis, such as multiple condition coverage or some criteria based on data flow