Test Case Generation with PathCrawler/LTest: How to Automate an Industrial Testing Process

Sébastien Bardin[1] **Nikolai Kosmatov**[1] Bruno Marre[1]
David Mentré[2] Nicky Williams[1]

[1]CEA, List, Software Safety and Security Lab, Paris-Saclay, France

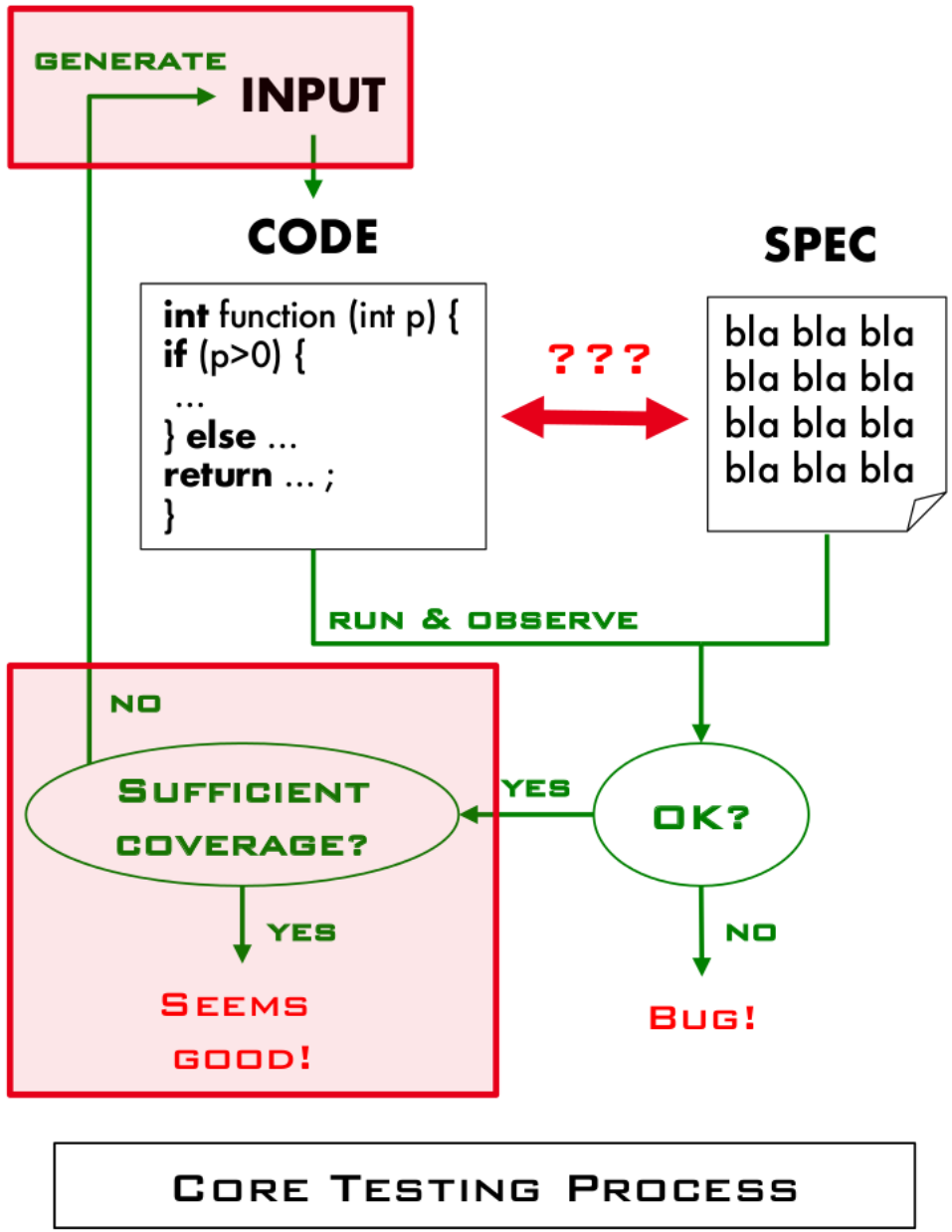
[2]Mitsubishi Electric R&D Centre Europe (MERCE), Rennes, France

