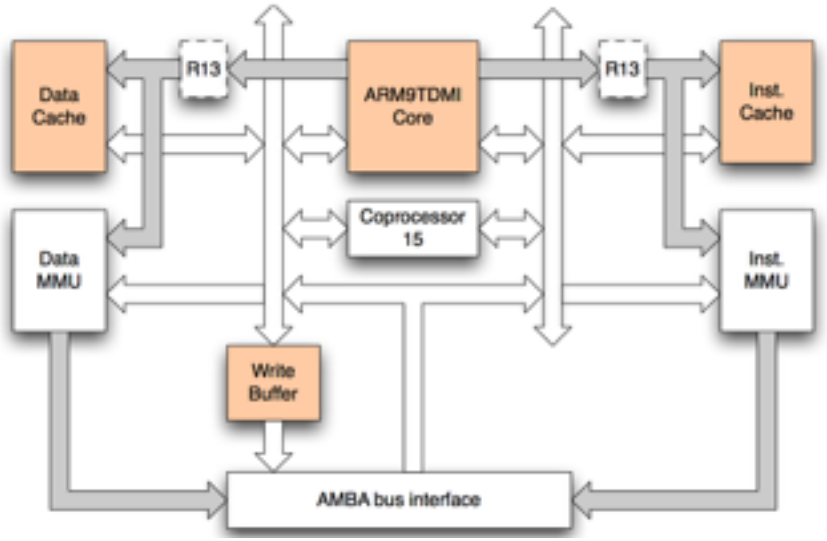
# Timed Automata for Modeling Caches and Pipelines

Franck Cassez
Macquarie University
Sydney, Australia

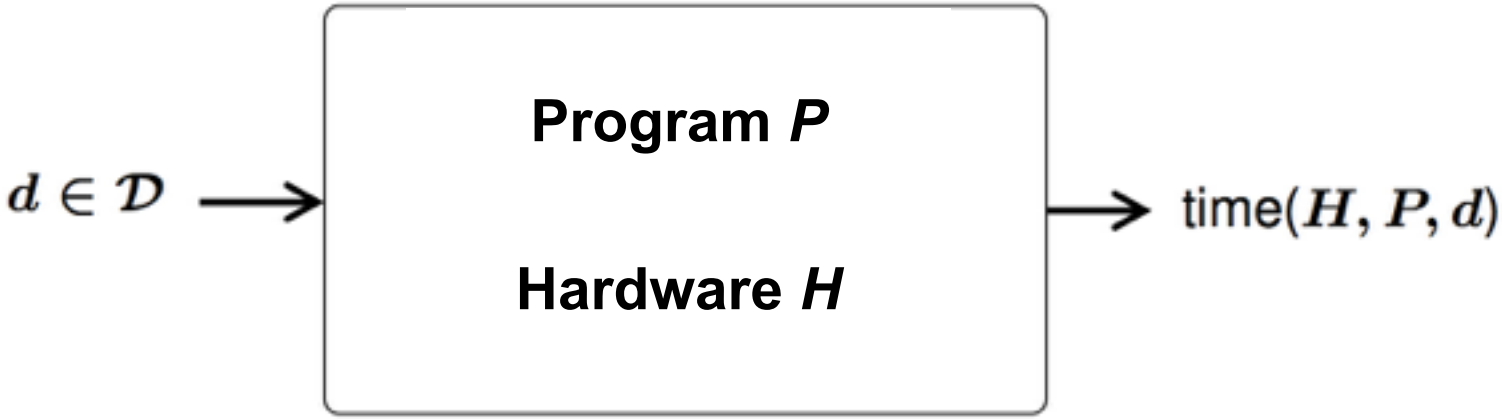
Pablo Gonzales
University of Cantabria
Santander, Spain

# Problem

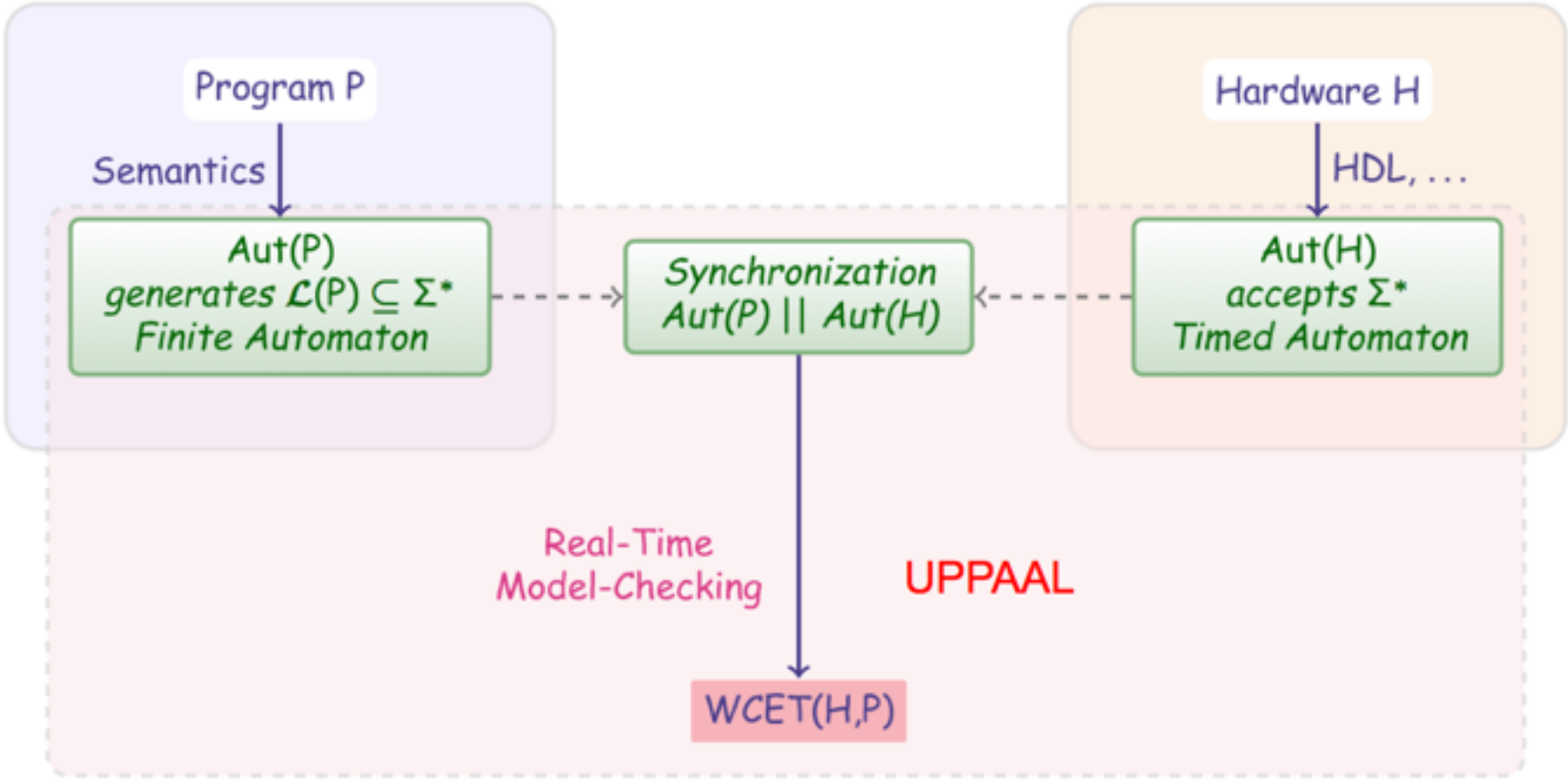
```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  dafffff0    ble   24 <fib+0x24>
```



