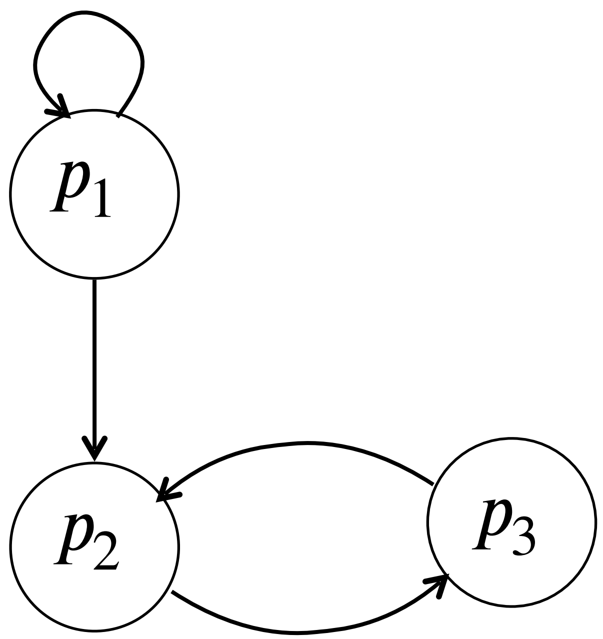
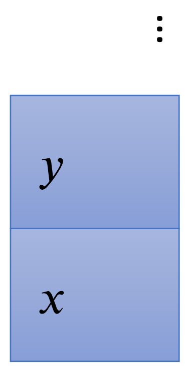
Differential Testing of Pushdown Reachability with a Formally Verified Oracle

Anders Schlichtkrull
Morten Konggaard Schou
Jiří Srba
Aalborg University

Dmitriy Traytel
University of Copenhagen

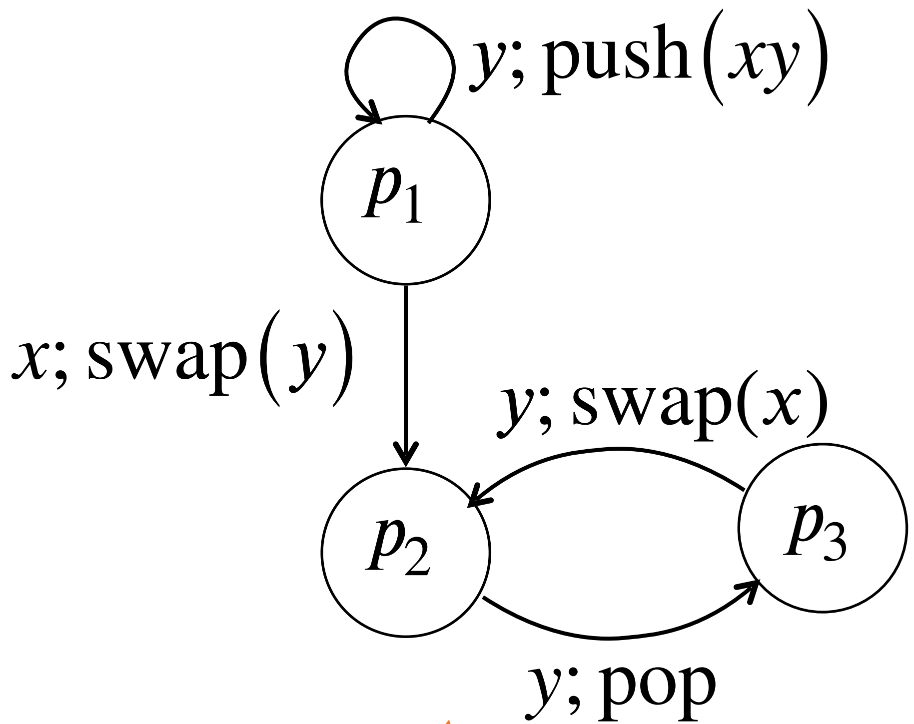
Background: Pushdown Systems - A small example

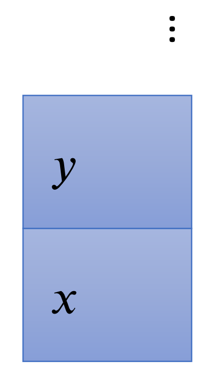




Finite automaton + Unbounded stack

Background: Pushdown Systems - A small example

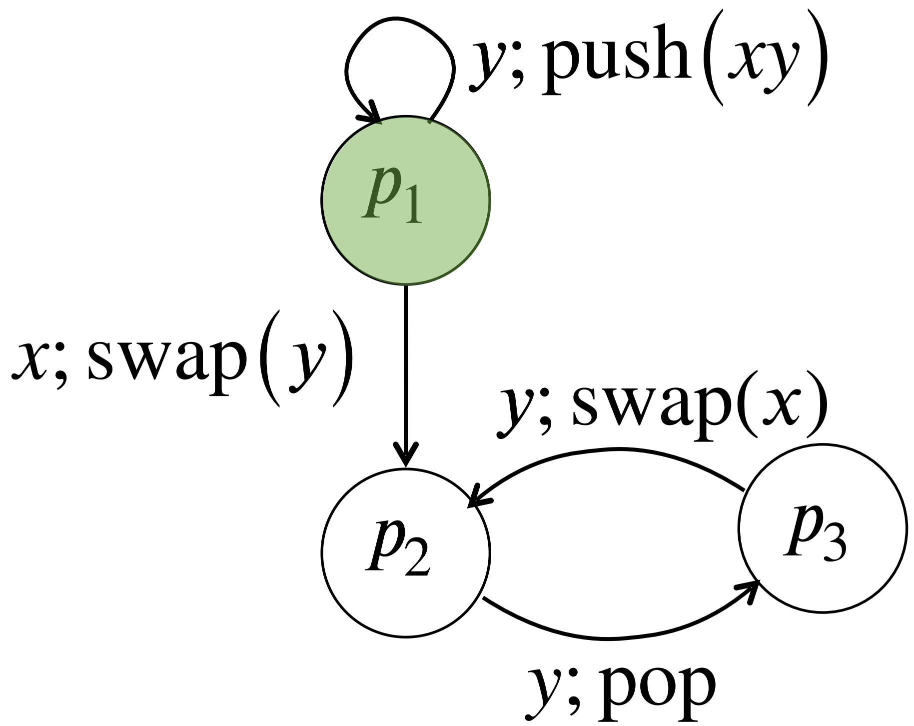


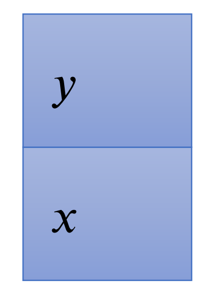


Finite automaton + Unbounded stack

Rules push, swap or pop labels

Background: Pushdown Systems - A small example



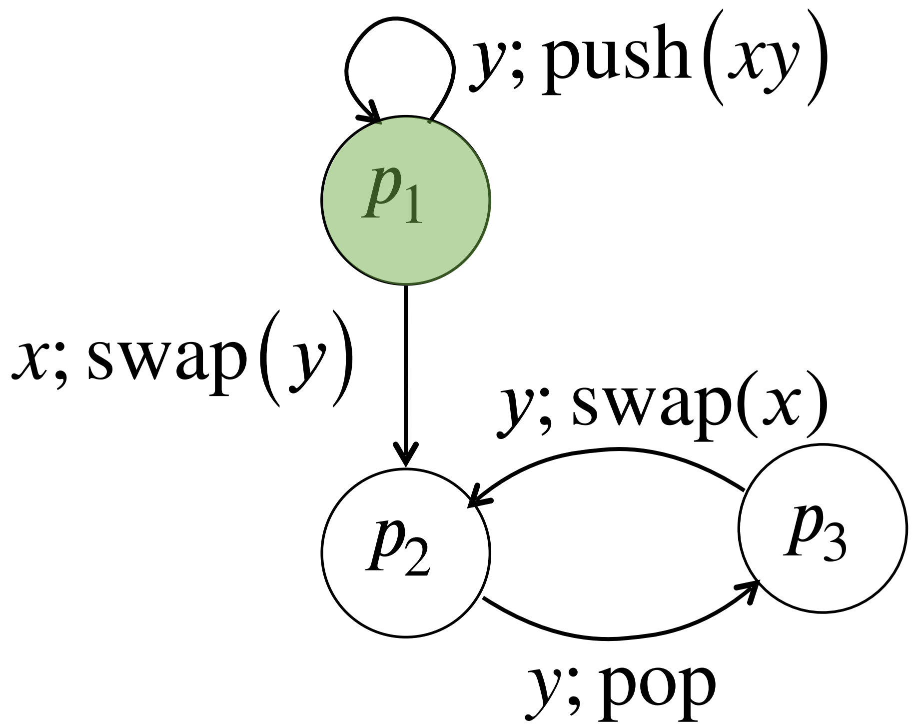


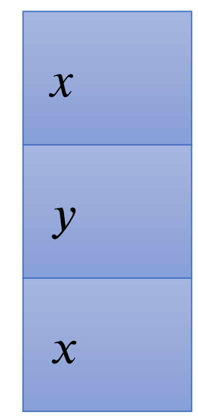
$\langle p_1, yx \rangle$

Finite automaton + Unbounded stack

Rules push, swap or pop labels

Background: Pushdown Systems - A small example



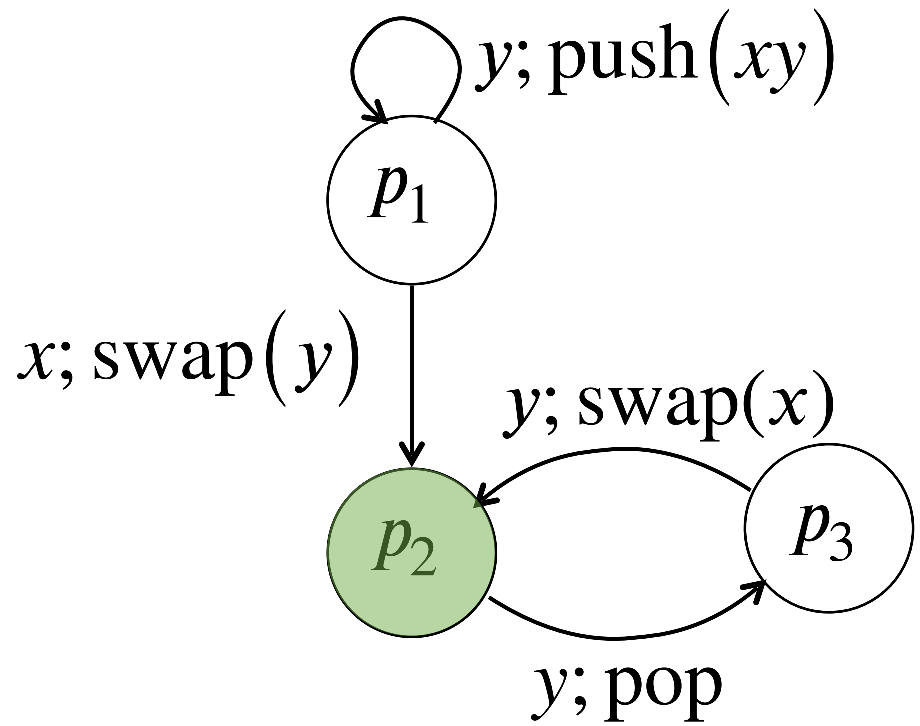


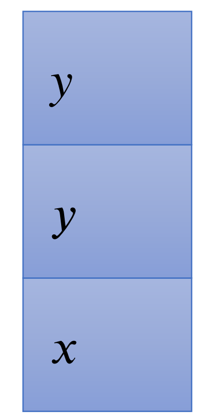
$$\langle p_1, yx \rangle \Rightarrow \langle p_1, xyx \rangle$$

Finite automaton + Unbounded stack

Rules push, swap or pop labels

Background: Pushdown Systems - A small example



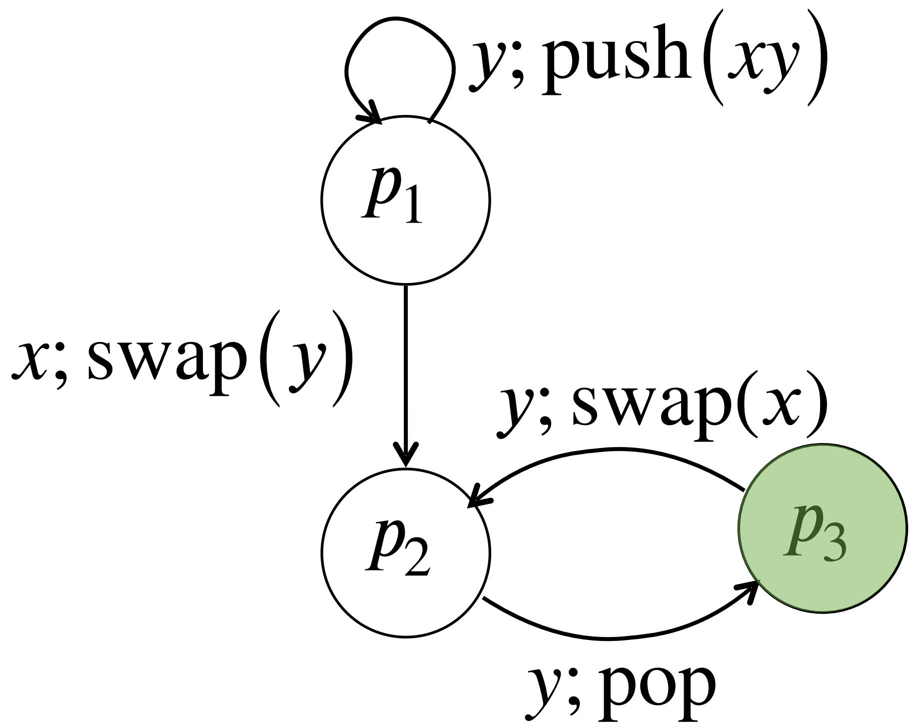


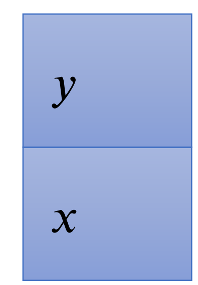
$$\langle p_1, yx \rangle \Rightarrow \langle p_1, xyx \rangle \Rightarrow \langle p_2, yyx \rangle$$

Finite automaton + Unbounded stack

Rules push, swap or pop labels

Background: Pushdown Systems - A small example



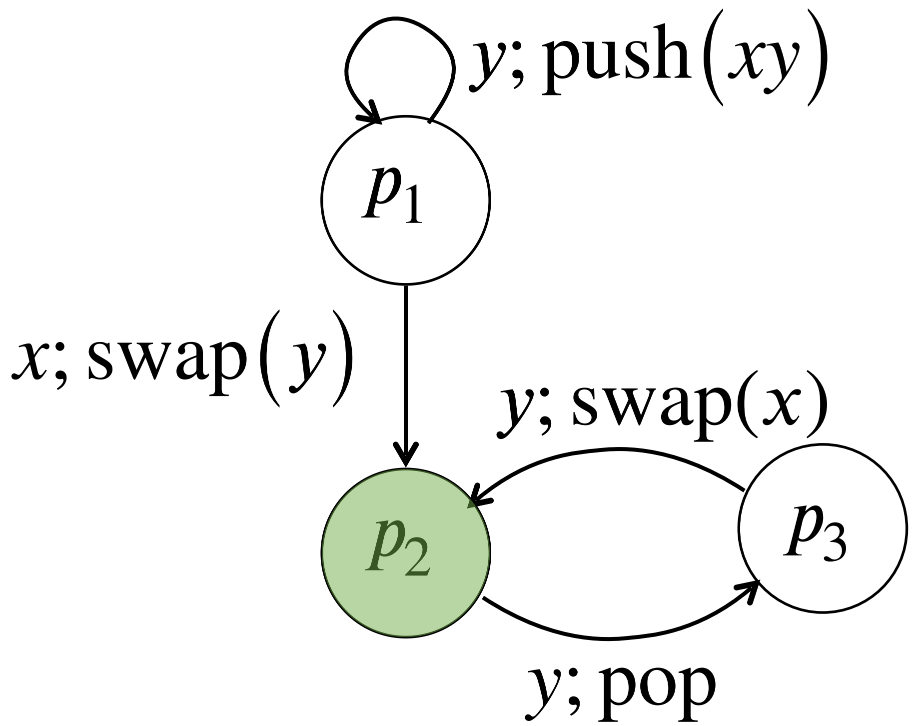


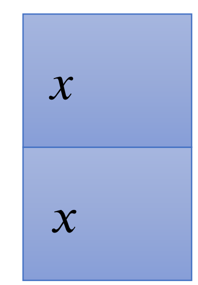
$$\langle p_1, yx \rangle \Rightarrow \langle p_1, xyx \rangle \Rightarrow \langle p_2, yyx \rangle \Rightarrow \langle p_3, yx \rangle$$

Finite automaton + Unbounded stack

Rules push, swap or pop labels

Background: Pushdown Systems - A small example





$$\langle p_1, yx \rangle \Rightarrow \langle p_1, xyx \rangle \Rightarrow \langle p_2, yyx \rangle \Rightarrow \langle p_3, yx \rangle \Rightarrow \langle p_2, xx \rangle$$

Finite automaton + Unbounded stack

Rules push, swap or pop labels

Background: Pushdown Reachability

We are interested in reachability!

E.g. to check for safety:

Can a system get into some bad configuration?

Or more generally:

Let C be a set of *initial configurations*.

Let C' be a set of *final configurations*.

Is C' reachable from C? $\exists c \in C, c' \in C' . \, c \Rightarrow^* c'$

Applications:

- Interprocedural control-flow analysis of recursive programs
- Static analysis of Java, C and C++.
- Communication network analysis
- Analysis MPLS communication protocols.

Background: Reachability

- For set $C \subseteq P \times \Gamma^*$ of configurations define:
  - Predecessors: $pre*(C) \ = \{c \mid \exists c' \in C, \ \ c \Rightarrow^* c'\}$
    - All configurations that can reach C.
  - Successors: $post^*(C) = \{c \mid \exists c' \in C, \ \ c' \Rightarrow^* c\}$
    - All configurations that C can reach.

