Verification of Real-Time Systems
Caches in WCET Analysis

Jan Reineke

Department of Computer Science
Saarland University
Saarbrücken, Germany

Advanced Lecture, Summer 2015
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges

3 Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
   ■ Challenges
   ■ Formalization of LRU and Logical Abstraction
   ■ Must Analysis
   ■ May Analysis
   ■ Remaining Challenges

3 Beyond Least-Recently-Used
   ■ Predictability Metrics
   ■ Relative Competitiveness
   ■ Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Cache vs Memory Performance

\[ x = a + b; \]

- LOAD \ r2, _a
- LOAD \ r1, _b
- ADD \ r3, r2, r1

Execution time on Motorola PowerPC 755:

Execution Time (Clock Cycles)

[Bar chart showing execution time for best and worst cases]
Caches

- How they work:
  - dynamically
  - managed by replacement policy

- Why they work: principle of locality
  - spatial
  - temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

![Diagram of CPU, Cache, and Main Memory]

Capacity: 32 KB
Latency: 3 cycles

Main Memory:
2 MB
100 cycles

- Why they work: principle of locality
  - spatial
  - temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

![Diagram of cache and main memory]

- Why they work: *principle of locality*
  - spatial
  - temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

```
CPU -> Cache
  "miss" [ab]
  (Cache capacity: 32 KB, Cache latency: 3 cycles)

Main Memory (Main memory capacity: 2 MB, Main memory latency: 100 cycles)
```

- Why they work: principle of locality
  - spatial
  - temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

  ![Diagram of CPU, Cache, and Main Memory with "miss" and latency information]

  - Capacity: 32 KB
  - Latency: 3 cycles
  - Main Memory: 2 MB

- Why they work: *principle of locality*
  - spatial
  - temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

- Why they work: principle of locality
  - spatial
  - temporal
Caches

■ How they work:
  ▶ dynamically
  ▶ managed by replacement policy

![Diagram]

■ Why they work: principle of locality
  ▶ spatial
  ▶ temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

  ![Diagram showing CPU, Cache, Main Memory with capacities and latencies]

  Capacity: 32 KB
  Latency: 3 cycles

  2 MB
  100 cycles

- Why they work: principle of locality
  - spatial
  - temporal
Caches

- How they work:
  - dynamically
  - managed by replacement policy

- Why they work: principle of locality
  - spatial
  - temporal
Fully-Associative Caches

Address:

Tag

Block offset

log₂(8 ⋅ b)

= ?

Yes: Hit!

No: Miss!

MUX

Tag

Data Block

... associativity

Tag

Data Block

Tag

Data Block

Data
Set-Associative Caches

Special cases:
- direct-mapped cache: only one line per cache set
- fully-associative cache: only one cache set
Cache Replacement Policies

- Least-Recently-Used (LRU) used in Intel Pentium I and MIPS 24K/34K
- First-In First-Out (FIFO or Round-Robin) used in Motorola PowerPC 56x, Intel XScale, ARM9, ARM11
- Pseudo-LRU (PLRU) used in Intel Pentium II-IV and PowerPC 75x
- Most Recently Used (MRU) as described in literature

Each cache set is treated independently:
→ Set-associative caches are compositions of fully-associative caches.
“Physical” Cache Implementation

- One Tag RAM per cache way: enables parallel lookup
- “Meta Data” RAM maintains additional information:
  - Replacement policy status
  - Data caches: “dirty” bits
  - Shared caches: cache coherence information
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
   ■ Challenges
   ■ Formalization of LRU and Logical Abstraction
   ■ Must Analysis
   ■ May Analysis
   ■ Remaining Challenges

3 Beyond Least-Recently-Used
   ■ Predictability Metrics
   ■ Relative Competitiveness
   ■ Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
   ■ Challenges
   ■ Formalization of LRU and Logical Abstraction
   ■ Must Analysis
   ■ May Analysis
   ■ Remaining Challenges

3 Beyond Least-Recently-Used
   ■ Predictability Metrics
   ■ Relative Competitiveness
   ■ Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Cache Analysis

Two types of cache analyses:

1. **Local guarantees: classification of individual accesses**
   - May-Analysis \(\rightarrow\) Overapproximates cache contents
   - Must-Analysis \(\rightarrow\) Underapproximates cache contents

2. **Global guarantees: bounds on cache hits/misses**
   - Special case: Persistence analysis

- Cache analyses mostly for LRU
- In practice: FIFO, PLRU, …
Challenges for Cache Analysis

Always a cache hit/always a miss?

read x
write z
read y
read z
Challenges for Cache Analysis

Always a cache hit/always a miss?

