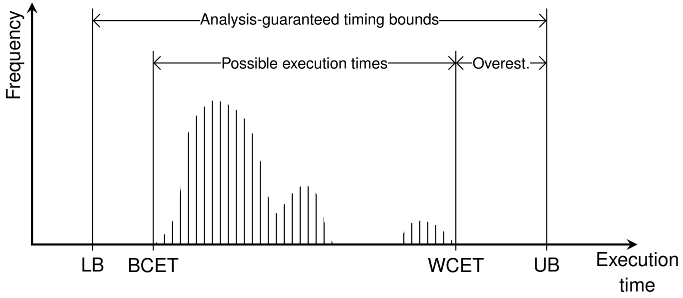
Relational Cache Analysis for Static Timing Analysis

Sebastian Hahn   Daniel Grund

ECRTS 2012

UdS-CS-Logo.pdf

Static Timing Analysis



- influence of the hardware on execution time
  - caches, pipelines, . . .
- tight bounds require micro-architectural analysis, e.g. cache analysis

Static Cache Analysis

- Approximate cache content at each program point
- Classify memory references as cache hit or cache miss

Must-cache analysis

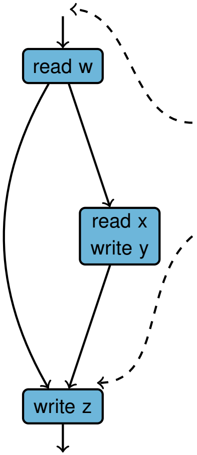
- under-approximation
- classify hits

May-cache analysis

- over-approximation
- classify misses

Static Cache Analysis

Challenges



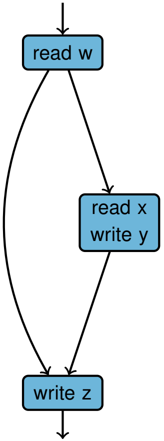
Initial cache contents unknown

Need to combine analysis information

Need to determine information about w, x, y, z

Static Cache Analysis

Two-step Approach



$w \in [\texttt{0x60810}, \texttt{0x60810}]$

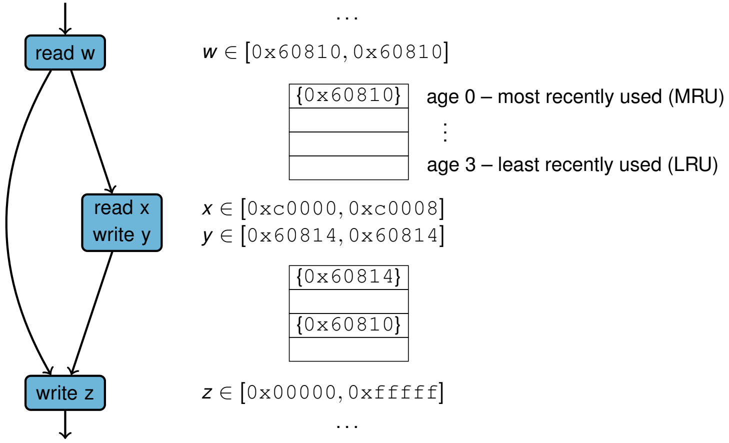
$x \in [\texttt{0xc0000}, \texttt{0xc0008}]$
$y \in [\texttt{0x60814}, \texttt{0x60814}]$

$z \in [\texttt{0x00000}, \texttt{0xfffff}]$

1. Approximate accessed addresses by Value Analysis

Static Cache Analysis for LRU replacement policy

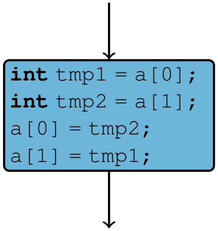
Two-step Approach



read w
read x
write y
write z
...
$w \in [\texttt{0x60810}, \texttt{0x60810}]$
{0x60810}
age 0 – most recently used (MRU)
⋮
age 3 – least recently used (LRU)
$x \in [\texttt{0xc0000}, \texttt{0xc0008}]$
$y \in [\texttt{0x60814}, \texttt{0x60814}]$
{0x60814}
{0x60810}
$z \in [\texttt{0x00000}, \texttt{0xfffff}]$
...