- C, pre*(C), and post*(C) can be infinite.

- Rephrasing reachability of C' from C:
  $C \cap pre*(C') \neq \emptyset$?
  Is there a configuration that is initial and can reach the finals?

- For regular sets: We can calculate pre* and decide reachability!

Background: Reachability

- For set $C \subseteq P \times \Gamma^*$ of configurations define:
  - Predecessors: $pre*(C) \ = \{c \mid \exists c' \in C, \ \ c \Rightarrow^* c'\}$
    - All configurations that can reach C.
  - Successors: $post^*(C) = \{c \mid \exists c' \in C, \ \ c' \Rightarrow^* c\}$
    - All configurations that C can reach.

- C, pre*(C), and post*(C) can be infinite.

- Rephrasing reachability of C' from C:
  $C' \cap post*(C) \neq \varnothing$?
  Is there a configuration that is final and that the initials can reach?

- For regular sets: We can calculate post* and decide reachability!

Background: $pre*$ preserves regularity [Büchi 1964]

For a regular set of configurations C, $pre*(C)$ is regular. [Büchi 1964]

Saturation procedure in polynomial time [CONCUR'97, INFINITY'97]
Algorithm with improved complexity [S. Schwoon 2002]
Fast algorithm in practice [ATVA'21]

Add transitions to P-Automaton until saturation.

Background: $post^*$ preserves regularity [Büchi 1964]

For a regular set of configurations C, $post^*(C)$ is regular. [Büchi 1964]

Saturation procedure in polynomial time [CONCUR’97, INFINITY’97]
Algorithm with improved complexity [S. Schwoon 2002]
Fast algorithm in practice [ATVA'21]

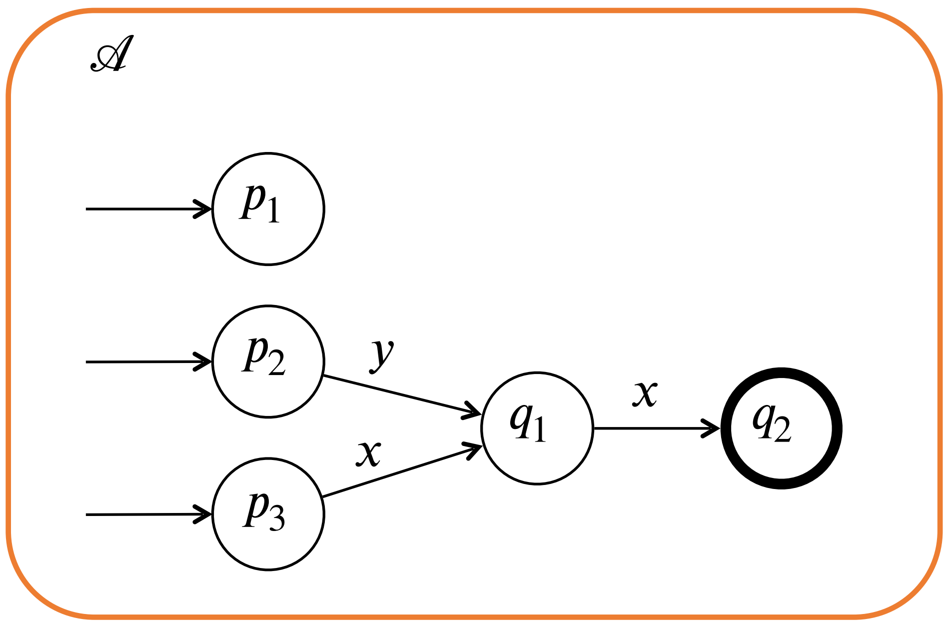
Add transitions to P-Automaton until saturation.

Background: P-automata

P-automata are representations of regular sets of configuration.
Here is an example:

example

$\mathscr{P}$-automaton



Set of configurations

C

$lang(\mathscr{A}) =$
$\{\langle p_2, yx\rangle, \langle p_3,\ xx\rangle\}$

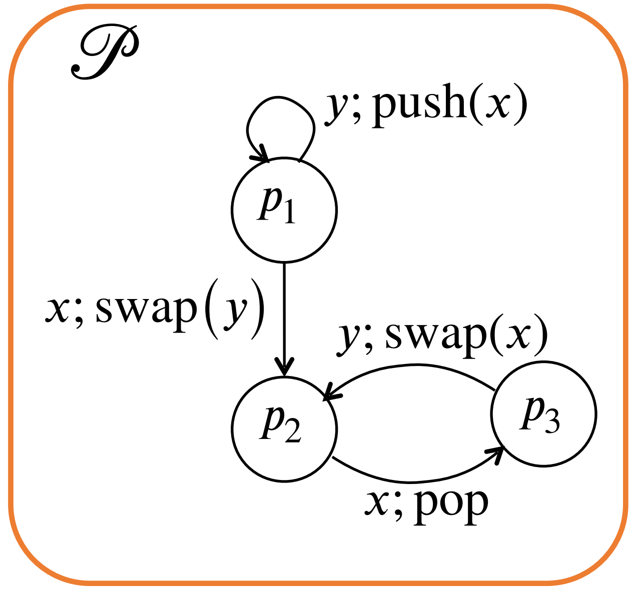
Sets recognized by some $\mathscr{P}$-*automaton* are called *regular*.

For these sets pre* can be calculated!

We will now look at why and how.

Background: Example of *pre** saturation procedure

Pushdown system



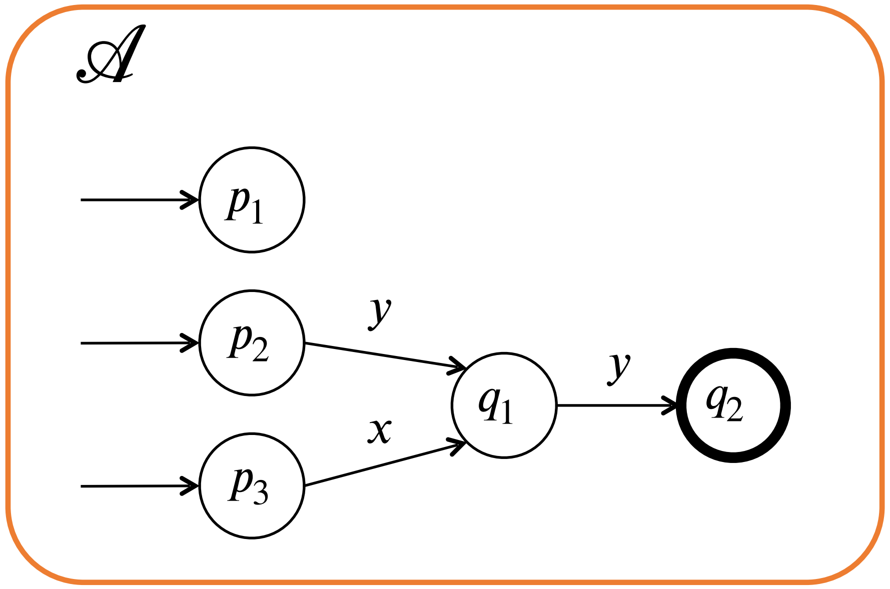
Set of configurations

C

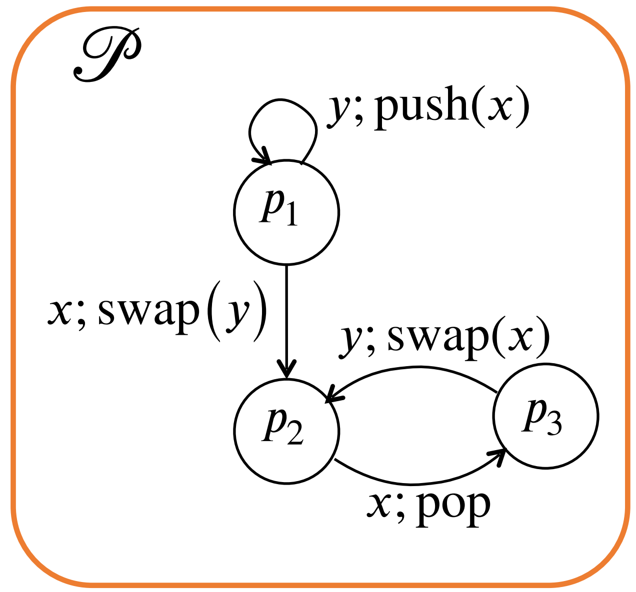
$$lang(\mathscr{A}) = \{\langle p_2, yx\rangle, \langle p_3, xx\rangle\}$$

Background: Example of pre^* saturation procedure

$\mathscr{P}$-automaton



Pushdown system



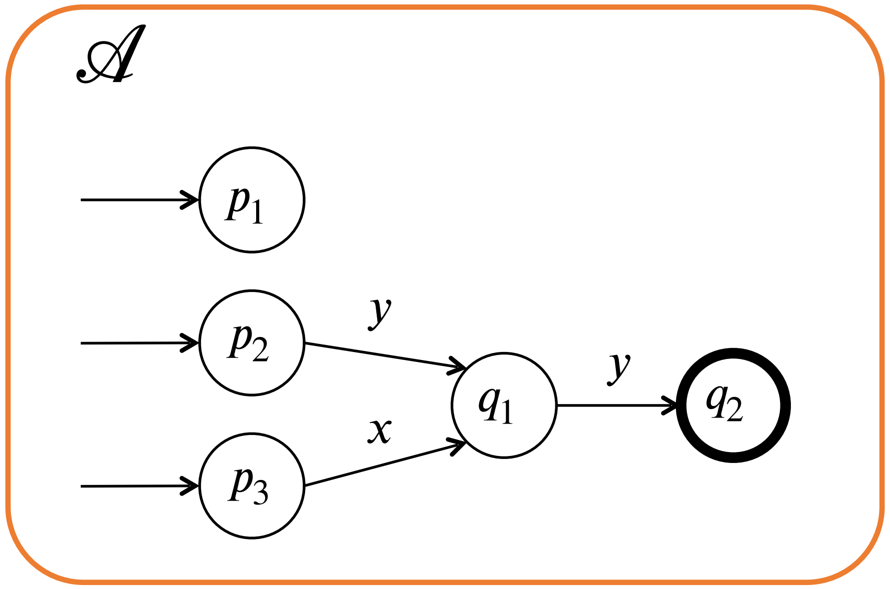
Set of configurations

C

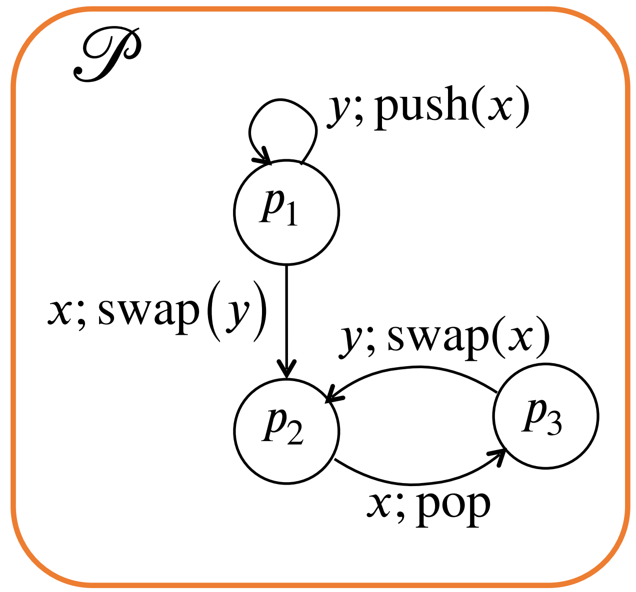
$$lang(\mathscr{A}) = \{\langle p_2, yx\rangle, \langle p_3, xx\rangle\}$$

Background: Example of *pre** saturation procedure

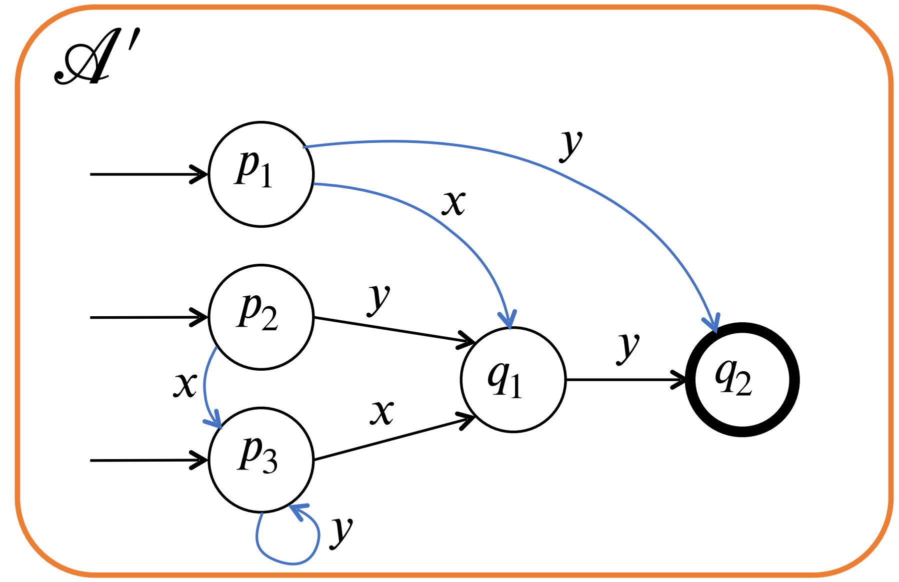
$\mathscr{P}$-automaton



Pushdown system



Saturated $\mathscr{P}$-automaton



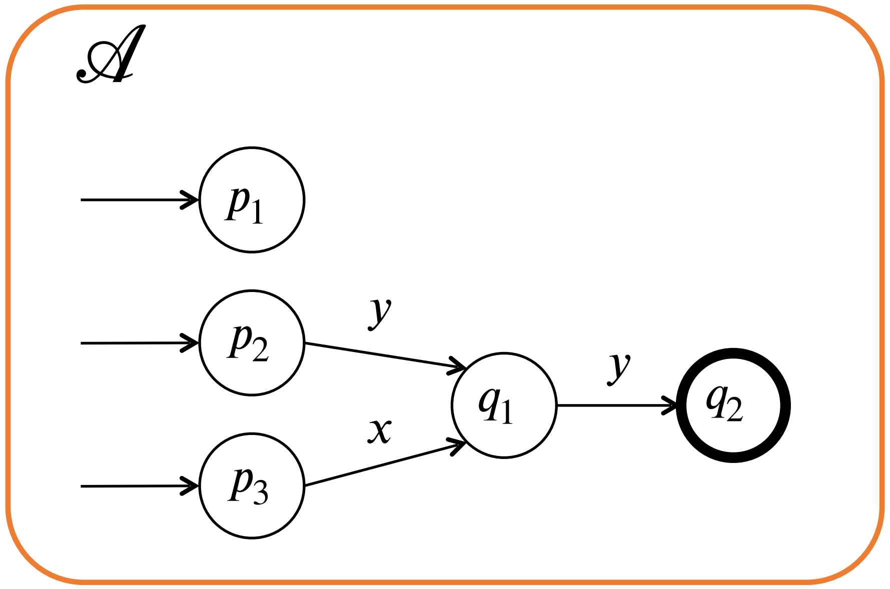
Set of configurations

C

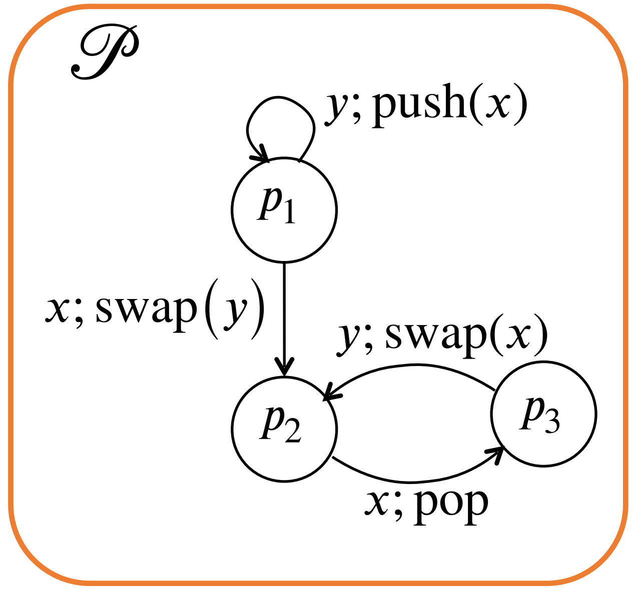
$$lang(\mathscr{A}) = \{\langle p_2, yx \rangle, \langle p_3, xx \rangle\}$$