1. Initial cache contents unknown.
2. Different paths lead to these points.
3. Cannot resolve address of z.
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics = set of states at each program point that any execution may encounter there

Two approximations:

\[
\text{Collecting Semantics} \quad \text{uncomputable}
\]
\[
\subseteq \quad \text{Cache Semantics} \quad \text{computable}
\]
\[
\subseteq \quad \gamma(\text{Abstract Cache Sem.}) \quad \text{efficiently computable}
\]
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics = set of states at each program point that any execution may encounter there

Two approximations:

- Collecting Semantics uncomputable
- $\subseteq$ Cache Semantics computable
- $\subseteq$ $\gamma($Abstract Cache Sem.$)$ efficiently computable
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

- Collecting Semantics     uncomputable
- Cache Semantics          computable
- $\gamma$ (Abstract Cache Sem.)  efficiently computable
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics = set of states at each program point that any execution may encounter there

Two approximations:

- Collecting Semantics uncomputable
- $\subseteq$ Cache Semantics computable
- $\subseteq \gamma(\text{Abstract Cache Sem.})$ efficiently computable
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics = set of states at each program point that any execution may encounter there

Two approximations:

- Uncomputable
- Computable

(\gamma \text{ (Abstract Cache Sem.)}) efficiently computable
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
   ■ Challenges
   ■ Formalization of LRU and Logical Abstraction
   ■ Must Analysis
   ■ May Analysis
   ■ Remaining Challenges

3 Beyond Least-Recently-Used
   ■ Predictability Metrics
   ■ Relative Competitiveness
   ■ Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Least-Recently-Used (LRU): Concrete Domain

Concrete states of $k$-way fully-associative cache captured by three functions corresponding to tag, data, and meta data RAMs:

\[
\begin{align*}
tag & : \{1, \ldots, k\} \rightarrow B, \\
data & : \{1, \ldots, k\} \rightarrow B^n, \\
meta & : \{1, \ldots, k\} \rightarrow \{0, \ldots, k - 1\},
\end{align*}
\]

where $meta$ captures the “age” of the memory block stored in the $i^{th}$ cache line. This is the only LRU-specific aspect.
Least-Recently-Used (LRU): A First Abstraction

1. Cache analysis is interested in *which* memory blocks are cached, not *what data* they hold $\rightarrow$ abstract from *data*.

2. For predicting hits and misses, it is irrelevant in which *physical* cache line a memory block is cached. Only the relative ages of different cache blocks are important.

A “logical” LRU cache state can be captured by

$$
age : \mathcal{B} \rightarrow \{0, \ldots, k - 1, k\},$$

where uncached blocks obtain age $k$. 
Least-Recently-Used (LRU): A First Abstraction

1. Cache analysis is interested in *which* memory blocks are cached, not *what data* they hold $\longrightarrow$ abstract from *data*.

2. For predicting hits and misses, it is irrelevant in which *physical* cache line a memory block is cached. Only the relative ages of different cache blocks are important.

A “logical” LRU cache state can be captured by

$$ age : \mathcal{B} \to \{0, \ldots, k - 1, k\}, $$

where uncached blocks obtain age $k$. Logical state be obtained from “physical” cache states as follows:

$$ \alpha(\text{tag, data, meta}) := \lambda b \in \mathcal{B} : \begin{cases} meta(i) & \text{if tag}(i) = b \\ k & \text{if tag}(i) \neq b \forall i \in \{1, \ldots, k\} \end{cases} $$
Least-Recently-Used (LRU): Logical Behavior

$$up(age, b) := \lambda b' \in B : \begin{cases} 
0 & : \text{if } b' = b \\
age(b') & : \text{if } age(b) \leq age(b') \\
age(b') + 1 & : \text{if } age(b) > age(b')
\end{cases}$$
Least-Recently-Used (LRU): Logical Behavior

\[
up(\text{age}, b) := \lambda b' \in \mathcal{B} : \begin{cases} 
0 & : \text{if } b' = b \\
\text{age}(b') & : \text{if } \text{age}(b) \leq \text{age}(b') \\
\text{age}(b') + 1 & : \text{if } \text{age}(b) > \text{age}(b') 
\end{cases}
\]

