Course 8

Model Checking Modulo Theories

Cubicle

An SMT based model checker for parameterized systems

Sylvain Conchon

LRI (UMR 8623), Université Paris-Sud
INRIA Saclay – Île-de-France

Contents

- A short tutorial on Cubicle
- Theoretical foundations
- Implementation details

Cubicle : an open source model checker

Cubicle is an open source software written in OCaml.

- It is based on the theoretical framework of Model Checking Modulo Theories [Ghilardi, Ranise]
- Its implementation relies on a lightweight and enhanced version of the SMT solver Alt-Ergo

Cubicle is the result of a fruitful collaboration between University of Paris-Sud and Intel corporation

Cubicle's input language

Inspired by the description language of the Murφ model checker (Dill's group at Stanford)

A program is only represented by a set of global variables and guarded commands.

Guarded commands

A guarded command consists of a predicate on the state variables, called a guard, and a set of assignments that update those variables to change the state.

The control structure is only a single infinite loop which repeatedly execute two steps:

1. evaluate all the guards, given the current values of global variables
2. arbitrarily choose one of the commands whose guard is true and execute it, updating the variables

Example 1 : sequential programs

```
X,Y = 0

L0:
    X := 1;
L1:
    Y := X + 1;
L2:
```

```
type loc = L0 | L1 | L2
var X : int
var Y : int
var PC : loc

init () { X = 0 && Y = 0 && PC = L0 }

transition t1 ()
requires { PC = L0 }
{ X := 1; PC := L1  }

transition t2 ()
requires { PC = L1 }
{ Y := X + 1; PC := L2  }
```

Example 2 : granularity

```
L0:
  X := 1;
L1:
  Y := X + 1;
L2:
```

Example 2 : granularity

```
L0:
  X := 1;
L1:
  EAX := X;
L2:
  EAX := EAX + 1;
L3:
  Y := EAX;
L4:
```

Example 2 : granularity

```
L0:
  X := 1;
L1:
  EAX := X;
L2:
  EAX := EAX + 1;
L3:
  Y := EAX;
L4:
```

```
type loc = L0 | L1 | L2 | L3 | L4
var X : int
var Y : int
var EAX : int
var PC : loc

init () { X = 0 && Y = 0 && PC = L0 }

transition t1 ()
requires { PC = L0 }{ X := 1; PC := L1  }

transition t2 ()
requires { PC = L1 }{ EAX := X; PC := L2  }

transition t3 ()
requires { PC = L2 }
{ EAX := EAX + 1; PC := L3  }

transition t4 ()
requires { PC = L3 }{ Y := EAX; PC := L4  }
```

Example 3 : threads

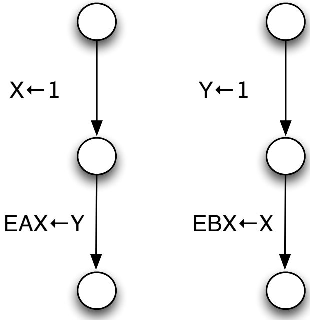
```
X,Y = 0
```

Thread 1

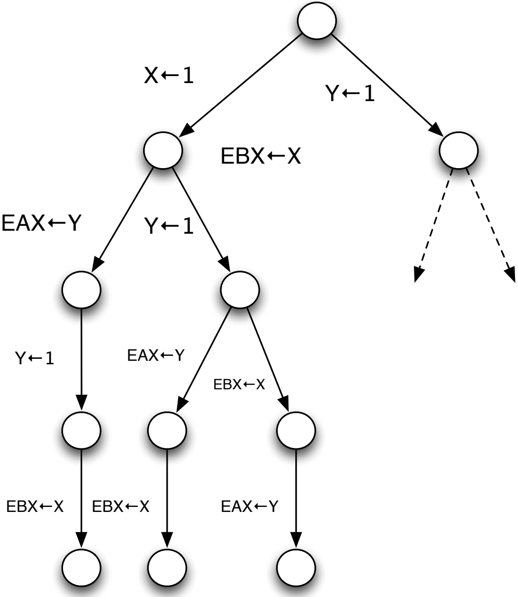
```
L0:
  X := 1;
L1:
  EAX := Y;
L2:
```