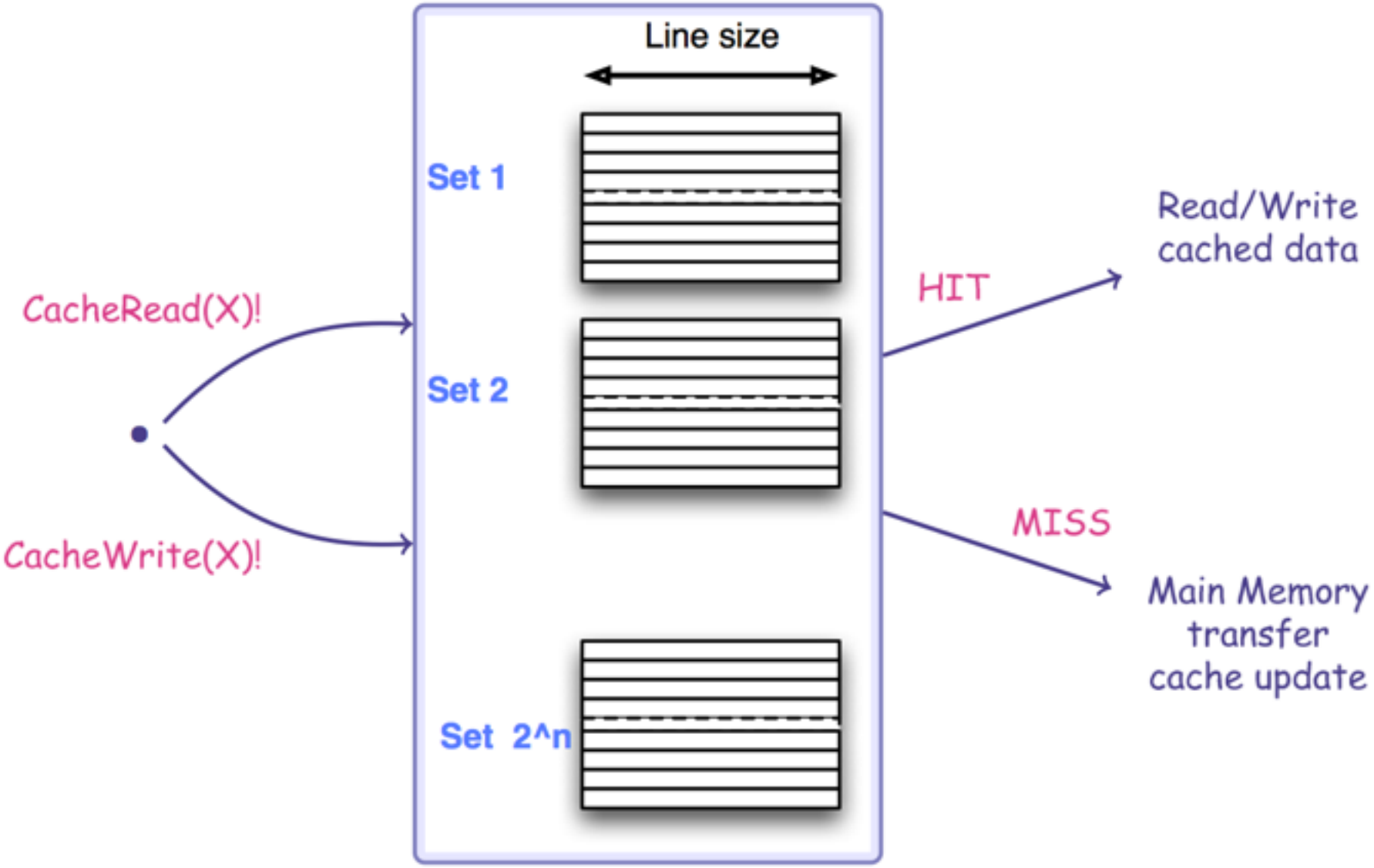

$$\text{WCET}(H, P) = \max_{d \in \mathcal{D}} \text{time}(H, P, d)$$



# Real-time model-checking

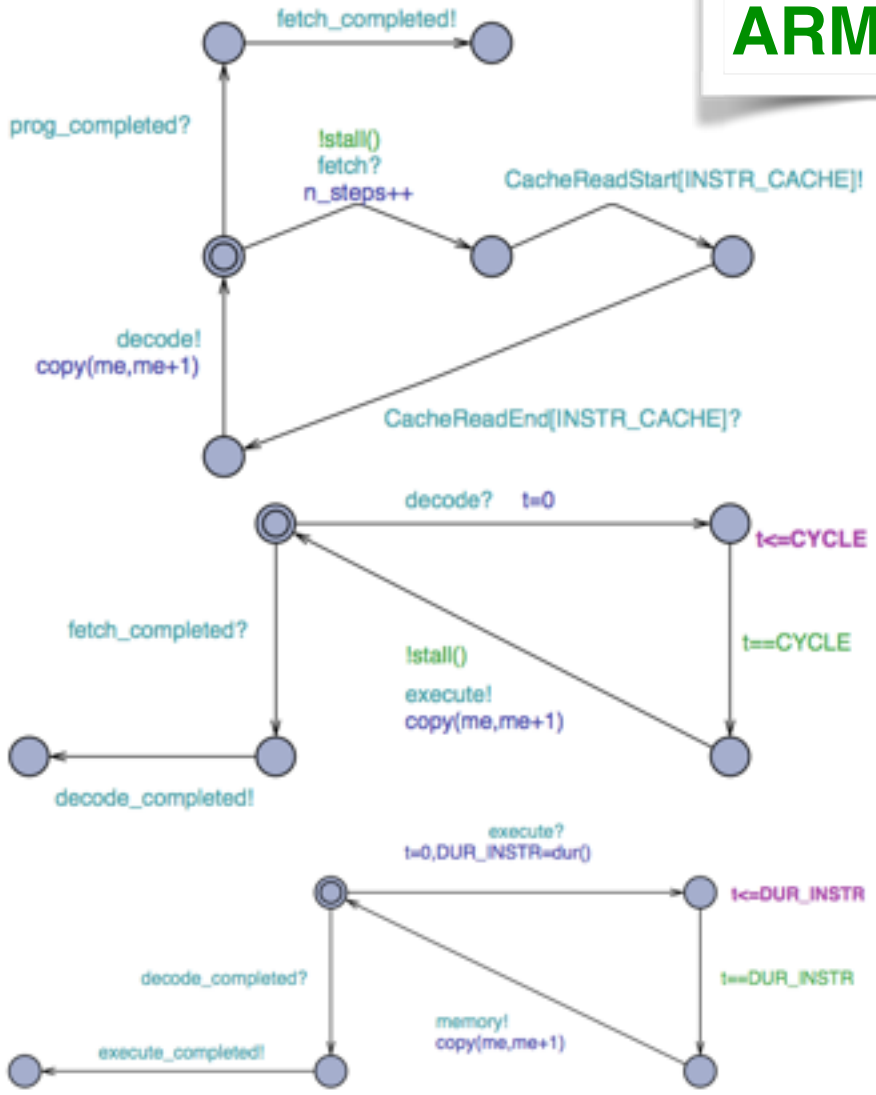


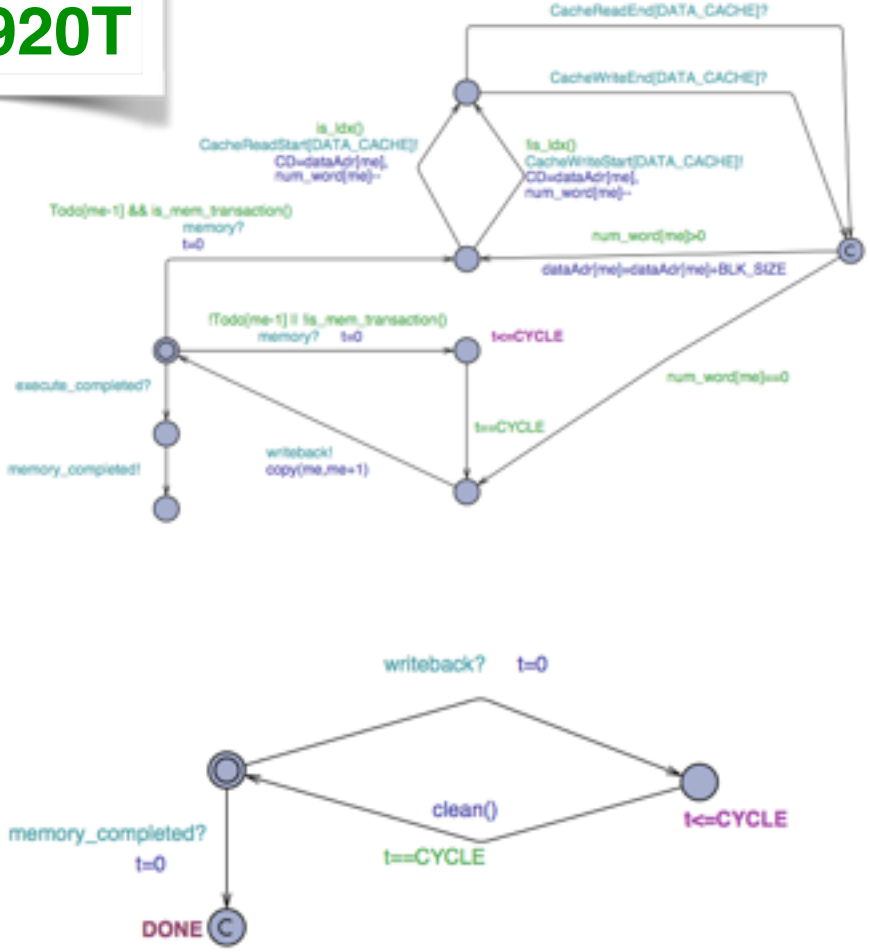
# Cache hit/miss



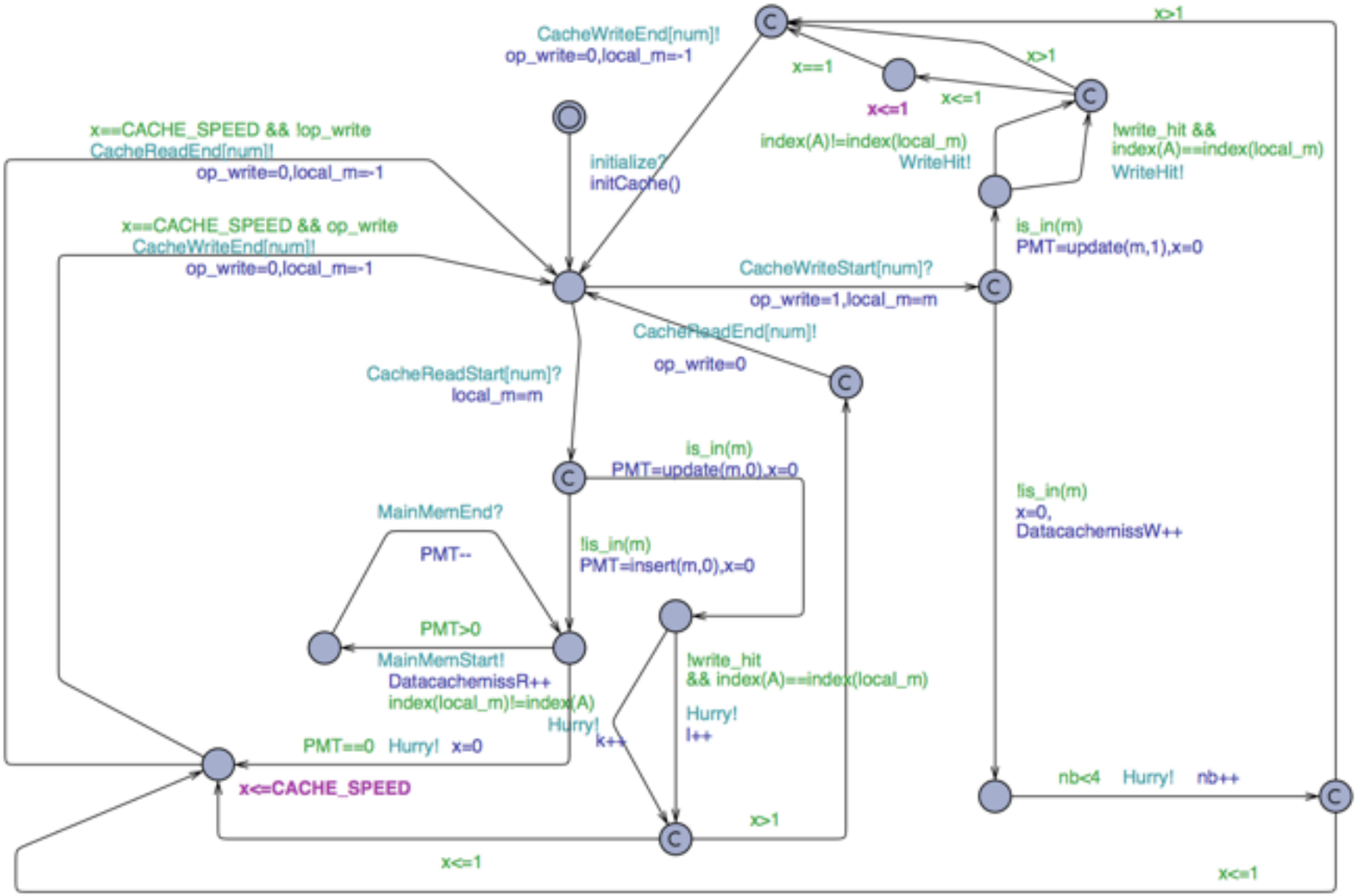
# Pipeline Model

**ARM920T**



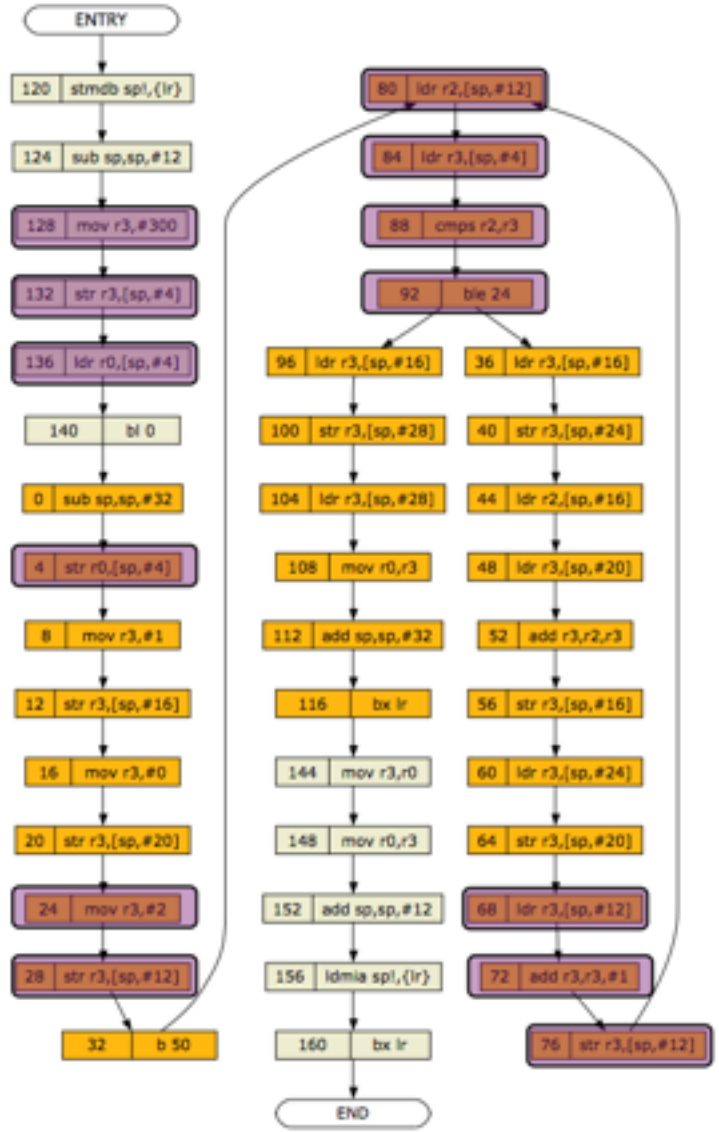
# Data cache


CacheWriteEnd[num]!
op_write=0,local_m=-1
x>1
x==1
x>1
x<=1
x<=1
x==CACHE_SPEED && !op_write
CacheReadEnd[num]!
op_write=0,local_m=-1
initialize?
initCache()
index(A)!=index(local_m)
WriteHit!
!write_hit &&
index(A)==index(local_m)
WriteHit!
x==CACHE_SPEED && op_write
CacheWriteEnd[num]!
op_write=0,local_m=-1
is_in(m)
PMT=update(m,1),x=0
CacheWriteStart[num]?
op_write=1,local_m=m
CacheReadEnd[num]!
op_write=0
CacheReadStart[num]?
local_m=m
is_in(m)
PMT=update(m,0),x=0
!is_in(m)
x=0,
DatacachemissW++
MainMemEnd?
PMT--
!is_in(m)
PMT=insert(m,0),x=0
PMT>0
MainMemStart!
DatacachemissR++
index(local_m)!=index(A)
Hurry!
k++
!write_hit
&& index(A)==index(local_m)
Hurry!
l++
PMT==0 Hurry! x=0
x<=CACHE_SPEED
nb<4 Hurry! nb++
x>1
x<=1
x<=1

# WCET-equivalent program

```
00000000 <fib>:
   0:  e24dd020     sub   sp, sp, #32
   4:  e58d0004     str   r0, [sp, #4]
   8:  e3a03001     mov   r3, #1
   c:  e58d3010     str   r3, [sp, #16]
  10:  e3a03000     mov   r3, #0