"Cache Miss": LRU has notion of age
Least-Recently-Used (LRU): Logical Behavior

\[ up(\text{age}, b) := \lambda b' \in B : \begin{cases} 
    0 & : \text{if } b' = b \\
    \text{age}(b') & : \text{if } \text{age}(b) \leq \text{age}(b') \\
    \text{age}(b') + 1 & : \text{if } \text{age}(b) > \text{age}(b') 
\end{cases} \]

"Cache Miss":

"Cache Hit":

LRU has notion of age
LRU: Must-Analysis: Abstract Domain

- Used to predict *cache hits*.
- Maintains *upper bounds on ages* of memory blocks.
- Upper bound $\triangleleft$ associativity $\rightarrow$ memory block definitely cached.

**Example**

<table>
<thead>
<tr>
<th>Abstract state:</th>
<th>...and its interpretation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x}$</td>
<td>Describes the set of all concrete cache states in which $x$, $s$, and $t$ occur,</td>
</tr>
<tr>
<td>${}$</td>
<td>$x$ with an age of 0,</td>
</tr>
<tr>
<td>${s,t}$</td>
<td>$s$ and $t$ with an age not older than 2.</td>
</tr>
<tr>
<td>${}$</td>
<td>$\forall ([{x}, {}, {s, t}, {}]) =$</td>
</tr>
<tr>
<td>age 0</td>
<td>${[x, s, t, a], [x, t, s, a], [x, s, t, b], \ldots}$</td>
</tr>
</tbody>
</table>
Sound Update – Local Consistency

Abstract Update

\[ (must) \rightarrow (must') \]

\[ \gamma \]

Lifted Concrete Update

\[ \gamma \]

concrete cache states

concrete cache states
Formally:

\[ up_{must}(\widehat{age}, b) := \lambda b' \in B : \begin{cases} 
0 & : \text{if } b' = b \\
\text{age}(b') & : \text{if } \widehat{age}(b) \leq \widehat{age}(b') \\
\text{age}(b') + 1 & : \text{if } \widehat{age}(b) > \widehat{age}(b') 
\end{cases} \]
LRU: Must-Analysis: Update

“Potential Cache Miss”:

“Definite Cache Hit”:

Why does $t$ not age in the second case?
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $\text{age} \sqcup \text{age}' := \lambda b \in B : \max\{\text{age}(b), \text{age}'(b)\}$

Graphically:

\[
\begin{array}{c}
{a} \\
{c,f} \\
{d}
\end{array} \sqcup \begin{array}{c}
{c} \\
{e} \\
{a} \\
{d}
\end{array} = \begin{array}{c}
\{\} \\
{a,c} \\
{d}
\end{array}
\]
LRU: Must-Analysis: Join
Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $\text{age} \sqcup \text{age}' := \lambda b \in B : \max\{\text{age}(b), \text{age}'(b)\}$

Graphically:
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $\text{age} \sqcup \text{age}^\prime := \lambda b \in B : \max\{\text{age}(b), \text{age}^\prime(b)\}$

Graphically:

```
  {a}   {c}   {}
  {}    {e}    {}
{c, f} {a}    {a, c}
{d}    {d}    {d}
```
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $age \sqcup age' := \lambda b \in B : \max\{age(b), age'(b)\}$

Graphically:

\[
\begin{array}{c}
\{a\} \\
\{} \\
\{c,f\} \\
\{d\}
\end{array}
\sqcup
\begin{array}{c}
\{c\} \\
\{e\} \\
\{a\} \\
\{d\}
\end{array}
= 
\begin{array}{c}
\{} \\
\{} \\
\{a,c\} \\
\{d\}
\end{array}
\]
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

Formally: $a_{ge} \sqcup a_{ge'} := \lambda b \in B : \max\{a_{ge}(b), a_{ge'}(b)\}$

Graphically:

$$
\begin{array}{c|c|c}
\{a\} & \{c\} & \{\}
\{\} & \{e\} & \{\}
\{c,f\} & \{a\} & \{a,c\}
\{d\} & \{d\} & \{d\}
\end{array}
$$
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $age \sqcup age' := \lambda b \in B : \max\{age(b), age'(b)\}$

Graphically:
LRU: Must-Analysis: Join
Need to combine information where control-flow merges.

