



# Cycle Accurate Profiler for ASIPs

---

Zdeněk Přikryl and Tomáš Hruška

Brno University of Technology,  
Faculty of Information Technology,  
Czech republic



# Outline

---

- Introduction
- State of the Art
- Concept of the Profiler
- Profiling
- Experimental results
- Conclusion and Future Work



# Introduction

---

- Hardware/software co-design
  - Partitioning of the tasks into HW/SW according to criteria (e.g. performance)
- Application Specific Instruction-set Processor (ASIP)
  - Specific functions of HW are available via application specific instructions
- ASIP Design Methodology
  - Architecture Description Languages (ADLs)

# Architecture Description Languages



---

- Describe ASIP in several ways
  - Instruction set, micro-architecture, etc.
- Automatic generation of a tool-chain
  - Assembler, simulator, profiler, etc.
- Allow fluent development of the application for the designed system



# ADL ISAC

---

- ISAC stands for Instruction Set Architecture C
- Developed within the Lissom project
  - Inspired by LISA language
  - New features
- Belongs into the so-called mixed ADLs
- Two parts
  - Description of resources (e.g. registers)
  - Description of instruction set and micro-architecture



# ADL ISAC Example

---

```
REGISTER bit[32] fedeinst, fedepc;
BUS bit[32] bus { /* parameters */ }

OPERATION fe {
    // activation of the decode phase
    ACTIVATION { IF (can_de) { 1% de } };
    // behavior of the fetch event
    BEHAVIOR { fedeinst = bus[pc]; fedepc = pc; };
}
OPERATION de {
    // activate the instruction analyzer ia, instruction is
    // stored in fedeinst, value of fedepc is needed for
    // debugging
    CODINGROOT { {ia(fedeinst[fedepc])} };
}
```



# Profilers (1/2)

---

- Profiler
  - Tracks processor activities and provides the information about utilization of particular parts (e.g. instruction set coverage)
- Architecture independent profilers
  - Use specific instrumentation (i.e. new instructions are injected into a source code)
- Architecture dependent profilers
  - Have all needed details about processor architecture itself



## Profilers (2/2)

---

- Low-level profilers
  - Use assembly language
  - Usually used for the optimization of ASIP itself
- High-level profilers
  - Use higher programmable language (e.g. ANSI C)
  - Used for the optimization of a running program