  14:  e58d3014     str   r3, [sp, #20]
  18:  e3a03002     mov   r3, #2
  1c:  e58d300c     str   r3, [sp, #12]
  20:  ea00000a     b     50 <fib+0x50>
  24:  e59d3010     ldr   r3, [sp, #16]
  28:  e58d3018     str   r3, [sp, #24]
  2c:  e59d2010     ldr   r2, [sp, #16]
  30:  e59d3014     ldr   r3, [sp, #20]
  34:  e0823003     add   r3, r2, r3
  38:  e58d3010     str   r3, [sp, #16]
  3c:  e59d3018     ldr   r3, [sp, #24]
  40:  e58d3014     str   r3, [sp, #20]
  44:  e59d300c     ldr   r3, [sp, #12]
  48:  e2833001     add   r3, r3, #1
  4c:  e58d300c     str   r3, [sp, #12]
  50:  e59d200c     ldr   r2, [sp, #12]
  54:  e59d3004     ldr   r3, [sp, #4]
  58:  e1520003     cmp   r2, r3
  5c:  dafffff0     ble   24 <fib+0x24>
  60:  e59d3010     ldr   r3, [sp, #16]
  64:  e58d301c     str   r3, [sp, #28]
  68:  e59d301c     ldr   r3, [sp, #28]
  6c:  e1a00003     mov   r0, r3
  70:  e28dd020     add   sp, sp, #32
  74:  e12fff1e     bx    lr

00000078 <main>:
  78:  e52de004     push  {lr}
  7c:  e24dd00c     sub   sp, sp, #12
  80:  e3a03f4b     mov   r3, #300
  84:  e58d3004     str   r3, [sp, #4]
  88:  e59d0004     ldr   r0, [sp, #4]
  8c:  ebffffdb     bl    0 <fib>
  90:  e1a03000     mov   r3, r0
  94:  e1a00003     mov   r0, r3
  98:  e28dd00c     add   sp, sp, #12
  9c:  e49de004     pop   {lr}
  a0:  e12fff1e     bx    lr
```