Background: Example of *pre** saturation procedure

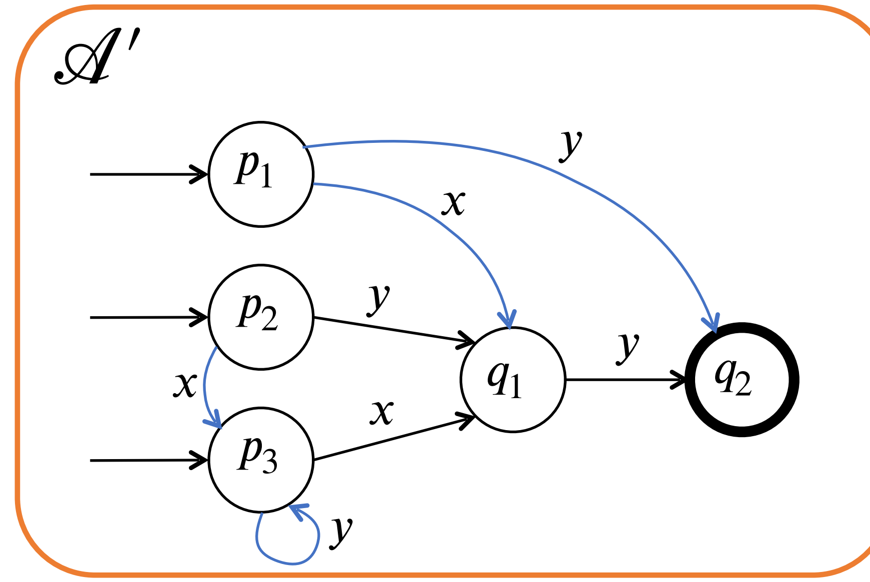
$\mathscr{P}$-automaton



Pushdown system



Saturated $\mathscr{P}$-automaton



Set of configurations

C

$$lang(\mathscr{A}) = \{\langle p_2, yx\rangle, \langle p_3, xx\rangle\}$$

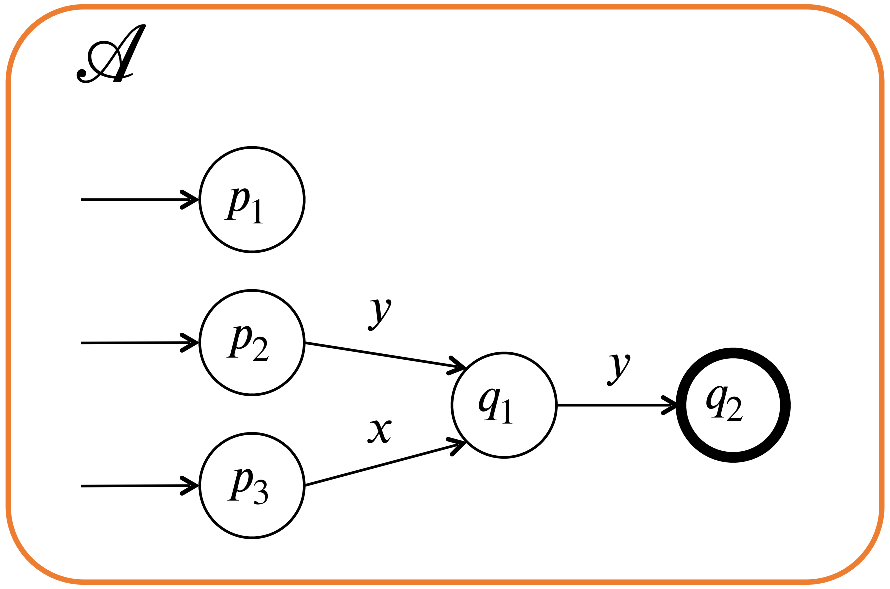
Set of predecessor configurations

$pre^*(C)$

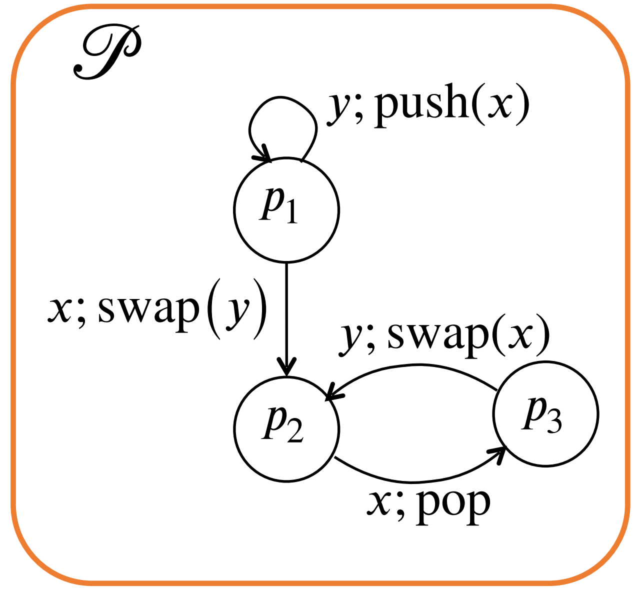
$$lang(\mathscr{A}') = \{\langle p_1, y\rangle, \langle p_1, xy\rangle, \langle p_2, yy\rangle\}$$
$$\cup \{\langle p_2, xy^n xy\rangle \mid n \geq 0\}$$
$$\cup \{\langle p_2, y^n xy\rangle \mid n > 0\}$$

Background: Example of *pre** saturation procedure

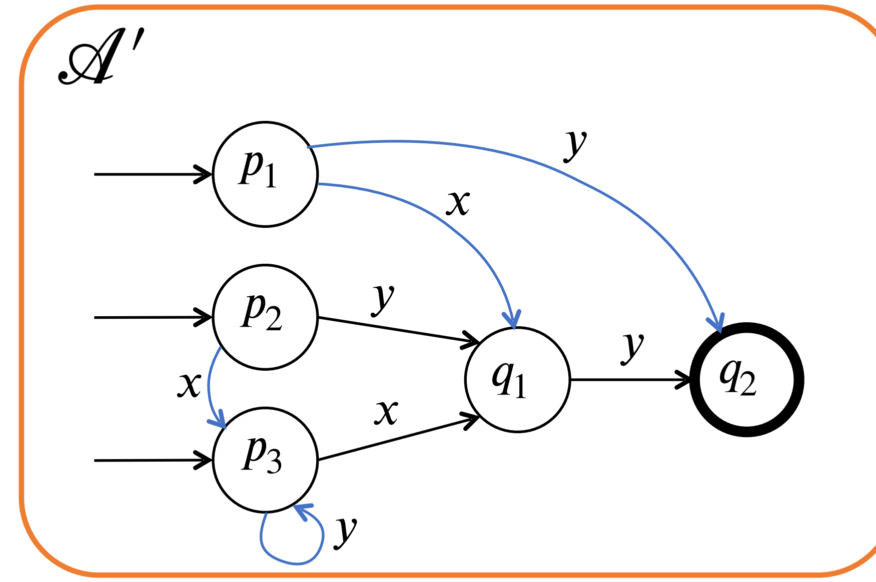
$\mathscr{P}$-automaton



Pushdown system



Saturated $\mathscr{P}$-automaton



Set of configurations

C

$$lang(\mathscr{A}) = \{\langle p_2, yx\rangle, \langle p_3, xx\rangle\}$$

Set of predecessor configurations

$pre^*(C)$

$$lang(\mathscr{A}') = \{\langle p_1, y\rangle, \langle p_1, xy\rangle, \langle p_2, yy\rangle\}$$
$$\cup \{\langle p_2, xy^n xy\rangle \mid n \geq 0\}$$
$$\cup \{\langle p_2, y^n xy\rangle \mid n > 0\}$$

This can be done in polynomial time.

PDAAAL

An implementation of pushdown reachability.

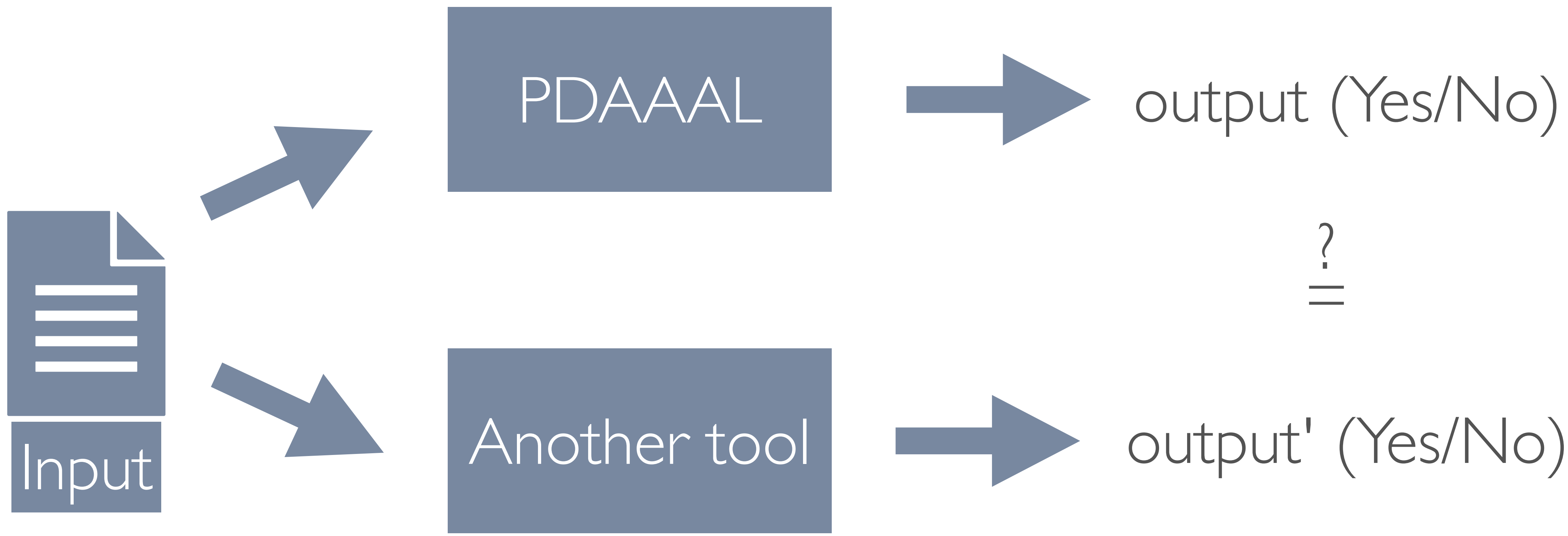
Used in particular for analysis of MPLS communication protocols.

Based on pre*, post* and a new algorithm dual*. [ATVA'21]

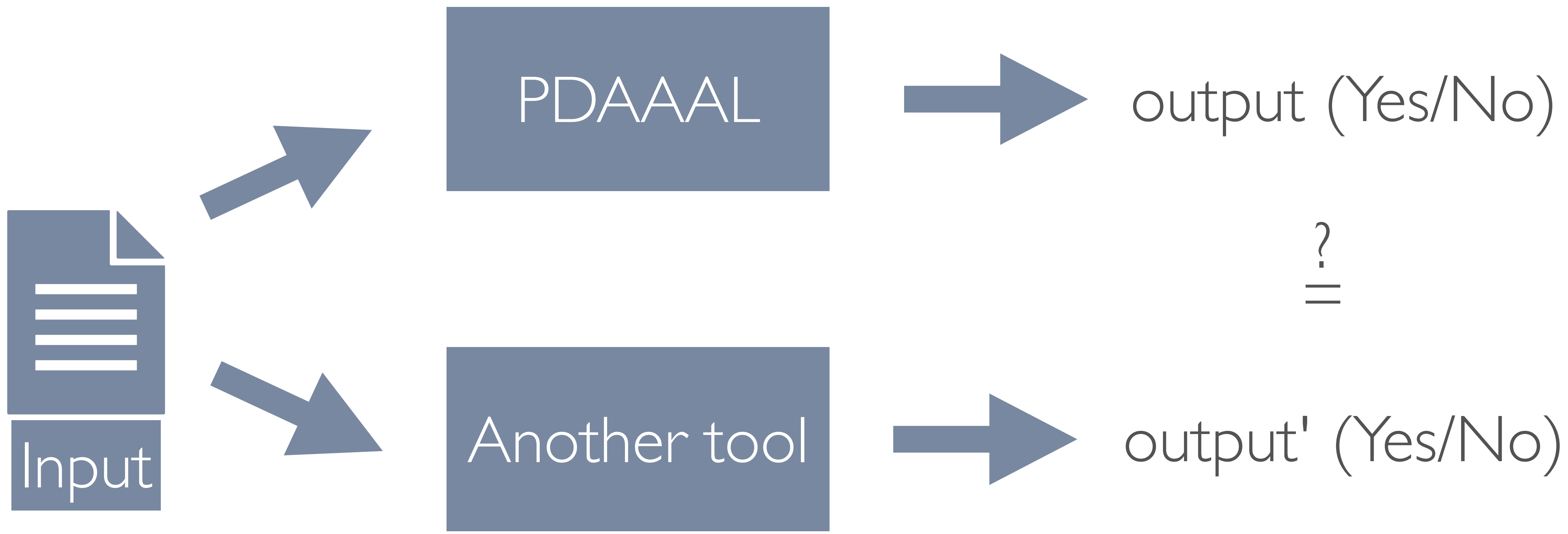
What if PDAAAL has a bug?

We want to avoid that!

Differential testing

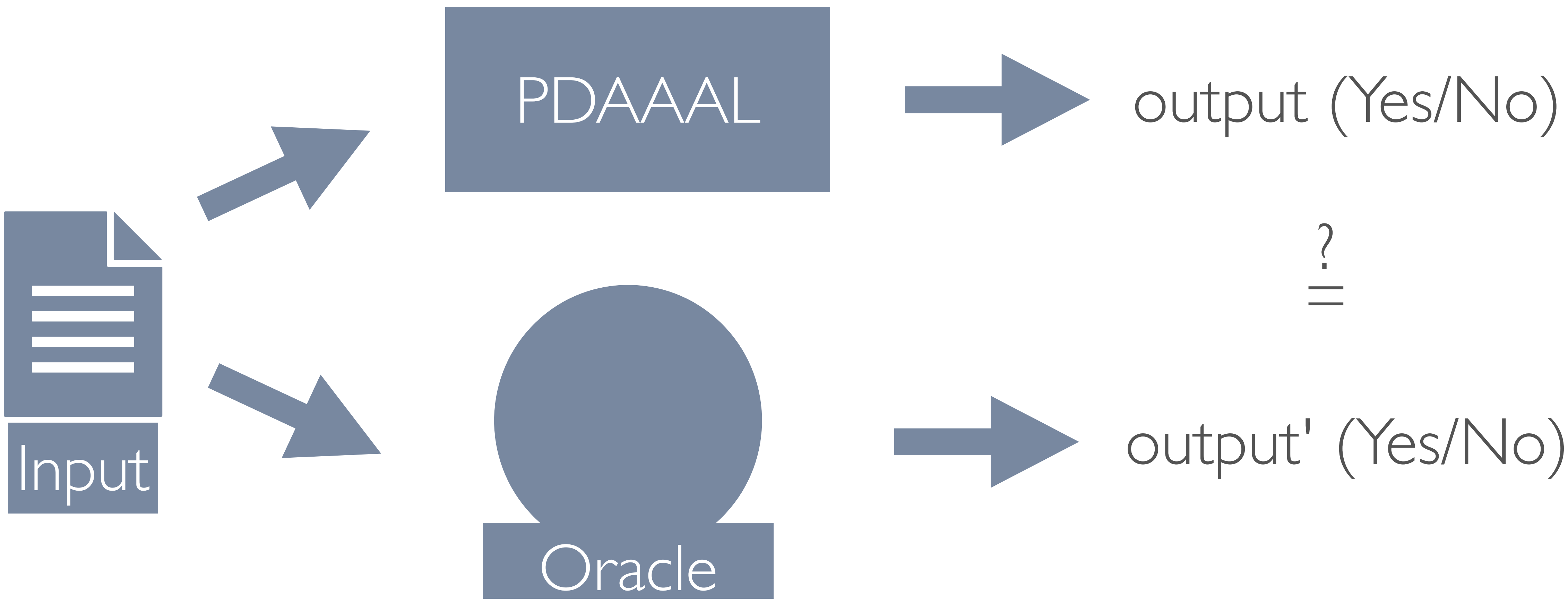


Differential testing

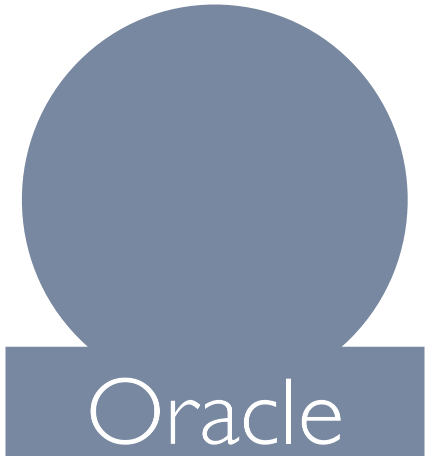


What is the ideal tool to test against?

Differential testing



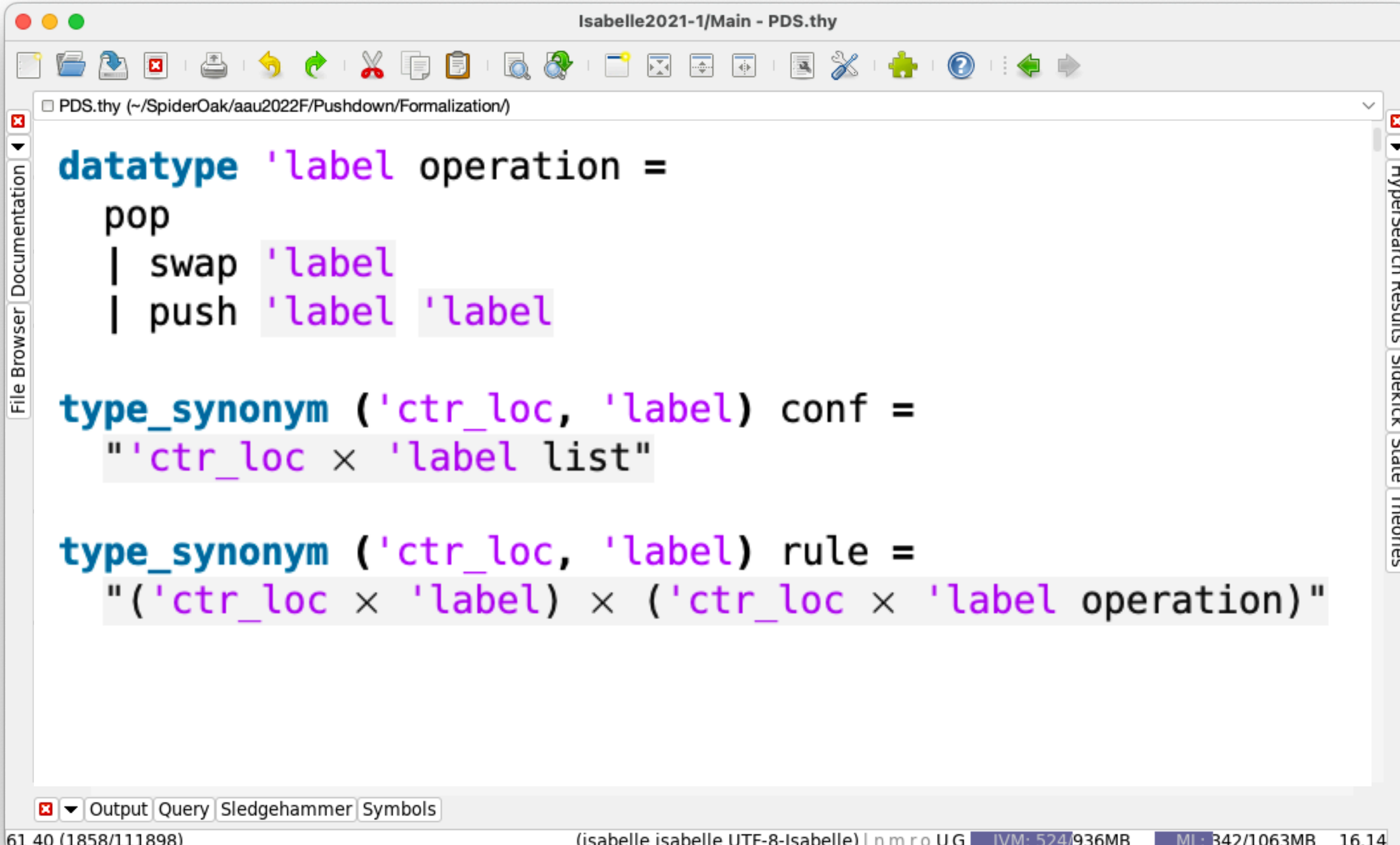
Oracle

How do we build an 

?

