Static Analysis of Least Recently Used Caches: Complexity, Optimal Analysis, and Applications to Worst-Case Execution Time and Security

Valentin Touzeau

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France

October 8, 2019

PhD supervisors: David Monniaux, Grenoble Alpes University, France
Claire Maïza, Grenoble Alpes University, France
Reviewers: Björn Lisper, Mälardalen University, Sweden
Kenneth McMillan, Microsoft Research, Redmond, USA
Examiners: Frédéric Pétrot, Grenoble Alpes University, France
Jan Reineke, Saarland University, Saarbrücken, Germany
Hugues Cassé, Paul Sabatier University, Toulouse, France
Sébastien Faucou, Nantes University, France

Cache memory

Core

Cache memory

Core

Main Memory

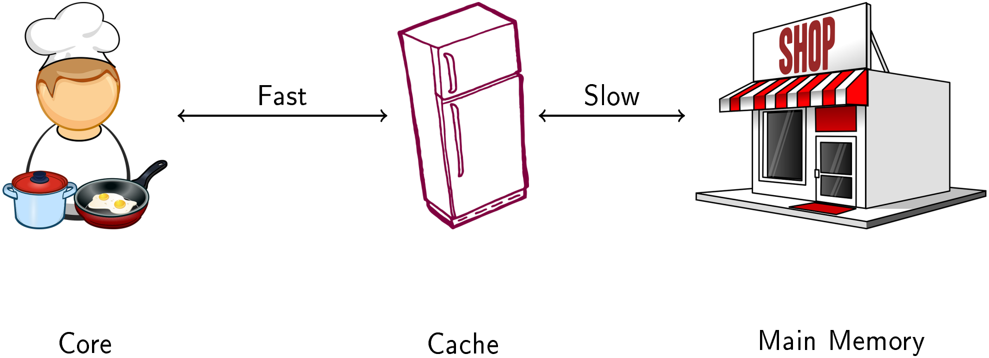
Cache memory

Slow

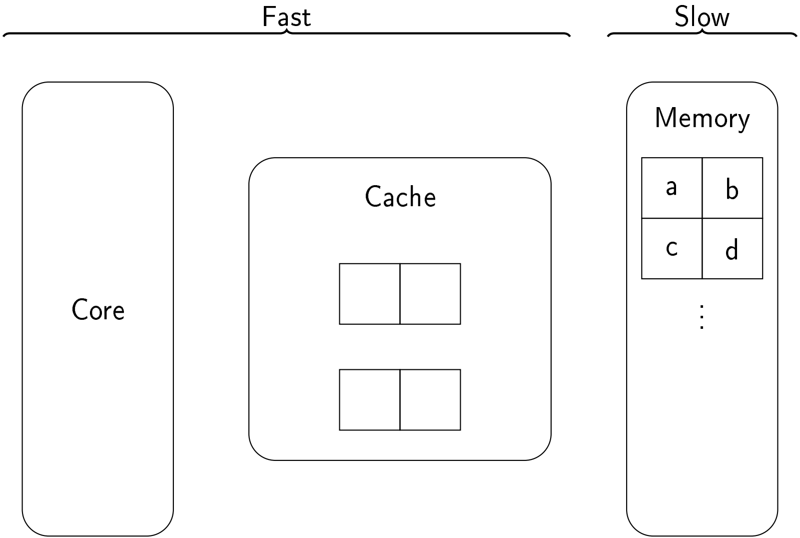
Core

Main Memory

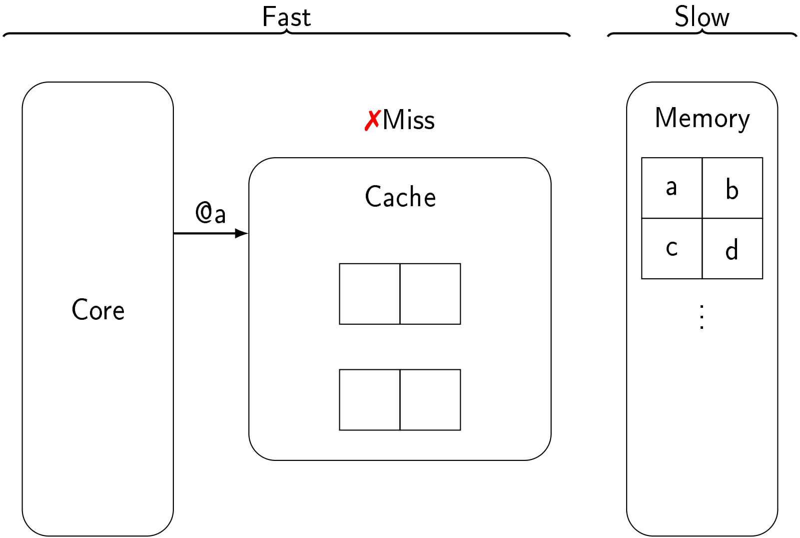
Cache memory



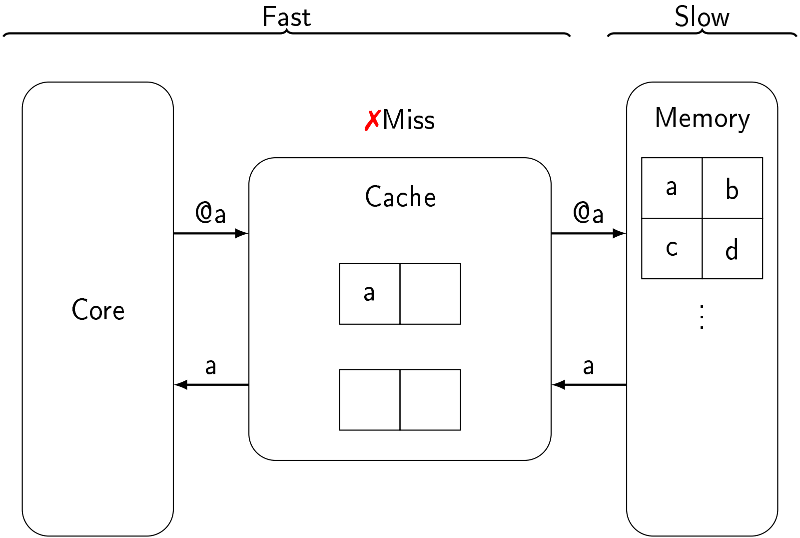
Cache memory



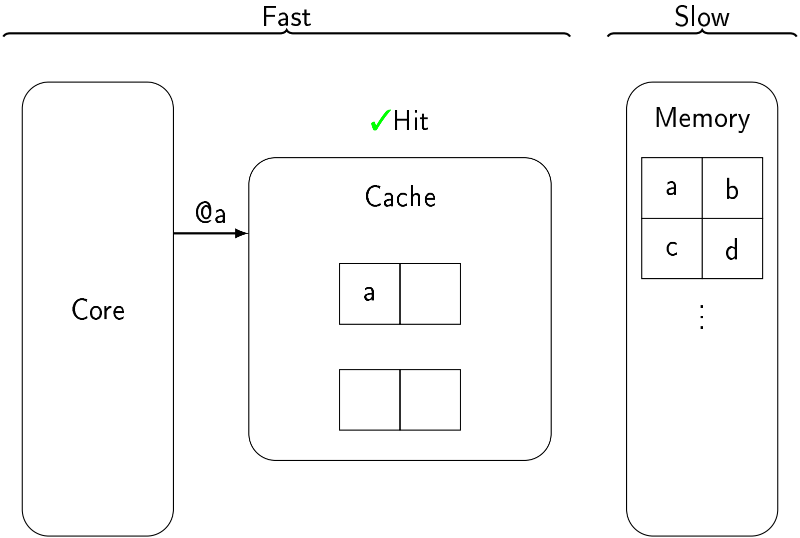
Cache memory



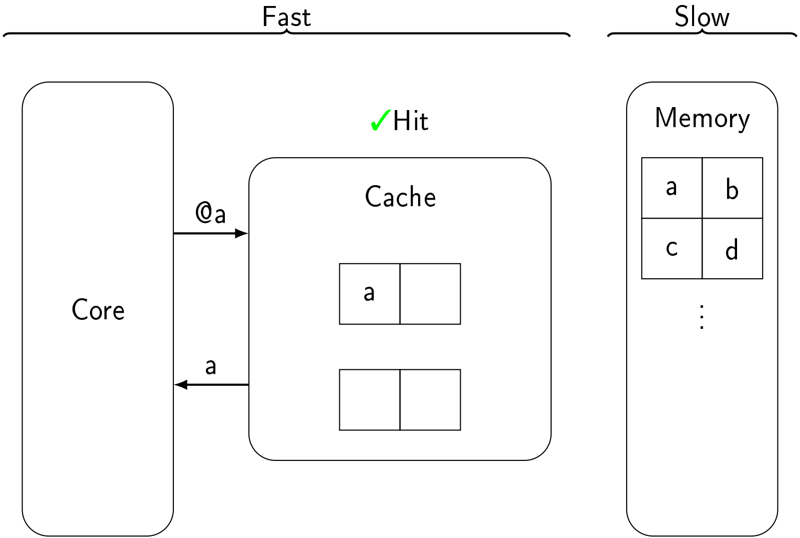