# Limitations

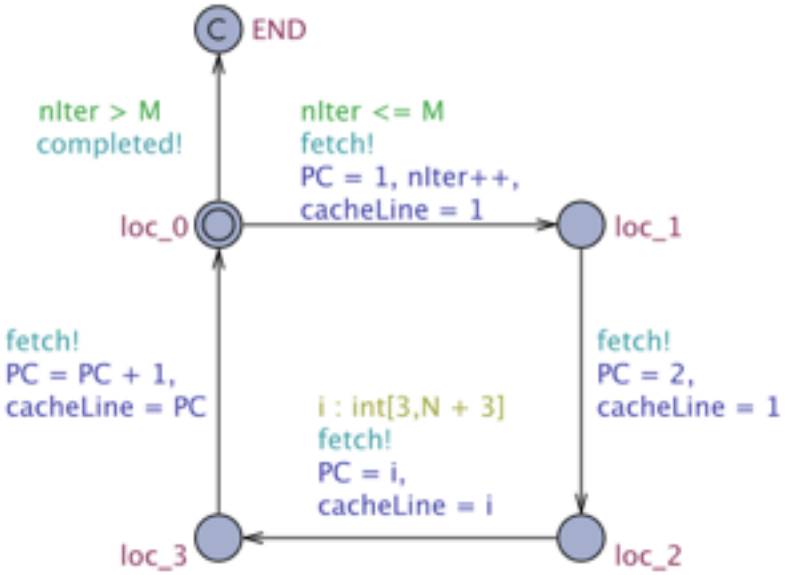
**explicit cache representation** — array: 1000+ “lines”

**fixed initial state for all caches** — empty cache

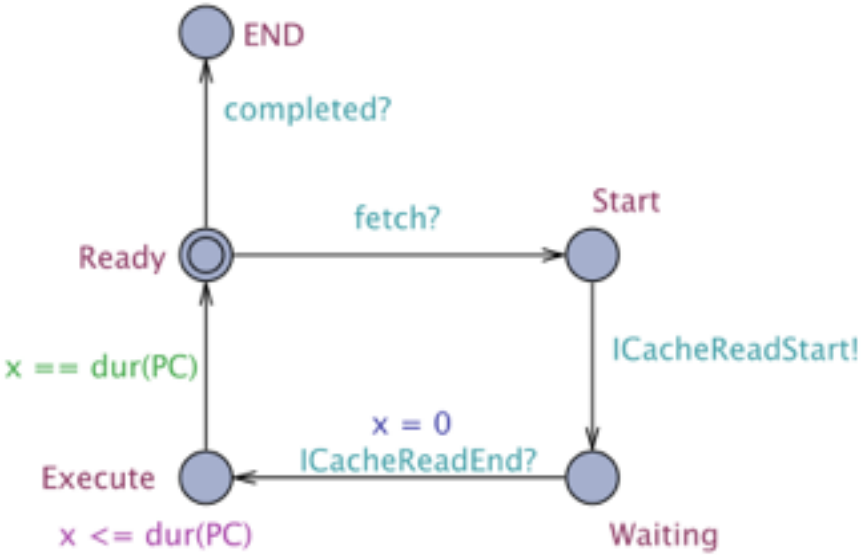
**may miss pruning in real-time model checking** — cache of size 1

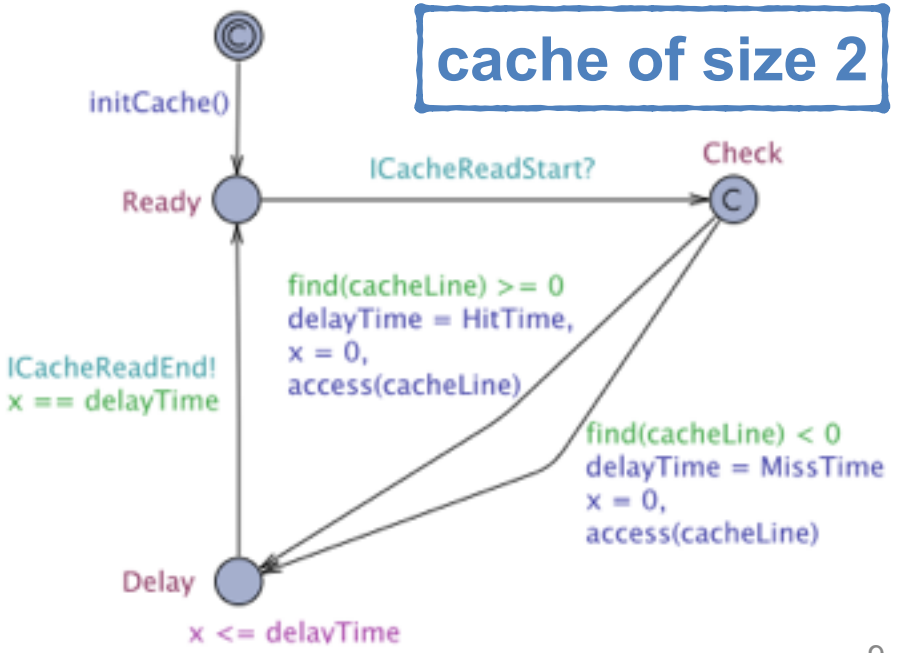

```
1   i = 0;
2   while ( i <= 10 && bool_non_det() ) {
3     ...
4     i = i + 1
5   }
6   //  second loop
7   j = 0 ;
8   while ( j <= 100 ) {
9     // do something
10    j = j + 1
11  }
```

# 2-stage pipeline & concrete cache




```
 1  nIter = 0;
 2  while ( Inter <= M ) {
 3    nIter = nIter + 1
 4    skip
 5    switch ( i ) {   // i ∈ [3, N + 3]
 6      case 3     : skip ; break
 7      case 4     : skip ; break
 8      ...
 9      case N + 3: skip ; break
10    }
11  }
```



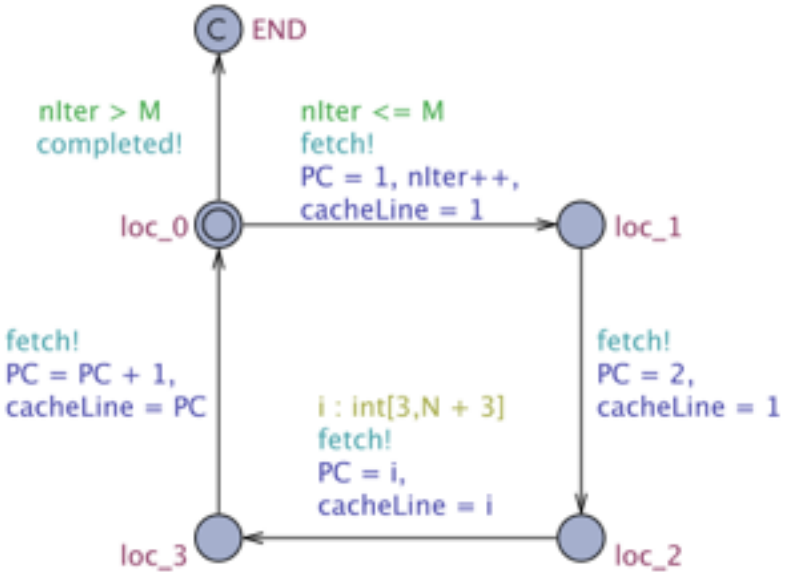
# Cache state equivalences

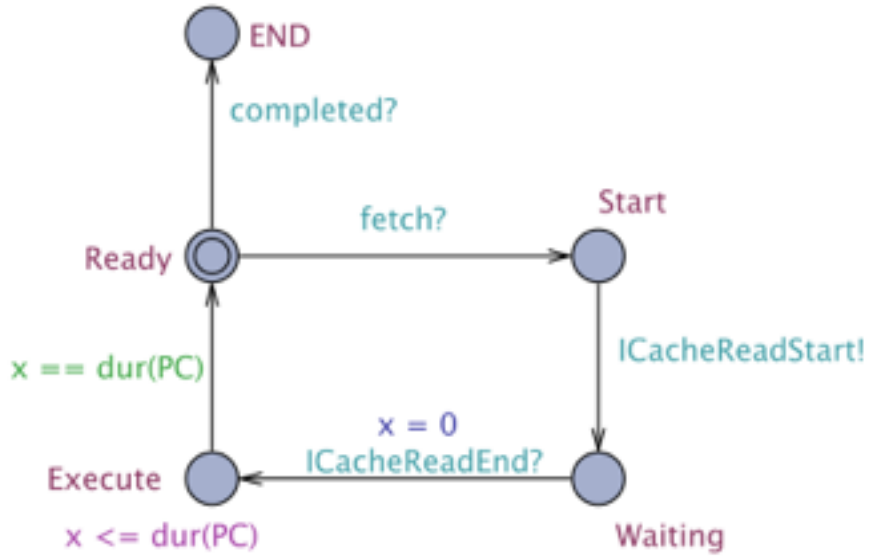
***run*** = sequence of (program state, hit/miss)