With Isabelle/HOL!

Isabelle/HOL



Isabelle/HOL

Isabelle2021-1/Main - PDS.thy

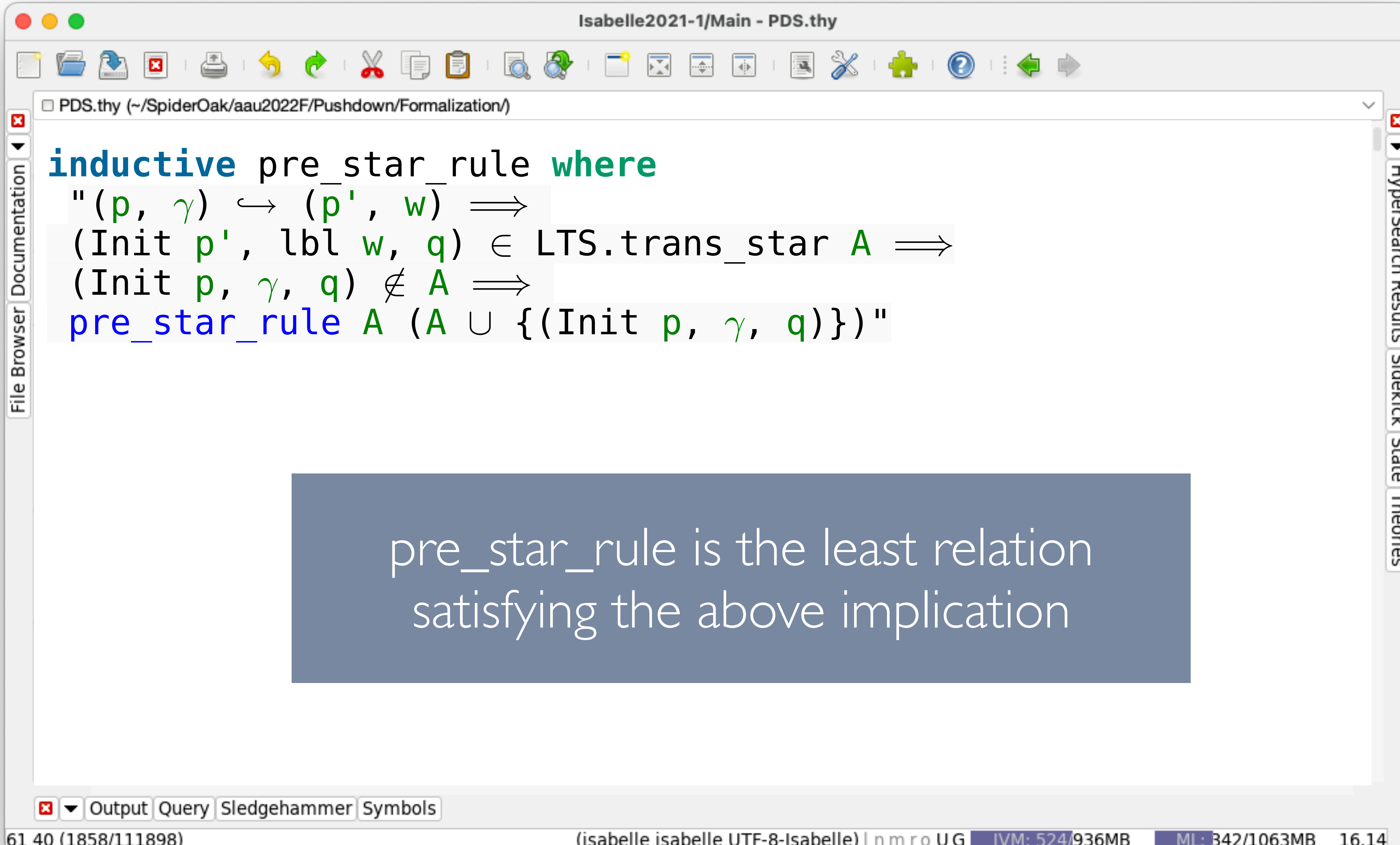
PDS.thy (~/SpiderOak/aau2022F/Pushdown/Formalization/)

```
fun lbl where
  "lbl pop = []"
| "lbl (swap γ) = [γ]"
| "lbl (push γ  γ') = [γ, γ']"

fun accepts where
  "accepts A (p,w) ⟷
     (∃q ∈ finals. (Init p,w,q) ∈ LTS.trans_star A )"

fun lang where
  "lang A = {c. accepts A c}"

fun pre_star where
  "pre_star C = {c'. ∃c ∈ C. c' ⇒* c}"
```

Output Query Sledgehammer Symbols

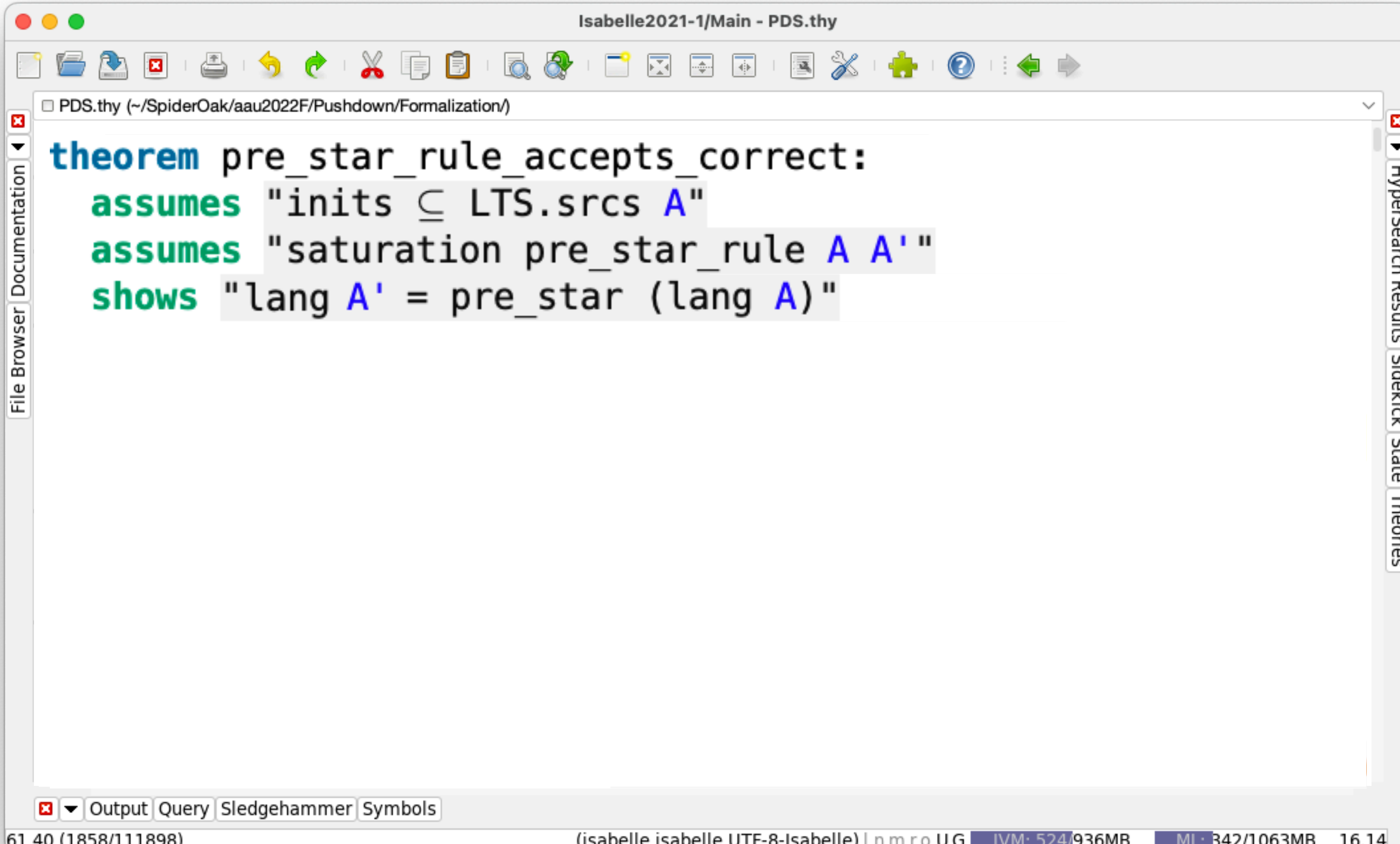
61,40 (1858/111898) (isabelle,isabelle,UTF-8-Isabelle) | n m r o U G JVM: 524/936MB ML: 342/1063MB 16.14

Isabelle/HOL



pre_star_rule is the least relation satisfying the above implication

Isabelle/HOL



Isabelle/HOL

Isabelle2021-1/Main - PDS.thy

PDS.thy (~/SpiderOak/aau2022F/Pushdown/Formalization/)

```
theorem pre_star_rule_accepts_correct:
  assumes "inits ⊆ LTS.srcs A"
  assumes "saturation pre_star_rule A A'"
  shows "lang A' = pre_star (lang A)"
proof (rule; rule)
  fix c :: "'ctr_loc × 'label list"
  define p where "p = fst c"
  define w where "w = snd c"
  assume "c ∈ pre_star (lang A)"
  then have "(p,w) ∈ pre_star (lang A)"
    unfolding p_def w_def by auto
  then have "∃p' w'. (p',w') ∈ lang A ∧ (p,w) ⇒* (p',w')"
    unfolding pre_star_def by force
  then obtain p' w' where "(p',w') ∈ lang A ∧ (p,w) ⇒* (p',w
```

Output Query Sledgehammer Symbols

61,40 (1858/111898) (isabelle,isabelle,UTF-8-Isabelle) | n m r o U G JVM: 524/936MB ML: 342/1063MB 16.14

Formalized Correctness Theorem

We prove similar theorems for post* and dual*.

Total effort:

4400 Isabelle LOC. ~2 person months.

We followed [S. Schwoon 2002].

Lehrstuhl für Informatik VII
der Technischen Universität München

Model-Checking Pushdown Systems

Stefan Schwoon

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

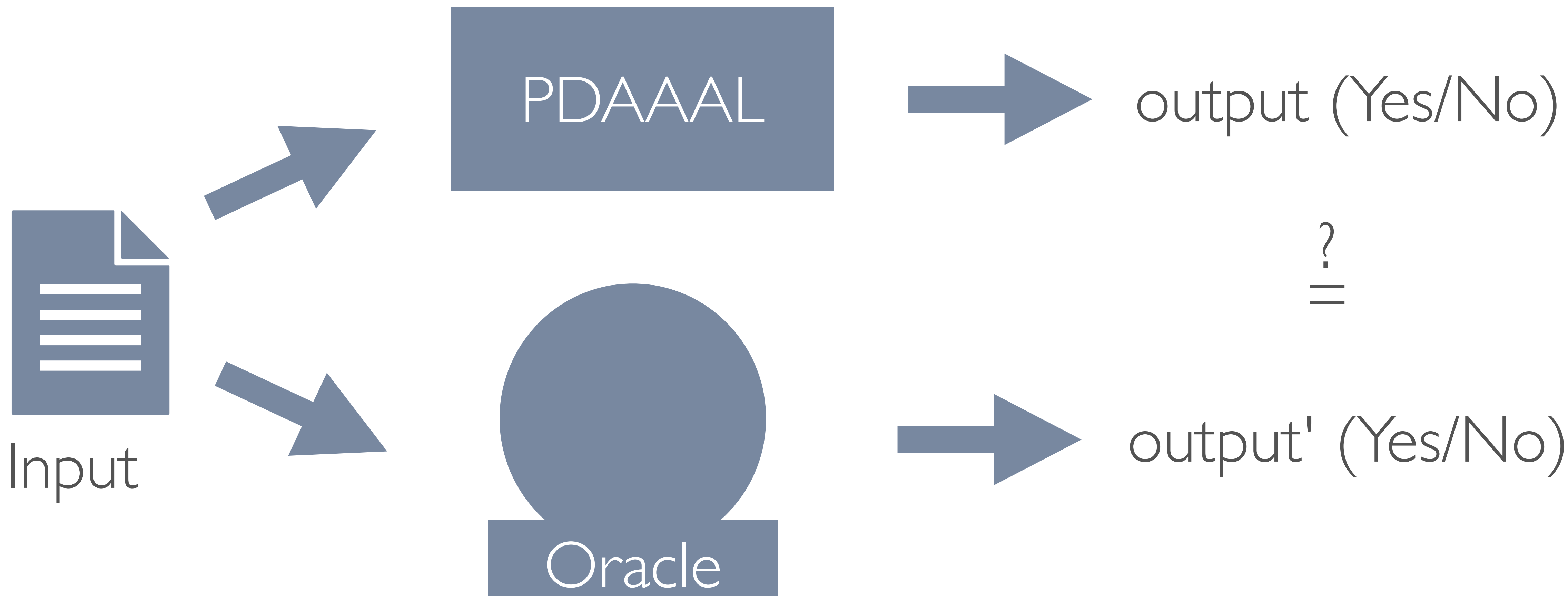
Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. T. Nipkow, Ph.D.

Prüfer der Dissertation: 1. Univ.-Prof. Dr. Dr.h.c. W. Brauer
2. Prof. Dr. J. Esparza,

Differential testing



What do we pick for the input?

What do we pick for the input?

Phase 1

- Real world test from the network domain
- In total: 25 512 test cases.

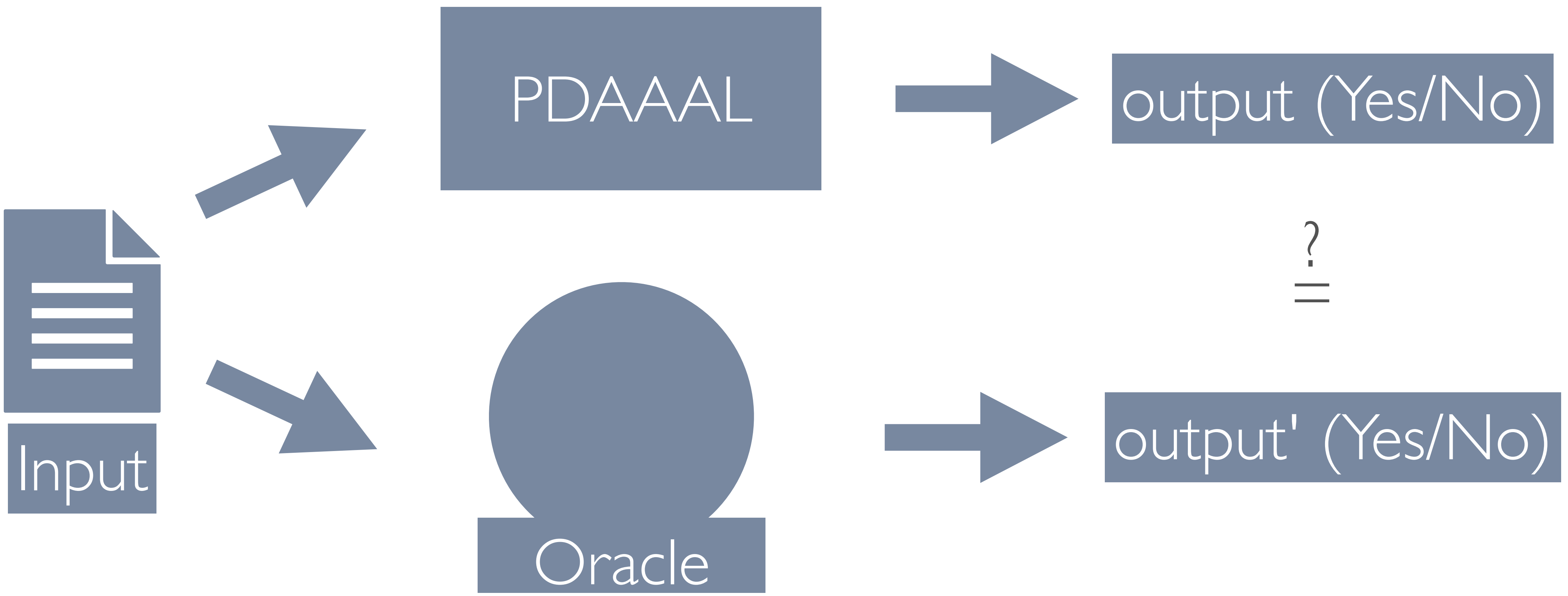
Phase 2

- Randomly generated pushdown systems
  - 4 control locations, 5 labels, up to 200 pushdown rules, up to 13 automata transitions
- In total: 15 000 test cases.