Cache memory



Cache memory



Cache memory



Motivation

Real-time systems

Cache memory induce variation on the execution time.
Worst Case Execution Time estimation ⇒ cache analysis

Security

Cache may leak secret data[1]
Bounding leakage[2] ⇒ cache analysis

[1] J. Kelsey, B. Schneier, D. A. Wagner, C. Hall, Journal of Computer Security 2000
[2] G. Doychev, B. Köpf, L. Mauborgne, J. Reineke, ACM Trans. Inf. Syst. Secur. 2015

Our Goal

Predicting memory accesses outcome (hit/miss).

Our Goal

Predicting memory accesses outcome (hit/miss).

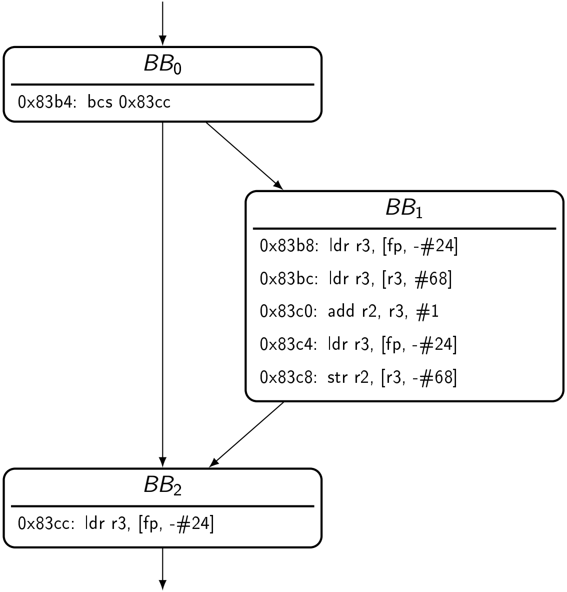
Undecidable in general

Context

```
0x83b4: bcs 0x83cc
0x83b8: ldr r3, [fp,-#24]
0x83bc: ldr r3, [r3,#68]
0x83c0: add r2, r3,#1
0x83c4: ldr r3, [fp,-#24]
0x83c8: str r2, [r3,#68]
0x83cc: ldr r3, [fp,-#24]
```

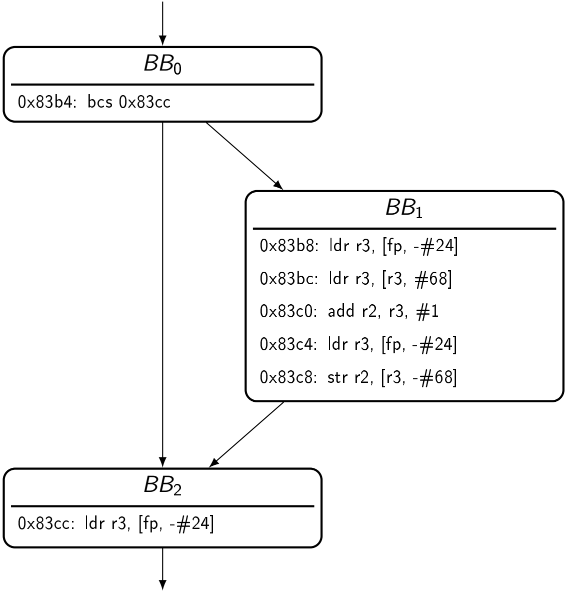
Context

```
0x83b4:  bcs  0x83cc
0x83b8:  ldr  r3,  [fp,-#24]
0x83bc:  ldr  r3,  [r3,#68]
0x83c0:  add  r2,  r3,#1
0x83c4:  ldr  r3,  [fp,-#24]
0x83c8:  str  r2,  [r3,#68]
0x83cc:  ldr  r3,  [fp,-#24]
```



