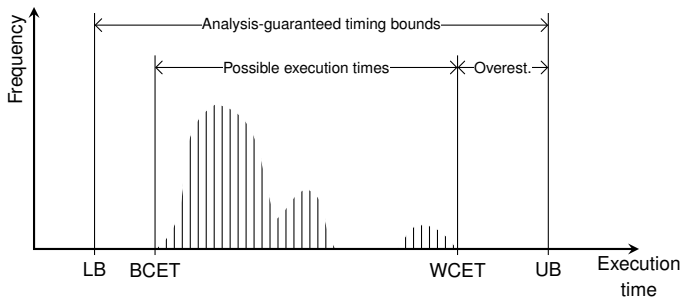


Relational Cache Analysis for Static Timing Analysis

Sebastian Hahn Daniel Grund

ECRTS 2012





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

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

Must-cache analysis

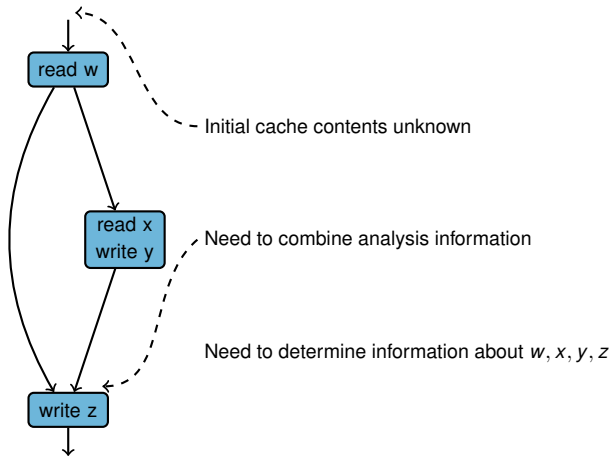
- ▶ under-approximation
- ▶ classify hits

May-cache analysis

- ▶ over-approximation
- ▶ classify misses

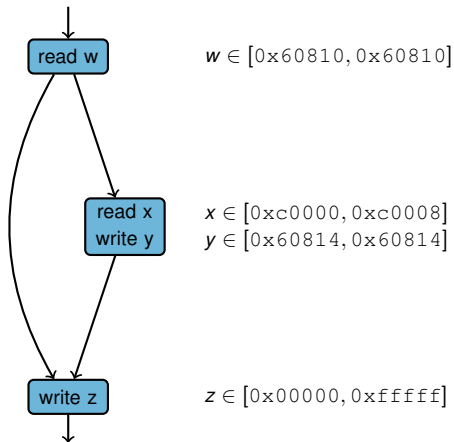
Static Cache Analysis

Challenges



Static Cache Analysis

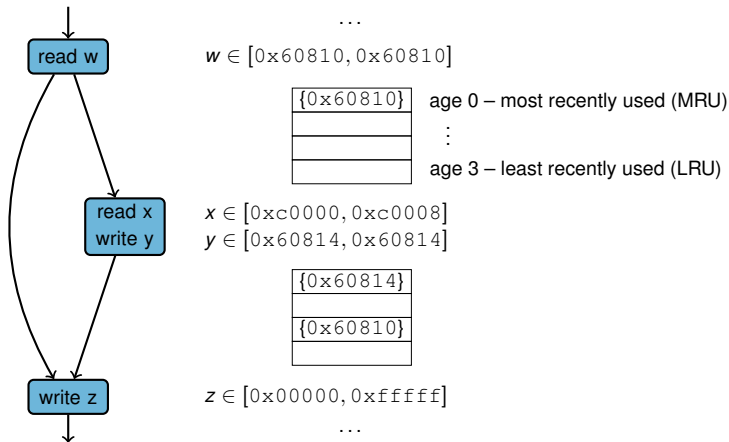
Two-step Approach



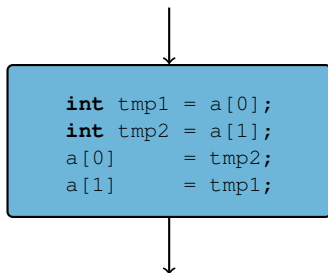
1 Approximate accessed addresses by Value Analysis