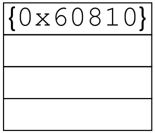
1. Approximate accessed addresses by Value Analysis
2. Approximate cached memory blocks by Cache Analysis

Example

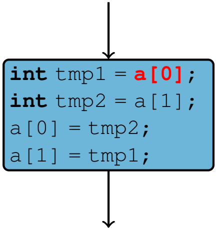


```
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```

1. Address Information
   `a[0],a[1]` $\in$ `[0x00000, 0xfffff]`
2. Cache Information

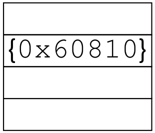


Example



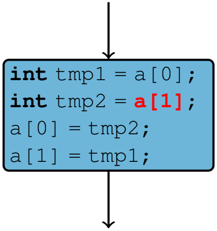
```
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```

1. Address Information
   $\texttt{a[0]}, \texttt{a[1]} \in [\texttt{0x00000}, \texttt{0xfffff}]$
2. Cache Information



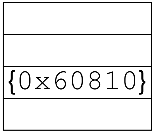
Intangibility of Memory Blocks
$\Rightarrow$ not guaranteed to be cached

Example



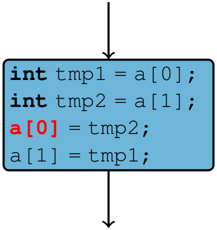
```
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```

1. Address Information
   $\texttt{a[0]}, \texttt{a[1]} \in [\texttt{0x00000}, \texttt{0xfffff}]$
2. Cache Information



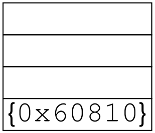
Intangibility of Memory Blocks
$\Rightarrow$ not guaranteed to be cached

Example



```
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```

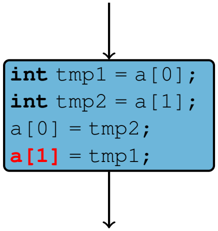
1. Address Information
   $\texttt{a[0]}, \texttt{a[1]} \in [\texttt{0x00000}, \texttt{0xfffff}]$
2. Cache Information



Intangibility of Memory Blocks
$\Rightarrow$ not guaranteed to be cached

Excessive Information Loss

Example



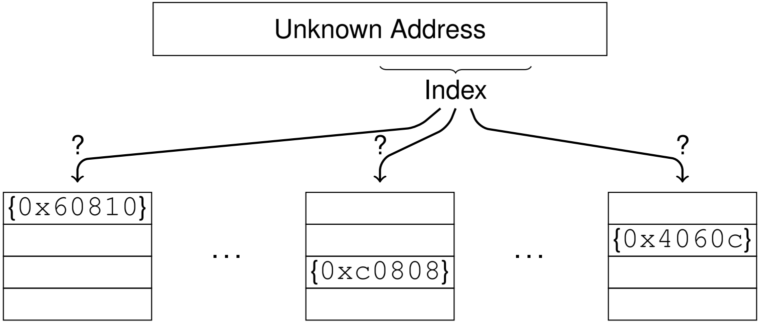
```
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```

1. Address Information
   $\texttt{a[0]}, \texttt{a[1]} \in [\texttt{0x00000}, \texttt{0xfffff}]$
2. Cache Information

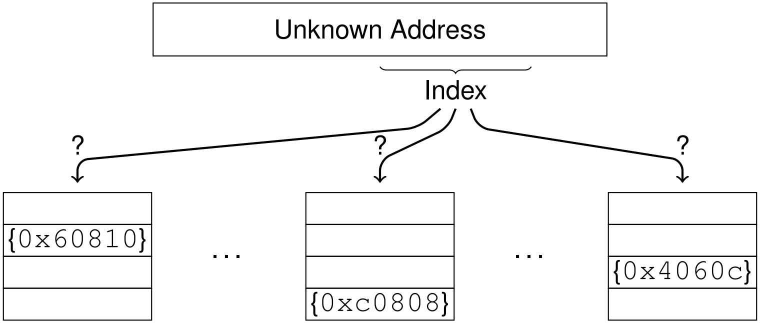
Intangibility of Memory Blocks
$\Rightarrow$ not guaranteed to be cached

Excessive Information Loss
$\Rightarrow$ not guaranteed to be cached anymore

Multiple Cache Sets



Multiple Cache Sets



Multiple Aging

$\Rightarrow$ any cache set might be affected

Tagline

Precisely determined addresses

are not necessary for

precise cache analysis.

Tagline

Precisely determined addresses

are not necessary for

precise cache analysis.

But relations between addresses.

Outline

Symbolic Names

Definition (Symbolic Name)

Unique identifier for an occurrence of an address expression

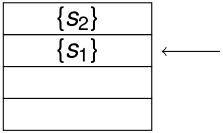
```
...
int tmp1 = a[0];
int tmp2 = a[1];
a[0]     = tmp2;
a[1]     = tmp1;
...
```

After Compilation