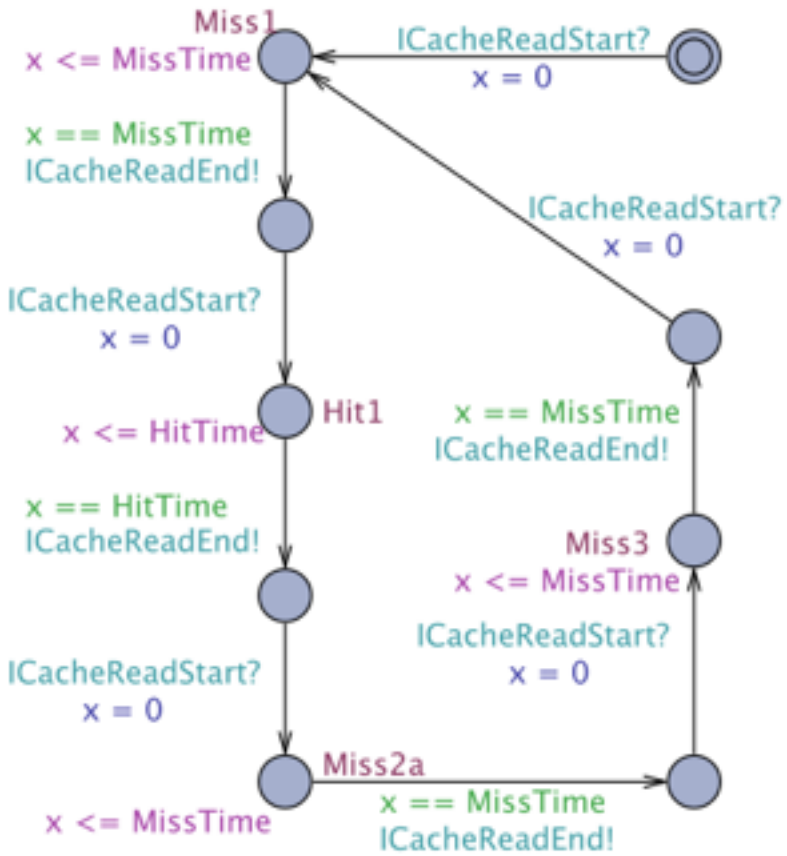
**given a program state *s*, two cache states *c* and *c'***
**are equivalent**
**iff they generate the same runs**

**solution: compute the cache state equivalence**

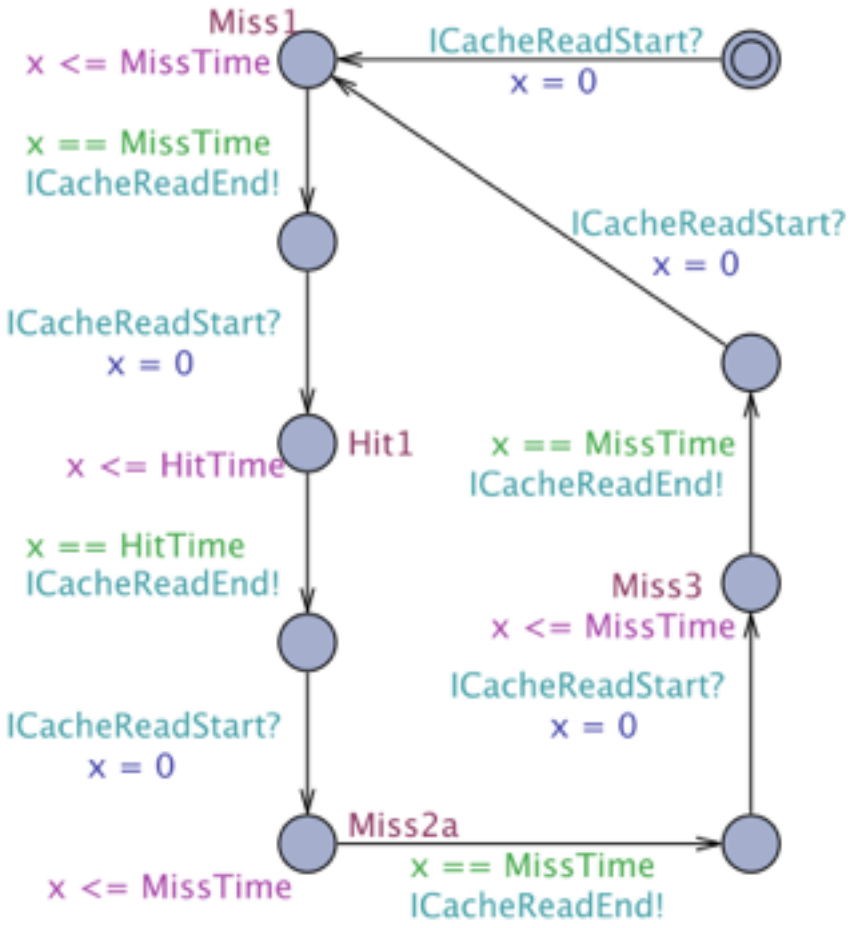
# Small cache models



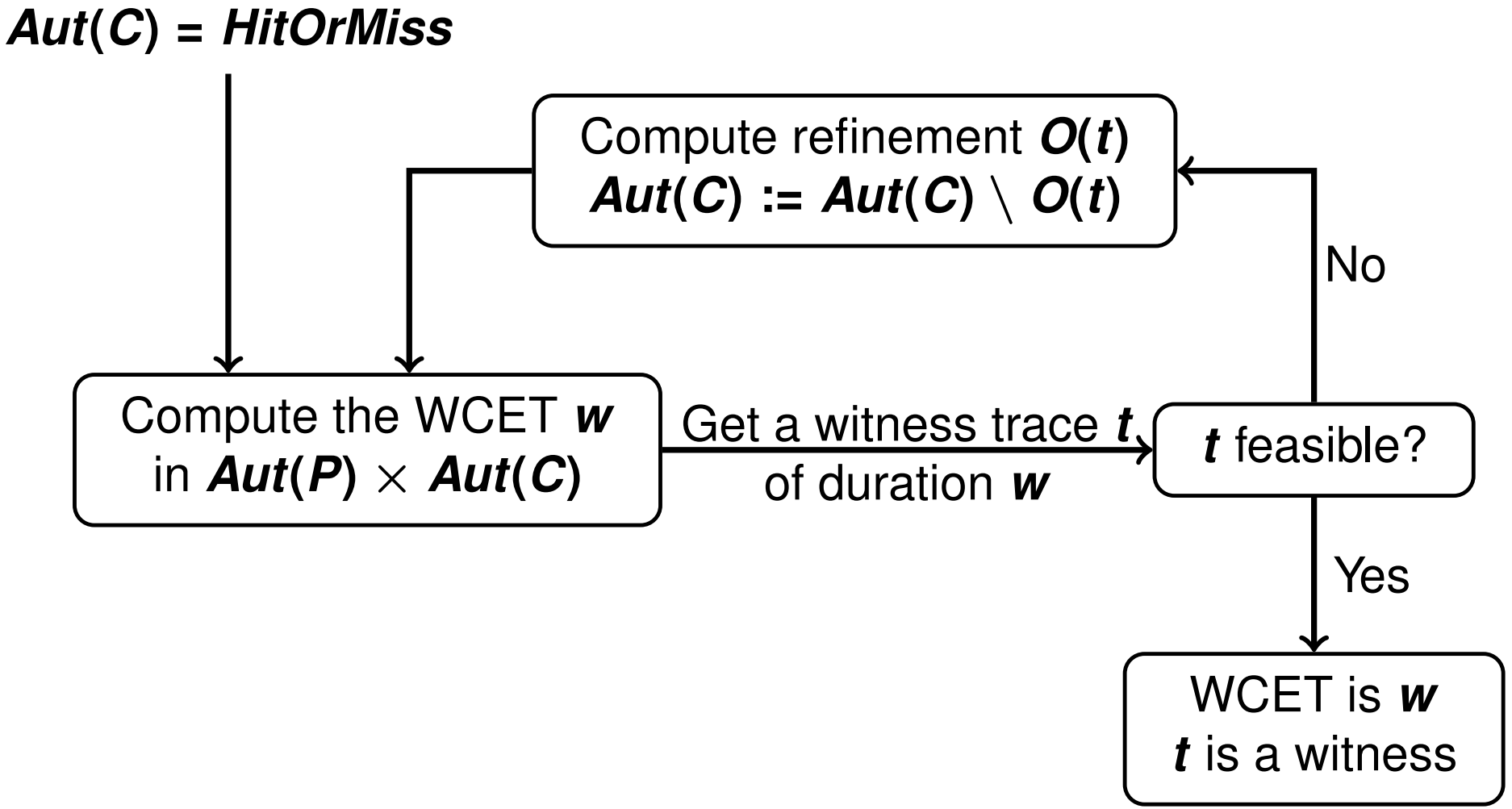



# Small cache models




| *N* | States Explored | | WCET |
|---|---|---|---|
| | Explicit Model | Small Model | |
| 1 | 549 | 147 | 396 |
| 2 | 1055 | 196 | 396 |
| 3 | 1626 | 245 | 396 |
| 4 | 2267 | 294 | 396 |
| 5 | 2953 | 343 | 396 |
| 6 | 3699 | 392 | 396 |
| 7 | 4505 | 441 | 396 |
| 8 | 5371 | 490 | 396 |
| 9 | 6297 | 539 | 396 |
| 10 | 7283 | 588 | 396 |