Thread 2

```
L0:
  Y := 1;
L1:
  EBX := X;
L2:
```


X←1
EAX←Y
Y←1
EBX←X

Example 3 : interleaving semantics


X←1
Y←1
EBX←X
EAX←Y
Y←1
Y←1
EAX←Y
EBX←X
EBX←X
EBX←X
EAX←Y

Example 3

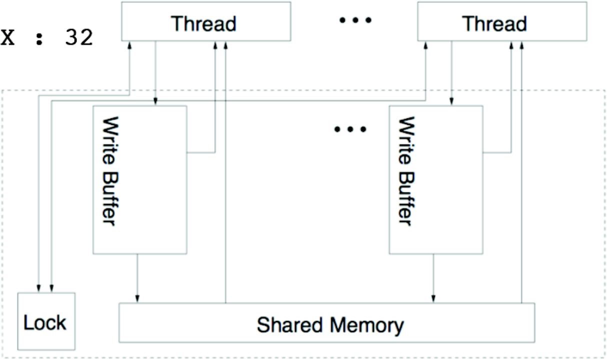
```
type location = L0 | L1 | L2
var X : int
var Y : int
var EAX : int
var EBX : int
var PC1 : location
var PC2 : location

init () { PC1 = L0 && PC2 = L0 && X = 0 && Y = 0 }

transition writex_1 ()
requires { PC1 = L0 } { X := 1; PC1 := L1 }

transition ready_1 ()
requires { PC1 = L1  } { EAX := Y; PC1 := L2 }

transition writey_2 ()
requires { PC2 = L0 } { Y := 1; PC2 := L1 }

transition readx_2 ()
requires { PC2 = L1 } { EBX := X; PC2 := L2 }
```

Example 3 : safety property

- Characterize bad states
- Use Cubicle to show that bad states cannot be reach from initial states

```
type loc = L0 | L1 | L2
var X : int
var Y : int
var EAX : int
var EBX : int
var PC1 : loc
var PC2 : loc
init () { PC1 = L0 && PC2 = L0 && X = 0 && Y = 0 }
unsafe () { PC1 = L2 && PC2 = L2 && EAX = 0 && EBX = 0 }
...
```

Example 4 : encoding asynchronous writes


Thread
...
Thread
Write Buffer
...
Write Buffer
Lock
Shared Memory

```
[x] ← 1
EAX ← [y]
```

```
[y] ← 1
EBX ← [x]
```

EAX : 32

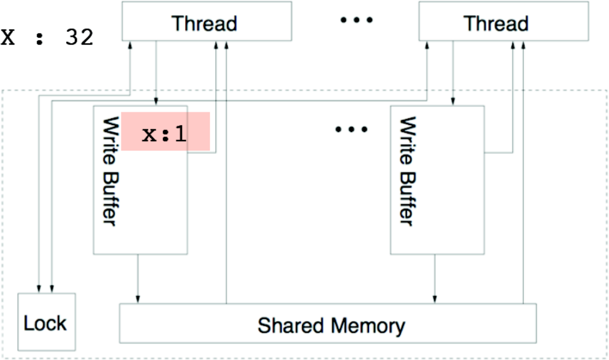
EBX : 47

x : 0   y : 0

Example 4 : encoding asynchronous writes

```
[x] ← 1
EAX ← [y]
```

```
[y] ← 1
EBX ← [x]
```