```
...
add r5, r1, 0      bind   s2
ld r10, [r5]       deref. s2
add r6, r1, 4      bind   s3
ld r11, [r6]       deref. s3
st [r5], r11       deref. s2
st [r6], r10       deref. s3
...
```

Abstract Caches

- symbolic names as abstract cache elements

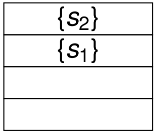


during execution, the memory block represented by s_1 has at most age 1

- symbolic names abstract from concrete addresses
  $\rightarrow$ all memory references tangible

Relations between Symbolic Names

```
...
add r6, r1, 4      bind   s3
ld  r11, [r6]      deref. s3
...
```



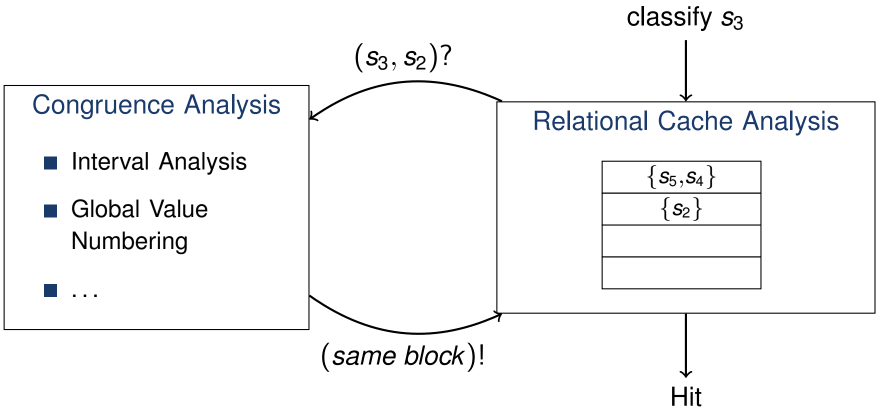
$\xrightarrow{\text{deref. } s_3}$

Classify reference s_3 as hit?
How are s_1 and s_2 affected?

Use of relational information

- s_3 and s_1 denote the **same memory block** $\rightarrow$ classify reference as hit
- s_3 and s_2 map to **different cache sets** $\rightarrow$ s_2 not affected (e.g. no aging)

Overall Framework



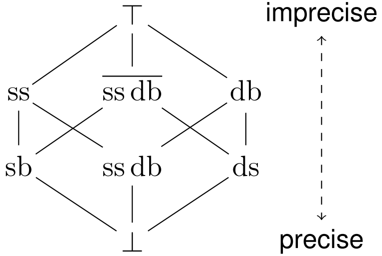
Relations

$$\begin{tikzcd}
 & \top \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\
\mathrm{ss} \arrow[d, no head] \arrow[dr, no head] & \overline{\mathrm{ss}\,\mathrm{db}} \arrow[dl, no head] \arrow[dr, no head] & \mathrm{db} \arrow[dl, no head] \arrow[d, no head] \\
\mathrm{sb} \arrow[dr, no head] & \mathrm{ss}\,\mathrm{db} \arrow[d, no head] & \mathrm{ds} \arrow[dl, no head] \\
 & \bot &
\end{tikzcd}$$

Relation	Meaning
ss	same cache set
ds	different cache set
sb	same block
db	different block
$\mathrm{ss}\,\mathrm{db}$	ss and db
$\overline{\mathrm{ss}\,\mathrm{db}}$	ds or sb

sb	classify hits
$\mathrm{ss}\,\mathrm{db}$	account for cache conflicts
ds	exclude possible eviction

Relations



Relation	Meaning
ss	same cache set
ds	different cache set
sb	same block
db	different block
$\mathrm{ss\,db}$	ss and db
$\overline{\mathrm{ss\,db}}$	ds or sb

Induces partial order $\sqsubseteq$: $\mathrm{sb} \sqsubseteq \overline{\mathrm{ss\,db}}$ and $\mathrm{ds} \sqsubseteq \overline{\mathrm{ss\,db}}$

sb	classify hits
$\mathrm{ss\,db}$	account for cache conflicts
ds	exclude possible eviction

Congruence Information

Partial Execution Trace τ

$$
\begin{aligned}
&\langle s_1 \mapsto \texttt{0x60810} \rangle \\
\circ&\langle s_1 \rangle \\
\circ&\langle s_2 \mapsto \texttt{0xbffc0} \rangle \\
\circ&\langle s_2 \rangle \\
\circ&\langle s_3 \mapsto \texttt{0xbffc4} \rangle \\
\circ&\langle s_3 \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{rel}(\tau, s_1, s_3) \\
=&\widehat{\mathit{rel}}(\mathit{last}(\tau, s_1), \mathit{last}(\tau, s_3)) \\
=&\widehat{\mathit{rel}}(\texttt{0x60810}, \texttt{0xbffc4}) \\
=&\mathrm{ds}
\end{aligned}
$$

Congruence Information

Partial Execution Trace τ

$$\begin{aligned}
&\langle s_1 \mapsto \texttt{0x60810} \rangle \\
\circ&\langle s_1 \rangle \\