ISOLA 2018, Limassol, Cyprus
November 7, 2018

Context: White-Box Testing

Testing process

- Generate a test input
- Run it and check for errors
- Estimate coverage: if enough stop, else loop



Context: White-Box Testing and Coverage Criteria

Coverage criterion = objectives to be fulfilled by the test suite

- Criteria guide automation
- Criteria can be part of industrial practice or normative requirements

Goal:

Automate unit test-case generation in an industrial setting

Industrial Partner: MERCE

Mitsubishi Electric

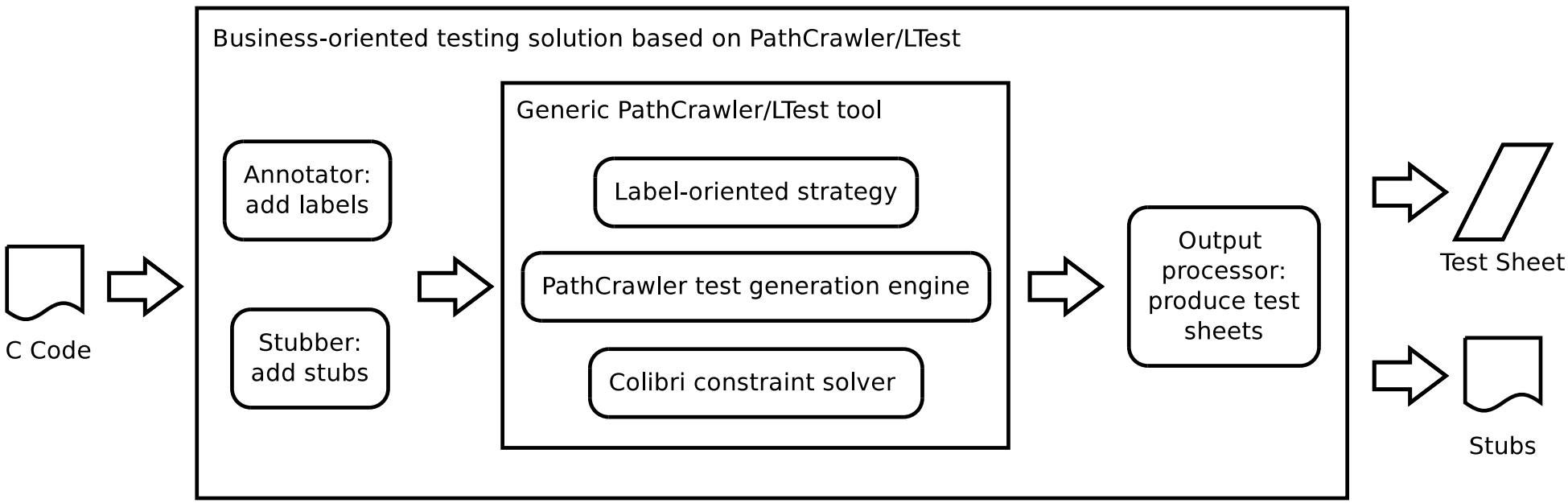
- a wide range of activities (Home Products, Space Systems Automotive Equipment, Transportation Systems, Energy Systems etc.)
- many software intensive, safety critical products
- products often require certification criteria
  - e.g. railway EN 50128 SIL4 or automotive ISO 26262 ASIL D
- up to 65% of the cost is due to testing

A research branch of Mitsubishi Electric:
MERCE (Mitsubishi Electric R&D Centre Europe)

Main ingredients of the solution:

- **PathCrawler:** a white-box test generation tool for C code
  - based on DSE (Dynamic Symbolic Execution)
- **Colibri:** a constraint solver used by **PathCrawler**
- **Labels:** a generic specification mechanism for criteria
  - can easily encode a large class of criteria
  - a semantic view, with a formal treatment
- **LTest**: offers an efficient test generation technique for labels
  - an optimized version of DSE (Dynamic Symbolic Execution)
  - no exponential blowup of the search space
- **Adaptation to the business unit needs** by MERCE

Global architecture of the solution:



Outline

Colibri constraint solving library

Colibri provides:

- basic constraints, filtering, labeling (different heuristics)
- bounded and modular integer arithmetics
- real and floating point arithmetics

- main author: Bruno Marre
- winner in floating point category at SMT-COMP 2017, 2018
- implemented at CEA List in Constraint Logic Programming

Outline

PathCrawler test-case generation tool

- based on DSE (Dynamic Symbolic Execution)
- sound and relatively complete
  - can justify why a test objective is uncovered
- main author: Nicky Williams
- implemented at CEA List in Constraint Logic Programming
- online version `http://pathcrawler-online.com`

Dynamic Symbolic Execution [dart,cute,pathcrawler,exe,sage,pex,klee,...]