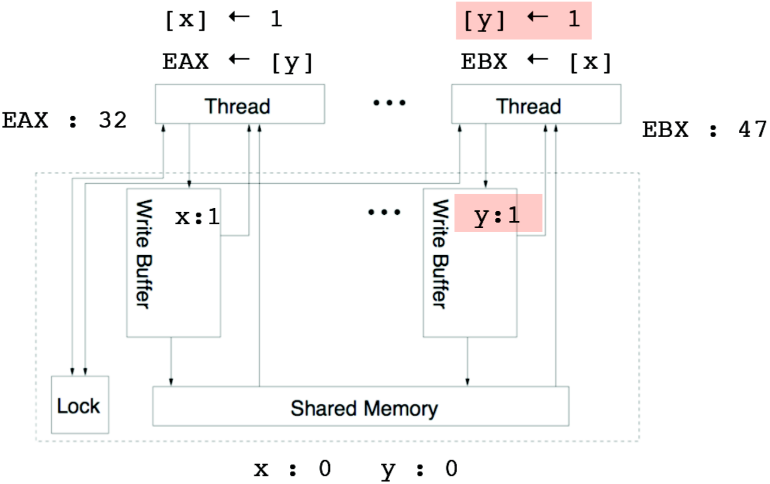

Thread
…
Thread
Write Buffer
x:1
…
Write Buffer
Lock
Shared Memory

EAX : 32

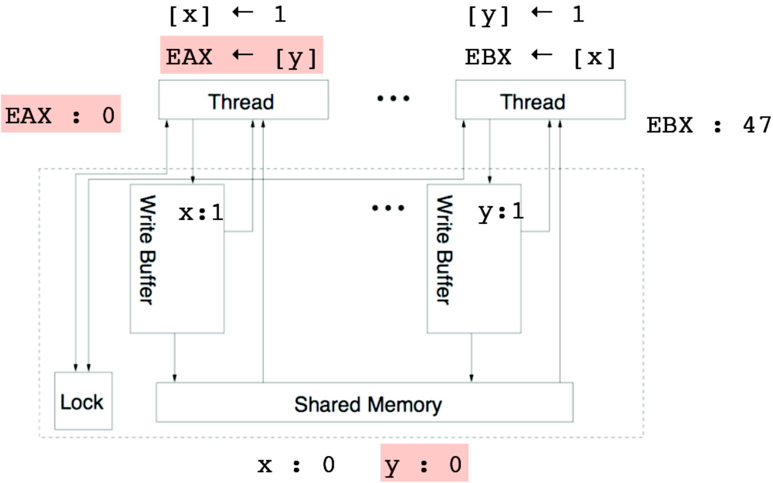
EBX : 47

x : 0   y : 0

Example 4 : encoding asynchronous writes


[x] ← 1
EAX ← [y]
[y] ← 1
EBX ← [x]
EAX : 32
Thread
...
Thread
EBX : 47
Write Buffer
x:1
...
Write Buffer
y:1
Lock
Shared Memory
x : 0 y : 0

Example 4 : encoding asynchronous writes



`[x] ← 1`

`EAX ← [y]`

`[y] ← 1`

`EBX ← [x]`

`EAX : 0`

Thread

...

Thread

`EBX : 47`

Write Buffer

`x:1`

...

Write Buffer

`y:1`

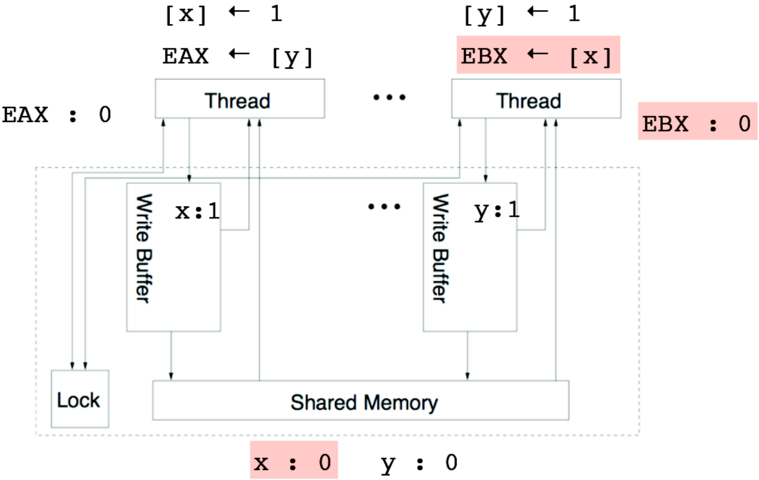
Lock

Shared Memory

`x : 0`

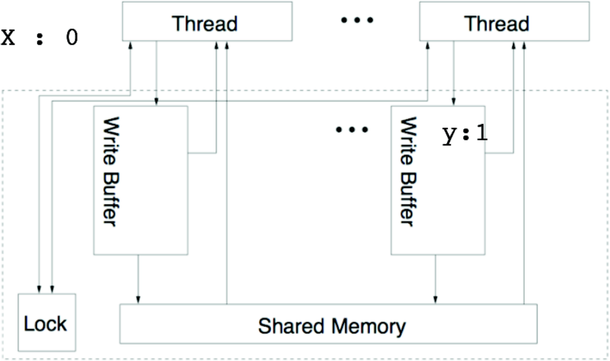
`y : 0`

Example 4 : encoding asynchronous writes


[x] ← 1
EAX ← [y]
[y] ← 1
EBX ← [x]
EAX : 0
Thread
…
Thread
EBX : 0
Write Buffer
x:1
…
Write Buffer
y:1
Lock
Shared Memory
x : 0
y : 0