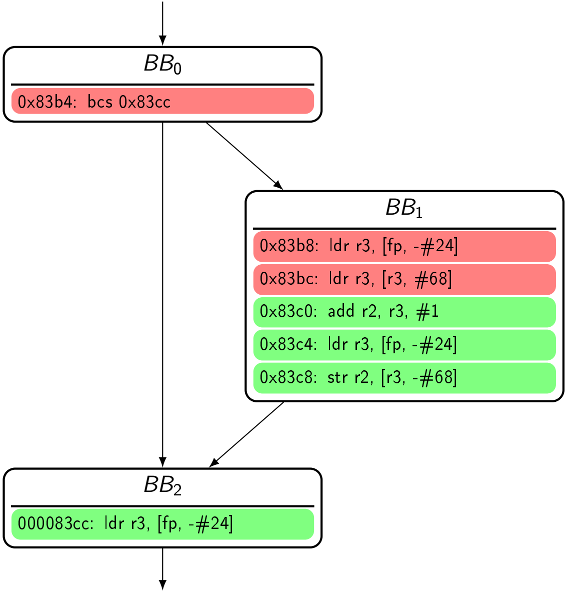
Context

```
0x83b4:  bcs  0x83cc
0x83b8:  ldr  r3,  [fp,-#24]
0x83bc:  ldr  r3,  [r3,#68]

0x83c0:  add  r2,  r3,  #1
0x83c4:  ldr  r3,  [fp,-#24]
0x83c8:  str  r2,  [r3,#68]
0x83cc:  ldr  r3,  [fp,-#24]
```

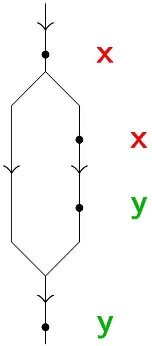


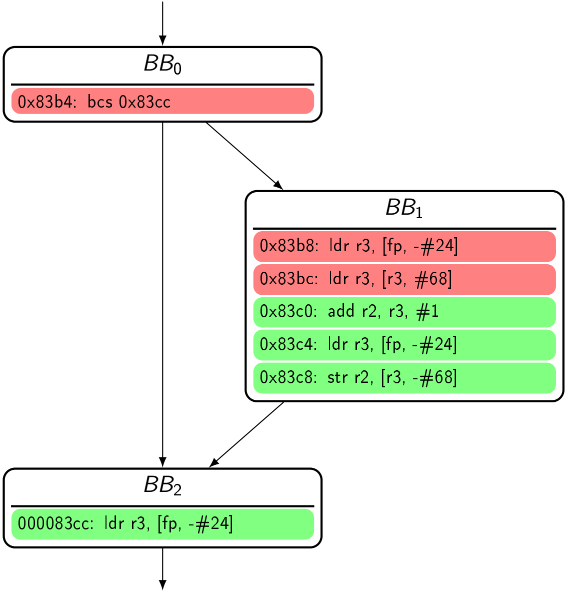
Context

```
0x83b4:  bcs  0x83cc
0x83b8:  ldr  r3,  [fp,-#24]
0x83bc:  ldr  r3,  [r3,#68]

0x83c0:  add  r2,  r3,  #1
0x83c4:  ldr  r3,  [fp,-#24]
0x83c8:  str  r2,  [r3,#68]
0x83cc:  ldr  r3,  [fp,-#24]
```

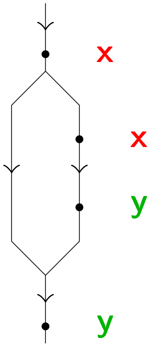


Context


x
x
y
y

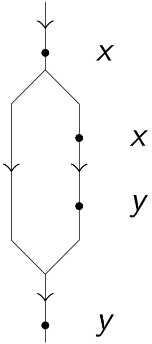

BB0
0x83b4: bcs 0x83cc
BB1
0x83b8: ldr r3, [fp, -#24]
0x83bc: ldr r3, [r3, #68]
0x83c0: add r2, r3, #1
0x83c4: ldr r3, [fp, -#24]
0x83c8: str r2, [r3, -#68]
BB2
000083cc: ldr r3, [fp, -#24]

Context

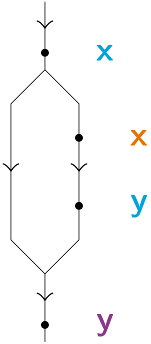


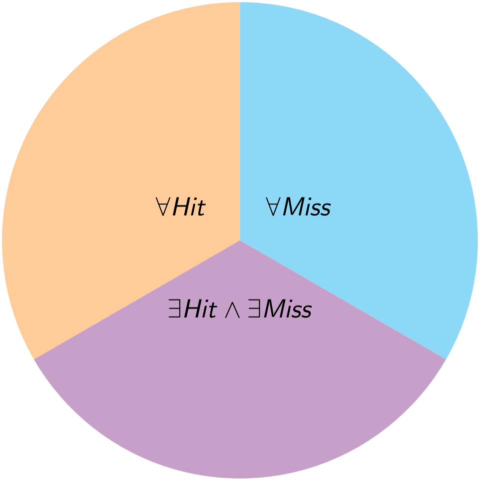
- All paths feasible
- Instruction cache only

Classification of memory accesses

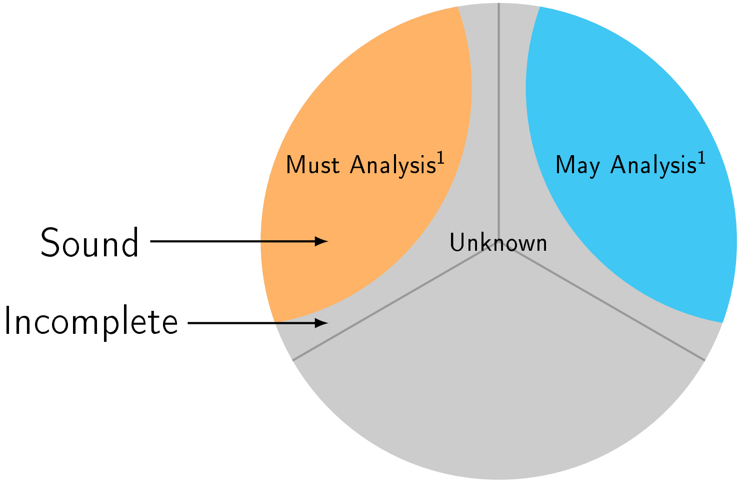


Classification of memory accesses



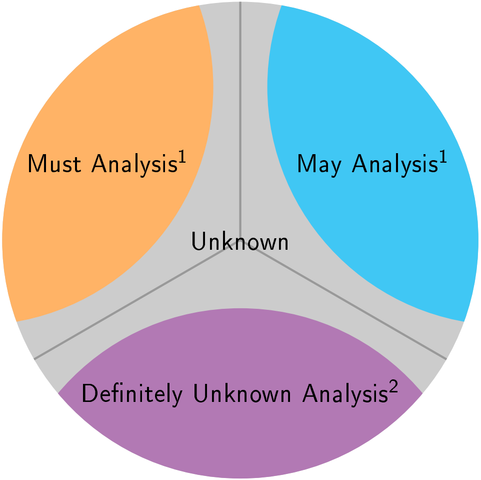


Our approach



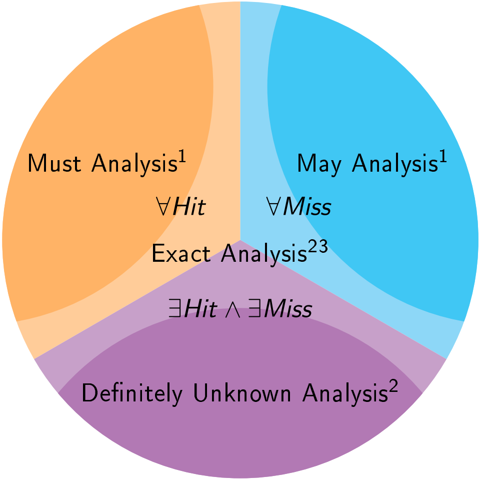
[1] M. Alt, C. Ferdinand, F. Martin, R. Wilhelm, SAS96