Join should be conservative (ensures \( \gamma \) is monotone):
- \( \gamma(A) \subseteq \gamma(A \cup B) \)
- \( \gamma(B) \subseteq \gamma(A \cup B) \)

Formally: \( \text{age} \cup \text{age}' \) := \( \lambda b \in B : \max\{\text{age}(b), \text{age}'(b)\} \)

Graphically:

\[
\begin{array}{c|c|c}
\{a\} & \{c\} & \{\} \\
\{\} & \{e\} & \{\} \\
\{c,f\} & \{a\} & \{a,c\} \\
\{d\} & \{d\} & \{d\}
\end{array}
\]

How many memory blocks can be in the must-cache? What about the ascending chain condition?
Example: Must-Analysis

entry \[\text{\{}\text{\}, \text{\}, \text{\}, \text{\}}\]

exit \[\text{\}}\]

Diagram:

- Node A
- Node B
- Node C
- Node D

Arrows connect nodes A to B, B to C, C to D, and D back to A.
Example: Must-Analysis

entry \[ \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \]

\[ \bot \cup [\emptyset, \emptyset, \emptyset, \emptyset] = [\emptyset, \emptyset, \emptyset, \emptyset] \]

exit \[ \bot \]
Example: Must-Analysis

entry $[\{\}, \{\}, \{\}, \{\}]$

$\bot \cup [\{\}, \{\}, \{\}, \{\}] = [\{\}, \{\}, \{\}, \{\}]$

$[\{A\}, \{\}, \{\}, \{\}]$

$[\{A\}, \{\}, \{\}, \{\}]$

exit $\bot$
Example: Must-Analysis

entry \[
\emptyset, \emptyset, \emptyset, \emptyset, \emptyset
\]

\[
\bot \sqcup \emptyset, \emptyset, \emptyset, \emptyset = \emptyset, \emptyset, \emptyset, \emptyset
\]

\[
\{A\}, \emptyset, \emptyset, \emptyset, \emptyset
\]

\[
\{A\}, \emptyset, \emptyset, \emptyset, \emptyset
\]

\[
\{B\}, \{A\}, \emptyset, \emptyset \sqcup \{C\}, \{A\}, \emptyset, \emptyset = \emptyset, \{A\}, \emptyset, \emptyset
\]

exit \[
\bot
\]
Example: Must-Analysis

entry: $[\emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$

$[\{D\}, \{\}, \{A\}, \{\}] \sqcup [\{\}, \{\}, \{\}, \{\}] = [\{\}, \{\}, \{\}, \{\}]$

$[\{A\}, \{\}, \{\}, \{\}]$

$[\{B\}, \{A\}, \{\}, \{\}] \sqcup [\{C\}, \{A\}, \{\}, \{\}] = [\{\}, \{A\}, \{\}, \{\}]$

exit: $[\{D\}, \{\}, \{A\}, \{\}]$

No cache hits can be predicted :-(

Jan Reineke
Caches in WCET Analysis
Advanced Lecture, 2015
Context-Sensitive Analysis/Virtual Loop-Unrolling

- Problem:
  - The first iteration of a loop will always result in cache misses.
  - Similarly for the first execution of a function.
- Solution: *Pull*
  - Virtually Unroll Loops: Distinguish the first iteration from others
  - Distinguish function calls by calling context.

Virtually unrolling the loop once:

- Accesses to $A$ and $D$ are provably hits after the first iteration
- Accesses to $B$ and $C$ can still not be classified. Within each execution of the loop, they may only miss once.