Static Cache Analysis for LRU replacement policy

Two-step Approach



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



1. Address Information

$a[0], a[1] \in [0x00000, 0xfffff]$

2. Cache Information

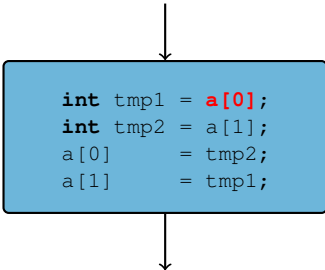
{0x60810}

1. Address Information

$a[0], a[1] \in [0x00000, 0xfffff]$

2. Cache Information

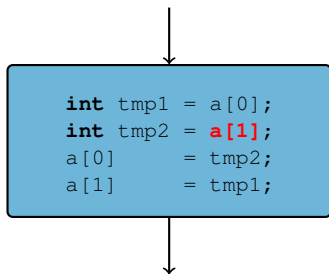
{0x60810}



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

Intangibility of Memory Blocks

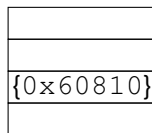
⇒ not guaranteed to be cached



1. Address Information

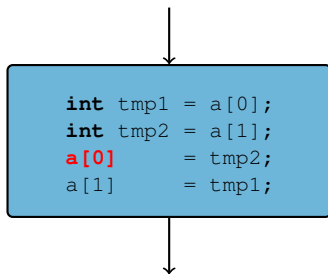
$a[0], a[1] \in [0x00000, 0xfffff]$

2. Cache Information



Intangibility of Memory Blocks

⇒ not guaranteed to be cached



1. Address Information

$a[0], a[1] \in [0x00000, 0xfffff]$

2. Cache Information

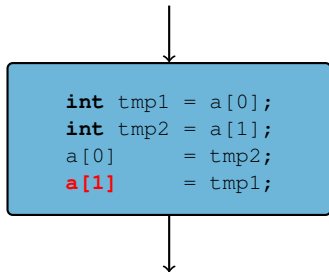
{0x60810}

Intangibility of Memory Blocks

⇒ not guaranteed to be cached

Excessive Information Loss

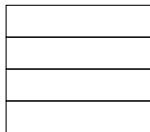
Example



1. Address Information

$a[0], a[1] \in [0x00000, 0xfffff]$

2. Cache Information

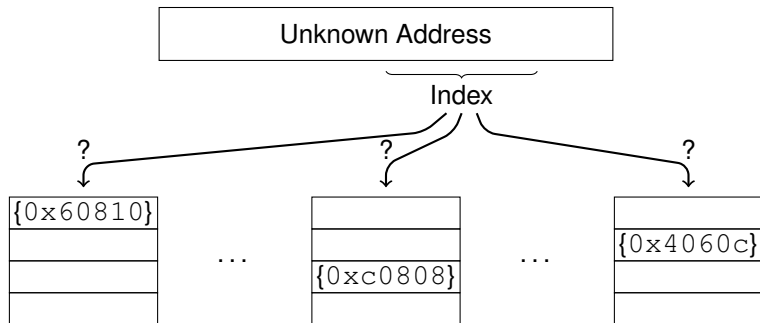


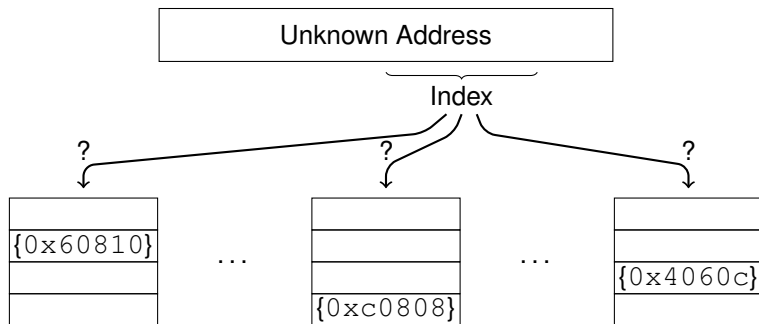
Intangibility of Memory Blocks

⇒ not guaranteed to be cached

Excessive Information Loss

⇒ not guaranteed to be cached anymore





Multiple Aging

⇒ any cache set might be affected

Precisely determined addresses
are not necessary for
precise cache analysis.