\circ&\langle s_2 \mapsto \texttt{0xbffc0} \rangle \\
\circ&\langle s_2 \rangle \\
\circ&\langle s_3 \mapsto \texttt{0xbffc4} \rangle \\
\circ&\langle s_3 \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathit{rel}(\tau, s_1, s_3) \\
=&\widehat{\mathit{rel}}(\mathit{last}(\tau, s_1), \mathit{last}(\tau, s_3)) \\
=&\widehat{\mathit{rel}}(\texttt{0x60810}, \texttt{0xbffc4}) \\
=&\,\mathrm{ds}
\end{aligned}$$

Partial Execution Trace τ'

$$\begin{aligned}
&\langle s_1 \mapsto \texttt{0x60810} \rangle \\
\circ&\langle s_1 \rangle \\
\circ&\langle s_2 \mapsto \texttt{0xbffc4} \rangle \\
\circ&\langle s_2 \rangle \\
\circ&\langle s_3 \mapsto \texttt{0xbffc8} \rangle \\
\circ&\langle s_3 \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathit{rel}(\tau', s_1, s_3) \\
=&\widehat{\mathit{rel}}(\mathit{last}(\tau', s_1), \mathit{last}(\tau', s_3)) \\
=&\widehat{\mathit{rel}}(\texttt{0x60810}, \texttt{0xbffc8}) \\
=&\,\mathrm{ss}\,\mathrm{db}
\end{aligned}$$

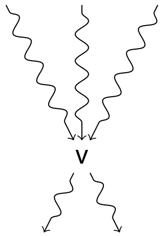
$\rightarrow$ congruence information has to safely account for both cases

Congruence Information

Congruence information modelled as one function

$$cgr_v : \mathcal{N} \times \mathcal{N} \to \mathcal{R}$$

per program location v.



Definition (Validity of Congruence Information)

Let $\mathcal{T}_v$ be the set of partial execution traces up to program location v.
cgr_v is called valid if for all $\tau \in \mathcal{T}_v$ and for all $s, t \in \mathcal{N}$

$$cgr_v(s, t) \sqsupseteq rel(\tau, s, t).$$

Computing Congruence Information

Global Value Numbering [Rosen, Wegman, and Zadeck, 1988]

$vn : expressions \rightarrow \mathbb{N}$

$vn(e_1) = vn(e_2) \Rightarrow e_1$ and e_2 compute the same value

Symbolic names s_1 and s_2 with associated address expressions ...

- address expressions e_1 and e_2, where $vn(e_1) = vn(e_2)$
  $\Rightarrow \mathrm{sb}$ relation
- address expressions e_1 and $e_2 + linesize$, where $vn(e_1) = vn(e_2)$
  $\Rightarrow \mathrm{ds}$ relation

Relational Cache Analysis

Similar to Ferdinand's must cache analysis [Ferdinand, 1997], but

- symbolic names as abstract cache elements instead of memory blocks
  $\rightarrow$ abstract from concrete addresses
- more general congruence information instead of address information
  $\rightarrow$ e.g. the address information can be used to compute relations

Outline

Evaluation Setting

Implementation

- FIRM Intermediate representation [Braun, Buchwald, and Zwinkau, 2011]
- x86 assembler graph produced by compilation
- interval analysis and global value numbering as congruence analyses

Three application areas

1. stack-relative memory accesses
2. array accesses within one loop iteration
3. input-dependent memory accesses

Input-dependent Memory Accesses

Taken from Mälardalen benchmarks [Gustafson, Betts, Ermedahl, and Lisper, 2010]