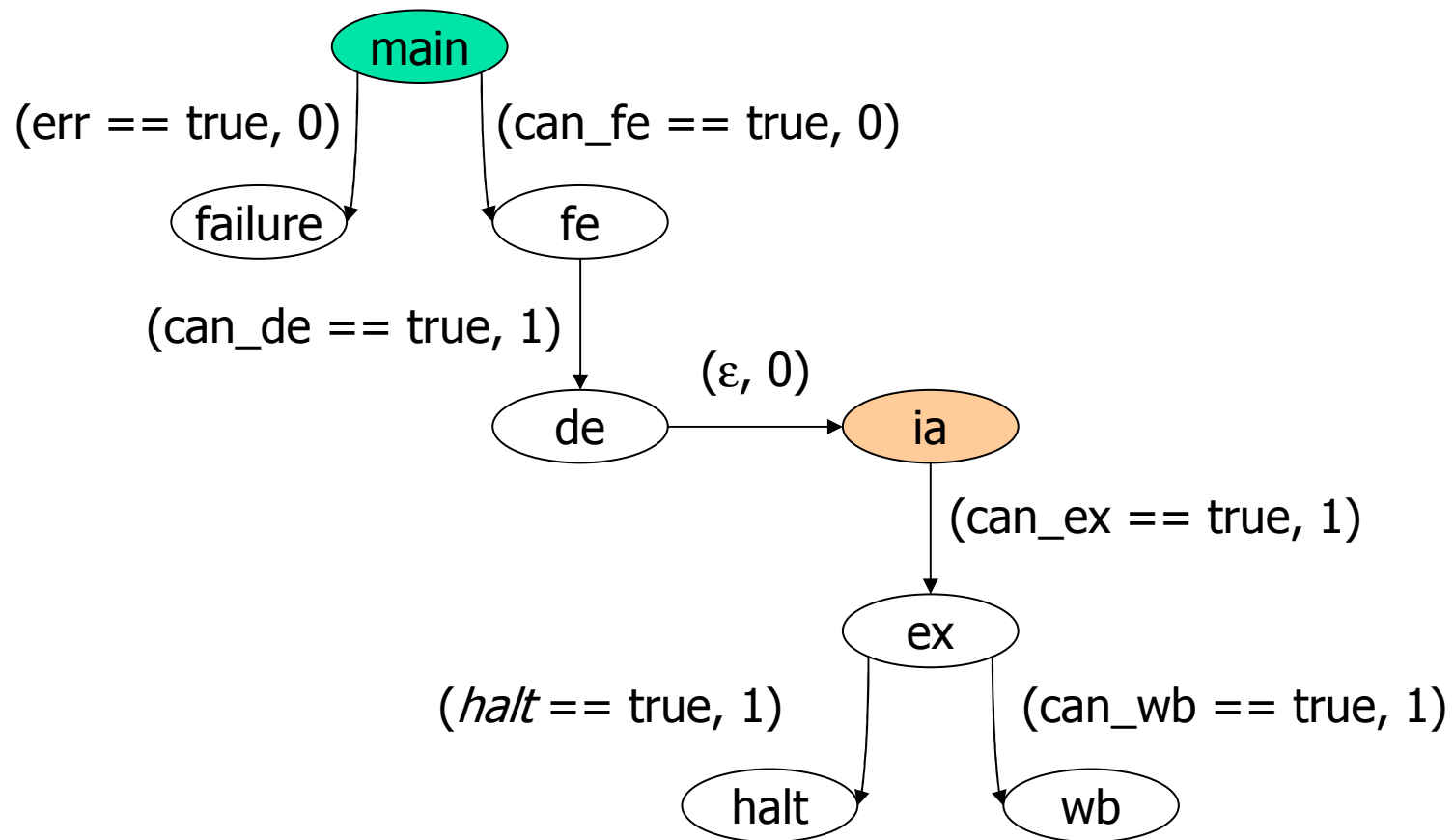


# Concept of the Profiler (1/4)

---

- Built in two phases
- From ASIP model an activation tree is created
- Activation tree is quadruple  $G = (E, H, s, z)$ 
  - $E$  is a partially ordered set of nodes (events)
  - $H$  is a set of edges  $H \subseteq E \times E$
  - $s$  is a node rating  $s: E \rightarrow 2^{H^*}$
  - $z$  is a edge rating  $z: H \rightarrow C \times D$ , where  $C$  is a set of valid ANSI C conditions including empty condition  $\varepsilon$ , and  $D \subset \mathbf{N}_0$  is a set of delays