Precisely determined addresses
are not necessary for
precise cache analysis.

But relations between addresses.

1 Introduction and Problem

2 Relational Cache Analysis

- Symbolic Names
- Relational Framework
- Relations and Congruence Information
- Cache Analysis

3 Implementation and Evaluation

Definition (Symbolic Name)

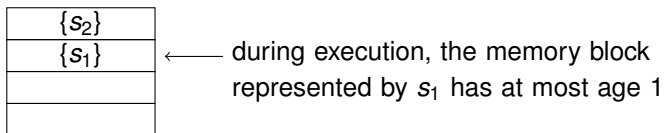
Unique identifier for an occurrence of an address expression

After Compilation

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

```
...  
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  
...
```

- symbolic names as abstract cache elements



- symbolic names abstract from concrete addresses
→ all memory references tangible

Relations between Symbolic Names

```

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

```

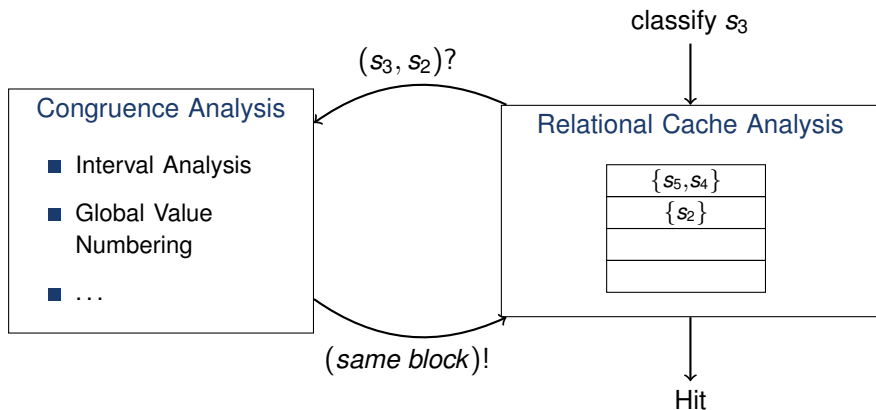
{s ₂ }
{s ₁ }

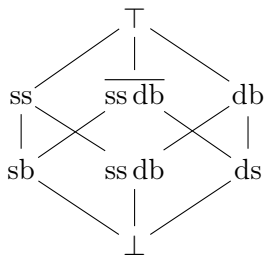
deref. s₃ →

Classify reference s₃ as hit?
How are s₁ and s₂ affected?

Use of relational information

- s₃ and s₁ denote the **same memory block** → classify reference as hit
- s₃ and s₂ map to **different cache sets** → s₂ not affected (e.g. no aging)



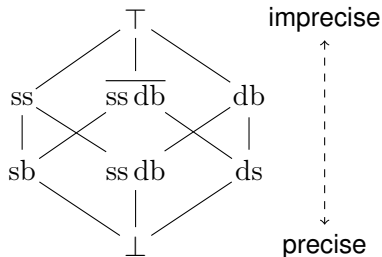


Relation	Meaning
ss	same cache set
ds	different cache set
sb	same block
db	different block
ss db	ss and db
<u>ss db</u>	ds or sb

sb classify hits

ss db account for cache conflicts

ds exclude possible eviction



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

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

sb classify hits

ss db account for cache conflicts

ds exclude possible eviction

Congruence Information

Partial Execution Trace τ

$\langle s_1 \mapsto 0x60810 \rangle$
 $\circ \langle s_1 \rangle$
 $\circ \langle s_2 \mapsto 0xbffc0 \rangle$
 $\circ \langle s_2 \rangle$
 $\circ \langle s_3 \mapsto 0xbffc4 \rangle$
 $\circ \langle s_3 \rangle$