```
void fdct(int *block, int lx) {
  ...
  /* Pass 1: process rows. */
  ...
  /* Pass 2: process columns. */

  for (i = 0; i<8; i++) {
    tmp0 = block[0]    + block[7*lx];
    tmp7 = block[0]    - block[7*lx];
    tmp1 = block[lx]   + block[6*lx];
    tmp6 = block[lx]   - block[6*lx];
    ...
    block[0]    = ...
    block[6*lx] = ...
    block[7*lx] = ...
    block[lx]   = ...
    ...
    /* advance to next column */
    block++;
  }
}
```

Configuration			references classified as always hit	
line size	assoc.	sets	State-of-the-art	Relational
4	4	4	11	15
4	8	4	26	37
8	4	4	36	42
8	8	4	54	67
16	4	4	56	67
16	8	4	73	84

Evaluation

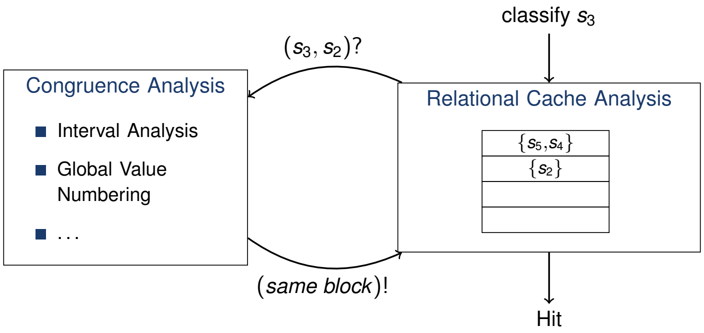
Qualitative Result

Relational Cache Analysis
is at least as precise as
Ferdinand's Cache Analysis

Summary and Conclusion

Absolute address information not needed for precise cache analysis

- Symbolic names abstract from concrete addresses
- Congruence analysis module provides relations between symbolic names



Future Work

Congruence Analysis

- New congruence analyses e.g.,
  - the Value-Set Analysis by Balakrishnan and Reps
  - the Congruence Analysis by Wegener at WCET 2012
- Effects of different congruence analyses on the analysis precision

Cache Analysis

- Improve abstract domain
- May analysis

Applications

- Analysing accesses to dynamically allocated data structures

Stack-relative Memory Accesses