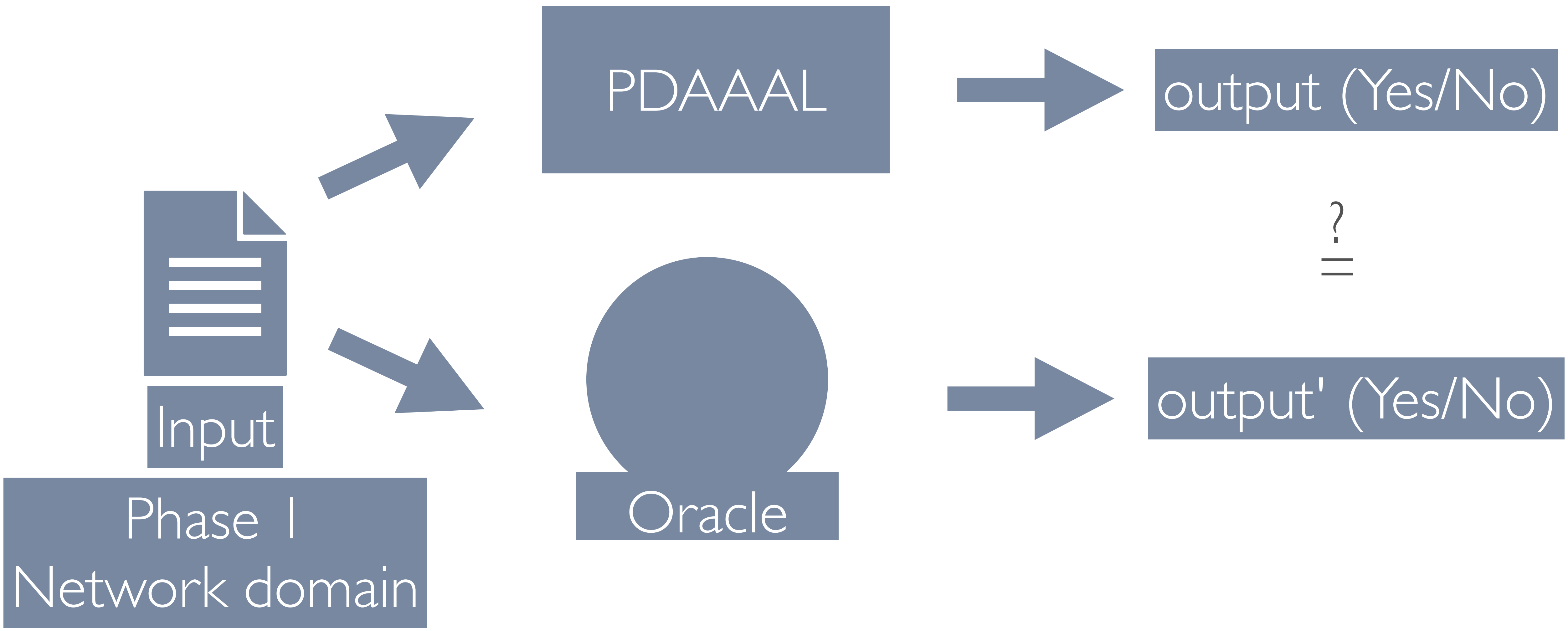
Phase 3

- Exhaustive testing up to a small size
  - 2 control locations, 2 stack symbols, up to 2 pushdown rules, automata with respectively 2 and 1 initial state, up to 2 automata transitions
- In total: 27 000 000 test cases.

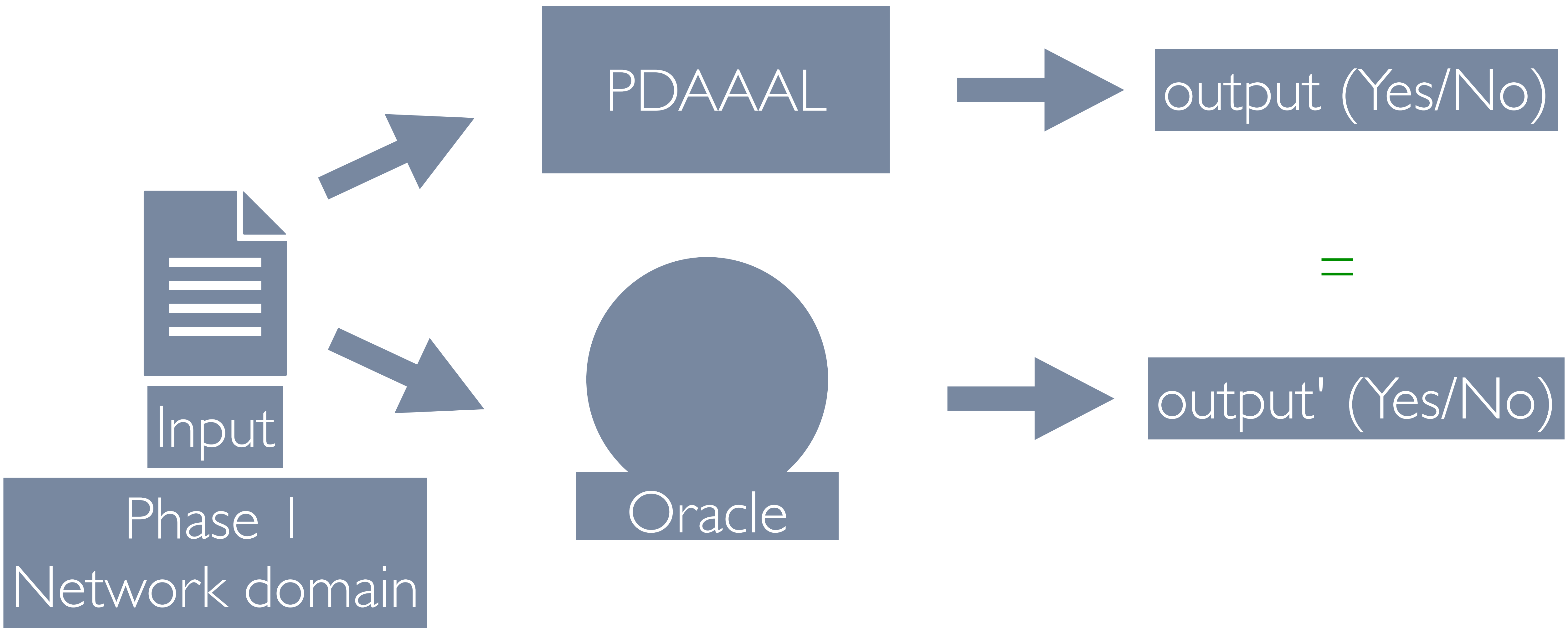
Differential testing



Differential testing

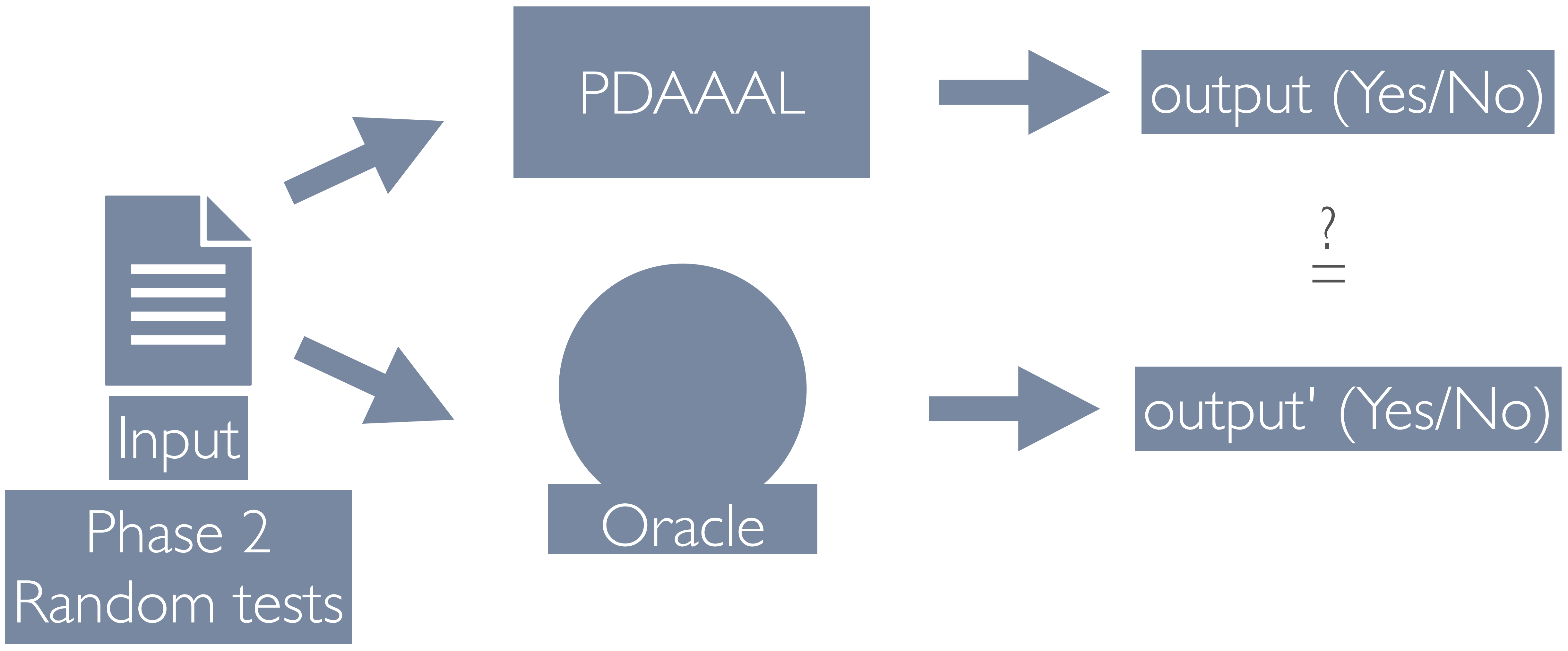


Differential testing

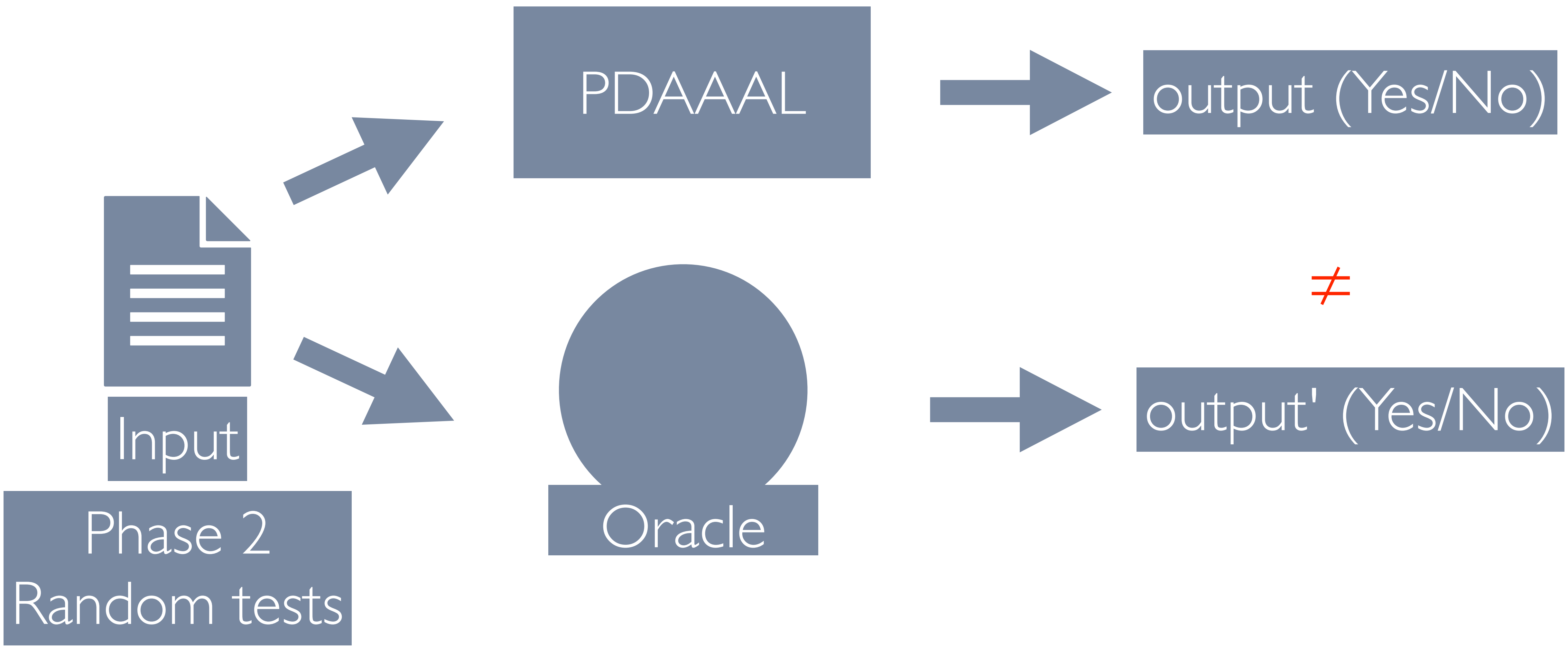


PDAAL and the oracle agree!

Differential testing



Differential testing



~1500 test where they disagree!

Let's look at an example

$\Delta=\{$
$(p_0,D)\hookrightarrow(p_0,\text{swap } A),$
$(p_0,E)\hookrightarrow(p_0,\text{push } B\ E),$
$(p_0,D)\hookrightarrow$
$(p_0,\text{push } D\ D),$
$(p_0,D)\hookrightarrow(p_0,\text{pop}),$
$(p_0,D)\hookrightarrow(p_1,\text{swap } A),$
$(p_0,A)\hookrightarrow(p_1,\text{push } C\ A),$
$(p_0,E)\hookrightarrow(p_2,\text{push } A\ E),$

$(p_1,B)\hookrightarrow(p_1,\text{swap } C),$
$(p_1,E)\hookrightarrow(p_1,\text{swap } C),$
$(p_1,A)\hookrightarrow(p_2,\text{swap } A),$
$(p_1,D)\hookrightarrow(p_2,\text{swap } D),$
$(p_1,C)\hookrightarrow(p_2,\text{swap } E),$
$(p_1,C)\hookrightarrow(p_3,\text{swap } D),$
$(p_1,D)\hookrightarrow(p_3,\text{pop}),$
$(p_2,B)\hookrightarrow(p_0,\text{push } A\ B),$
$(p_2,A)\hookrightarrow(p_0,\text{push } C\ A),$

$(p_2,B)\hookrightarrow(p_2,\text{swap } E),$
$(p_2,E)\hookrightarrow(p_3,\text{push } A\ E),$
$(p_2,B)\hookrightarrow(p_3,\text{push } C\ B),$
$(p_3,D)\hookrightarrow$
$(p_0,\text{push } B\ D),$
$(p_3,C)\hookrightarrow(p_0,\text{push } E\ C),$
$(p_3,C)\hookrightarrow(p_0,\text{swap } E),$
$(p_3,C)\hookrightarrow(p_1,\text{push } A\ C),$
$(p_3,B)\hookrightarrow(p_1,\text{pop}),$

$(p_0,B)\hookrightarrow(p_2,\text{push } D\ B),$
$(p_0,C)\hookrightarrow(p_2,\text{swap } D),$
$(p_0,E)\hookrightarrow(p_2,\text{swap } E),$
$(p_0,E)\hookrightarrow(p_3,\text{push } B\ E),$
$(p_0,C)\hookrightarrow(p_3,\text{swap } E),$
$(p_1,B)\hookrightarrow(p_0,\text{swap } C),$
$(p_1,D)\hookrightarrow(p_0,\text{swap } C),$
$(p_1,C)\hookrightarrow(p_0,\text{swap } B),$
$(p_1,C)\hookrightarrow(p_0,\text{swap } E),$

$(p_2,C)\hookrightarrow(p_0,\text{push } C\ C),$
$(p_2,D)\hookrightarrow$
$(p_0,\text{push } B\ D),$
$(p_2,C)\hookrightarrow(p_1,\text{push } C\ C),$
$(p_2,A)\hookrightarrow(p_1,\text{push } B\ A),$
$(p_2,A)\hookrightarrow(p_2,\text{push } A\ A),$
$(p_2,C)\hookrightarrow(p_2,\text{swap } A),$
$(p_2,E)\hookrightarrow(p_2,\text{swap } A),$
$(p_2,A)\hookrightarrow(p_2,\text{push } B\ A),$

$(p_3,E)\hookrightarrow(p_2,\text{swap } C),$
$(p_3,B)\hookrightarrow(p_2,\text{push } D\ B),$
$(p_3,E)\hookrightarrow(p_3,\text{swap } A),$
$(p_3,A)\hookrightarrow(p_3,\text{push } C\ A),$
$(p_3,E)\hookrightarrow(p_3,\text{swap } D),$
$(p_3,C)\hookrightarrow(p_3,\text{pop})\}$

$$
\begin{aligned}
A_1 &= \{(\text{Init } p_0, B, \text{Noninit } q_1), (\text{Init } p_0, D, \text{Noninit } q_0),\\
&\quad (\text{Init } p_2, B, \text{Noninit } q_0), (\text{Init } p_3, A, \text{Noninit } q_2),\\
&\quad (\text{Noninit } q_0, D, \text{Noninit } q_1), (\text{Noninit } q_2, C, \text{Noninit } q_0)\}\\
F_1 &= \{\} \qquad F_1^{ni} = \{q_1\}\\
A_2 &= \{(\text{Init } p_2,\ A,\ \text{Noninit } q_0)), (\text{Init } p_2,\ B,\ \text{Noninit } q_0)\}\\
F_2 &= \{p_0, p_2\} \quad F_2^{ni} = \{\}
\end{aligned}
$$