# Concept of the Profiler (2/4)



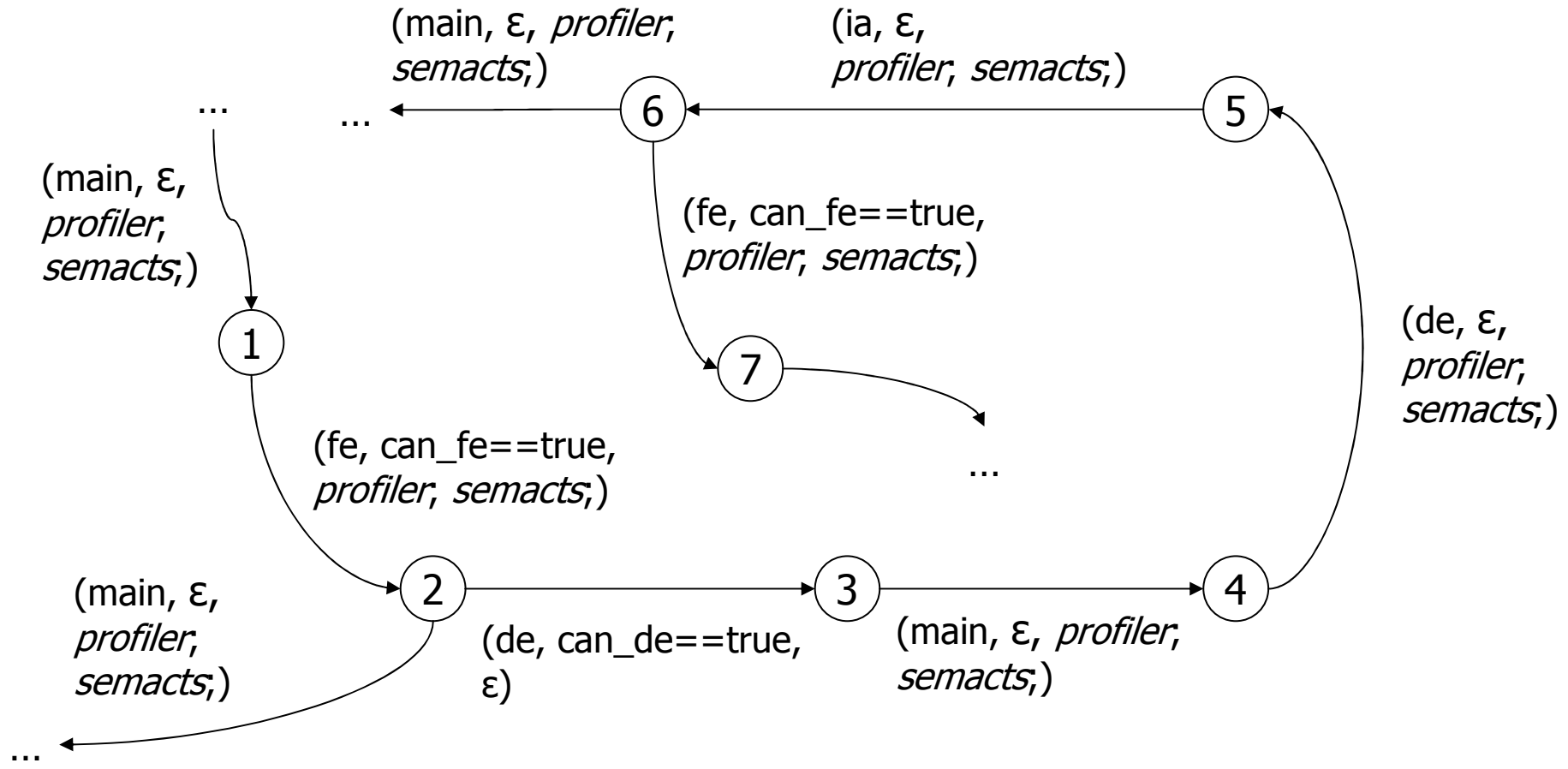


# Concept of the Profiler (3/4)

---

- From the activation tree an event automaton is created
- Event automaton is a septuple  
 $M = (Q, \Sigma, P, s, F, S, z)$ 
  - $Q, s, F$  analogical to finite automaton
  - $\Sigma$  is an input alphabet containing events
  - $P$  is a finite set of transitions in form  $c : pa \rightarrow q$ , where  $c$  is valid ANSI C condition including empty condition  $\varepsilon$ , states  $p, q \in Q$  and  $a \in \Sigma$
  - $S$  is a set of *semantic actions*
  - $z$  is a relation  $z \subseteq P \times S$