```
int comp(int a1, int a2, int a3,
            int b1, int b2, int b3,
            int c1, int c2, int c3) {
  int p1 = a2 * b3 + a3 * b2;
  int p2 = a3 * b1 + a1 * b3;
  int p3 = a1 * b2 + a2 * b1;

  int p4 = a2 * c3 + a3 * c2;
  int p5 = a3 * c1 + a1 * c3;
  int p6 = a1 * c2 + a2 * c1;

  int p7 = b2 * c3 + b3 * c2;
  int p8 = b3 * c1 + b1 * c3;
  int p9 = b1 * c2 + b2 * c1;

  return p1 * c1 + p2 * c2 + p3 * c3 +
         p4 * b1 + p5 * b2 + p6 * b3 +
         p7 * a1 + p8 * a2 + p9 * a3;
}
```

Configuration			Precise SP `0xc000`	
ls	*k*	*n*	traditional	relational
4	4	4	18	18
4	8	4	18	18
8	4	4	25	25
8	8	4	25	25
16	4	4	28	28
16	8	4	28	28

Configuration			Imprecise SP `0xc000` - `0xc008`	
ls	*k*	*n*	traditional	relational
4	4	4	0	14
4	8	4	0	15
8	4	4	0	15
8	8	4	0	18
16	4	4	0	18
16	8	4	0	18

Array Reuse Within One Loop Iteration

```
int a[50][50], b[50];