Our approach



[1] M. Alt, C. Ferdinand, F. Martin, R. Wilhelm, SAS96
[2] V. Touzeau, C. Maïza, D. Monniaux, J. Reineke, CAV17

Our approach



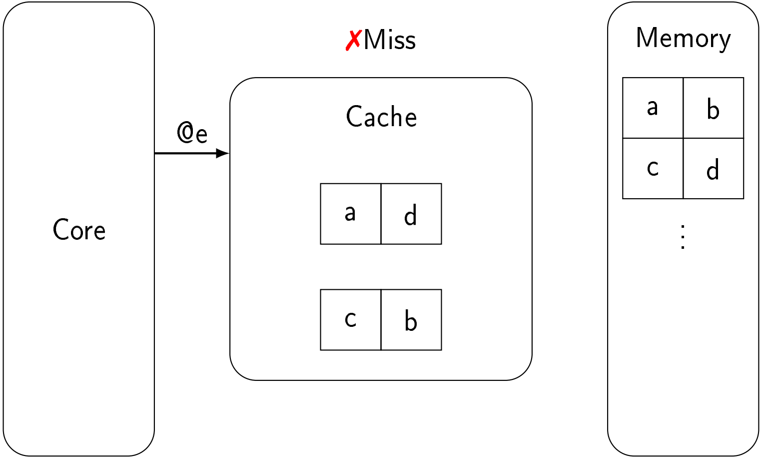
[1] M. Alt, C. Ferdinand, F. Martin, R. Wilhelm, SAS96
[2] V. Touzeau, C. Maïza, D. Monniaux, J. Reineke, CAV17
[3] V. Touzeau, C. Maïza, D. Monniaux, J. Reineke, POPL19

Overview

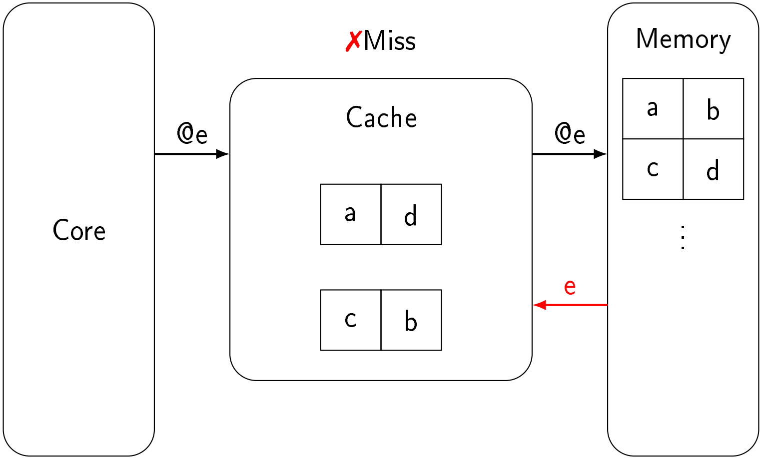
Assumptions and restrictions

- We analyze simple control-flow (no dynamic jump, no function pointers, etc.)
- We assume that all paths are feasible
- We assume simple processor (in-order, no speculation)
- We focus on instruction cache
- We analyze Least Recently Used (LRU) caches