# Trace abstraction refinement



# Conclusion & ongoing work

## Advantages

- no assumption on initial state of the cache
- reduced state space in real-time model-checking

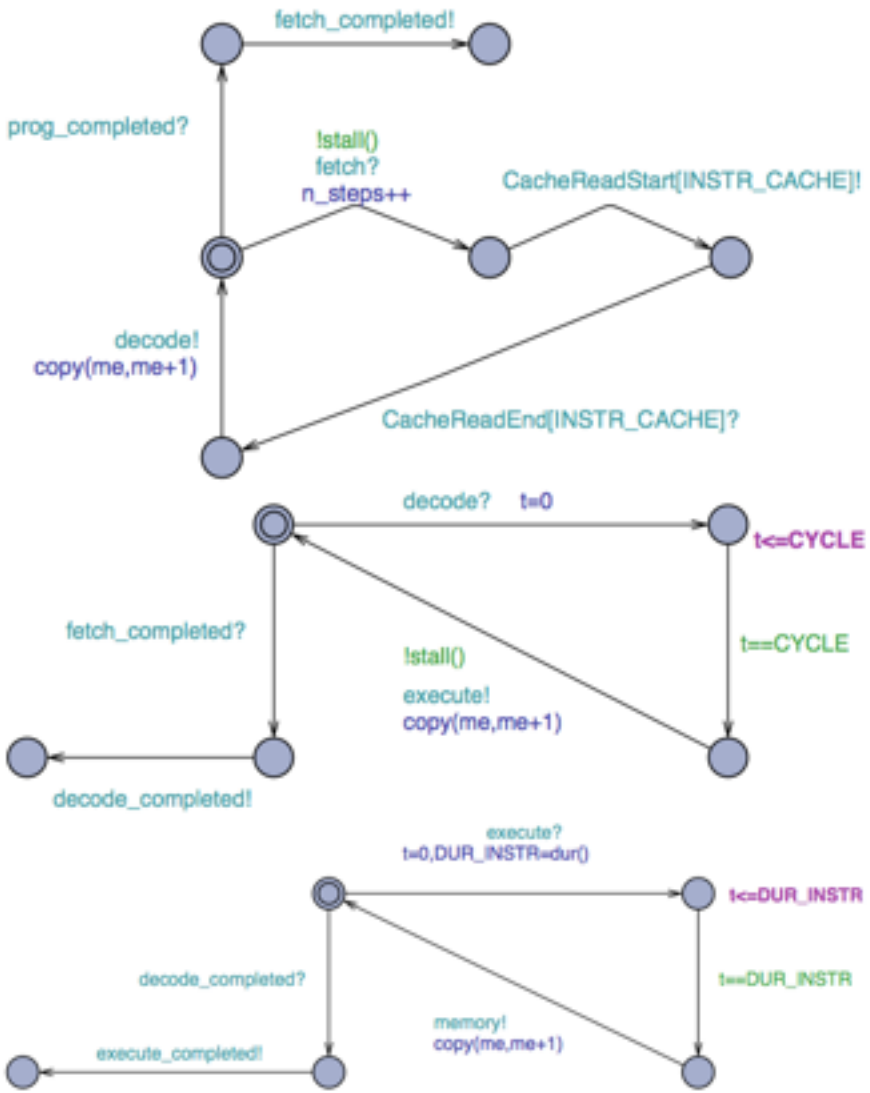
## Ongoing

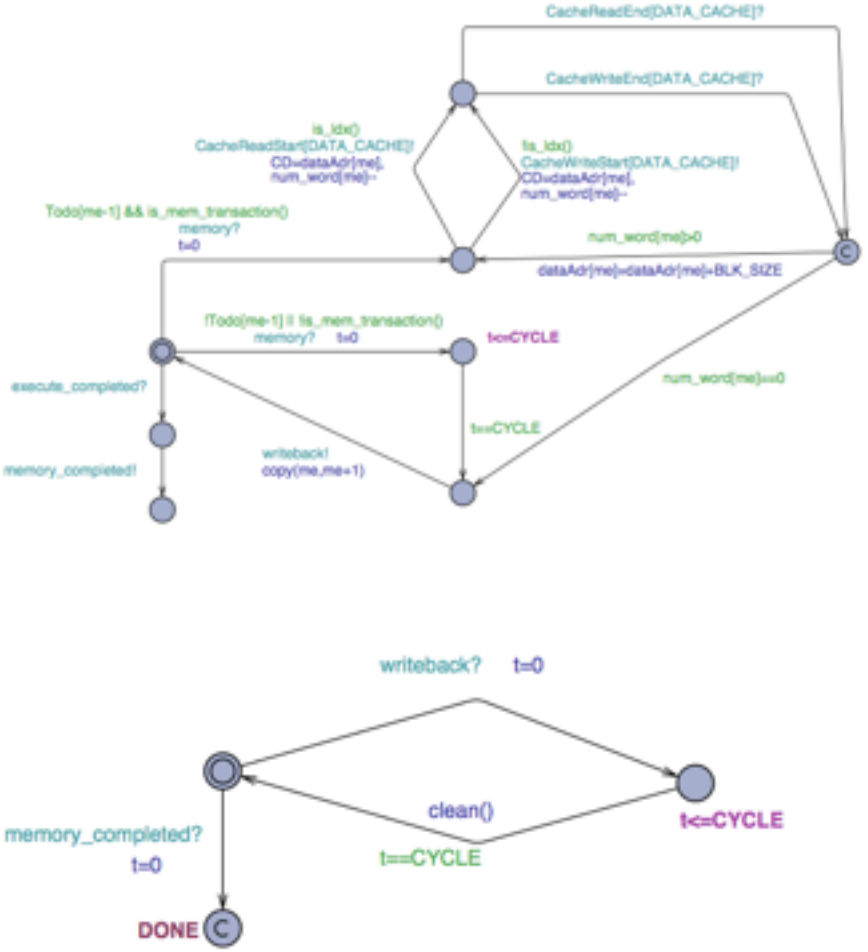
- implement and test on Malärdalen Univ. benchmarks
- extend technique to data cache