int main(void) {
  int i, j, n = 50, w;
  for (i = 0; i <= n; i++) {
    w = 0;
    for (j = 0; j <= n; j++) {
      a[i][j] = (i + 1) + (j + 1);
      if (i == j)
        a[i][j] *= 10;
      else
        a[i][j] *= 2;
      w += a[i][j];
    }
    b[i] = w;
  }
  return 0;
}
```

Configuration			Number of references classified as always hit	
ls	*k*	*n*	traditional	relational
4	4	4	0	3
4	8	4	0	3
8	4	4	3	6
8	8	4	3	6
16	4	4	5	8
16	8	4	5	8

Congruence Information

Partial Execution Trace τ

$$\begin{array}{l} \quad\langle s_1 \mapsto \texttt{0x60810} \rangle \\ \circ\langle s_1 \rangle \end{array}$$

$$\begin{aligned} cgr(s_1, s_3) &= \widehat{rel}(last(\tau, s_1), last(\tau, s_3)) \\ &= \widehat{rel}(\texttt{0x60810}, \bot) \\ &= \bot \end{aligned}$$

Congruence Information

Partial Execution Trace τ'

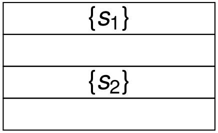
$$\begin{aligned}
&\langle s_1 \mapsto \texttt{0x60810} \rangle \\
\circ&\langle s_1 \rangle \\
\circ&\langle s_2 \mapsto \texttt{0xbffc0} \rangle \\
\circ&\langle s_2 \rangle \\
\circ&\langle s_3 \mapsto \texttt{0xbffc4} \rangle \\
\circ&\langle s_3 \rangle
\end{aligned}$$

$$\begin{aligned}
cgr(s_1, s_3) &= \widehat{rel}(last(\tau, s_1), last(\tau, s_3)) \\
&= \widehat{rel}(\texttt{0x60810}, \perp) \\
&= \perp
\end{aligned}$$

$$\begin{aligned}
cgr(s_1, s_3) &= \widehat{rel}(last(\tau', s_1), last(\tau', s_3)) \\
&= \widehat{rel}(\texttt{0x60810}, \texttt{0xbffc4}) \\
&= \mathrm{ds}
\end{aligned}$$

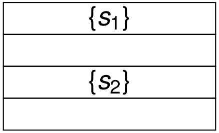
$\rightarrow$ congruence information depends on program location

Abstract Cache Domain



$$\mathcal{N} \rightarrow \{0, \ldots, k-1, \infty\}$$

Abstract Cache Domain

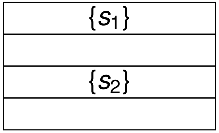


$+ \qquad cgr_v(s_1, s_2) = \mathrm{sb}$

$$(\mathcal{N} \to \mathcal{AB}^{\leq}) \times (\mathcal{N} \times \mathcal{N} \to \mathcal{R})$$

Abstract Cache Domain

Effective Age Bound — Aliasing Problem



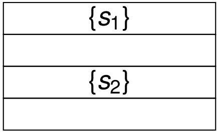
$$+ \qquad cgr_v(s_1, s_2) = \mathrm{sb}$$

$$\downarrow$$

$$eab^{\leq} : (\mathcal{N} \to \mathcal{AB}^{\leq}) \times (\mathcal{N} \times \mathcal{N} \to \mathcal{R}) \to (\mathcal{N} \to \mathcal{AB}^{\leq})$$
$$eab^{\leq}(ab, cgr_v) = \lambda s \in \mathcal{N}.\, \min\{ab(t) \mid t \in \mathcal{N} \wedge cgr_v(s,t) = \mathrm{sb}\}$$

Abstract Cache Domain

Effective Age Bound — Normalisation

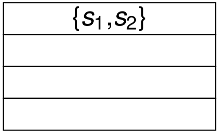


$+ \quad cgr_v(s_1, s_2) = \mathrm{sb}$

$$\downarrow$$

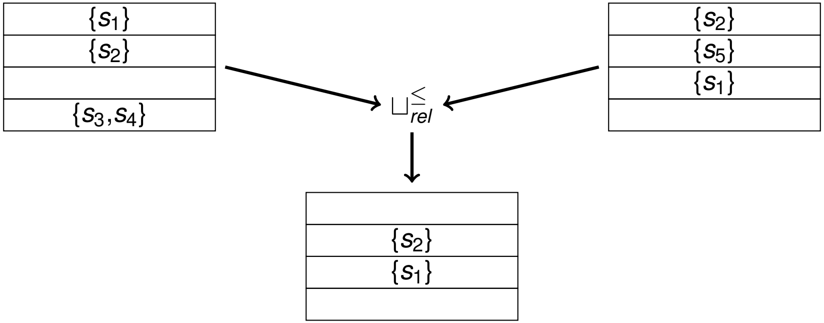
$$eab^{\leq} : (\mathcal{N} \to \mathcal{AB}^{\leq}) \times (\mathcal{N} \times \mathcal{N} \to \mathcal{R}) \to (\mathcal{N} \to \mathcal{AB}^{\leq})$$
$$eab^{\leq}(ab, cgr_v) = \lambda s \in \mathcal{N}.\min\{ab(t) \mid t \in \mathcal{N} \wedge cgr_v(s,t) = \mathrm{sb}\}$$

$$\downarrow$$



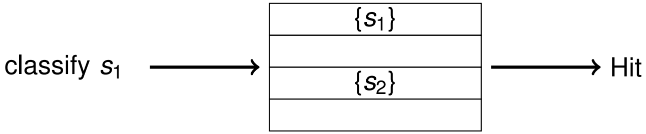
$+ \quad cgr_v(s_1, s_2) = \mathrm{sb}$

Join



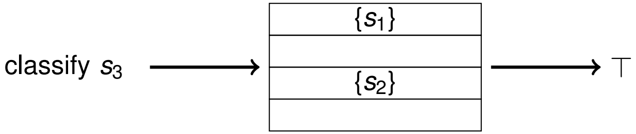