Example 4 : encoding asynchronous writes

```
[x] ← 1            [y] ← 1
EAX ← [y]          EBX ← [x]
```

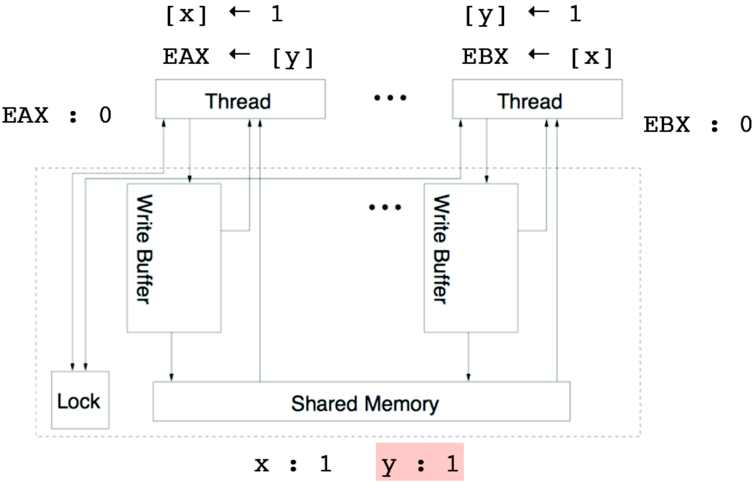

Thread
...
Thread
Write Buffer
...
Write Buffer
y:1
Lock
Shared Memory

EAX : 0

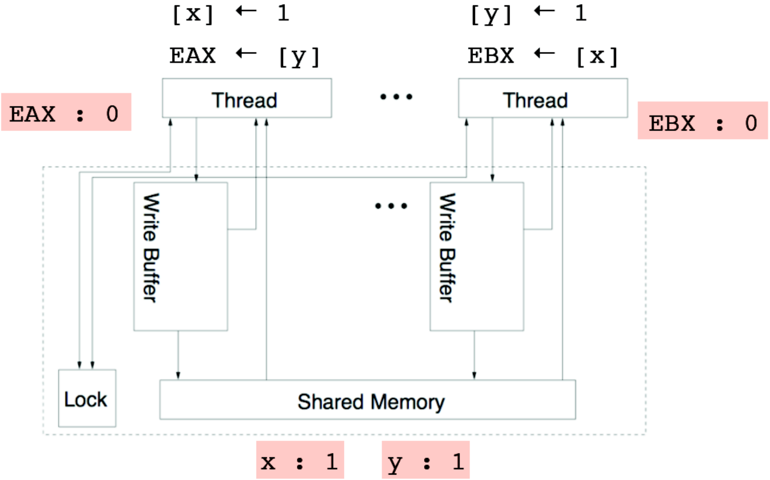
EBX : 0

x : 1   y : 0

Example 4 : encoding asynchronous writes


[x] ← 1
EAX ← [y]
[y] ← 1
EBX ← [x]
EAX : 0
Thread
• • •
Thread
EBX : 0
Write Buffer
• • •
Write Buffer
Lock
Shared Memory
x : 1
y : 1

Example 4 : encoding asynchronous writes



[x] ← 1
EAX ← [y]
[y] ← 1
EBX ← [x]
EAX : 0
Thread
• • •
Thread
EBX : 0
Write Buffer
• • •
Write Buffer
Lock
Shared Memory
x : 1
y : 1

Example 4 : encoding asynchronous writes

```
...
var WriteBuffer1 : int
var WriteBuffer2 : int
var Sync1 : bool
var Sync2 : bool
var PC1 : location
var PC2 : location
...

transition sync_buffer1 ()
requires { Sync1 = True }
{ X := WriteBuffer1;
  Sync1 := False; }

transition sync_buffer2 ()
requires { Sync2 = True }
{ Y := WriteBuffer2;
  Sync2 := False; }
```

```
transition write_x ()
requires { PC1 = L1 }
{ WriteBuffer1 := 1;
  Sync1 := True;
  PC1 := L2; }

transition read_y ()
requires { PC1 = L2 }
{ EAX := Y;
  PC1 := L3; }

transition write_y ()
requires { PC2 = L1 }
{ WriteBuffer2 := 1;
  Sync2 := True;
  PC2 := L2; }

transition read_x ()
requires { PC2 = L2 }
{ EBX := X;
  PC2 := L3; }
```