Let's look at an example

$\Delta=\{$
$(p_0,D)\hookrightarrow(p_0,\text{swap } A)$,
$(p_0,E)\hookrightarrow(p_0,\text{push } B\ E)$,
$(p_0,D)\hookrightarrow$
$(p_0,\text{push } D\ D)$,
$(p_0,D)\hookrightarrow(p_0,\text{pop})$,
$(p_0,D)\hookrightarrow(p_1,\text{swap } A)$,
$(p_0,A)\hookrightarrow(p_1,\text{push } C\ A)$,
$(p_0,E)\hookrightarrow(p_2,\text{push } A\ E)$,
$(p_1,B)\hookrightarrow(p_1,\text{swap } C)$,
$(p_1,E)\hookrightarrow(p_1,\text{swap } C)$,
$(p_1,A)\hookrightarrow(p_2,\text{swap } A)$,
$(p_1,D)\hookrightarrow(p_2,\text{swap } D)$,
$(p_1,C)\hookrightarrow(p_2,\text{swap } E)$,
$(p_1,C)\hookrightarrow(p_3,\text{swap } D)$,
$(p_1,D)\hookrightarrow(p_3,\text{pop})$,
$(p_2,B)\hookrightarrow(p_0,\text{push } A\ B)$,
$(p_2,A)\hookrightarrow(p_0,\text{push } C\ A)$,
$(p_2,B)\hookrightarrow(p_2,\text{swap } E)$,
$(p_2,E)\hookrightarrow(p_3,\text{push } A\ E)$,
$(p_2,B)\hookrightarrow(p_3,\text{push } C\ B)$,
$(p_3,D)\hookrightarrow$
$(p_0,\text{push } B\ D)$,
$(p_3,C)\hookrightarrow(p_0,\text{push } E\ C)$,
$(p_3,C)\hookrightarrow(p_0,\text{swap } E)$,
$(p_3,C)\hookrightarrow(p_1,\text{push } A\ C)$,
$(p_3,B)\hookrightarrow(p_1,\text{pop})$,

$(p_0,B)\hookrightarrow(p_2,\text{push } D\ B)$,
$(p_0,C)\hookrightarrow(p_2,\text{swap } D)$,
$(p_0,E)\hookrightarrow(p_2,\text{swap } E)$,
$(p_0,E)\hookrightarrow(p_3,\text{push } B\ E)$,
$(p_0,C)\hookrightarrow(p_3,\text{swap } E)$,
$(p_1,B)\hookrightarrow(p_0,\text{swap } C)$,
$(p_1,D)\hookrightarrow(p_0,\text{swap } C)$,
$(p_1,C)\hookrightarrow(p_0,\text{swap } B)$,
$(p_1,C)\hookrightarrow(p_0,\text{swap } E)$,
$(p_2,C)\hookrightarrow(p_0,\text{push } C\ C)$,
$(p_2,D)\hookrightarrow$
$(p_0,\text{push } B\ D)$,
$(p_2,C)\hookrightarrow(p_1,\text{push } C\ C)$,
$(p_2,A)\hookrightarrow(p_1,\text{push } B\ A)$,
$(p_2,A)\hookrightarrow(p_2,\text{push } A\ A)$,
$(p_2,C)\hookrightarrow(p_2,\text{swap } A)$,
$(p_2,E)\hookrightarrow(p_2,\text{swap } A)$,
$(p_2,A)\hookrightarrow(p_2,\text{push } B\ A)$,
$(p_3,E)\hookrightarrow(p_2,\text{swap } C)$,
$(p_3,B)\hookrightarrow(p_2,\text{push } D\ B)$,
$(p_3,E)\hookrightarrow(p_3,\text{swap } A)$,
$(p_3,A)\hookrightarrow(p_3,\text{push } C\ A)$,
$(p_3,E)\hookrightarrow(p_3,\text{swap } D)$,
$(p_3,C)\hookrightarrow(p_3,\text{pop})\}$

It's hard to diagnose the problem on big examples like this.

$$
\begin{aligned}
A_1 &= \{(\text{Init } p_0, B, \text{Noninit } q_1), (\text{Init } p_0, D, \text{Noninit } q_0),\\
&\quad (\text{Init } p_2, B, \text{Noninit } q_0), (\text{Init } p_3, A, \text{Noninit } q_2),\\
&\quad (\text{Noninit } q_0, D, \text{Noninit } q_1), (\text{Noninit } q_2, C, \text{Noninit } q_0)\}\\
F_1 &= \{\} \qquad F_1^{ni} = \{q_1\}\\
A_2 &= \{(\text{Init } p_2,\ A,\ \text{Noninit } q_0)), (\text{Init } p_2,\ B,\ \text{Noninit } q_0)\}\\
F_2 &= \{p_0, p_2\} \quad F_2^{ni} = \{\}
\end{aligned}
$$

Delta Debugging

- We implement Delta Debugging.
- ***Fully automatic*** minimization of failing test cases.

- A failing test case is seen as a set of features.
- Delta Debugging checks if some subset also fails.
- Does so until a minimal set is found.

- We fix the set of features to contain
  - (i) each pushdown rule,
  - (ii) each transition in either of the P-automata,
  - (iii) each final state in a P-automaton
    - (as opposed to it not being final).

Result

$$
\begin{aligned}
\Delta &= \{(\mathsf{p}_0, D) \hookrightarrow (\mathsf{p}_0, \mathsf{pop})\} \\
\mathsf{A}_1 &= \{(\mathsf{Init}\ \mathsf{p}_0, \mathsf{D}, \mathsf{Noninit}\ \mathsf{q}_0), (\mathsf{Noninit}\ \mathsf{q}_0, \mathsf{D}, \mathsf{Noninit}\ \mathsf{q}_1)\} \\
\mathsf{F}_1 &= \{\} \qquad \mathsf{F}_1^{\mathsf{ni}} = \{\mathsf{q}_1\} \\
\mathsf{A}_2 &= \{\} \\
\mathsf{F}_2 &= \{\mathsf{p}_0\} \quad \mathsf{F}_2^{\mathsf{ni}} = \{\}
\end{aligned}
$$

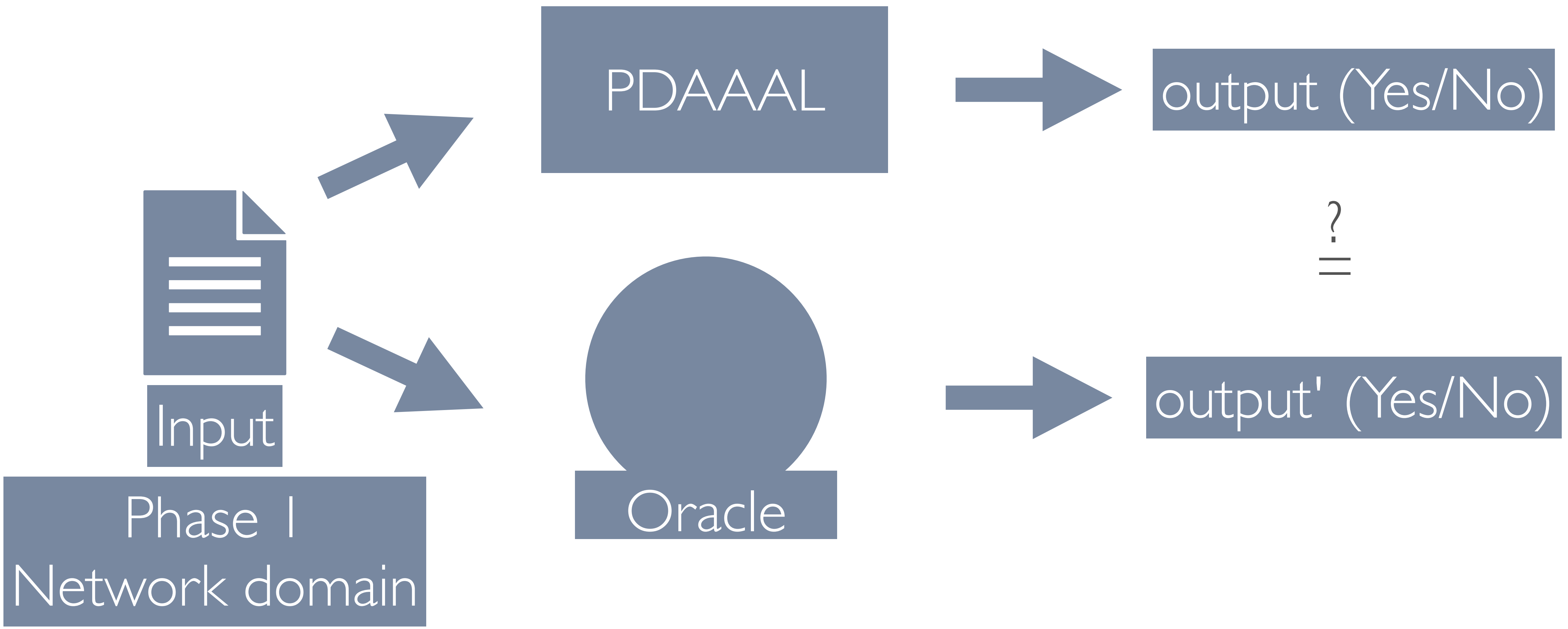
What was the problem?

- We looked at a number of minimized failing test cases.
- Common trait: P-automaton A' accepted the empty word.

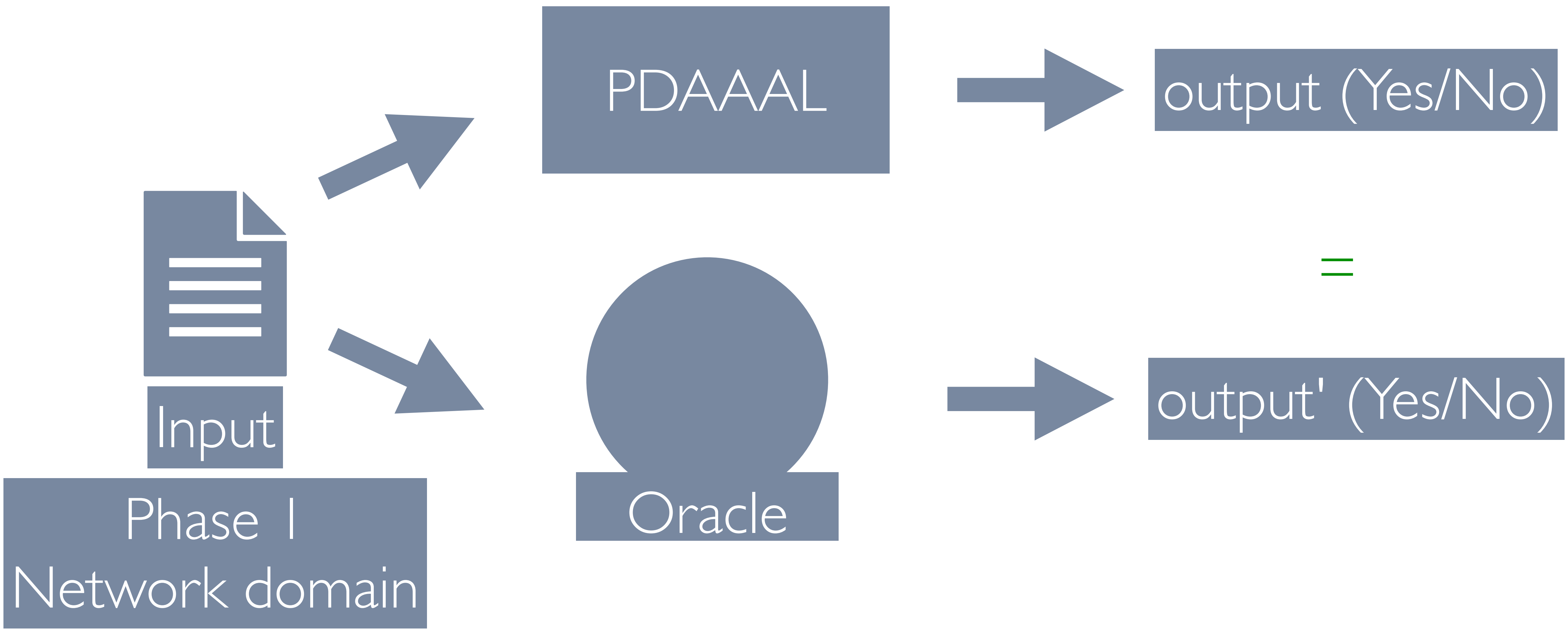
- The problem:
  - ε-transitions were not handled correctly by PDAAALs intersection algorithm.

We fixed the bug and ran the tests.

Differential testing

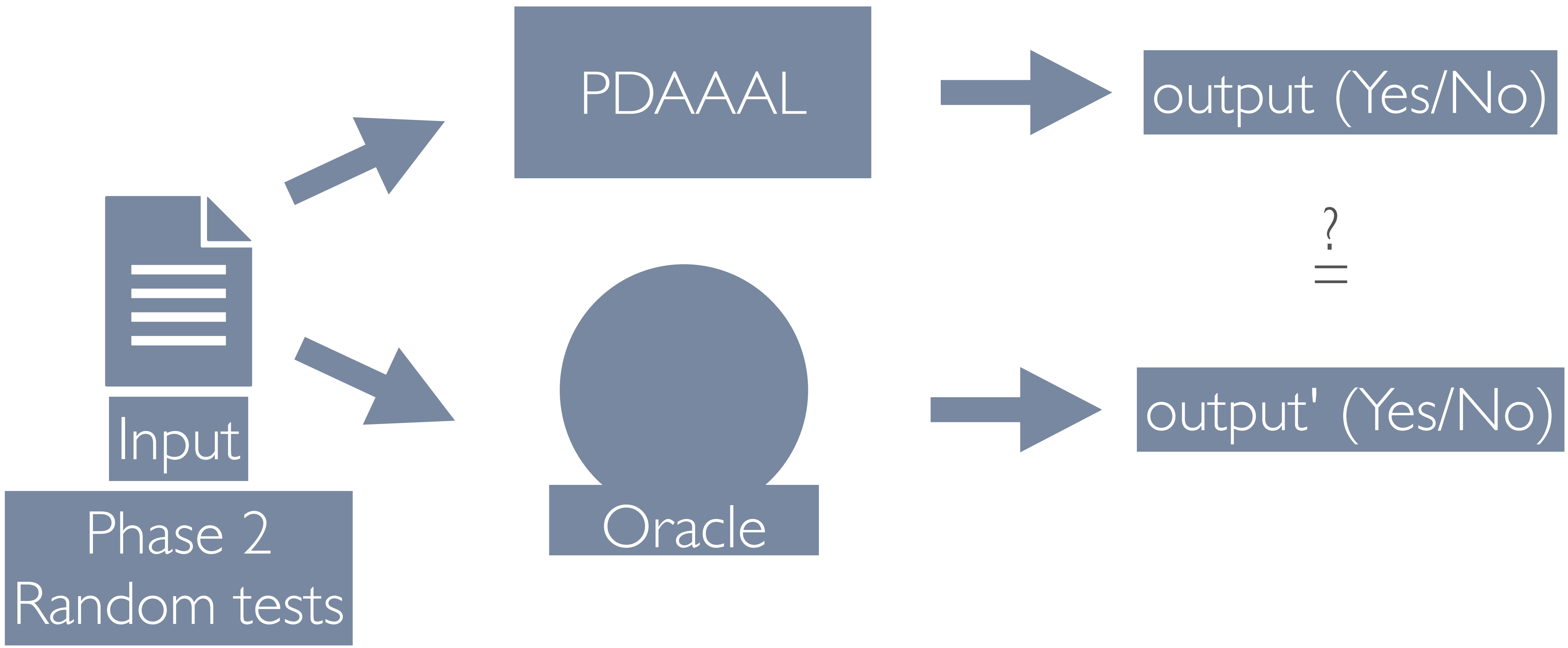


Differential testing

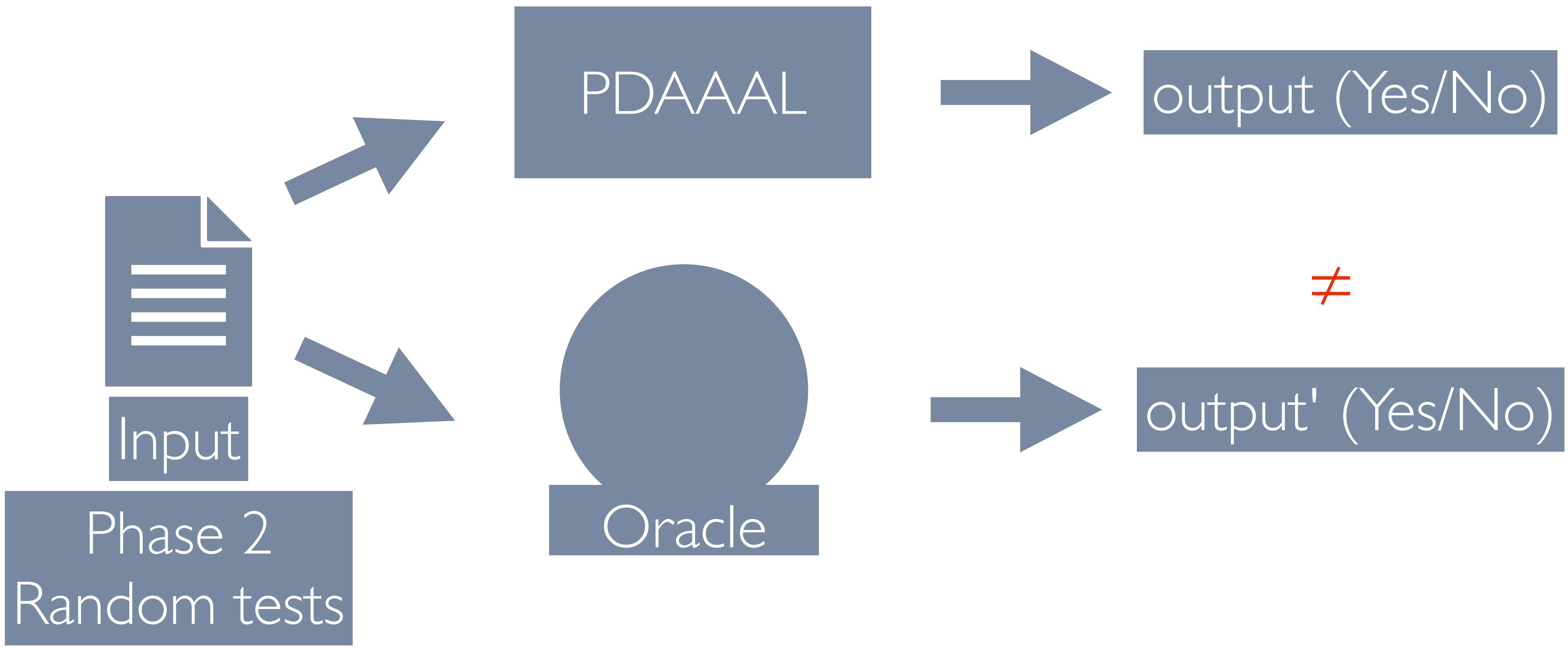


PDAAL and the oracle agree!