# Concept of the Profiler (4/4)





# Profiling (1/3)

---

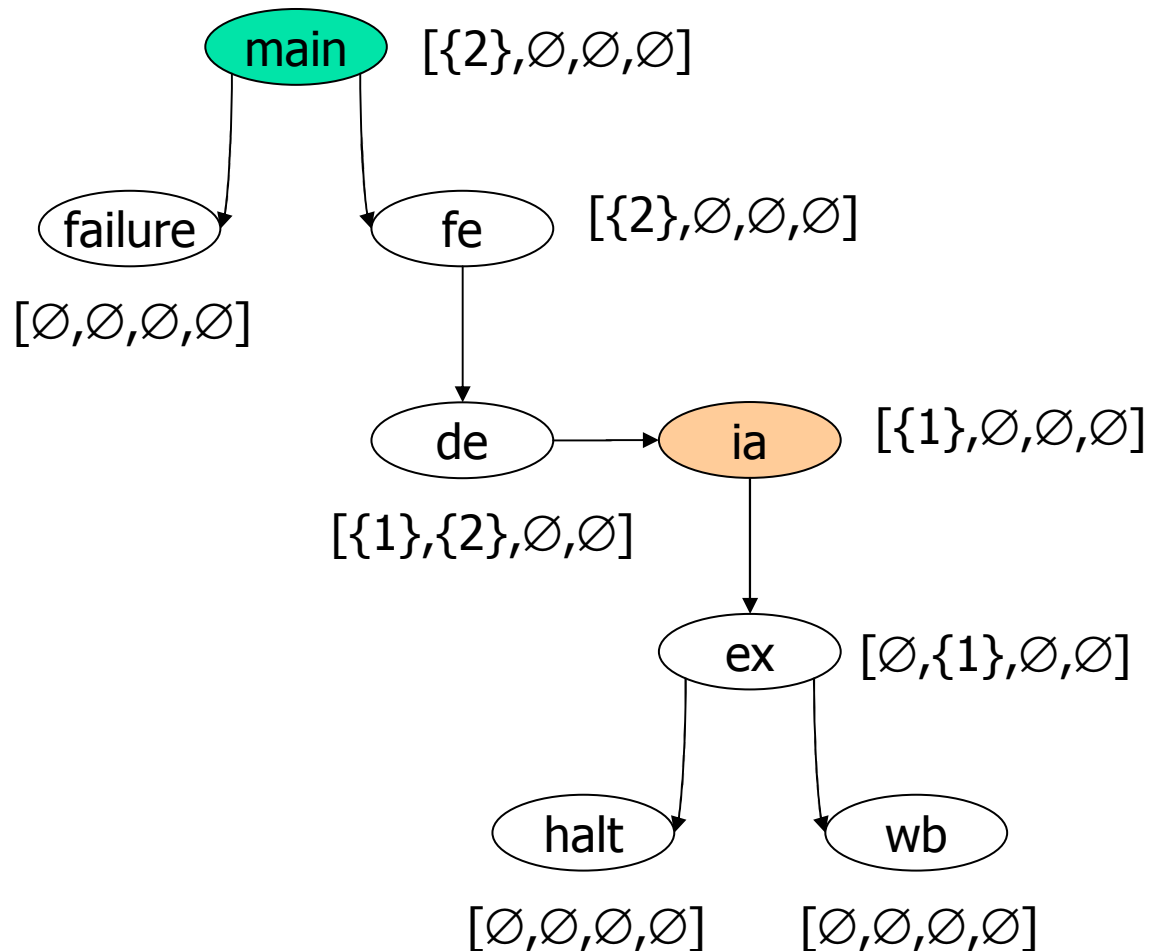
- For each executed instruction, there is an object collecting information about it (e.g. its textual form)
- Special queue is created for every activation tree node
  - In the queue, several object identifications can be stored
- Distribution of object identifications is very important
- If needed, a developer can give hints to the profiler in ASIP model

# Profiling (2/3)

- Source code

1. xor r3, r3
2. mov r1, #10
3. load r2, #0x1f
4. add r3, r1, r2
5. save r3, #0x2f
6. halt
7. nop
8. nop
9. nop