Parameterized systems

Modeling and verifying programs involving an arbitrary number of processes

- Replication of components
- Unknown or very large number of components

Typical examples : cache coherence protocols, mutual exclusion algorithms, fault-tolerant protocols

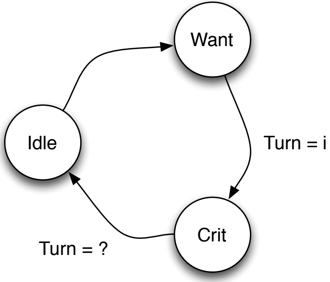
Array-based transition systems

Cubicle handles parameterized systems through :

- a new built-in data type proc (with unspecified cardinality)
- state variables defined as arrays indexed by process identifiers
- initial states described with a universally-quantified formula over processes
- bad states described with existentially-quantified formulas over processes
- transitions parameterized by process identifiers

Example 5 : array-based transition systems

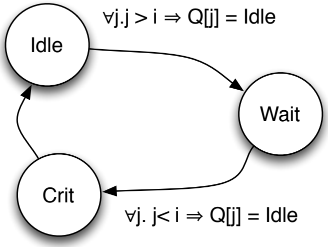
A Dekker-like mutual exclusion algorithm



```
var Turn : proc
array Req[proc] : bool
array CS[proc] : bool

 init (i) { Req[i]=False && CS[i]=False }
 unsafe (i j) { CS[i]=True && CS[j]=True }

transition req (i)
requires { Req[i] = False }
{ Req[i] := True }

transition enter (i)
requires {Req[i]=True && CS[i]=False && Turn=i}
{ CS[i] := True }

transition exit (i)
requires { CS[i]=True }
{ Turn := ? ;
  CS[i] := False;
  Req[i] := False }
```

Example 6 : global conditions

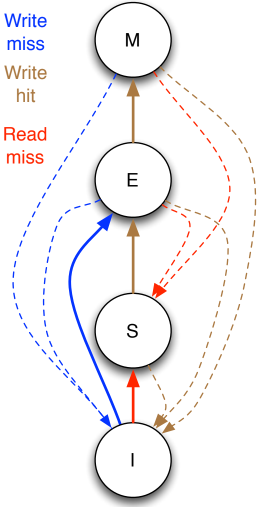
A Bakery-like mutual exclusion algorithm


Idle
∀j.j > i ⇒ Q[j] = Idle
Wait
Crit
∀j. j< i ⇒ Q[j] = Idle

```
type t = Idle | Wait | Crit
array Q[proc] : t

init (i) { Q[i]=Idle }
unsafe (i j) { Q[i]=Crit && Q[j]=Crit }

transition wait (i)
requires { Q[i]=Idle &&
           forall_other j. (j<i || Q[j]=Idle) }
{ Q[i] := Wait }

transition enter (i)
requires { Q[i]=Wait &&
           forall_other j. (i<j || Q[j]=Idle) }
{ Q[i] := Crit }

transition exit (i)
requires { Q[i] = Crit }
{ Q[i] := Idle }
```

Example 7 : case analysis

MESI protocol


Write miss
Write hit
Read miss
M
E
S
I

```
type state = M | E | S | I
array St[proc]: state
init (i) { St[i] = I }

transition read_miss ( i )
requires { St[i] = I }
{ St[j] := case  | j = i : S
                 | St[j] = E : S
                 | St[j] = M : S
                 | _ : St[j]      }

transition write_miss ( i )
requires { St[i] = I }
{ St[j] := case | j = i : E | _ : I }

transition write_hit_1 ( i )
requires { St[i] = E }{ St[i] := M }

transition write_hit_2 ( i )
requires { St[i] = S }
{ St[j] := case | j = i : E | _ : I }
```

Language limitations

Cubicle's input language is still limited; it could be further improved.