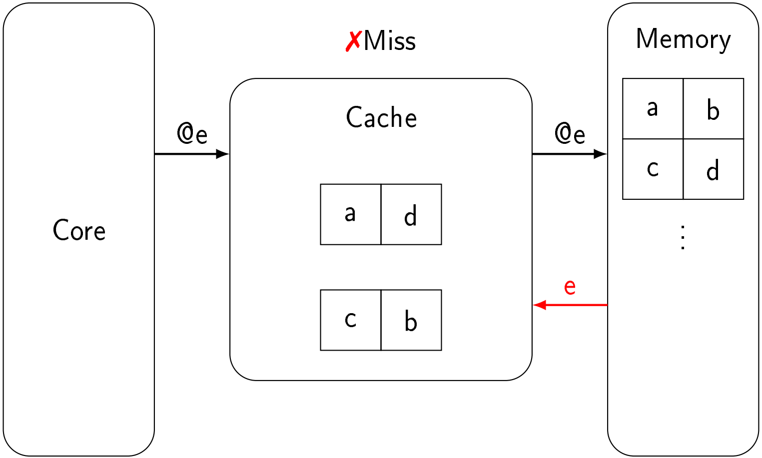
Replacement policy



Replacement policy



Replacement policy

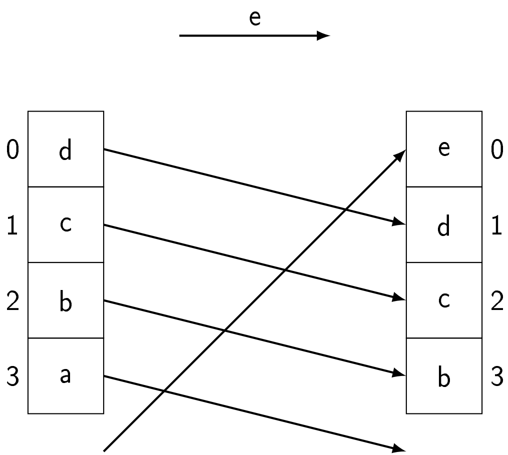


Examples of replacement policy

LRU, PLRU, FIFO, MRU, Pseudo-RR

Least Recently Used Policy

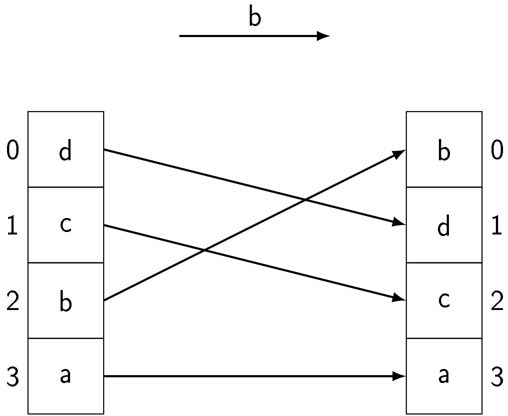
Miss



Block age: logical position of the block

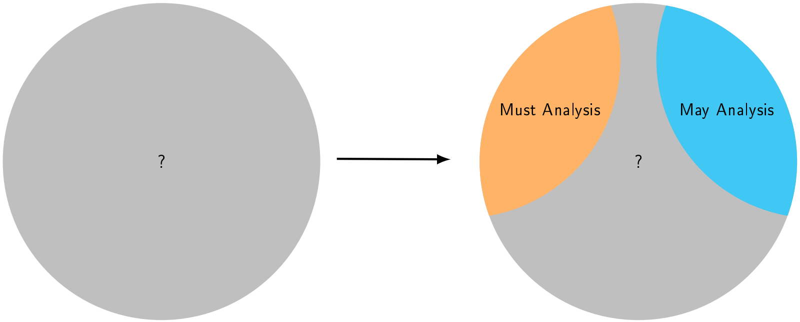
Least Recently Used Policy

Hit



Block age: logical position of the block

May/Must Analysis[1]



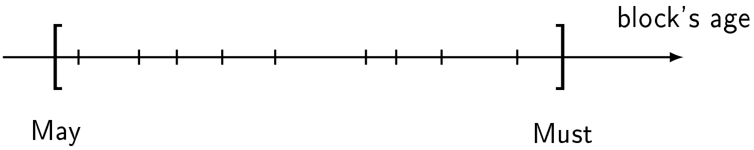
[1] M. Alt, C. Ferdinand, F. Martin, R. Wilhelm, SAS96

May/Must Analysis

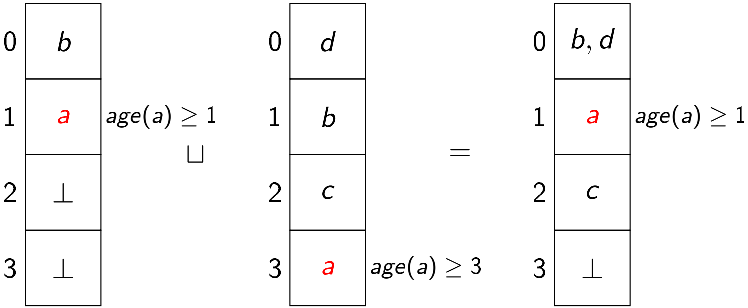
Abstraction of cache states

May/Must overapproximates block ages:

- May: safe lower bound
- Must: safe upper bound

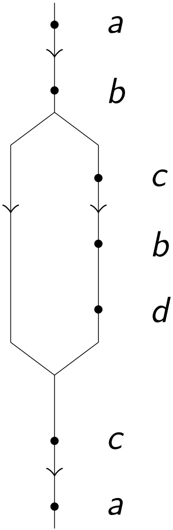


Example: May join



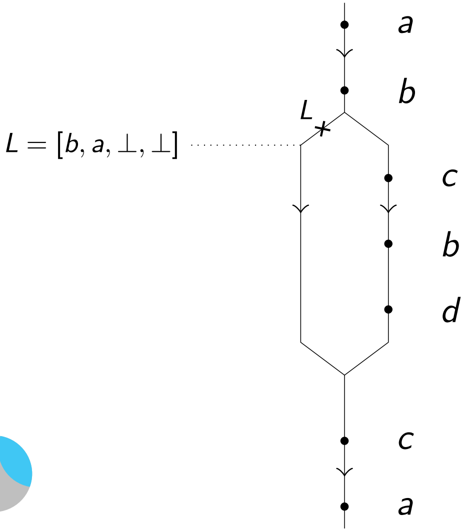
May/Must Analysis

Control Flow Graph



May/Must Analysis

May Analysis

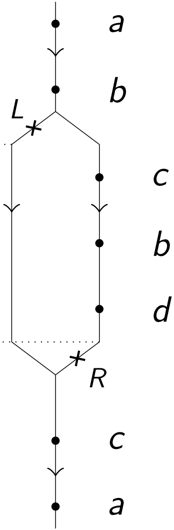


May/Must Analysis

May Analysis

$L = [b, a, \perp, \perp]$

$R = [d, b, c, a]$

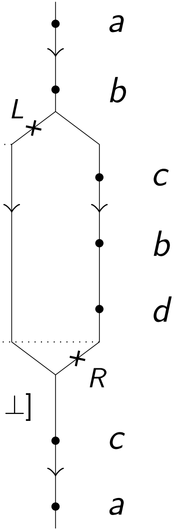


May Analysis

$L = [b, a, \perp, \perp]$

$R = [d, b, c, a]$

$L \sqcup_{May} R = [\{b, d\}, a, c, \perp]$



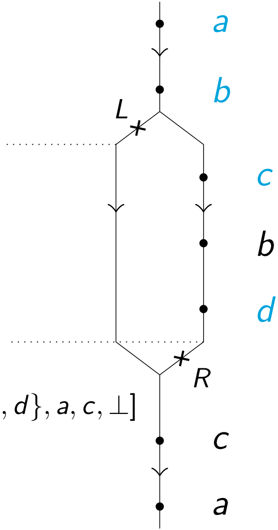
May/Must Analysis

May Analysis

$L = [b, a, \bot, \bot]$

$R = [d, b, c, a]$

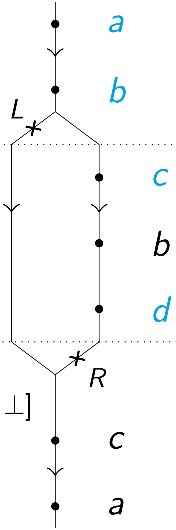
$L \sqcup_{May} R = [\{b, d\}, a, c, \bot]$



May/Must Analysis

May Analysis

Must Analysis



$L = [b, a, \bot, \bot]$

$L = [b, a, \bot, \bot]$

$R = [d, b, c, a]$

$R = [d, b, c, a]$

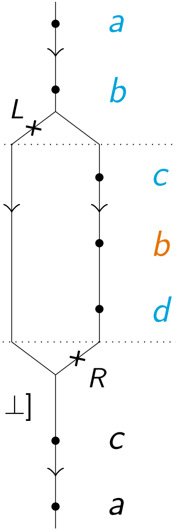
$L \sqcup_{May} R = [\{b, d\}, a, c, \bot]$

