BEST: a Binary Executable Slicing Tool
and its use to improve Model Checking-based WCET Analysis

Armel Mangean[1] Jean-Luc Béchennec[2] Mikaël Briday[3] Sébastien Faucou[3]

IRCCyN, UMR CNRS 6597
[1]École Centrale de Nantes, [2]CNRS , [3]Université de Nantes

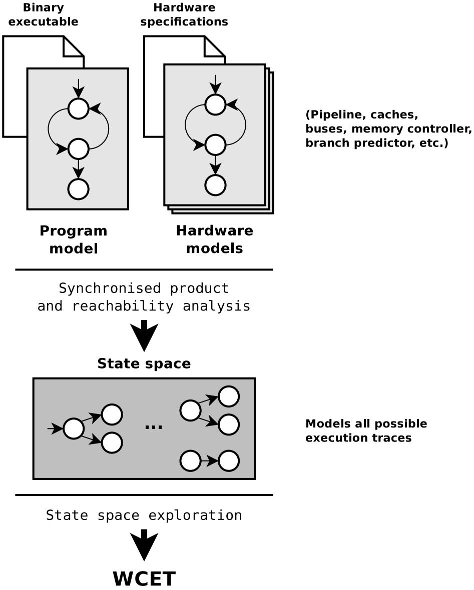
July 5, 2016

16th International Workshop on Worst-Case Execution Time Analysis

Introduction

Introduction

Motivation



Introduction

Motivation

modularity	network of timed automata
tightness	exact cache analysis
	▶ arbitrary policies (not only LRU nor PLRU)
witness	initial hardware and software configuration
binary level	no high level source code analysis
	▶ compiler independent

Introduction
Challenge

Limitations

- suffer of the state space explosion
  - tailored for embedded microcontrollers

Challenges

- abstracting models of hardware components [4]
- **abstracting models of programs** [1, 3, 6]
  - Cassez et al., 2013

Program Abstraction using Program Slicing

Program Abstraction using Program Slicing

Overview of Program Slicing

Introduced by Weiser in 1981 [7]

- given a *program* $P \subseteq L \times I$, $\forall (l, i), (l, i') \in P, i = i'$ with
  - L a finite set of labels
  - I a finite set of instructions operating over V
  - V the set of variables of P
- and a *criterion* $C = (l, v)$ with
  - $l \in L$ a label and
  - $v \subseteq V$ a subset of variables
- a *slice* S_C is a subset of P
  with the same semantics as P wrt. criterion C

The slice $S_{(l,v)}$

- is a valid program
- that computes values for the subset v
  - same as with the original program P
  - to the point of execution l
- is obtained by deleting zero or more “lines” from P

Program Abstraction using Program Slicing

Overview of Program Slicing

```
00003000 <_start>:
    3000:  li     r1,1          ;r1  <- 1
    3004:  ori    r1,r1,49296 ;ri  <- r1 | 49296
    3008:  bl     3010          ;call main
0000300c <loop>:
    300c:  b      300c          ;branch
00003010 <main>:
    3010:  li     r8,29         ;r8  <- 29
    3014:  li     r10,1         ;r10 <- 1
    3018:  mtctr r8             ;ctr <- r8
    301c:  li     r9,1          ;r9  <- 1
    3020:  b      3028          ;branch
    3024:  mr     r9,r3         ;r9  <- r3
    3028:  add    r3,r9,r10     ;r3  <- r9+r10
    302c:  mr     r10,r9        ;r10 <- r9
    3030:  bdnz   3024          ;ctr--,
                                ;branch if ctr!=0
    3034:  blr                  ;return
```

$$C = (3030, \{ctr\})$$

Program Abstraction using Program Slicing

Overview of Program Slicing

```
00003000 <_start>:
    3000:  li     r1,1          ;r1  <- 1
    3004:  ori    r1,r1,49296 ;ri  <- r1 | 49296
    3008:  bl     3010          ;call main
0000300c <loop>:
    300c:  b      300c          ;branch
00003010 <main>:
    3010:  li     r8,29         ;r8  <- 29
    3014:  li     r10,1         ;r10 <- 1
    3018:  mtctr r8             ;ctr <- r8
    301c:  li     r9,1          ;r9  <- 1
    3020:  b      3028          ;branch
    3024:  mr     r9,r3         ;r9  <- r3
    3028:  add    r3,r9,r10     ;r3  <- r9+r10
    302c:  mr     r10,r9        ;r10 <- r9
    3030:  bdnz   3024          ;ctr--,
                                ;branch if ctr!=0
    3034:  blr                  ;return
```

$$C = (3030, \{ctr\})$$

Program Abstraction using Program Slicing

Overview of Program Slicing

- dataflow equation-based or graph-based
  - fixpoint computation or
  - reachability analysis
- slicing binary executables
  - a closed issue [5] (although not trivial)
  - multiple graph computation from a CFG
  - reachability analysis on the final graph