Differential testing



Differential testing



I test where they disagree!

Fixing another bug

We minimized the input.
We diagnosed the problem.

It involves PDAAAL's implementation of early termination.

PDAAALs algorithm

Given a PDA, a P-automaton A, and a P-automaton A':

1. Calculate pre*(A')
2. Calculate intersection automaton: A ∩ pre*(A')
3. If A ∩ pre*(A') is non-empty

        return "Reachable"

    else

        return "Non-reachable"

PDAAAL does not do 1., 2., 3. in sequence like this.
Instead it follows this idea:
When an edge is added in the calculation of pre*(A'):

    Corresponding edges are added in calculation of intersection.
    Check if intersection is non-empty.

Fixing the bug

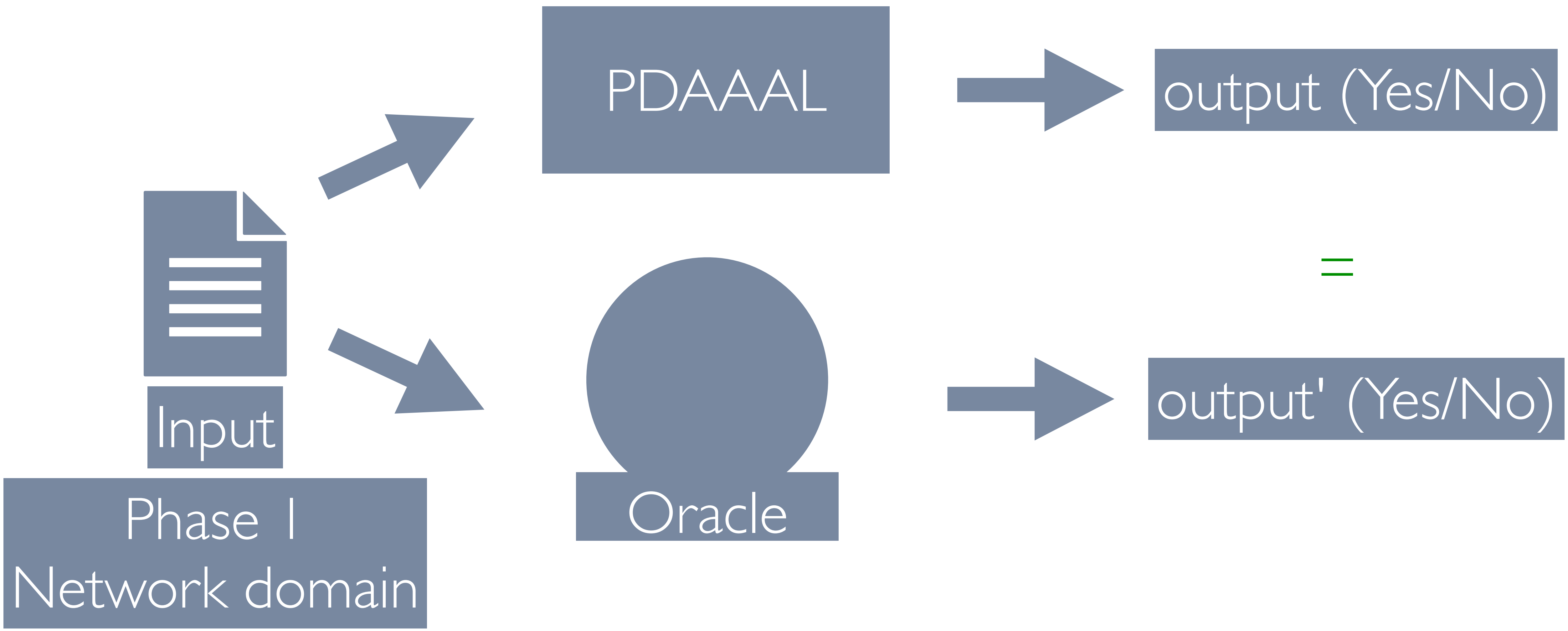
Algorithm: Automata transitions updated *before* nonemptiness check.
In PDAAAL: Automata transitions updated *after* nonemptiness check.

Should happen *before*.

Commit fixing the problem:

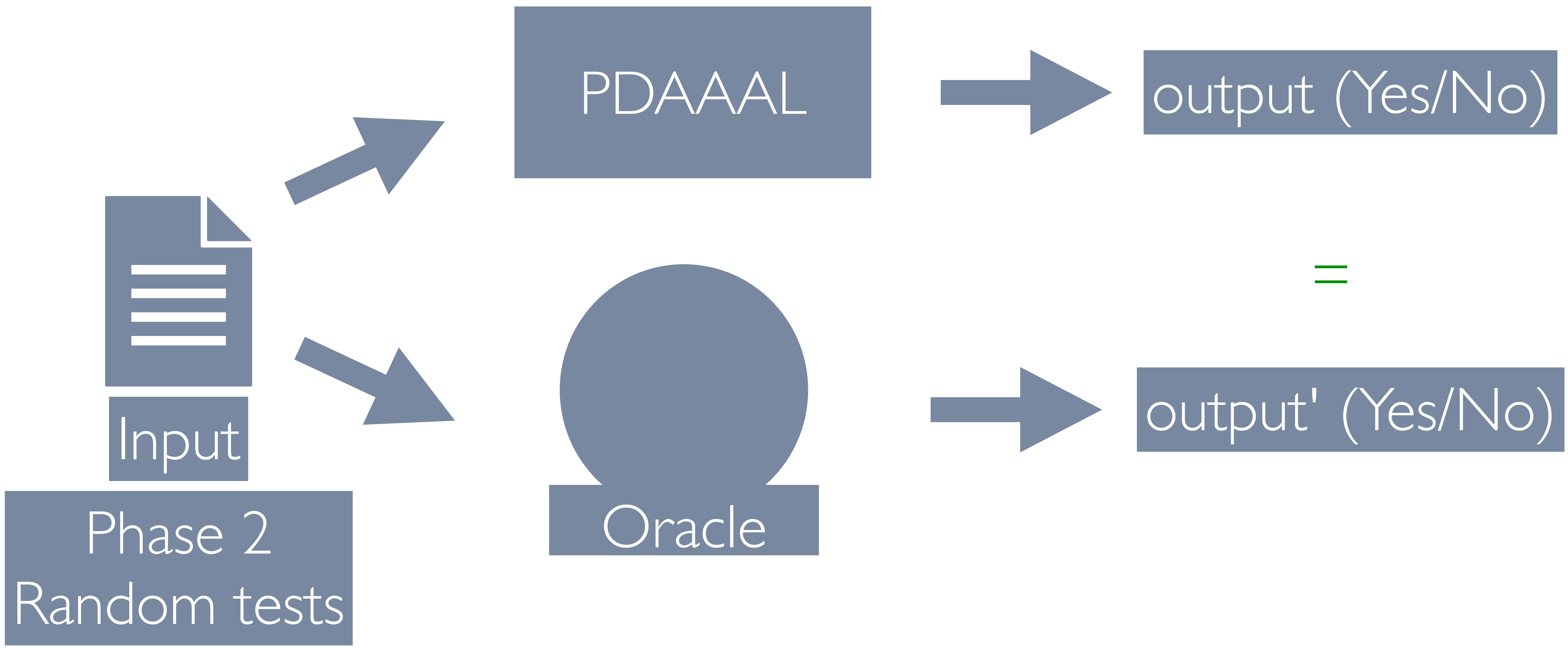
```
2 src/include/pdaaal/Solver.h

@@ -119,10 +119,10 @@ namespace pdaaal {
              if (res.second) { // New edge is not already in edges (rel U workset).
                  _workset.emplace(from, label, to);
                  if (trace != nullptr) { // Don't add existing edges
+                     _automaton.add_edge(from, to, label, trace_ptr_from<W>(trace));
                      if constexpr (ET) {
                          _found = _found || _early_termination(from, label, to, trace_ptr_from<W>(trace));
                      }
-                     _automaton.add_edge(from, to, label, trace_ptr_from<W>(trace));
                  }
              }
          };
```

Differential testing



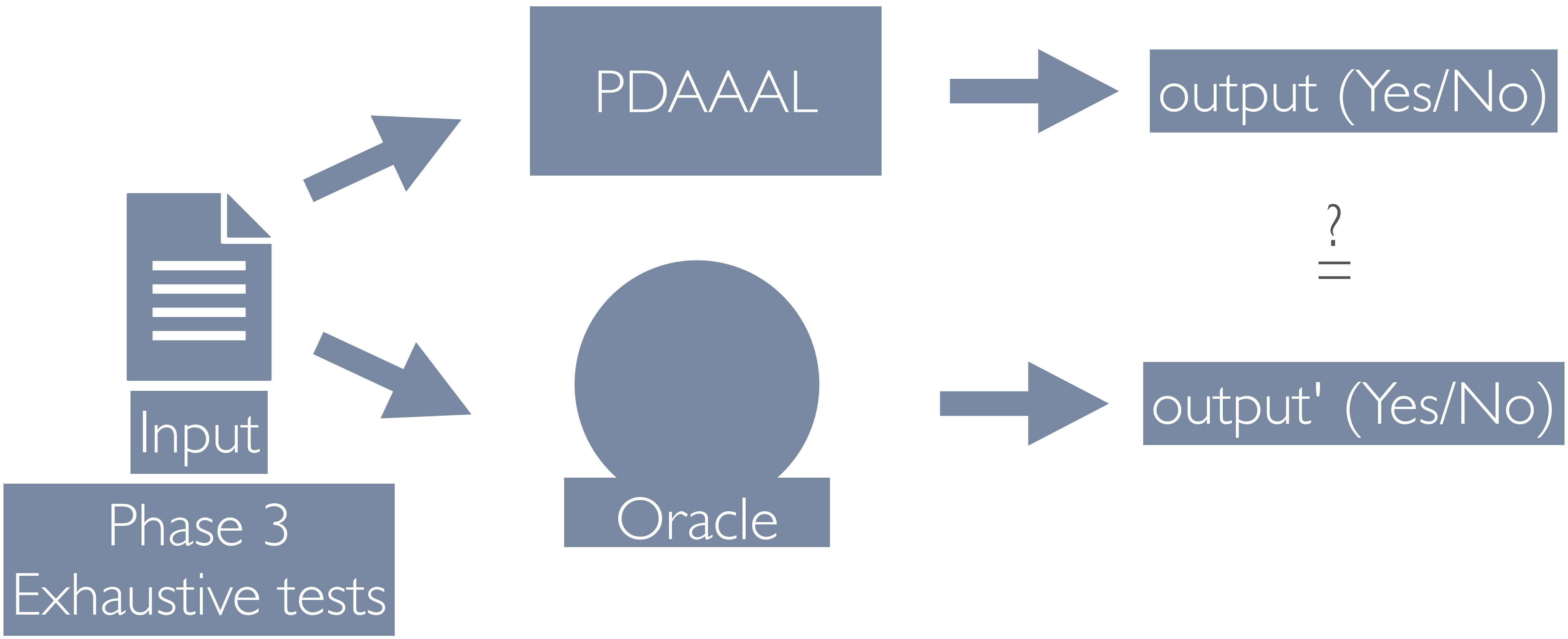
PDAAL and the oracle agree!

Differential testing

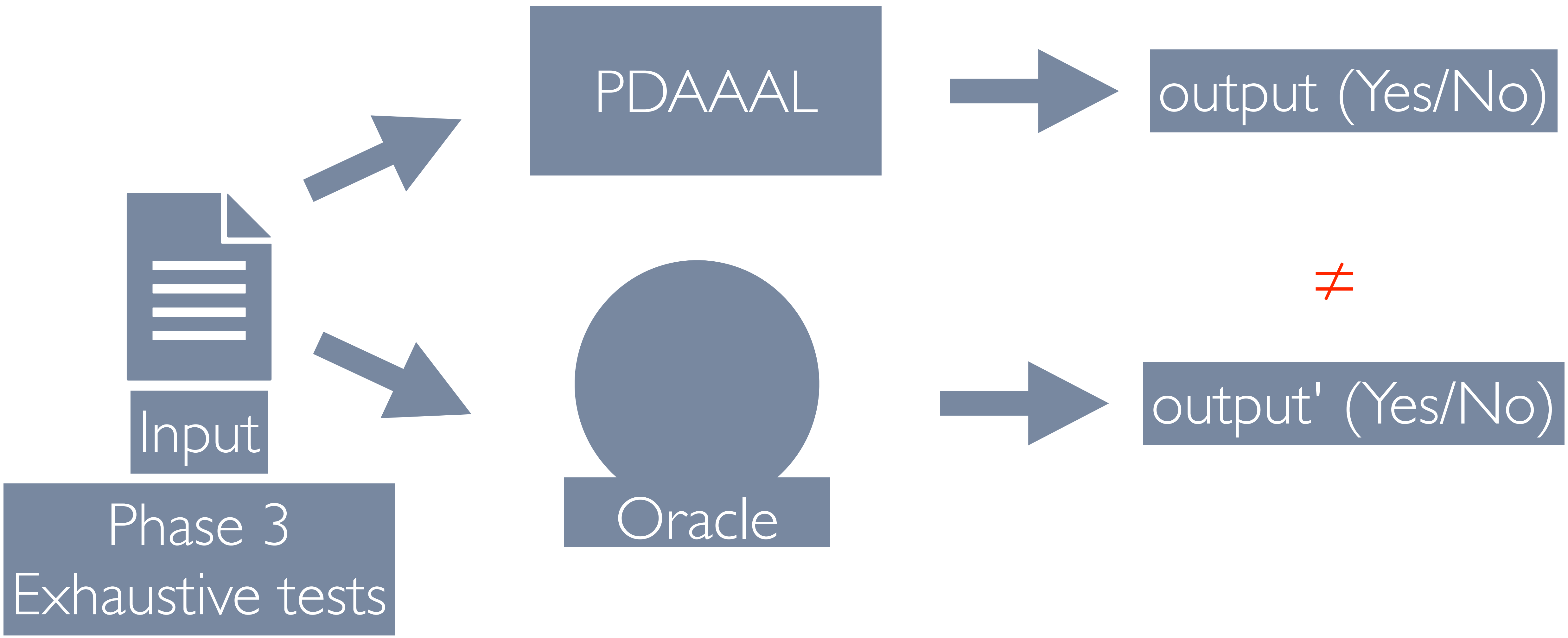


PDAAL and the oracle agree!

Differential testing



Differential testing



Thousands of test where they disagree!

Fixing another bug

We minimized the input.
We diagnosed the problem.

It involves PDAAL's parser.

Fixing another bug

PDAAAL's parser

Assumes rules can be added without knowing labels in advance.

Pushdown rule data structure

Assumes all labels are known in advance.

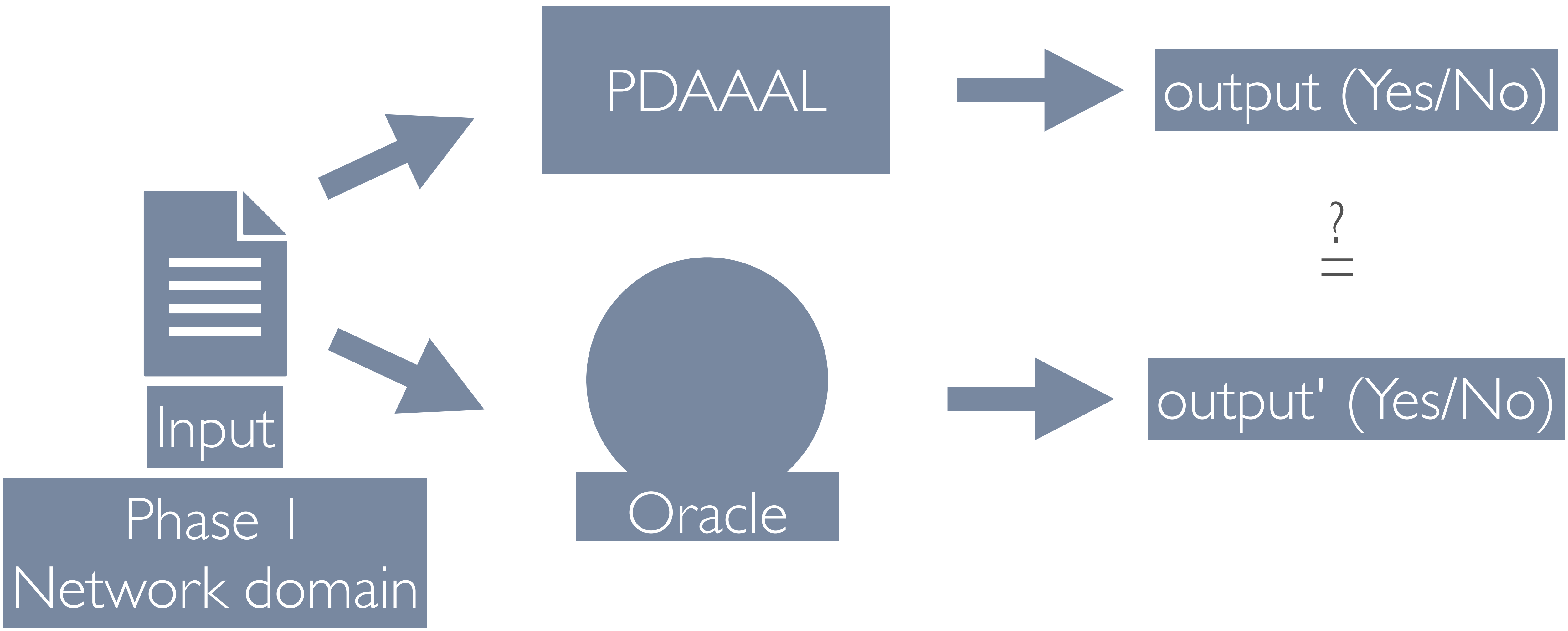
Fixing another bug

PDAAL's parser	Pushdown rule data structure
Assumes rules can be added without knowing labels in advance.	Assumes all labels are known in advance.