$L \sqcup_{Must} R = [\bot, b, \bot, a]$

May/Must Analysis

May Analysis

Must Analysis

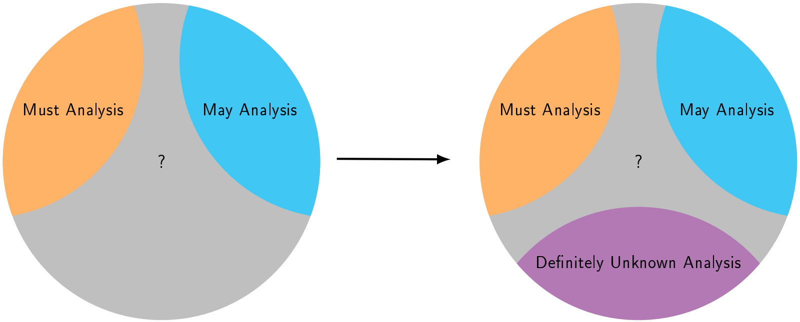


$L = [b, a, \bot, \bot]$ $\quad$ $L = [b, a, \bot, \bot]$

$R = [d, b, c, a]$ $\quad$ $R = [d, b, c, a]$

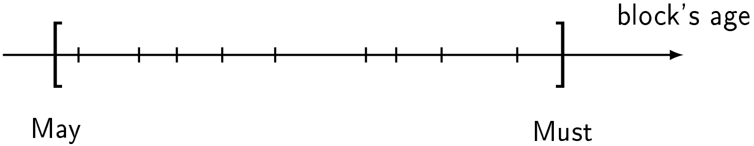
$L \sqcup_{May} R = [\{b, d\}, a, c, \bot]$ $\quad$ $L \sqcup_{Must} R = [\bot, b, \bot, a]$

Definitely Unknown [3]

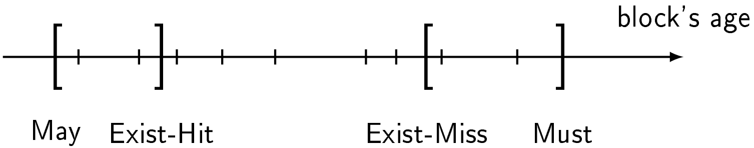


[3] V. Touzeau, C. Maïza, D. Monniaux, J. Reineke, CAV17

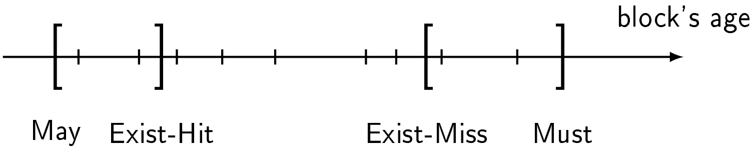
Definitely Unknown Analysis



Definitely Unknown Analysis



Definitely Unknown Analysis

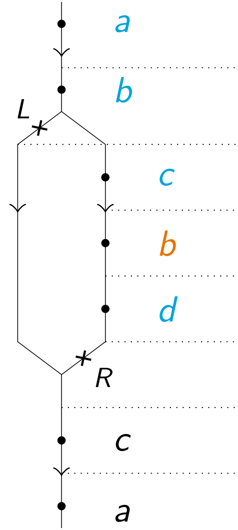


Definitely Unknown = Exist-Miss + Exist-Hit

- Ensure the existence of a path leading to a miss
- Ensure the existence of a path leading to a hit