  $\rightarrow$ Persistence Analysis
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges

3 Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
LRU: May-Analysis: Abstract Domain

- Used to predict cache misses.
- Maintains lower bounds on ages of memory blocks.
- Lower bound \( \geq \) associativity

\[ \rightarrow \text{memory block definitely not cached.} \]

**Example**

Abstract state:

| age 0 | \{x,y\} | \{} | \{s,t\} | \{u\} |

\[ \gamma([\{x,y\}, \{}, \{s,t\}, \{u\}]) = \{[x,y,s,t], [y,x,s,t], [x,y,s,u], \ldots \} \]

...and its interpretation:

Describes the set of all concrete cache states in which no memory blocks except \( x, y, s, t, \) and \( u \) occur,

- \( x \) and \( y \) with an age of at least 0,
- \( s \) and \( t \) with an age of at least 2,
- \( u \) with an age of at least 3.
LRU: May-Analysis: Update

Formally:

\[ \text{up}_{\text{may}}(\hat{\text{age}}, b) := \lambda b' \in B : \begin{cases} 
0 & : \text{if } b' = b \\
\text{age}(b') & : \text{if } \hat{\text{age}}(b) < \hat{\text{age}}(b') \\
\text{age}(b') + 1 & : \text{if } \hat{\text{age}}(b) \geq \hat{\text{age}}(b') \neq k \\
k & : \text{if } \hat{\text{age}}(b') = k
\end{cases} \]
LRU: May-Analysis: Update

"Definite Cache Miss":

\[
\begin{array}{c}
\{x,u\} \\
\{s,t\} \\
\{y\}
\end{array} \quad \Rightarrow \\ 
\begin{array}{c}
\{z\} \\
\{x,u\} \\
\{s,t\}
\end{array}
\]

"Potential Cache Hit":

\[
\begin{array}{c}
\{x,u\} \\
\{s,t\} \\
\{y\}
\end{array} \quad \Rightarrow \\ 
\begin{array}{c}
\{s\} \\
\{x,u\} \\
\{y,t\}
\end{array}
\]

Why does \( t \) age in the second case?
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

Formally: $\text{age} \sqcup \text{age}' := \lambda b \in B : \min\{\text{age}(b), \text{age}'(b)\}$

Graphically:

```
\begin{array}{ccc}
\{a,b\} & \{c\} & \{a,b,c\} \\
\{\}\quad & \{e\} & \{e\} \\
\{c,f\} & \{a\} & \{f\} \\
\{d\} & \{d\} & \{d\} \\
\end{array}
```
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

Formally: $age \sqcup age' := \lambda b \in B : \min\{age(b), age'(b)\}$

Graphically:

\[
\begin{array}{c}
\{a,b\} \\
\{\} \\
\{c,f\} \\
\{d\}
\end{array}
\sqcup
\begin{array}{c}
\{c\} \\
\{e\} \\
\{a\} \\
\{d\}
\end{array}
= 
\begin{array}{c}
\{a,b,c\} \\
\{e\} \\
\{f\} \\
\{d\}
\end{array}
\]
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $age \sqcup age' := \lambda b \in B : \min\{age(b), age'(b)\}$

Graphically:

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{a,b}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{e}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{f}</td>
<td>{d}</td>
</tr>
<tr>
<td>{c}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{d}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{a,b, c}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{e}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{f}</td>
<td>{d}</td>
</tr>
<tr>
<td>{c}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{d}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

Formally: $\text{age} \sqcup \text{age}' := \lambda b \in B : \min\{\text{age}(b), \text{age}'(b)\}$

Graphically:

```
  {a,b}   \{c\}   \{a,b,c\}
    \{\}   \{e\}   \{e\}
    \{c,f\} \{a\}   \{f\}
    \{d\}  \{d\}   \{d\}
```
**LRU: May-Analysis: Join**

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

Formally: $\text{age} \cup \text{age}' := \lambda b \in B : \min\{\text{age}(b), \text{age}'(b)\}$

Graphically:

\[
\begin{array}{c}
\{a,b\} \\
\{\} \\
\{c,f\} \\
\{d\}
\end{array}
\quad \sqcup 
\quad
\begin{array}{c}
\{c\} \\
\{e\} \\
\{a\} \\
\{d\}
\end{array}
\quad = 
\begin{array}{c}
\{a,b,c\} \\
\{e\} \\
\{f\} \\
\{d\}
\end{array}
\]
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures $\gamma$ is monotone):

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

Formally: $age \cup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

$$
\begin{array}{c|c|c}
\{a,b\} & \{c\} & \{a,b,c\} \\
\{\} & \{e\} & \{e\} \\
\{c,f\} & \{a\} & \{f\} \\
\{d\} & \{d\} & \{d\}
\end{array}
$$

How many memory blocks can be in the must-cache?