✓ very powerful approach to white-box test generation

✓ many tools and successful case-studies since mid 2000's

✓ arguably one of the most wide-spread use of formal methods

PathCrawler test-case generation tool

- based on DSE (Dynamic Symbolic Execution)
- sound and relatively complete
  - can justify why a test objective is uncovered
- main author: Nicky Williams
- implemented at CEA List in Constraint Logic Programming
- online version `http://pathcrawler-online.com`

Dynamic Symbolic Execution [dart,cute,pathcrawler,exe,sage,pex,klee,...]

✓ very powerful approach to white-box test generation

✓ many tools and successful case-studies since mid 2000's

✓ arguably one of the most wide-spread use of formal methods

✗ lack of support for many coverage criteria

Outline

Labels, a mechanism to specify test objectives

Basic definitions

Given a program P, a label l is a pair (loc, φ), where:

- φ is a well-defined predicate at location loc in P
- φ contains no side-effects

Example:

```
statement_1;
// l1:  x==y
// l2:  !(x==y)
if (x==y && a<b)
    {...};
statement_3;
```

Simulation of coverage criteria by labels: CC

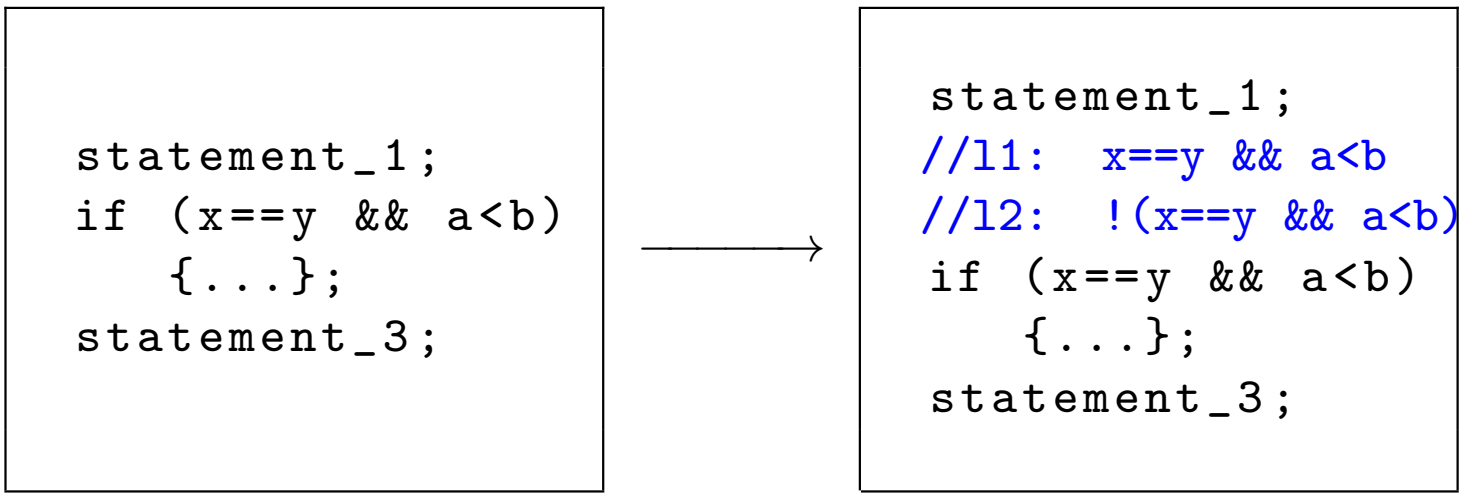
```
statement_1;
if (x==y && a<b)
   {...};
statement_3;
```



```
statement_1;
// l1:  x==y
// l2:  !(x==y)
// l3:  a<b
// l4:  !(a<b)
if (x==y && a<b)
   {...};
statement_3;
```