$$ab \sqcup_{rel}^{\leq} ab' = \lambda s \in \mathcal{N}.\, \max(ab(s), ab'(s))$$

Classification



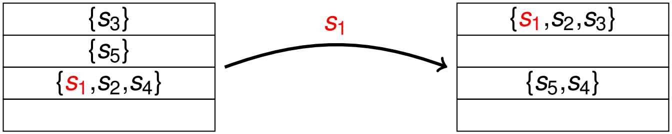
$$Class^{\leq}_{rel}(ab, s) := \begin{cases} \mathrm{H} & : ab(s) < \infty \\ \top & : \text{otherwise} \end{cases}$$

Classification



$$Class^{\leq}_{rel}(ab, s) := \begin{cases} \mathrm{H} & : ab(s) < \infty \\ \top & : \text{otherwise} \end{cases}$$

Update

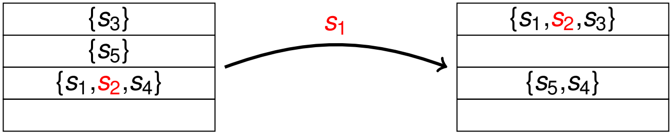


$$cgr_v(s_1, s_1) = \mathrm{sb}$$

$$U^{\leq}_{rel}(ab, s) := \lambda t. \begin{cases} 0 & : srel = \mathrm{sb} \\ ab(t) & : srel \in \{\mathrm{ds}, \overline{\mathrm{ss\,db}}\} \\ ab(t) & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) \leq ab(t) \\ ab(t) + 1 & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) < k - 1 \\ \infty & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) \geq k - 1 \end{cases}$$

where $srel = cgr_v(s, t)$.

Update

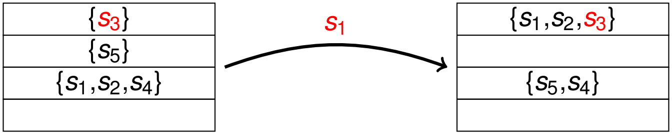


$$cgr_v(s_1, s_2) = \mathrm{sb}$$

$$U^{\leq}_{rel}(ab, s) := \lambda t. \begin{cases} 0 & : srel = \mathrm{sb} \\ ab(t) & : srel \in \{\mathrm{ds}, \overline{\mathrm{ss\,db}}\} \\ ab(t) & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) \leq ab(t) \\ ab(t) + 1 & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) < k - 1 \\ \infty & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) \geq k - 1 \end{cases}$$

where $srel = cgr_v(s, t)$.

Update

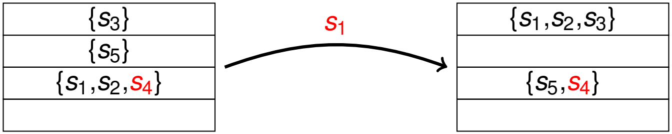


$$cgr_v(s_1, s_3) = \mathrm{ds}$$

$$U^{\leq}_{rel}(ab, s) := \lambda t. \begin{cases} 0 & : srel = \mathrm{sb} \\ ab(t) & : srel \in \{\mathrm{ds}, \overline{\mathrm{ss\,db}}\} \\ ab(t) & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) \leq ab(t) \\ ab(t) + 1 & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) < k - 1 \\ \infty & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) \geq k - 1 \end{cases}$$

where $srel = cgr_v(s, t)$.

Update

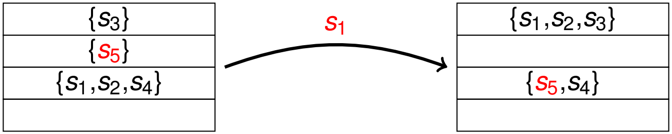


$$cgr_v(s_1, s_4) = \mathrm{ss\,db}$$

$$U_{rel}^{\leq}(ab, s) := \lambda t. \begin{cases} 0 & : srel = \mathrm{sb} \\ ab(t) & : srel \in \{\mathrm{ds}, \overline{\mathrm{ss\,db}}\} \\ ab(t) & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) \leq ab(t) \\ ab(t) + 1 & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) < k - 1 \\ \infty & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) \geq k - 1 \end{cases}$$

where $srel = cgr_v(s, t)$.

Update



$$cgr_v(s_1, s_5) = \mathrm{ss}$$

$$U_{rel}^{\leq}(ab, s) := \lambda t. \begin{cases} 0 & : srel = \mathrm{sb} \\ ab(t) & : srel \in \{\mathrm{ds}, \overline{\mathrm{ss\,db}}\} \\ ab(t) & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) \leq ab(t) \\ ab(t) + 1 & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) < k - 1 \\ \infty & : srel \sqsupseteq_{\mathcal{R}} \mathrm{ss\,db} \wedge ab(s) > ab(t) \wedge ab(t) \geq k - 1 \end{cases}$$

where $srel = cgr_v(s, t)$.