What about the ascending chain condition?
Outline

1  Caches

2  Cache Analysis for Least-Recently-Used
   ■ Challenges
   ■ Formalization of LRU and Logical Abstraction
   ■ Must Analysis
   ■ May Analysis
   ■ Remaining Challenges

3  Beyond Least-Recently-Used
   ■ Predictability Metrics
   ■ Relative Competitiveness
   ■ Sensitivity – Caches and Measurement-Based Timing Analysis

4  Summary
Accounting for Uncertainty about Memory Addresses

For instruction-cache analysis, there is no uncertainty about the accessed memory blocks. But what about data accesses?

So far: \( \text{up}(\text{age}, x) : \text{Cache} \times B \to \text{Cache} \).  

With uncertainty: \( \text{up}(\text{age}, X) : \text{Cache} \times 2^{B \cup \{\bot\}} \to \text{Cache} \).
Accounting for Uncertainty about Memory Addresses

For instruction-cache analysis, there is no uncertainty about the accessed memory blocks. But what about data accesses?

So far: \( \text{up}(\text{age}, x) : \widehat{\text{Cache}} \times \mathcal{B} \rightarrow \widehat{\text{Cache}}. \)

With uncertainty: \( \text{up}(\text{age}, X) : \widehat{\text{Cache}} \times 2^{\mathcal{B} \cup \{\bot\}} \rightarrow \widehat{\text{Cache}}. \)

**How to define** \( \text{up}(\text{age}, X)? \)

\[
\text{up}(\text{age}, X) := \bigcup \{ \text{mp}(\text{age}, x) \mid x \in X \}
\]
Accounting for Uncertainty about Memory Addresses

For instruction-cache analysis, there is no uncertainty about the accessed memory blocks. But what about data accesses?

So far: $\text{up}(\widehat{\text{age}}, x) : \text{Cache} \times \mathcal{B} \rightarrow \text{Cache}$. 

With uncertainty: $\text{up}(\widehat{\text{age}}, X) : \text{Cache} \times 2^\mathcal{B}\text{U}\{\bot\} \rightarrow \text{Cache}$. 

**How to define $\text{up}(\widehat{\text{age}}, X)$?**

$$\text{up}(\widehat{\text{age}}, X) := \bigsqcup \{ \text{up}(\widehat{\text{age}}, x) \mid x \in X \} \text{ where } \text{up}(\widehat{\text{age}}, \bot) := \widehat{\text{age}}$$

Practical effect?
Extension to Set-Associative Caches

One view: Set-associative cache = array of fully-associative caches

\[ up_{set}(\hat{s}, x) := \hat{s}[\text{index}(x) \mapsto up(\hat{s}(\text{index}(x)), x)], \]

where \( \text{index}(x) \) determines the set number of block \( x \).

Uncertainty can be accounted for as described before.

\[ \rightarrow \text{Excursion: Relational Cache Analysis} \]
Extension to Set-Associative Caches

One view: Set-associative cache = array of fully-associative caches

\[ \text{up}_{set}(\hat{s}, x) := \hat{s}[\text{index}(x) \mapsto \text{up}(\hat{s}(\text{index}(x)), x)], \]

where \( \text{index}(x) \) determines the set number of block \( x \).

Uncertainty can be accounted for as described before.

**Effect of address uncertainty on results?**

\[ \rightarrow \text{Excursion: Relational Cache Analysis} \]
Data Caches: Write-through vs Write-back

Two different write policies:
Write-through: Every write goes directly to main memory.
Write-back: Cache keeps track of “dirty” cache lines and writes back changes to main memory only when cache line is evicted.

What is preferable in terms of performance/analysis?