Program Abstraction using Program Slicing

Abstracting models of programs

An instruction has

- a timing behavior due to its
  - class of instruction $\rightarrow$ number of execution cycles
  - data dependencies $\rightarrow$ pipeline stall
  - memory access $\rightarrow$ cache delay
- and a semantics
  - updates the system state

$\rightarrow$ We can abstract semantics of some instructions while keeping the timing behavior of the program

$\rightarrow$ Variables used only by abstracted instructions can be removed from the model thus reducing the overall state space

Program Abstraction using Program Slicing

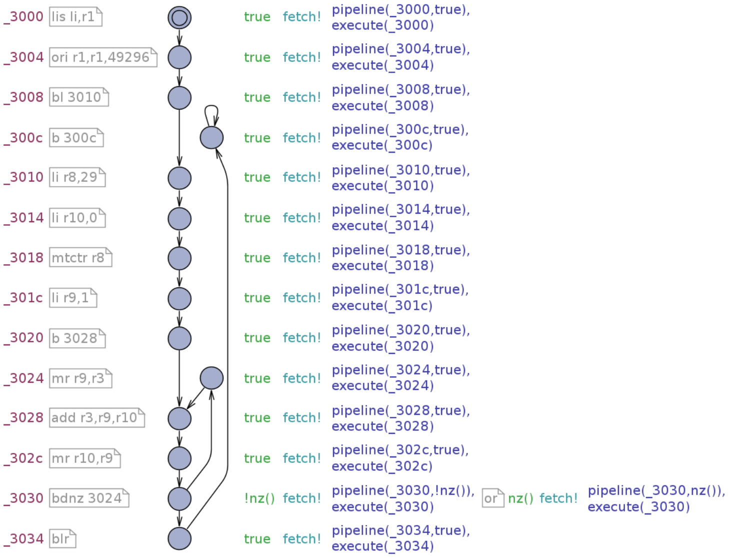
Abstracting models of programs

How to abstract a model of program?
(**but not its timing behavior**)

- abstract model must contain all paths from original model
  - i.e. contain all control instructions and their dependencies
- we can use program slicing to find these instructions
  - criteria are chosen wrt. the previous constraint as follows:
    $$\{(l, v) \mid \exists i,\ (l, i) \in P \\ \text{and } i \text{ is a conditional branching instruction} \\ \text{and } v \text{ is the subset of variables used by } i \text{ at } l\}$$

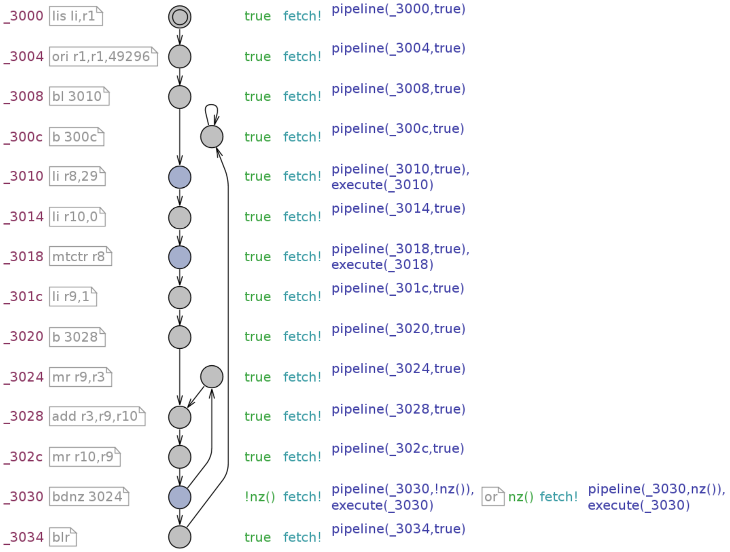
Program Abstraction using Program Slicing