Condition Coverage (**CC**)

Simulation of coverage criteria by labels: DC



Decision Coverage (**DC**)

Simulation of coverage criteria by labels: MCC

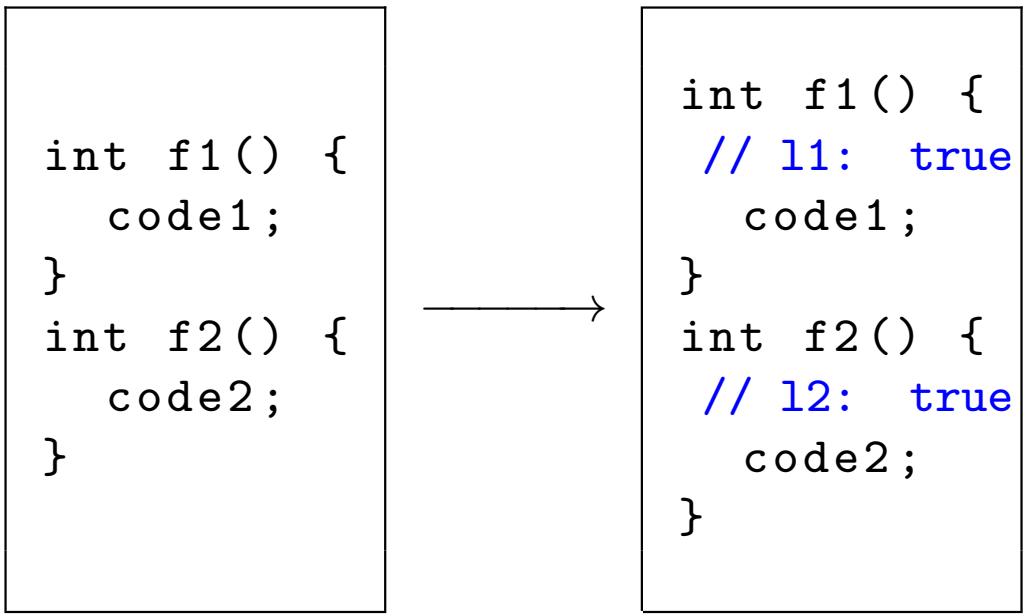
```
statement_1;
if (x==y && a<b)
   {...};
statement_3;
```



```
statement_1;
// l1:  x==y && a<b
// l2:  x==y && a>=b
// l3:  x!=y && a<b
// l4:  x!=y && a>=b
if (x==y && a<b)
   {...};
statement_3;
```

Multiple-Condition Coverage (**MCC**)

Simulation of coverage criteria by labels: FC



Function Coverage (**FC**)

Simulation results

Theorem

The following coverage criteria can be simulated by label coverage: **IC**, **DC**, **FC**, **CC**, **MCC**, *Input Domain Partition, Run-Time Errors.*

Theorem

For any finite set O of side-effect free mutation operators, weak mutation criterion $\mathbf{WM}_O$ *can be simulated by label coverage.*

Outline

The problem: how to efficiently adapt DSE to labels

Dynamic Symbolic Execution

× lack of support for many coverage criteria

Challenge: extend DSE to a large class of coverage criteria

- well-known problem
- recent efforts in this direction through instrumentation
  [Active Testing, Mutation DSE, Augmented DSE]
- limitations:
  - exponential explosion of the search space [APEX: 272x avg]
  - very implementation-centric mechanisms
  - unclear expressiveness

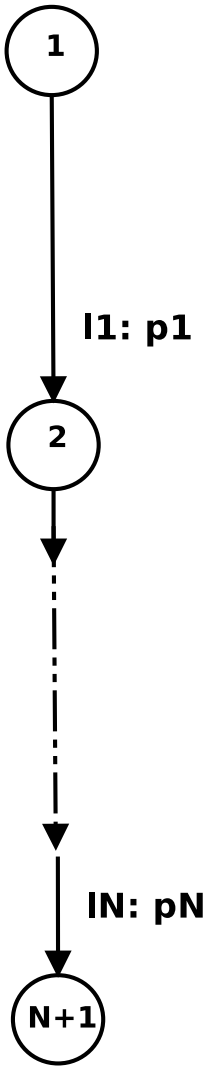
Our approach: a label-oriented test generation