Definitely Unknown

Exist-Miss Analysis



$[a, \bot, \bot, \bot]$

$L = [b, a, \bot, \bot]$

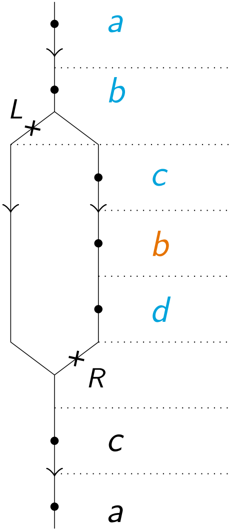
$[c, b, a, \bot]$

$[b, c, a, \bot]$

$R = [d, b, c, a]$

Definitely Unknown

Exist-Miss Analysis



$[a, \bot, \bot, \bot]$

$L = [b, a, \bot, \bot]$

$[c, b, a, \bot]$

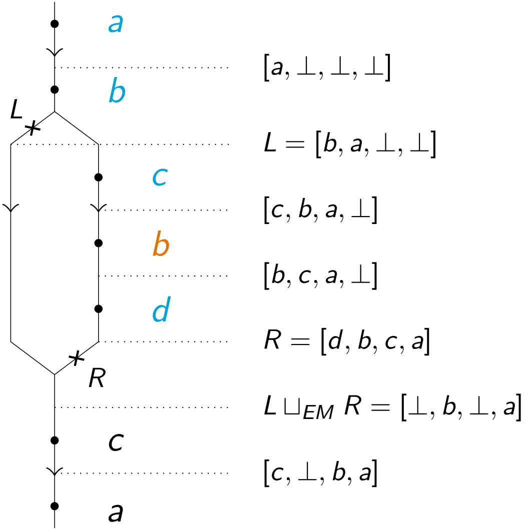
$[b, c, a, \bot]$

$R = [d, b, c, a]$

$L \sqcup_{EM} R = [\bot, b, \bot, a]$

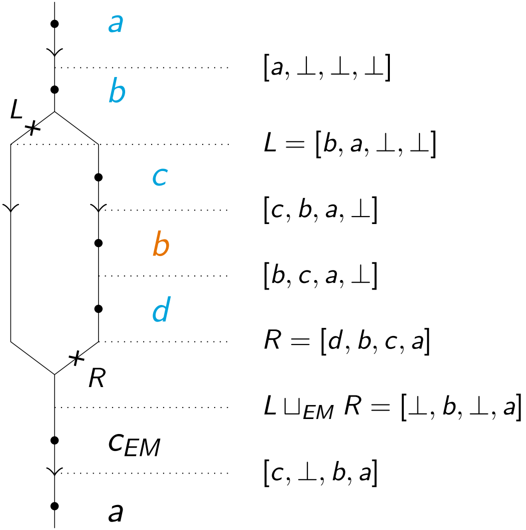
Definitely Unknown

Exist-Miss Analysis

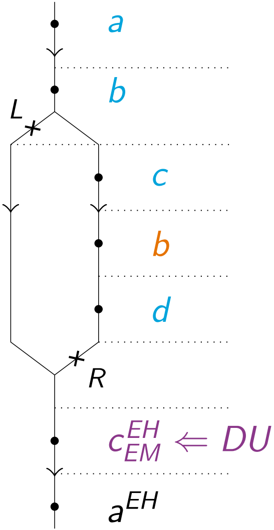


Definitely Unknown

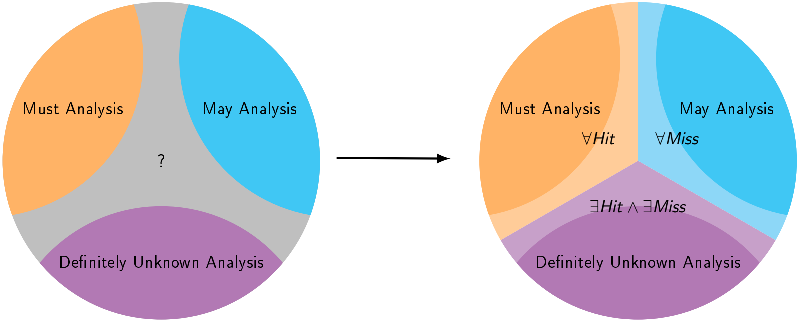
Exist-Miss Analysis



Definitely Unknown

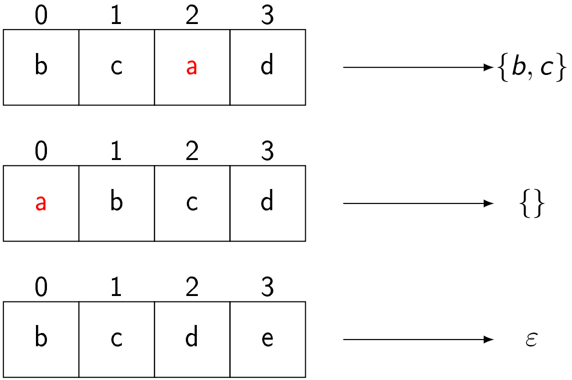


New Exact Analysis



Exact Analysis: Block Focusing

Abstraction relative to block a : set of younger blocks

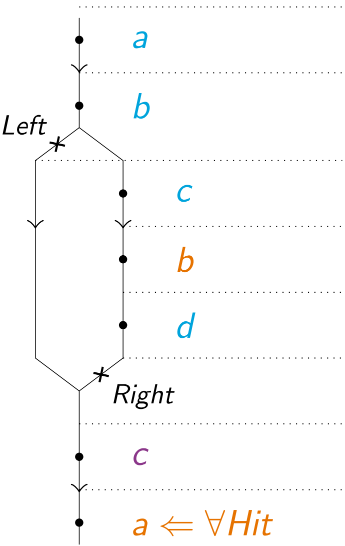


a is the most recently used

a is absent

Analyzing block ⓐ

Cache State Focused on block a



$\{\varepsilon\}$

$\{\emptyset\}$

$Left = \{\{b\}\}$

$\{\{b, c\}\}$

$Right = \{\{b, c, d\}\}$

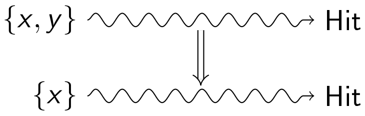
$Left \cup Right = \{\{b\}, \{b, c, d\}\}$

$\{\{b, c\}, \{b, c, d\}\}$

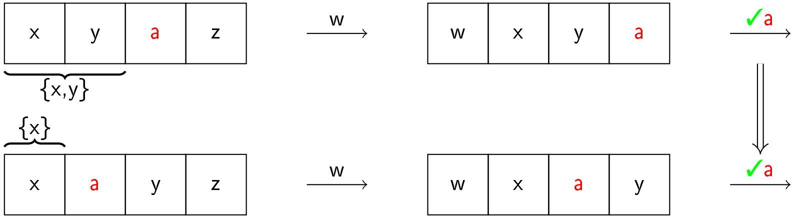
A more efficient analysis

Main idea: keep maximal younger sets when classifying hits.

Classifying hits

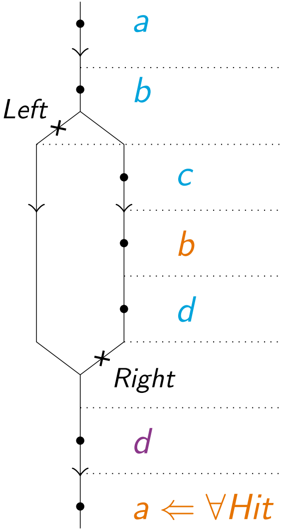


Example:



Example:

Control Flow Graph



Our Exact Analysis

ε

$\{\emptyset\}$

$Left = \{\{b\}\}$

$\{\{b, c\}\}$

$\{\{b, c\}\}$

$Right = \{\{b, c, d\}\}$

$Left \sqcup Right = \{\{b, c, d\}, $~~$\{b\}$~~$\}$

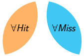
$\{\{b, c, d\}\}$

Experimental results

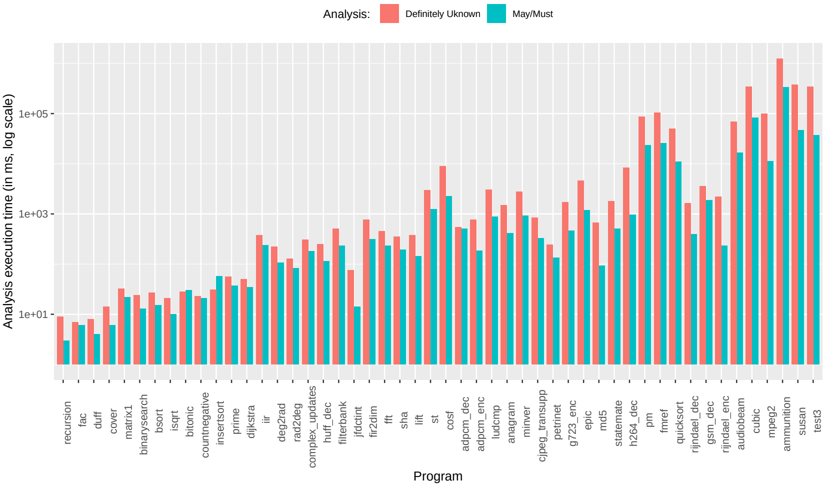
Experimental settings

- Our analyses are implemented in Otawa v2
- We rely on Cudd and libExtra for ZDD manipulation
- The Model Checker used is nuXmv
- We analyze the benchmarks from TacleBench
- ARM binaries contain between 18 and ~5000 memory accesses
- I-Cache contains 32 sets and 8 ways of 16 bytes (4KB in total)

DU Analysis efficiency

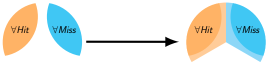


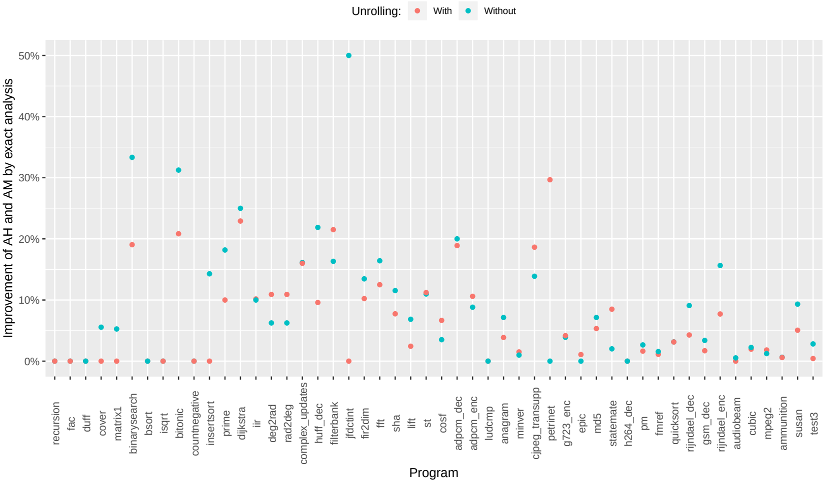
vs



- Definitely Unknown analysis is 2.72 times slower than the May/Must analysis.