Abstracting models of programs



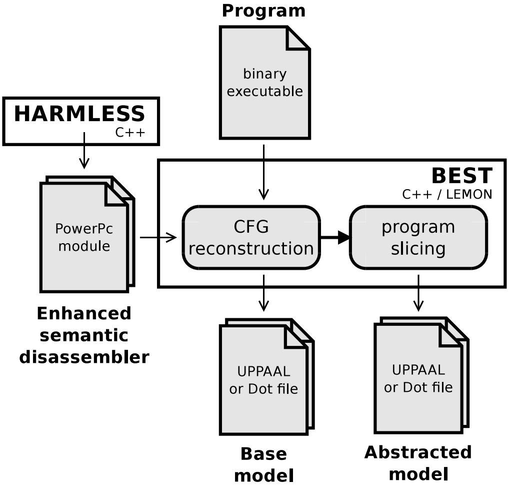
Program Abstraction using Program Slicing

Abstracting models of programs



Program Abstraction using Program Slicing

Tool implementation



Experimental results

Experimental results

Methodology

- use of Mälardalen WCET benchmarks
- excluding programs containing
  - switch-case statements
  - floating-point arithmetic
  - recursive programs
- multiple compilers and optimization options
  - GCC 5.3.1 (`-O0`, `-O1`, `-O2`, `-O3`)
  - COSMIC C 4.3.7 (`-no`, *default*)
  - targeting the PowerPC 32 bits instruction set
- sums up to 96 binaries
- use of Trampoline RTOS [2] services
  - not documented on our paper

Experimental results

Results

Source file	Gcc				Cosmic C	
	-O0	-O1	-O2	-O3	-no	*default*
`adpcm.c`	224/1858, 88%	357/966, 63%	421/1094, 62%	348/1775, 80%	398/1282, 69%	338/1064, 68%
Average	78%	63%	62%	65%	66%	63%
	67%				64%	

$\rightarrow$ number of instructions in the slice/total number of instructions, gain in percentage.
$\rightarrow$ execution time negligible (always < 1 sec.)

Experimental results

Results

Source file	GCC				COSMIC C	
	`-O0`	`-O1`	`-O2`	`-O3`	`-no`	*default*
`adpcm.c`	11/17, 35%	28/32, 13%	26/28, 7%	33/36, 8%	22/37, 41%	22/37, 41%
Average	38%	35%	36%	37%	59%	54%
	37%				57%	

$\rightarrow$ number of registers in the slice/total number of registers, gain in percentage.

$\rightarrow$ execution time negligible (always < 1 sec.)

Future work

Future work

- improve support of interprocedurality (straightforward)
- extend data dependency analysis to stack frames and initialized data
  - bigger slices but not necessarily bigger state space
- modeling the PowerPC e200z4 core
  - no data cache
  - instruction cache
    - 2 or 4-ways associative
    - pseudorandom (global FIFO)
  - branch prediction, . . .
- modeling the MPC5643L microcontroller
  - two PowerPC e200z4 cores
  - XBAR crossbar switch
    - multiple masters / multiple slaves
    - per slave policy (FP or RR)
- WCET analysis of parallel programs

Conclusion

- abstract models of program
  - for Model Checking-based WCET analysis
  - based on program slicing
- a binary executable slicing tool
  - instruction set independant
  - free sofware (GNU GPL)
  - promising experimental results

References

Florian Brandner and Alexander Jordan.
Refinement of worst-case execution time bounds by graph pruning.
Computer Languages, Systems & Structures, 2014.

Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, and Yvon Trinquet.
Trampoline An Open Source Implementation of the OSEK/VDX RTOS Specification.
In *IEEE International Conference on Emerging Technologies and Factory Automation*, 2006.

Franck Cassez and Jean-Luc Béchennec.
Timing Analysis of Binary Programs with UPPAAL.
In *International Conference on Application of Concurrency to System Design*, 2013.

Franck Cassez and Pablo González de Aledo Marugán.
Timed Automata for Modeling Caches and Pipelines.
In *Workshop on Models for Formal Analysis of Real Systems*, 2015.

Akos Kiss, Judit Jász, Gábor Lehotai, and Tibor Gyimóthy.
Interprocedural Static Slicing of Binary Executables.
In *International Workshop on Source Code Analysis and Manipulation*, 2003.

Armel Mangean, Jean-Luc Béchennec, Mikaël Briday, and Sébastien Faucou.
BEST: a Binary Executable Slicing Tool.
In *16th International Workshop on Worst-Case Execution Time Analysis*, 2016.

Mark Weiser.
Program Slicing.
In *International Conference on Software Engineering*, 1981.