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

Congruence Information

Partial Execution Trace τ

$\langle s_1 \mapsto 0x60810 \rangle$
 $\circ \langle s_1 \rangle$
 $\circ \langle s_2 \mapsto 0xbffc0 \rangle$
 $\circ \langle s_2 \rangle$
 $\circ \langle s_3 \mapsto 0xbffc4 \rangle$
 $\circ \langle s_3 \rangle$

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

Partial Execution Trace τ'

$\langle s_1 \mapsto 0x60810 \rangle$
 $\circ \langle s_1 \rangle$
 $\circ \langle s_2 \mapsto 0xbffc4 \rangle$
 $\circ \langle s_2 \rangle$
 $\circ \langle s_3 \mapsto 0xbffc8 \rangle$
 $\circ \langle s_3 \rangle$

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

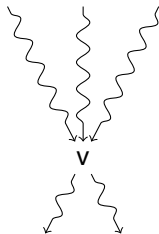
→ congruence information has to safely account for both cases

Congruence Information

Congruence information modelled as one function

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

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 $s, t \in \mathcal{N}$

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

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

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

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

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

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

- 1 Introduction and Problem
- 2 Relational Cache Analysis
 - Symbolic Names
 - Relational Framework
 - Relations and Congruence Information
 - Cache Analysis
- 3 Implementation and Evaluation