# UPPAAL models

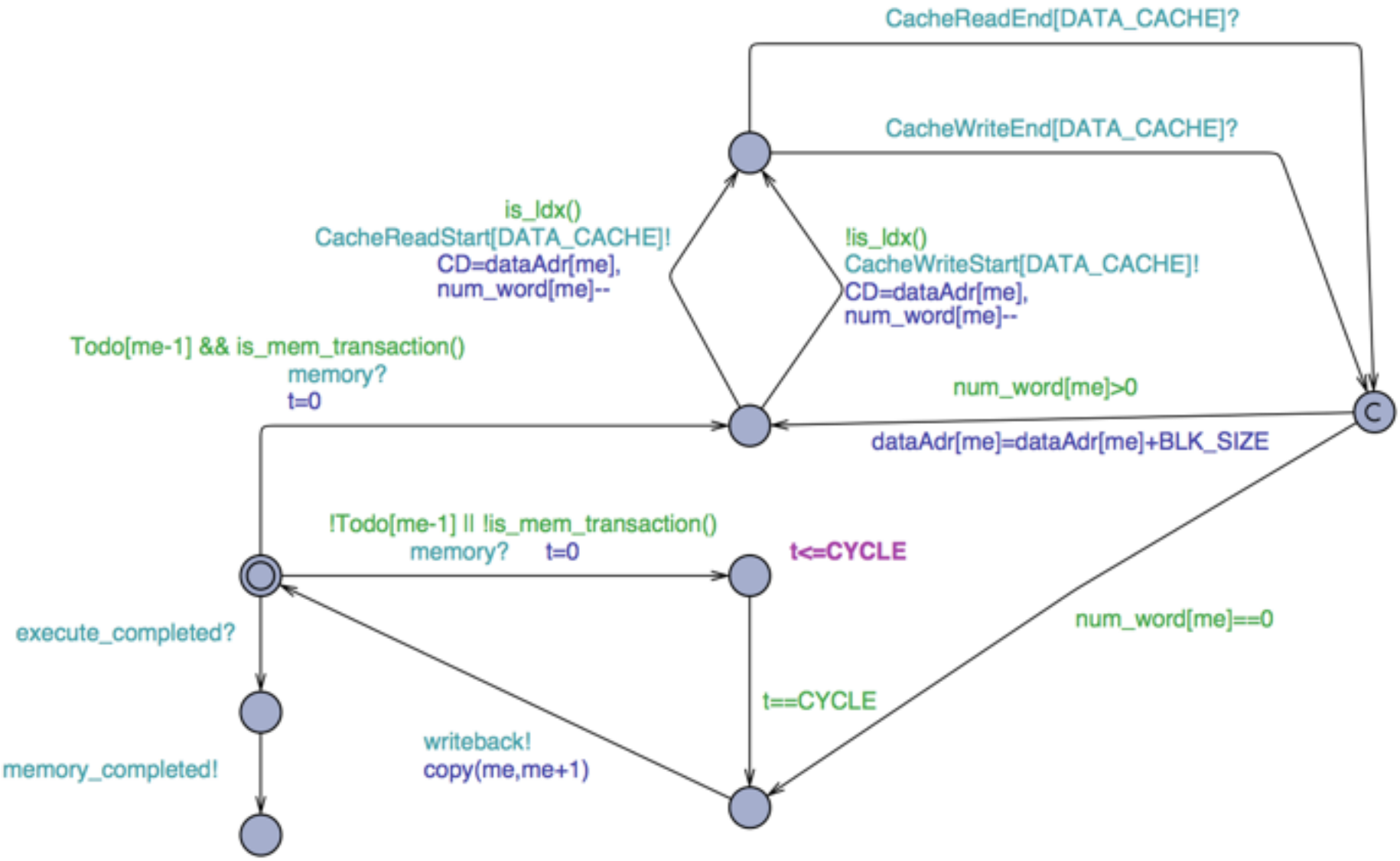
ACSD 2013

# Pipeline Model

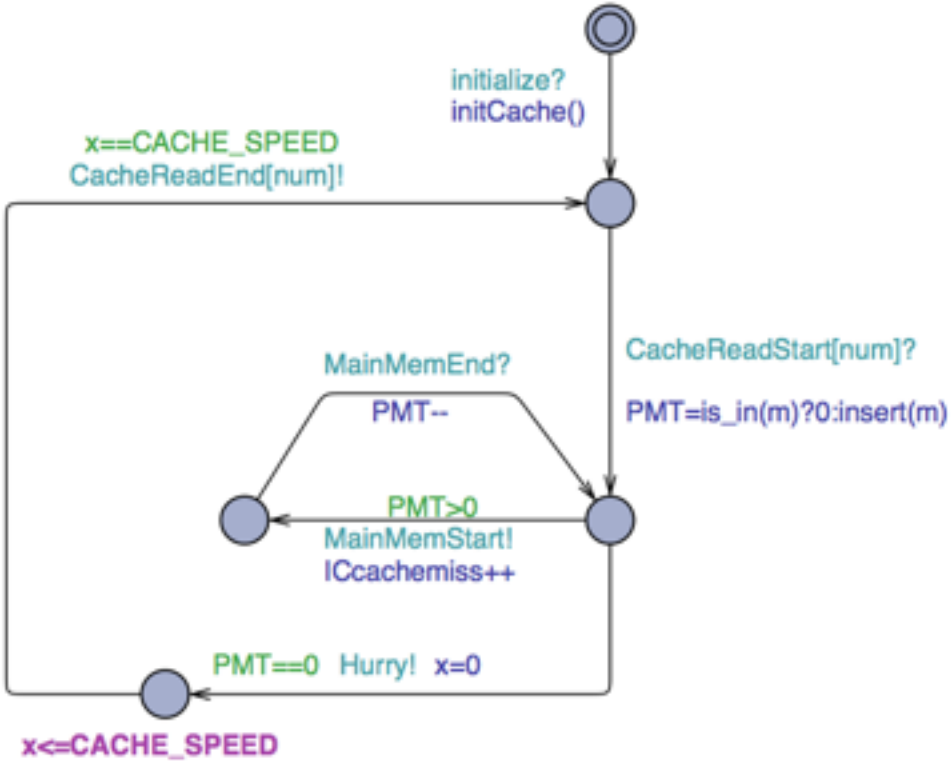


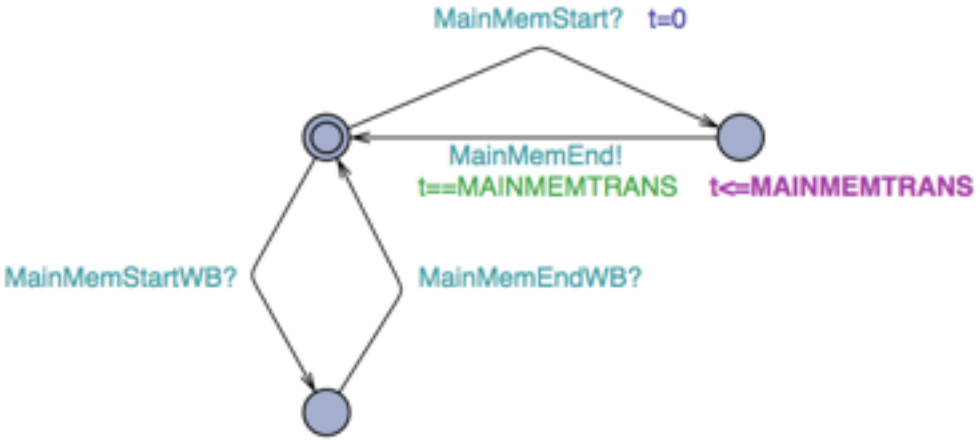



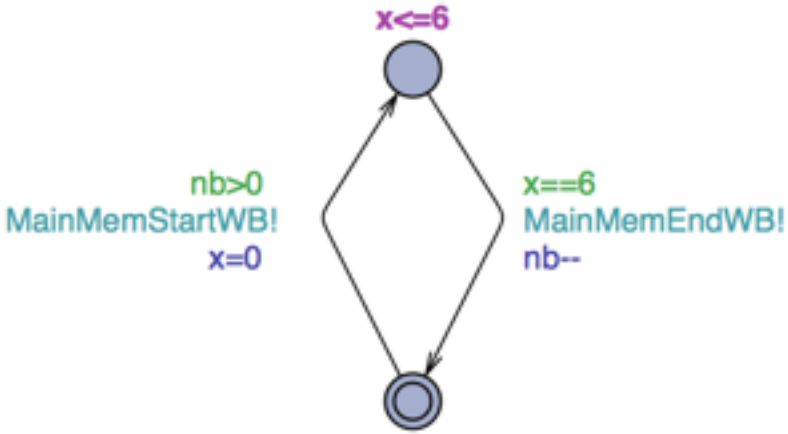
# Memory stage



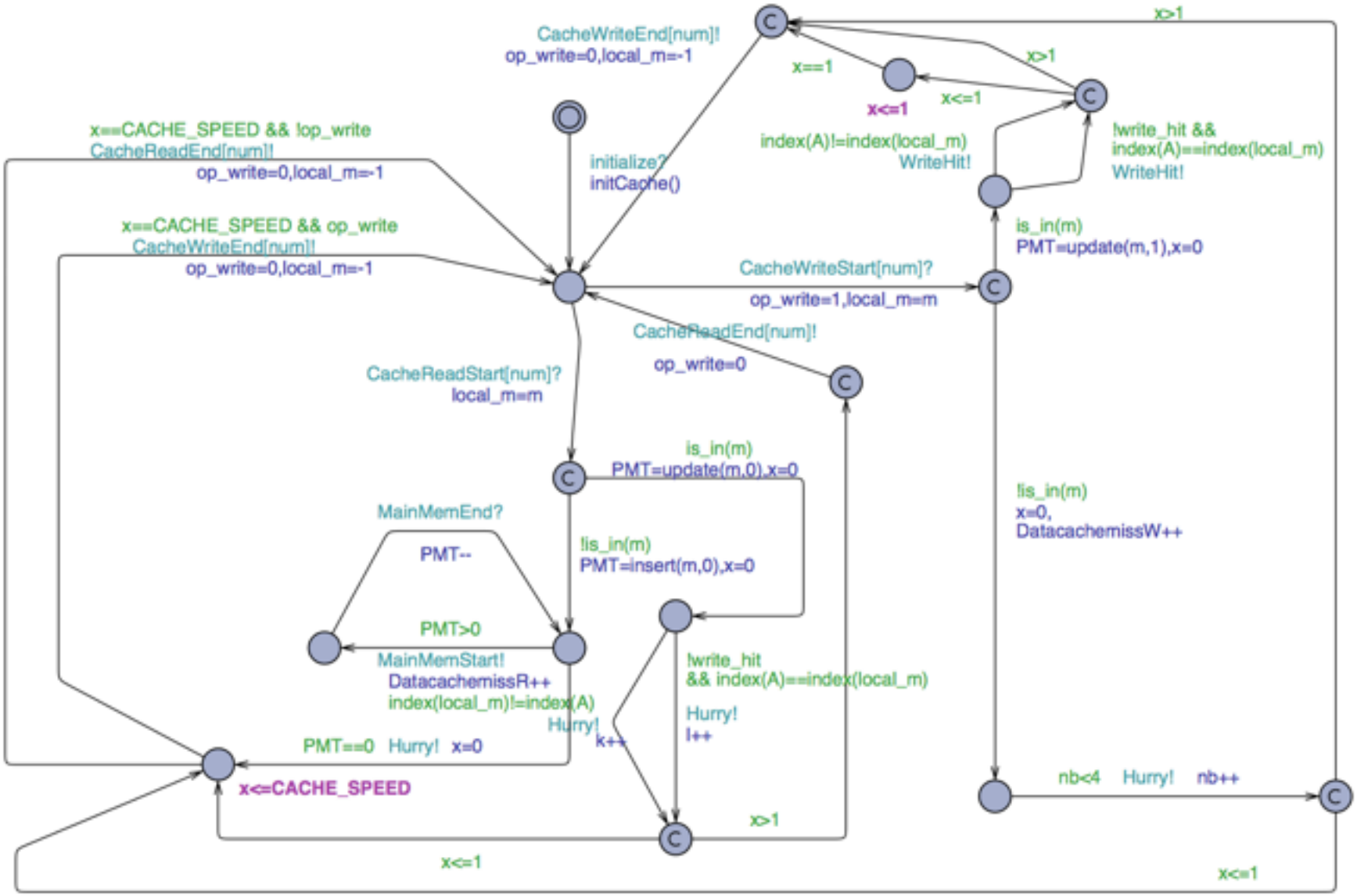
# Instruction cache and RAM



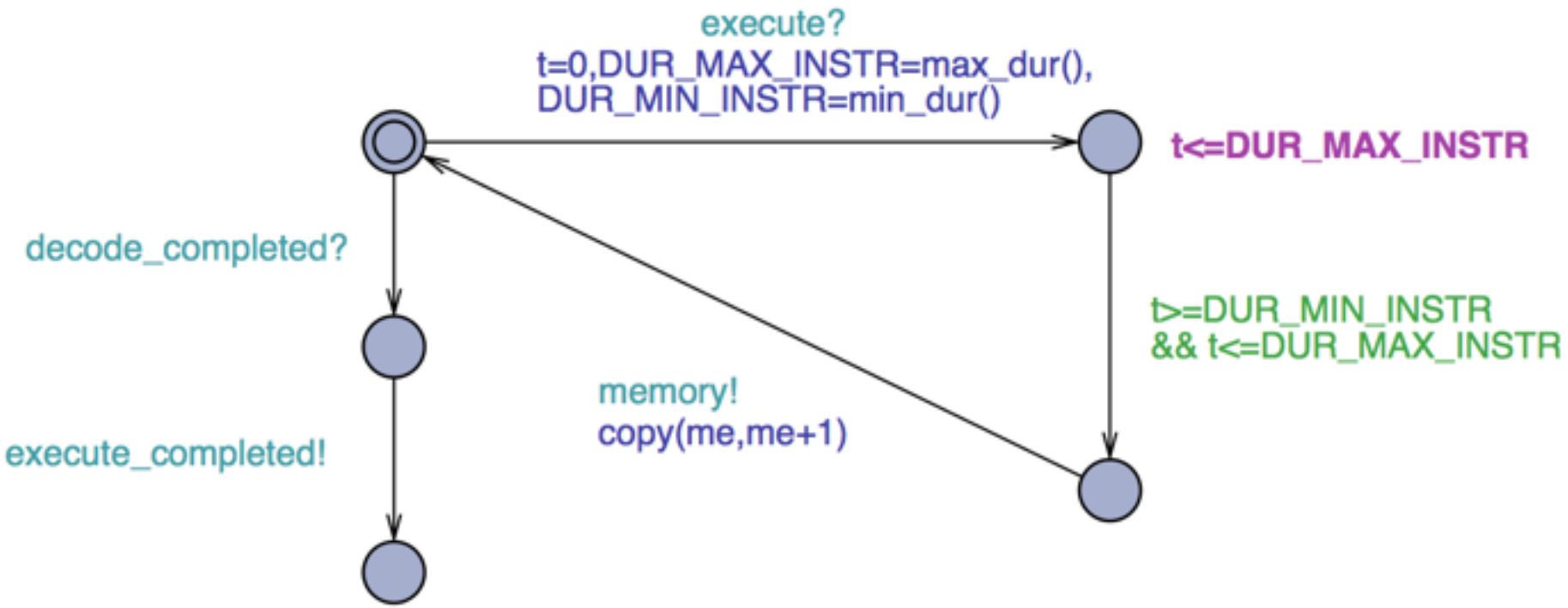



# Data cache


CacheWriteEnd[num]!
op_write=0,local_m=-1
x>1
x==1
x>1
x<=1
x<=1
x==CACHE_SPEED && !op_write
CacheReadEnd[num]!
op_write=0,local_m=-1
initialize?
initCache()
index(A)!=index(local_m)
WriteHit!
!write_hit &&
index(A)==index(local_m)
WriteHit!
x==CACHE_SPEED && op_write
CacheWriteEnd[num]!
op_write=0,local_m=-1
is_in(m)
PMT=update(m,1),x=0
CacheWriteStart[num]?
op_write=1,local_m=m
CacheReadEnd[num]!
op_write=0
CacheReadStart[num]?
local_m=m
is_in(m)
PMT=update(m,0),x=0
!is_in(m)
x=0,
DatacachemissW++
MainMemEnd?
PMT--
!is_in(m)
PMT=insert(m,0),x=0
PMT>0
MainMemStart!
DatacachemissR++
index(local_m)!=index(A)
Hurry!
k++
!write_hit
&& index(A)==index(local_m)
Hurry!
l++
PMT==0 Hurry! x=0
x<=CACHE_SPEED
nb<4 Hurry! nb++
x>1
x<=1
x<=1

# Execution stage (variable execution time)



# Experimental results (ACSD 2013)

| Program[⊕] | loc[†] | UPPAAL Time/States Explored[¶] | Computed BCET/WCET (C) | Measured BCET/WCET (M) | Error (%)[‡] | Slice[§] |
|---|---|---|---|---|---|---|
| **Single-Path Programs** | | | | | | |
| fib-O0 | 74 | 2s/74181 | 8098 | 8064 | **0.42%** | 47/131 |
| fib-O1 | 74 | 0.6s/22333 | 2597 | 2544 | **2.0%** | 18/72 |
| fib-O2 | 74 | 0.3s/9711 | 1209 | 1164 | **3.8%** | 22/71 |
| janne-complex-O0* | 65 | 1.7s/38038 | 4264 | 4164 | **2.4%** | 78/173 |
| janne-complex-O1* | 65 | 0.5s/14600 | 1715 | 1680 | **2.0%** | 30/89 |
| janne-complex-O2* | 65 | 0.5s/13004 | 1557 | 1536 | **1.3%** | 32/78 |
| fdct-O1 | 238 | 21s/60534 | 4245 | 4092 | **3.7%** | 100/363 |
| fdct-O2 | 238 | 3.24s/55285 | 19231 | 18984 | **1.3%** | 166/3543 |