- data types (records, several parameterized data types)
- programming constructs (sequences, loops)
- arithmetic expressions

Theoretical foundation : MCMT

Cubicle implements the Model Checking Modulo Theories (MCMT) framework [Ghilardi, Ranise] where system states and transitions are defined as first-order formulas

- Initial states are defined by a universally quantified formula
- Bad states are defined by special existentially quantified formulas, called cubes
- Transitions correspond to existentially quantified formulas

Example

```
type st = Idle | Want | Crit
var Turn : proc
array S[proc] : st

init (z) { S[z] = Idle }

unsafe (x y) {
  S[x] = Crit && S[y] = Crit
}

transition req (i)
requires { S[i] = Idle }
{ S[i] := Want  }
```

```
transition enter (i)
requires { S[i] = Want &&
           Turn = i }
{ S[i] := Crit }
```

```
transition exit (i)
requires { S[i] = Crit }
{ Turn := ? ;
  S[i] := Idle }
```

Example

`predicate` $R(t, w, c)$

`axiom` $init : \forall t, w, c.\ (\forall i. \neg w[i] \wedge \neg c[i]) \Rightarrow R(t, w, c)$

`axiom` $req : \forall t, t', w, w', c, c'.$
$$R(t, w, c) \wedge \exists i. \neg w[i] \wedge w' = w[i \leftarrow true] \wedge t' = t \wedge c' = c$$
$$\Rightarrow R(t', w', c')$$

`axiom` $enter : \forall t, t', w, w', c, c'.$
$$R(t, w, c) \wedge \exists i. w[i] \wedge \neg c[i] \wedge t = i \wedge c' = c[i \leftarrow true] \wedge t' = t \wedge w' = w$$
$$\Rightarrow R(t', w', c')$$

`axiom` $exit : \forall t, t', w, w', c, c'.$
$$R(t, w, c) \wedge \exists i. c[i] \wedge w' = w[i \leftarrow false] \wedge c' = c[i \leftarrow false]$$
$$\Rightarrow R(t', w', c')$$

`goal` $unsafe : \exists t, w, c, i, j. R(t, w, c) \wedge i \neq j \wedge c[i] \wedge c[j]$

Checking safety = Backward Reachability (BR)

I : inital states   U : unsafe states (cube) $\mathcal{T}$: transitions

---

```
BR ():
    V := ∅;
    push(Q, U);
    while not_empty(Q) do
        $\varphi$ := pop(Q)
        if $\varphi \wedge$ I sat then return unsafe
        if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then
            V := $V \cup \{\varphi\}$
            push(Q, $\mathtt{pre}_{\mathcal{T}}(\varphi)$)
    return safe
```

Checking safety = Backward Reachability (BR)

I : inital states U : unsafe states (cube) $\mathcal{T}$: transitions

---

```
BR ():
  V := ∅;
  push(Q, U);
  while not_empty(Q) do
    $\varphi$ := pop(Q)
    if $\varphi \land I$ sat then return unsafe (* SMT check *)
    if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then (* SMT check *)
      V := $V \cup \{\varphi\}$
      push(Q, $\mathrm{pre}_{\mathcal{T}}(\varphi)$)
  return safe
```

Pre-images

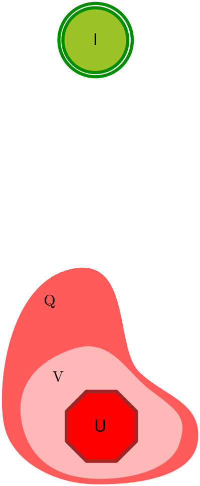
Given a transition $t \in \mathcal{T}$, $\texttt{pre}_t(\varphi)$ is a formula that describes the set of states from which a φ-state is reachable in one t-step

$$\texttt{pre}_{\mathcal{T}}(\varphi) = \bigvee_{t \in \mathcal{T}} \texttt{pre}_t(\varphi)$$

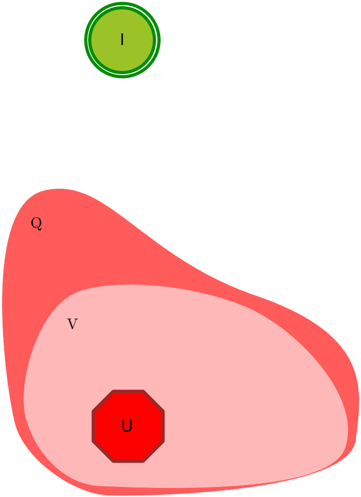
If φ is a cube, then $\texttt{pre}_{\mathcal{T}}(\varphi)$ can also be represented as a union of cubes