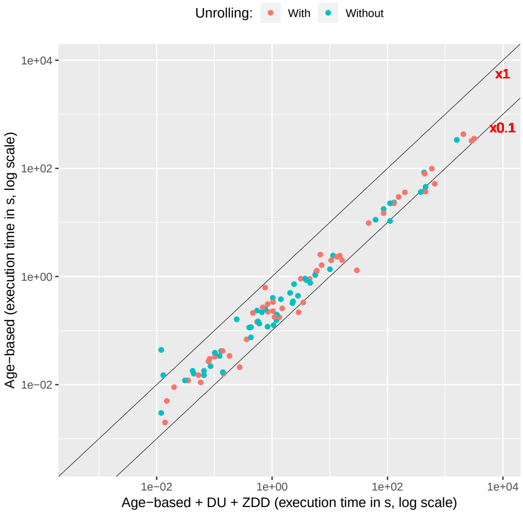
Classification improvement





- 18.2% more Always-Hit and Always-Miss with the exact approach.

Efficiency of the Exact Approach



- Exact approach is only 4.12 times slower in average.

Other contributions

Integration to WCET analysis

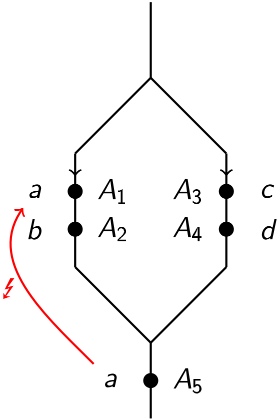
WCET estimation

Improvement of the WCET bound is modest (0.8% in average)

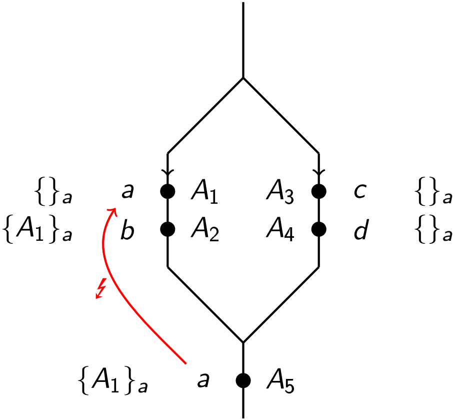
Analysis time

- Pipeline analysis is the WCET analysis bottleneck
- For unclassified accesses, both hit and miss cases must be considered
- Precise cache analysis thus reduce the state space
- Best improvement observed on big benchmark

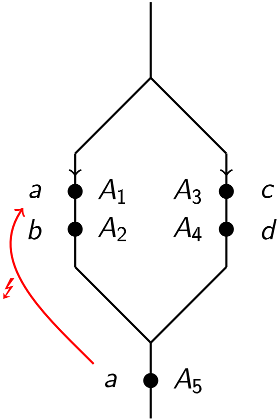
Application to Security



Application to Security



Application to Security



- This naive approach requires to store a high number of accesses
- Solution: reset the list of accesses when corresponding block is evicted

Complexity Results

General Problem

Given an access, how hard is it to find a path starting from an empty and leading to a hit/miss at this access?

Replacement Policy	LRU	PLRU	FIFO	MRU
Acyclic graph	NP-C	NP-C	NP-C	NP-C
Arbitrary graph	NP-C	PSPACE-C	PSPACE-C	PSPACE-C

Complexity Results

General Problem

Given an access, how hard is it to find a path starting from an empty and leading to a hit/miss at this access?

Replacement Policy	LRU	PLRU	FIFO	MRU
Acyclic graph	NP-C	NP-C	NP-C	NP-C
Arbitrary graph	NP-C	PSPACE-C	PSPACE-C	PSPACE-C

Except for MRU, all proofs are still valid in case of arbitrary initial cache state

Conclusion

Contributions

- A new analysis approximating the Definitely Unknwon accesses
- An approach providing a complete classification
- A theoretical study of the problem complexity
- Two applications, to security and WCET computation

Future work

- Partially take into account the program semantics.
  - take infeasible paths into account
  - extend the approach to persistent analysis
  - use classic abstract interpretation methods (trace partitioning, interprocedural analysis)
- Analyze data caches.
  - Requires precise address analysis
  - Take write policy into account
- Analyze other replacement policies.
  - Perform collecting semantics using hash-consing
  - Find abstract domain allowing performance and precision

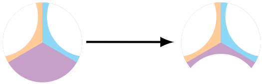
Collaboration

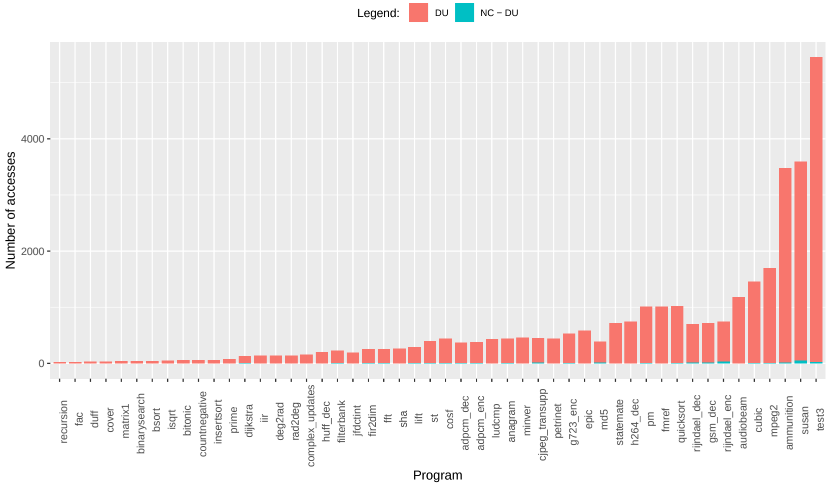
Many thanks to

- Zhenyu Bai
- Julien Balette-Pape
- Florian Barois
- Hugues Cassé
- Maeva Ramarijaona
- Maxime Raynal
- Jan Reineke

Thanks for your attention!

Definitely Unknown accesses





- 98.4% of Unknown accesses are Definitely Unknwon
- Exist-Hit and Exist-Miss highly reduce the state space