| **Single-Path Programs[‡] with MUL/MLA/SMULL instructions (duration of instruction depends on data)** | | | | | | |
| fdct-O0 | 238 | 124s/85008 | 11242/11800 | 11448 | **3.0%** | 253/831 |
| matmult-O0* | 162 | 217s/10531262 | 502849/529250 | 511584/528684 | **0.1%** | 158/314 |
| matmult-O1* | 162 | 25s/1112527 | 129967/156367 | 127356/153000 | **2.2%** | 71/172 |
| matmult-O2* | 162 | 121s/6780931 | 122045/148299 | 116844/140664 | **5.4%** | 75/288 |
| jfdcint-O0 | 374 | 92s/100861 | 12726/12918 | 12588 | **2.6%** | 159/792 |
| jfdcint-O1 | 374 | 12s/35419 | 4880/5072 | 4668 | **8.6%** | 25/325 |
| jfdcint-O2 | 374 | 5.38s/175661 | [16746,16938] | 16380 | **3.4%** | 56/2512 |
| **Multiple-Path Programs** | | | | | | |
| bs-O0 | 174 | 30s/1421274 | 478/1068 | 1056 | **1.1%** | 75/151 |
| bs-O1 | 174 | 23s/1214673 | 321/738 | 720 | **2.5%** | 28/82 |
| bs-O2 | 174 | 12s/655870 | 273/628 | 600 | **4.6%** | 28/65 |
| cnt-O0* | 115 | 4s/77002 | 9025/9027 | 8836 | **2.1%** | 99/235 |
| cnt-O1* | 115 | 1.4s/27146 | 4123/4123 | 3996 | **3.1%** | 42/129 |
| cnt-O2* | 115 | 9s/11490 | 3067/3067 | 2928 | **4.6%** | 39/263 |
| insertsort-O0* | 91 | 598.98s/24250738 | 3133 | 3108 | **0.8%** | 79/175 |
| insertsort-O1* | 91 | 353.80s/11455293 | 1533 | 1500 | **2.2%** | 40/115 |
| insertsort-O2* | 91 | 11.68s/387292 | 1326 | 1320 | **0.4%** | 43/108 |
| ns-O0* | 497 | 60s/3064316 | 940/30968 | 30732 | **0.8%** | 132/215 |
| ns-O1* | 497 | 8s/368720 | 605/11701 | 11568 | **1.1%** | 61/124 |
| ns-O2* | 497 | 55s/1030746 | 441/7280 | 7236 | **0.6%** | 566/863 |

[⊕] file-Ox indicates that file was compiled using gcc -Ox
[†] lines of code in the C source file
[‡] $\frac{(C-M)}{M} \times 100$ computed using the upper bound for $C$ and $M$
[§] Instructions in Slice/Instructions in Program
* Program selected for the WCET Challenge 2006
[¶] UPPAAL 4.1.11/Intel Pentium 5/3.1Ghz/16GB

# Cache hit/miss