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++;
  }
}

```

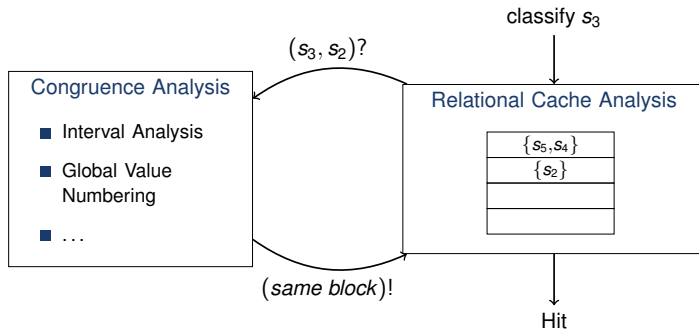
Configuration			references classified as always hit	
line size	assoc.	sets	State-of-the-art	Relational
4	4	4	11	15
4	8	4	26	37
8	4	4	36	42
8	8	4	54	67
16	4	4	56	67
16	8	4	73	84

Qualitative Result

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

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	
<i>ls</i>	<i>k</i>	<i>n</i>	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	
<i>ls</i>	<i>k</i>	<i>n</i>	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	
<i>l</i> s	<i>k</i>	<i>n</i>	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

Partial Execution Trace τ

$$\langle s_1 \mapsto 0x60810 \rangle$$
$$\circ \langle s_1 \rangle$$

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

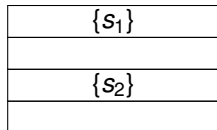
Partial Execution Trace τ'

- $\langle s_1 \mapsto 0x60810 \rangle$
- $\circ \langle s_1 \rangle$
- $\langle s_2 \mapsto 0xbffc0 \rangle$
- $\circ \langle s_2 \rangle$
- $\langle s_3 \mapsto 0xbffc4 \rangle$
- $\circ \langle s_3 \rangle$

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

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

→ congruence information depends on program location



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

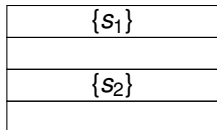
$\{s_1\}$
$\{s_2\}$

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

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

Abstract Cache Domain

Effective Age Bound — Aliasing Problem



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



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

Abstract Cache Domain

Effective Age Bound — Normalisation

$\{s_1\}$
$\{s_2\}$

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



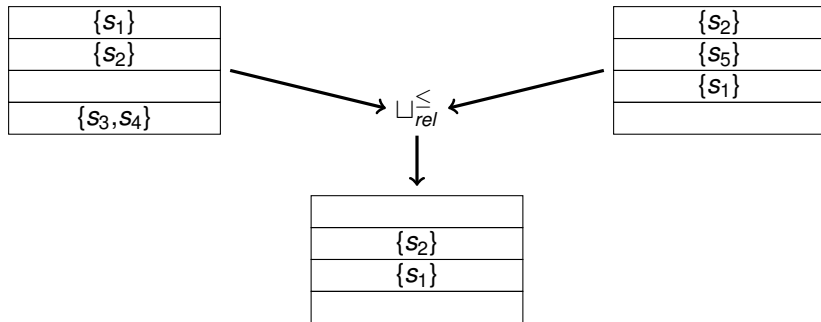
$$eab^{\leq} : (\mathcal{N} \rightarrow \mathcal{AB}^{\leq}) \times (\mathcal{N} \times \mathcal{N} \rightarrow \mathcal{R}) \rightarrow (\mathcal{N} \rightarrow \mathcal{AB}^{\leq})$$

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

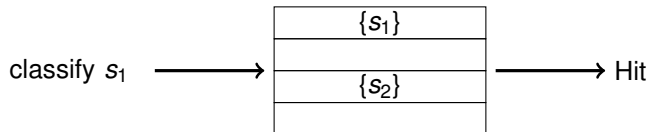


$\{s_1, s_2\}$

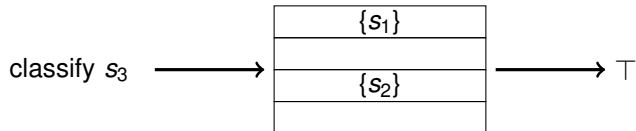
$$+ \quad cgr_v(s_1, s_2) = sb$$



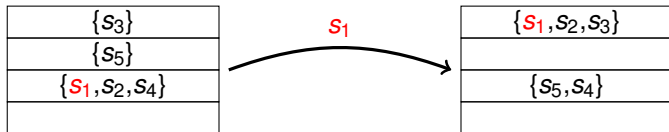
$$ab \sqcup_{rel}^{\leq} ab' = \lambda s \in \mathcal{N}. \max(ab(s), ab'(s))$$



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



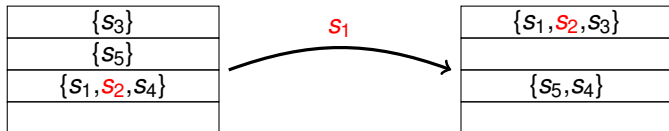
$$Class_{rel}^{\leq}(ab, s) := \begin{cases} H & : ab(s) < \infty \\ \top & : \text{otherwise} \end{cases}$$



$$cgr_v(s_1, s_1) = sb$$

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

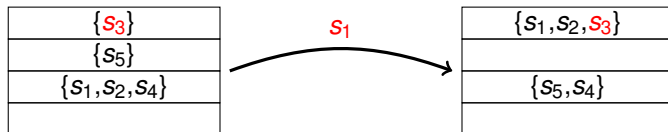
where $srel = cgr_v(s, t)$.



$$cgr_v(s_1, s_2) = sb$$

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

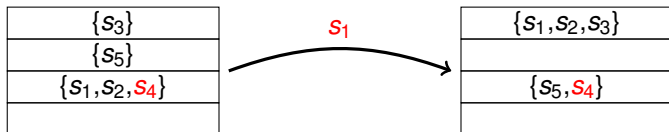
where $srel = cgr_v(s, t)$.



$$cgr_v(s_1, s_3) = ds$$

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

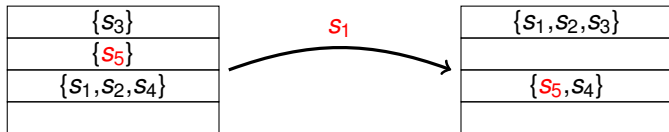
where $srel = cgr_v(s, t)$.



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

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

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



$$cgr_v(s_1, s_5) = ss$$

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

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