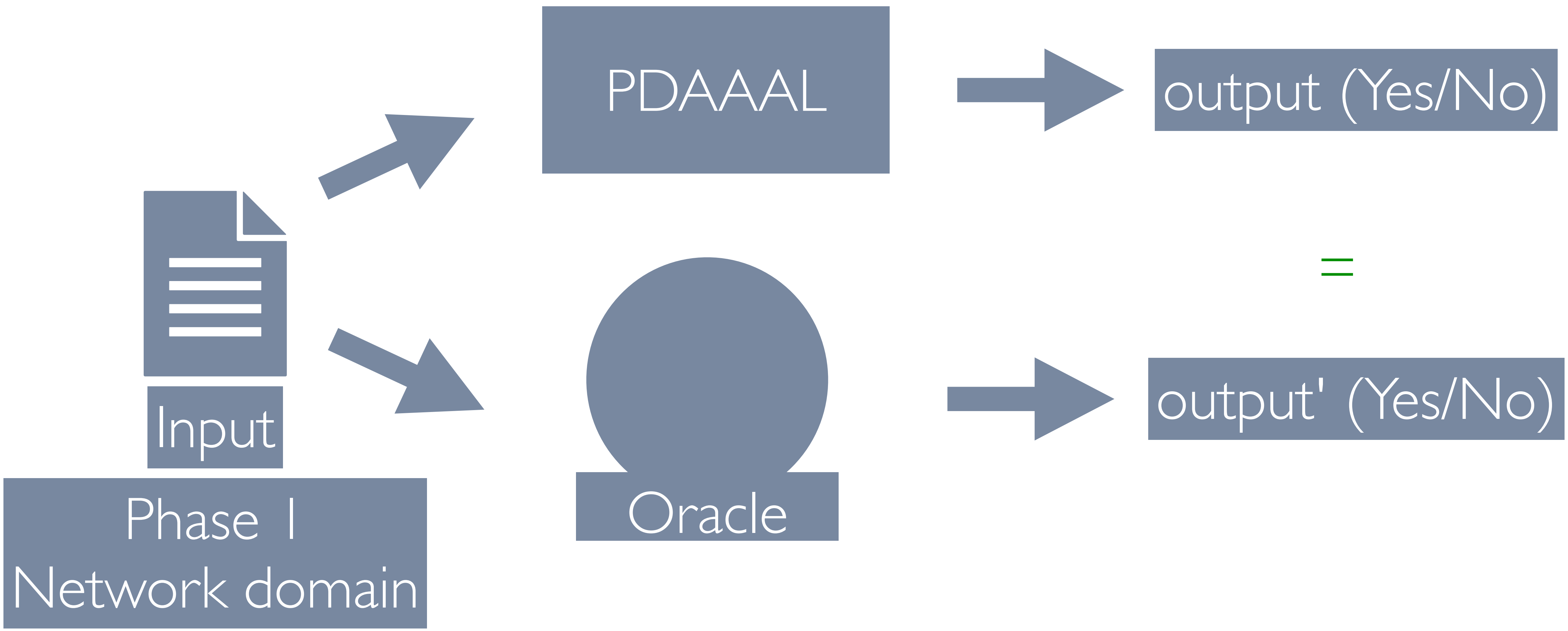
A mismatch!

We fixed the bug and ran the tests.

Differential testing after fixing the mistakes

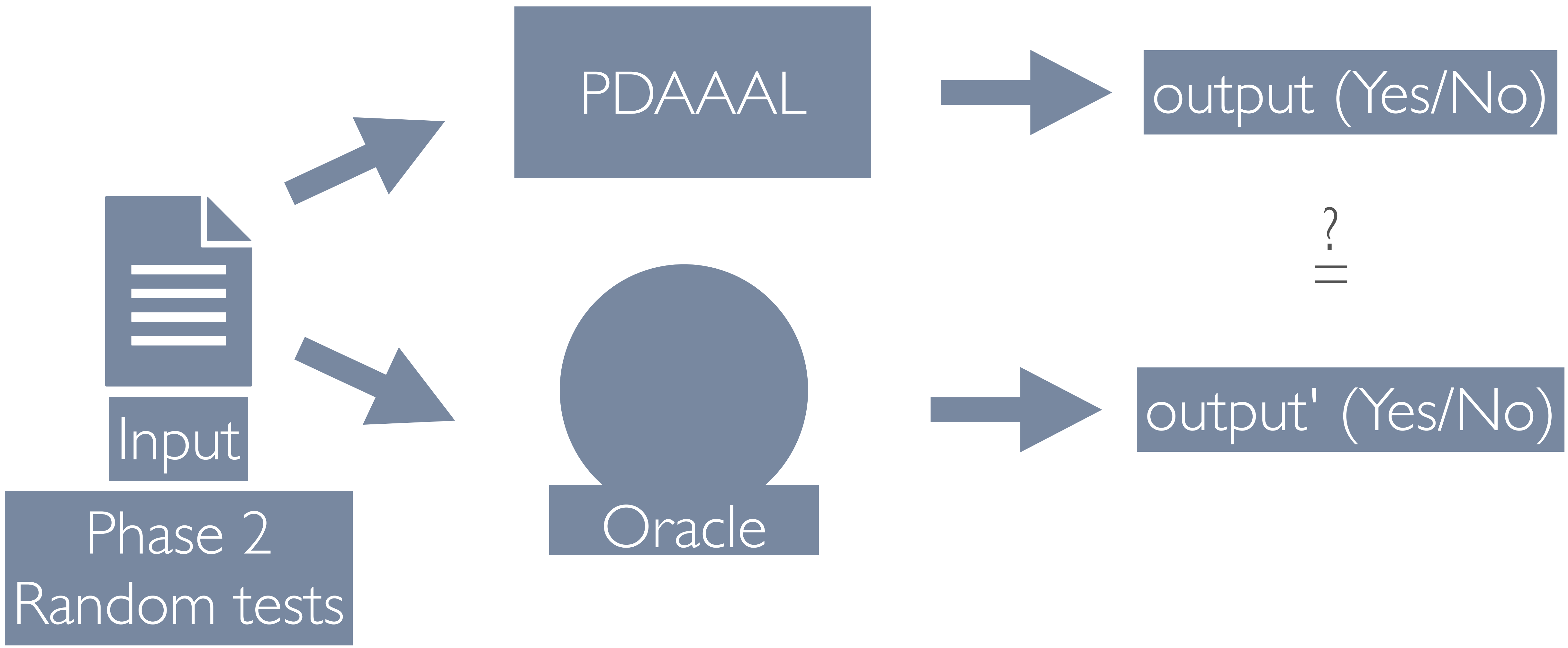


Differential testing after fixing the mistakes

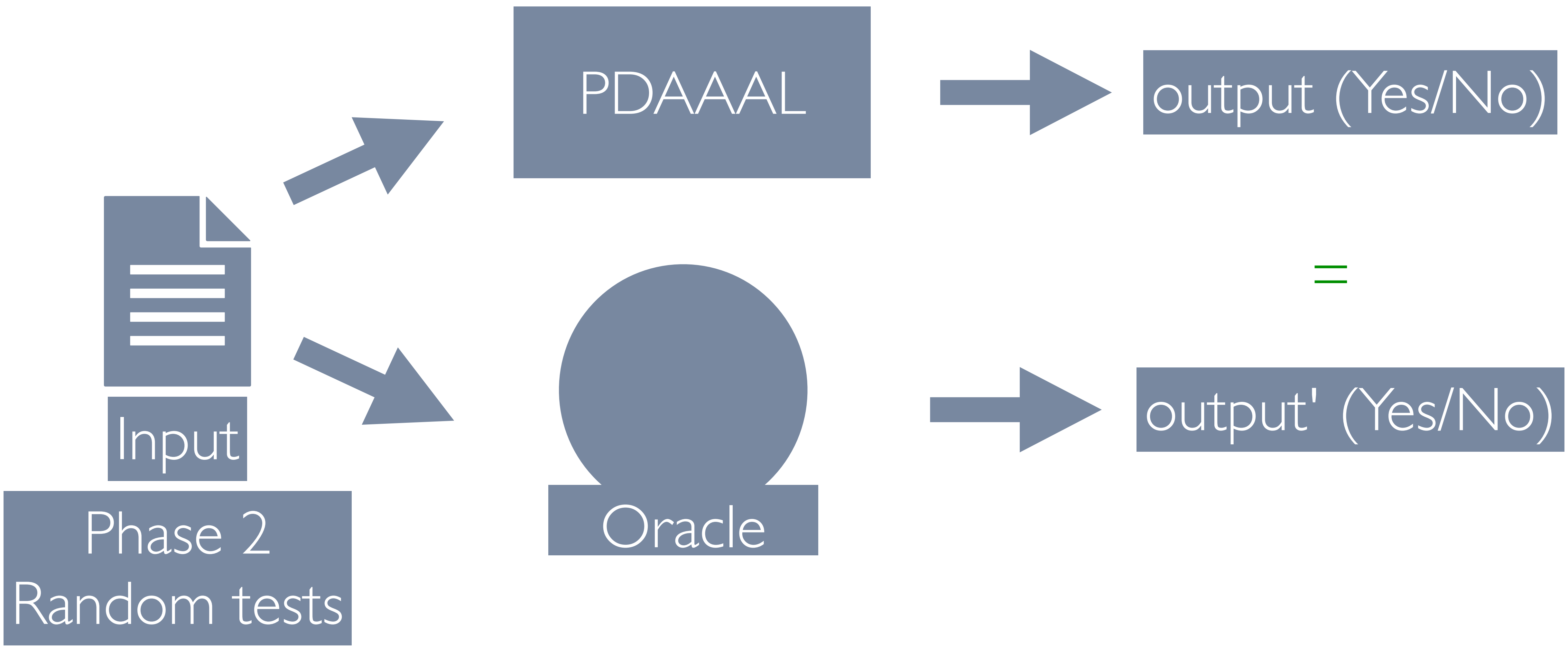


PDAAL and the oracle agree!

Differential testing after fixing the mistakes

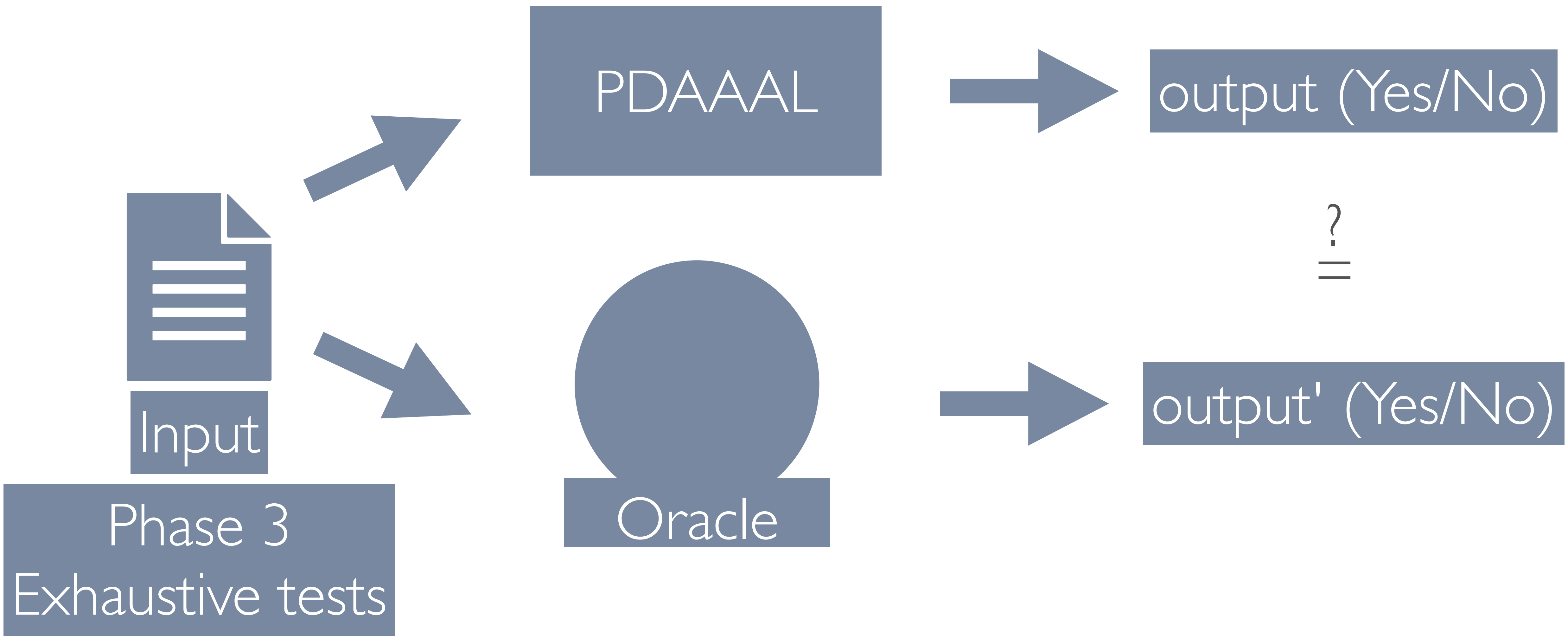


Differential testing after fixing the mistakes

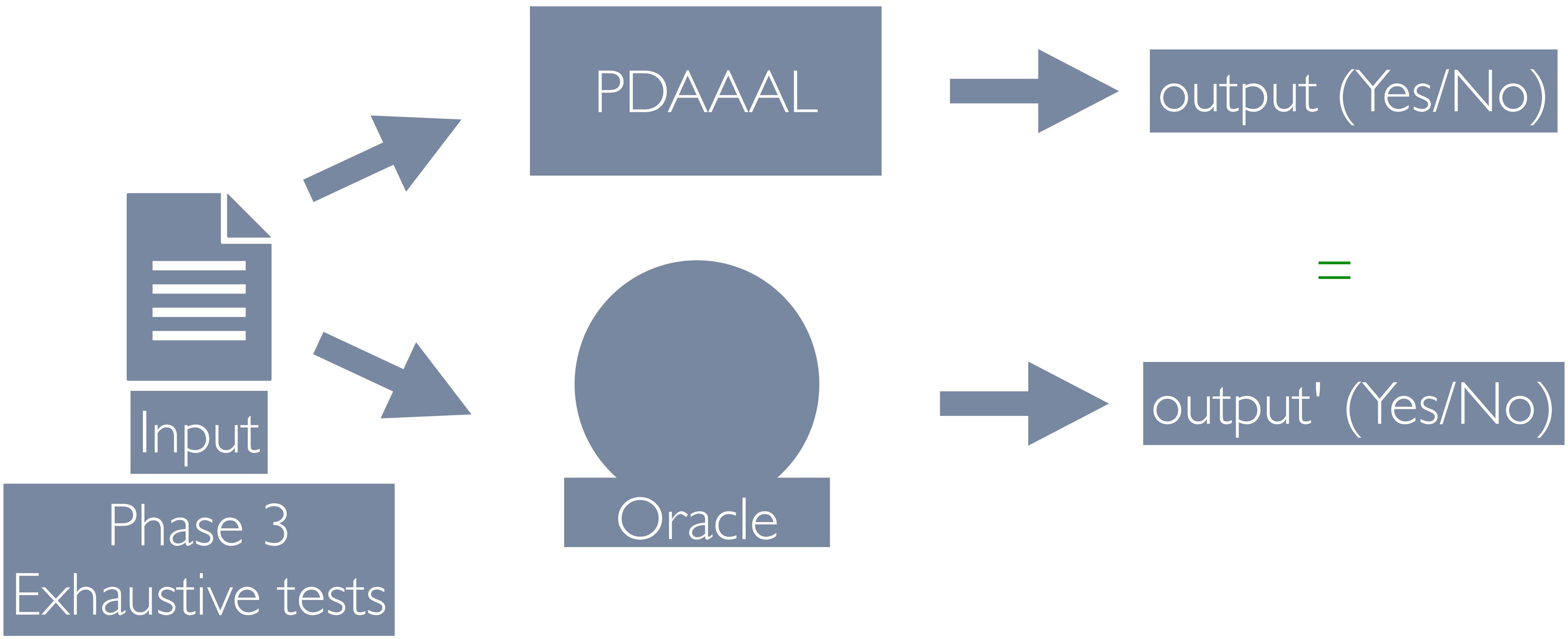


PDAAAL and the oracle agree!

Differential testing after fixing the mistakes



Differential testing after fixing the mistakes



PDAAL and the oracle agree!

Conclusion

On the "theory level":

We formalized correctness of post*, pre* and dual*.

Thus PDAAAL is based on correct algorithms.

On the "implementation level":

A fully automatic ***toolchain*** for improving tools for pushdown reachability

1. It does differential testing against our oracle.
2. It automatically minimizes counter examples using delta debugging.

The toolchain helped us find bugs in PDAAAL.

We have fixed these bugs.

Thank you 😀

Performance

For the network test cases

The average CPU time (on AMD EPYC 7642 processors at 1.5 GHz) per test case was 35 seconds for Isabelle, while PDAAAL used less than 0.02 seconds on most cases.

Execution

The execution of all tests in the three phases took 303 CPU days. We executed the tests on a compute cluster with 1 536 CPU cores.