Running BR

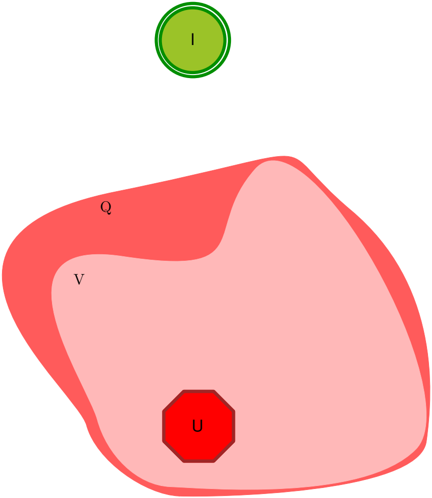
Running BR



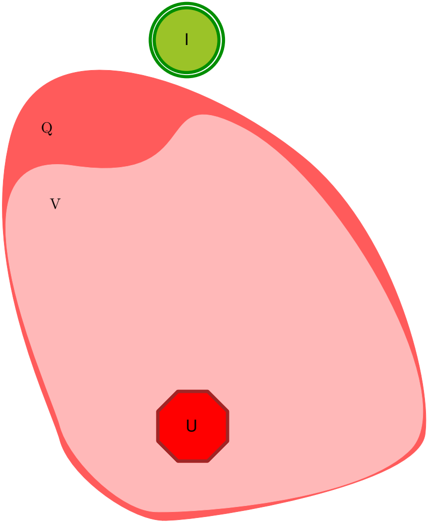
Running BR



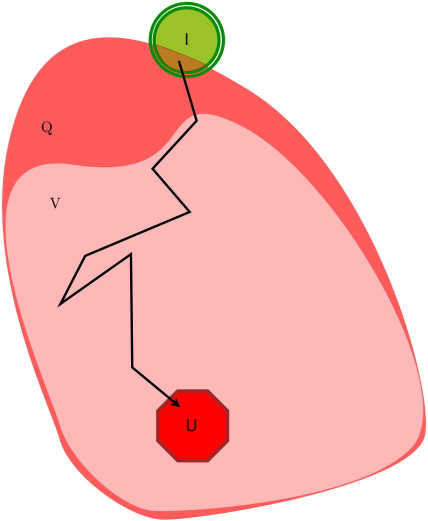
Running BR



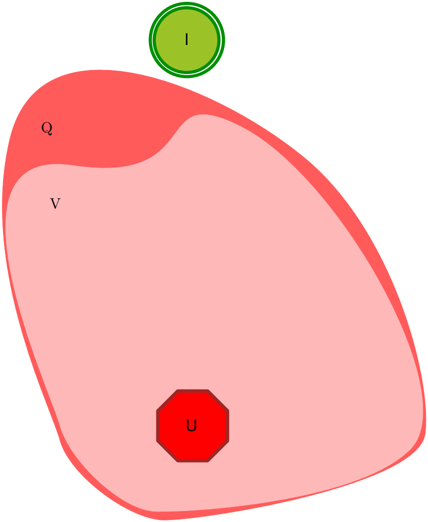
Running BR



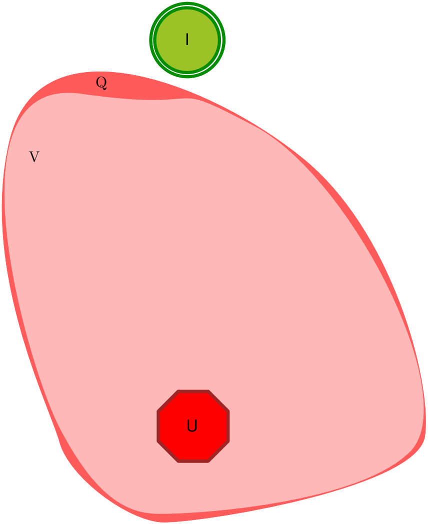
Running BR



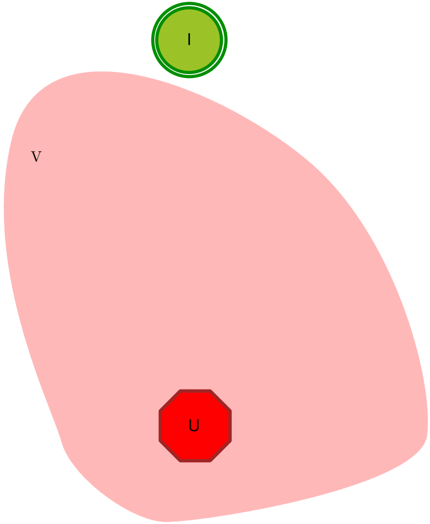
Running BR



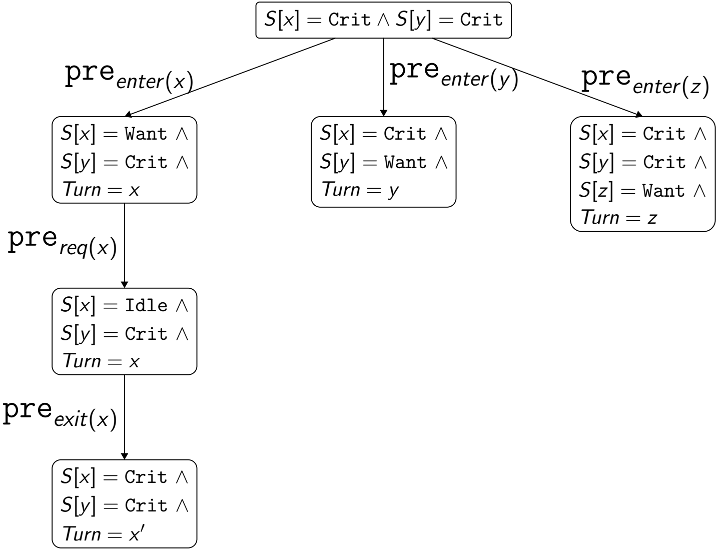
Running BR



Running BR



BR : example



$S[x] = \texttt{Crit} \wedge S[y] = \texttt{Crit}$

$\texttt{pre}_{enter(x)}$

$S[x] = \texttt{Want} \wedge S[y] = \texttt{Crit} \wedge Turn = x$

$\texttt{pre}_{req(x)}$

$S[x] = \texttt{Idle} \wedge S[y] = \texttt{Crit} \wedge Turn = x$

$\texttt{pre}_{exit(x)}$