Expermimental results

Detailed results

Source file	Gcc				Cosmic C	
	-O0	-O1	-O2	-O3	-no	*default*
adpcm.c	1858/224, 88%	966/357, 63%	1094/421, 62%	1775/348, 80%	1282/398, 69%	1064/338, 68%
bs.c	82/27, 67%	38/19, 50%	28/18, 36%	28/18, 36%	54/28, 48%	35/18, 49%
bsort100.c	141/39, 72%	65/24, 63%	58/18, 69%	58/18, 69%	74/34, 54%	66/34, 48%
cnt.c	193/44, 77%	104/38, 63%	87/24, 72%	1124/81, 93%	128/25, 80%	112/23, 79%
compress.c	725/271, 63%	529/214, 60%	530/247, 53%	752/316, 58%	591/253, 57%	501/228, 54%
crc.c	295/44, 85%	162/50, 69%	141/47, 67%	210/98, 53%	186/112, 40%	148/81, 45%
expint.c	187/33, 82%	135/50, 63%	27/5, 81%	27/5, 81%	115/48, 58%	93/40, 57%
fdct.c	662/11, 98%	185/6, 97%	205/6, 97%	692/3, 99%	317/6, 98%	218/6, 97%
fibcall.c	58/12, 79%	32/10, 69%	14/3, 79%	14/3, 79%	29/8, 72%	21/8, 62%
fir.c	137/21, 85%	79/34, 57%	79/33, 58%	79/33, 58%	87/35, 60%	74/30, 59%
janne_complex.c	75/21, 72%	39/21, 46%	40/29, 28%	36/26, 28%	41/20, 51%	30/20, 33%
jfdctint.c	505/29, 94%	195/9, 95%	219/9, 96%	795/6, 99%	255/9, 96%	205/9, 96%
matmult.c	196/36, 82%	120/42, 65%	110/43, 61%	103/38, 63%	135/20, 85%	118/20, 83%
ndes.c	936/162, 83%	474/160, 66%	563/236, 58%	886/522, 41%	653/118, 82%	576/112, 81%
ns.c	115/35, 66%	65/30, 54%	46/27, 41%	181/88, 51%	69/33, 52%	54/29, 46%
prime.c	146/63, 57%	56/38, 32%	48/30, 38%	31/14, 55%	95/46, 52%	82/42, 49%
Average	78%	63%	62%	65%	66%	63%
	67%				64%	

→ number of instructions in the slice / total number of instructions, gain in percentage.

Expermimental results

Detailed results

Source file	Gcc				Cosmic C	
	-O0	-O1	-O2	-O3	-no	*default*
adpcm.c	11/17, 35%	28/32, 13%	26/28, 7%	33/36, 8%	22/37, 41%	22/37, 41%
bs.c	7/11, 36%	10/13, 23%	9/10, 10%	9/10, 10%	10/14, 29%	11/13, 15%
bsort100.c	9/12, 25%	13/18, 28%	11/16, 31%	11/16, 31%	13/15, 13%	13/15, 13%
cnt.c	10/15, 33%	13/18, 28%	10/16, 38%	10/18, 44%	10/37, 73%	10/37, 73%
compress.c	15/19, 21%	26/31, 16%	30/33, 9%	32/35, 9%	21/37, 43%	21/37, 43%
crc.c	8/17, 53%	14/23, 39%	10/19, 47%	9/19, 53%	18/37, 51%	18/37, 51%
expint.c	8/13, 38%	16/26, 38%	4/11, 64%	4/11, 63%	14/37, 62%	14/37, 62%
fdct.c	6/13, 54%	4/21, 81%	4/30, 87%	3/33, 91%	3/35, 91%	3/35, 91%
fibcall.c	7/11, 36%	7/12, 42%	3/7, 57%	3/7, 57%	6/12, 50%	6/10, 40%
fir.c	7/16, 56%	13/22, 41%	14/21, 33%	14/21, 33%	15/37, 59%	15/37, 59%
janne_complex.c	7/12, 42%	6/9, 33%	6/8, 25%	7/9, 22%	7/36, 81%	7/8, 13%
jfdctint.c	8/11, 27%	3/15, 80%	4/25, 84%	4/33, 88%	3/35, 91%	3/34, 91%
matmult.c	10/19, 47%	15/20, 25%	15/19, 21%	13/19, 32%	8/37, 78%	8/37, 78%
ndes.c	9/17, 47%	21/27, 22%	23/26, 12%	27/28, 4%	16/37, 57%	15/37, 59%
ns.c	9/14, 36%	13/17, 24%	13/15, 13%	9/12, 25%	14/37, 62%	14/36, 61%
prime.c	10/13, 23%	6/9, 33%	6/9, 33%	6/8, 25%	11/36, 69%	12/36, 67%
Average	38%	35%	36%	37%	59%	54%
	37%				57%	

→ number of memory locations in the slice / total number of memory locations, gain in percentage.