- Clock cycle: 2

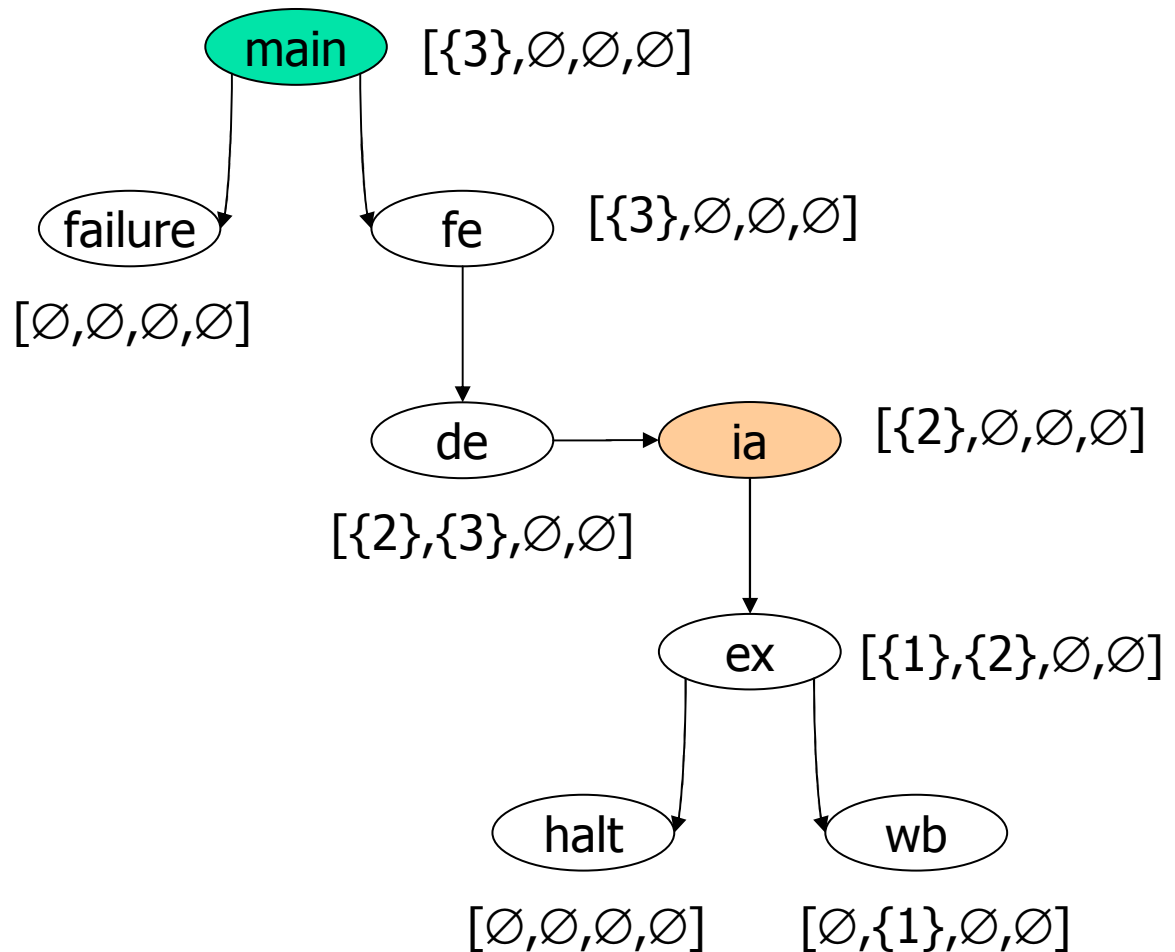


# Profiling (3/3)

## ■ Source code

1. xor r3, r3
2. mov r1, #10
3. load r2, #0x1f
4. add r3, r1, r2
5. save r3, #0x2f
6. halt
7. nop
8. nop
9. nop

## ■ Clock cycle: 3





# Experimental results

---

- Table compares speeds of our simulators and profilers
- Models are phase accurate without pipelines
- Several algorithms from MiBench testing suite were chosen

Processor	Simulator [MHz]	Profiler [MHz]	Slowdown
MIPS	30.25	1.98	15.27
Chili3	2.79	0.26	10.73
ARMv5	13.17	0.96	13.71



# Conclusion and Future Work

---

- Algorithm for profiler creation is based on the introduced formal models
- Created profiler is the low-level architecture dependent profiler
- In a lot of cases, the profiler is created without additional hints from the developer
- Profiler has several verbosity levels
  - Slowdowns are not same for different verbosity levels
- Current focus on the high-level profiler for C language



Thank You for Your Attention

---