DSE*: an efficient test generation technique for labels

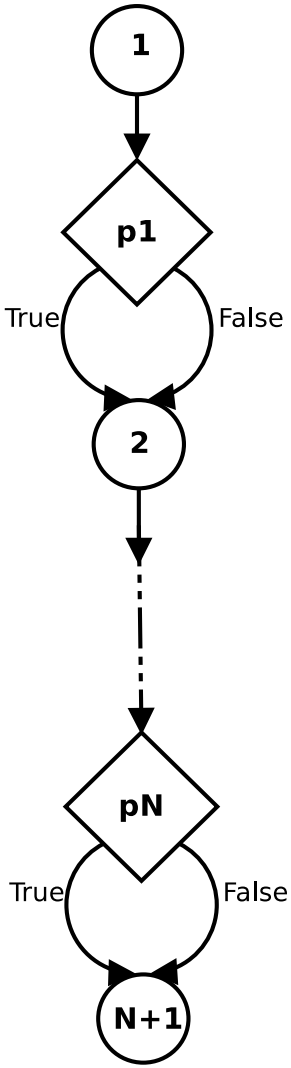
- Tight instrumentation totally prevents "complexification"
- Iterative Label Deletion: discards some redundant paths
- Both techniques can be implemented in a black-box manner

DSE*: Direct vs. Tight instrumentation of labels

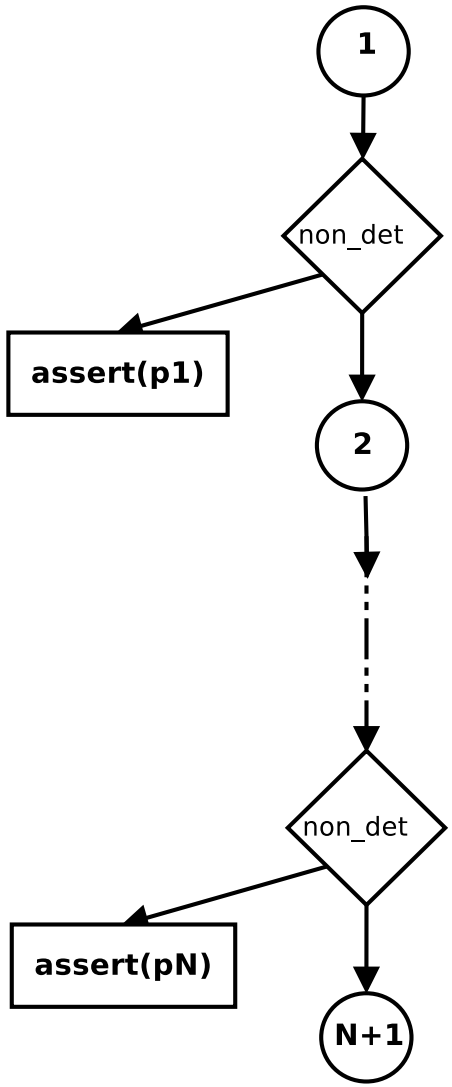
Program with N labels



Direct instrumentation

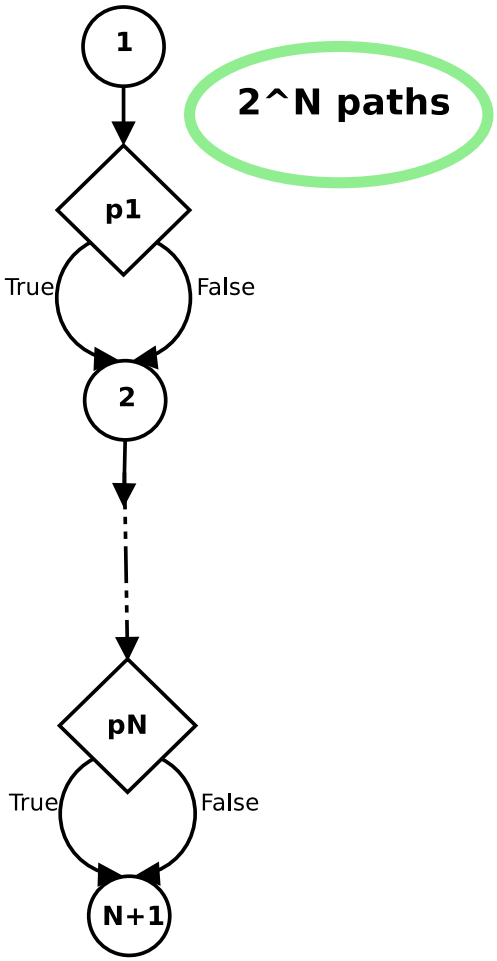


Tight Instrumentation

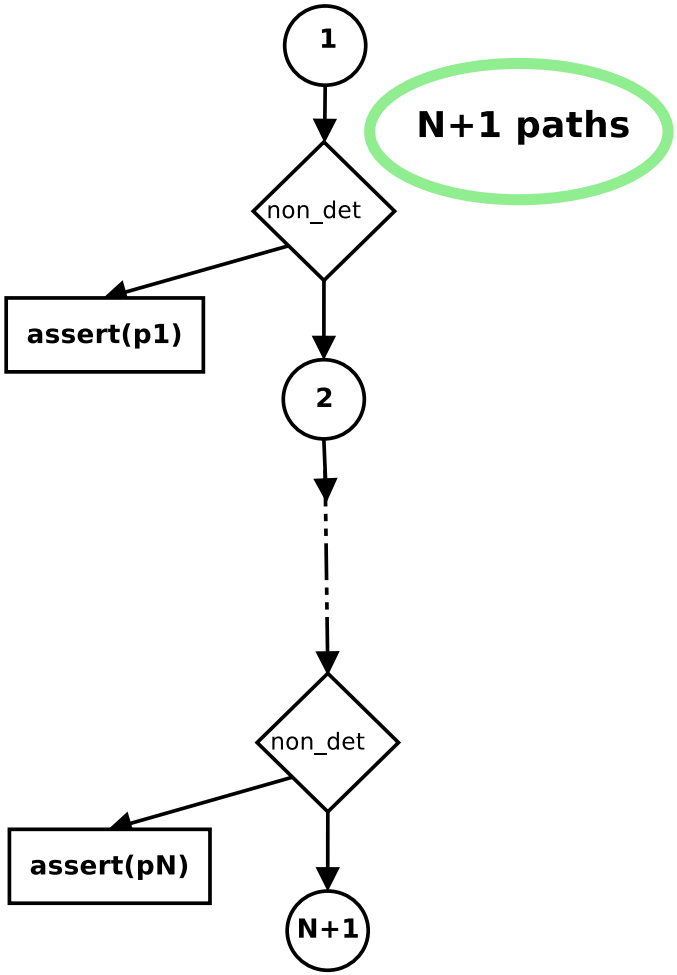


DSE*: Direct vs. Tight instrumentation of labels

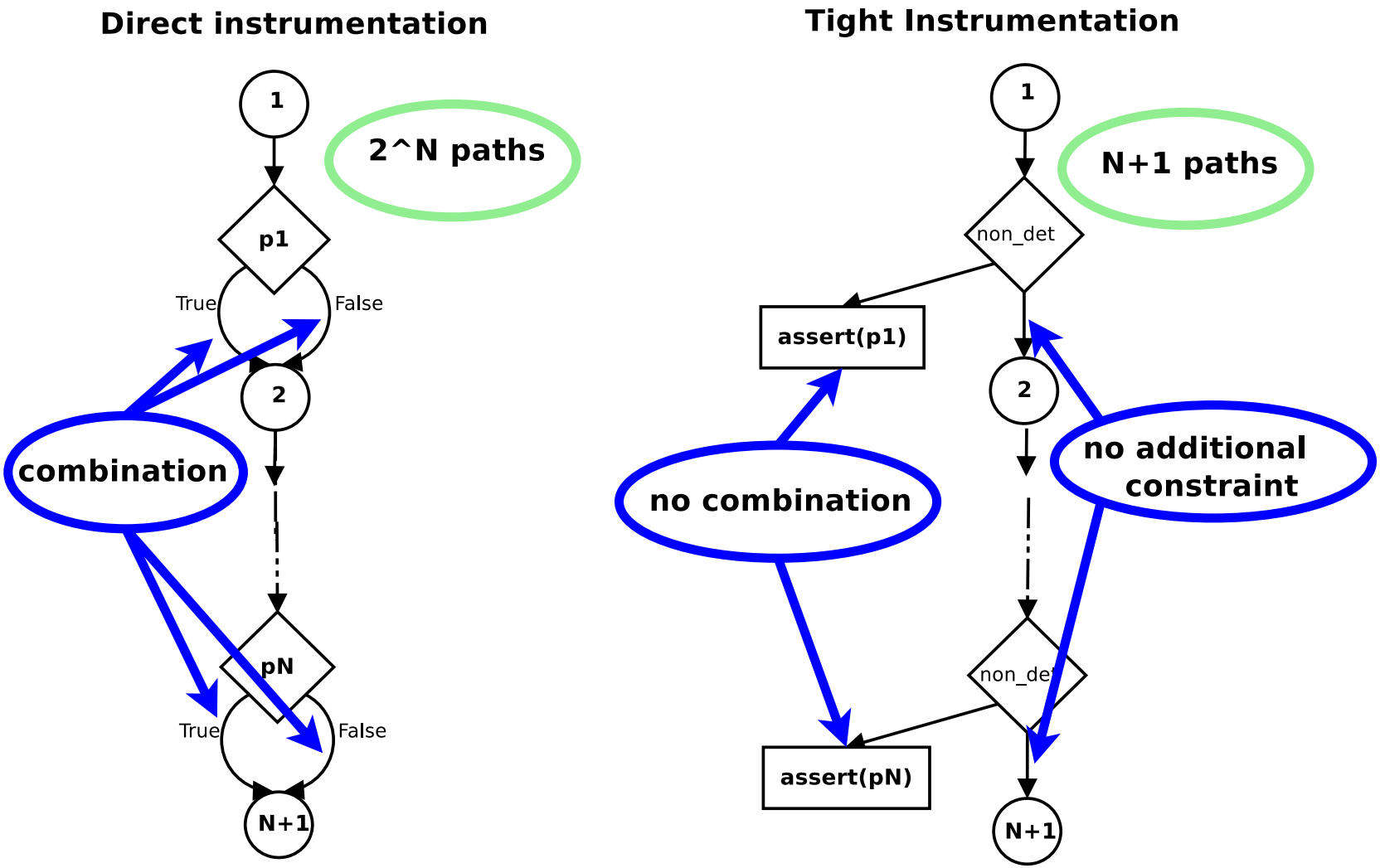
Direct instrumentation



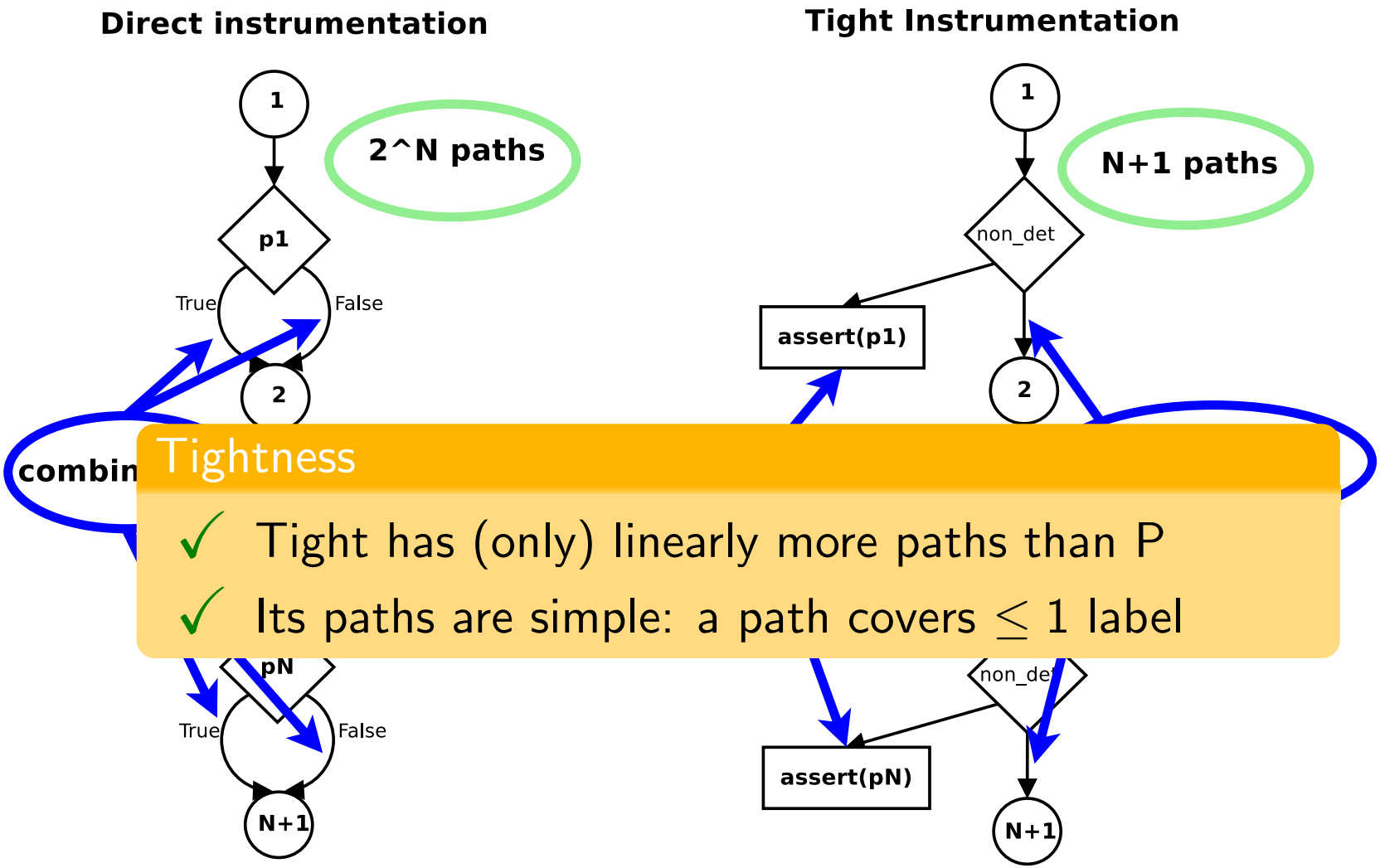
Tight Instrumentation



DSE*: Direct vs. Tight instrumentation of labels



DSE*: Direct vs. Tight instrumentation of labels



Tightness

- ✓ Tight has (only) linearly more paths than P
- ✓ Its paths are simple: a path covers ≤ 1 label

Some experimental results

DSE* dramatically improves test generation performances

- APEX reports an average overhead >272x
- DSE* leads to an average overhead of 2.4x

More about label based testing:

[Bardin et al., ICST 2014, TAP 2014, ICST 2015]
[Marcozzi et al., ICST 2017 (res.), ICST 2017 (tool), ICSE 2018]

Outline

Towards an industrial adoption

First, MERCE evaluated **PathCrawler/LTest** on manually annotated industrial code

- functions ranging from a few hundreds to one thousand lines
- **PathCrawler/LTest** covered all labels, taking a few seconds for small functions and $\sim$ 1 hour for the biggest (2^{145} paths)

MERCE developed additional modules for automatic annotation of labels, generation of stubs, generation of test sheets

MERCE evaluated the complete automatic tool

- on industrial code of 80,000 lines, 1,300 functions in 150 files
- the tool was able to parse and annotate 100% of the files and generate test cases for 86% of functions
- the generation took $<$ 1 day instead of $\sim$230 days manually

Those very good results are very encouraging for pushing the technology in business units

Future work

- Supporting a wider range of criteria (including dataflow, MCDC...)
  - a recent extension of labels to hyperlabels
- Further experiments
- Better adaptation of the tools to the needs of the users

A 3-year international (France-Luxembourg) grant of 760,000 EUR was allocated to support this work direction in 2019–2021

Lessons Learned: Key Factors for Industrial Usage

- close collaboration between tool developers and users
  - role of MERCE in adapting the tool to users' needs is crucial
- automation for integration in the current testing process
  - MERCE targeted a fully automated solution
- completeness of the tool or its capacity to justify the absence of a test input for a given test objective
- soundness and completeness of the tool are particularly important in the context of certification
- the performance of the tool is crucial for its integration
- the capacity to support criteria used in industry

Changing habits in an industrial process is always difficult!