$S[x] = \texttt{Crit} \wedge S[y] = \texttt{Crit} \wedge Turn = x'$

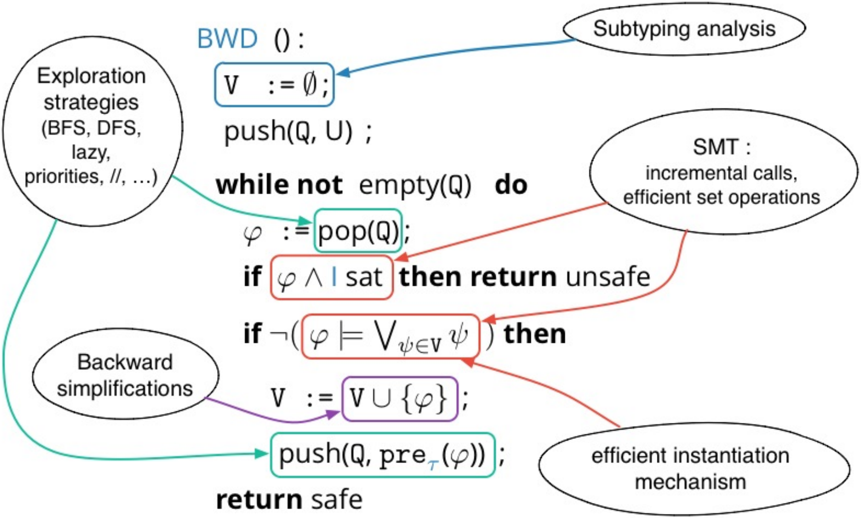
$\texttt{pre}_{enter(y)}$

$S[x] = \texttt{Crit} \wedge S[y] = \texttt{Want} \wedge Turn = y$

$\texttt{pre}_{enter(z)}$

$S[x] = \texttt{Crit} \wedge S[y] = \texttt{Crit} \wedge S[z] = \texttt{Want} \wedge Turn = z$

Implementation issues and optimizations



```
BWD ():
  $V := \emptyset$;
  push(Q, U) ;
  while not empty(Q) do
    $\varphi$ := pop(Q);
    if $\varphi \wedge I$ sat then return unsafe
    if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then
      $V := V \cup \{\varphi\}$;
      push(Q, $pre_\tau(\varphi)$);
  return safe
```

Exploration strategies (BFS, DFS, lazy, priorities, //, …)

Subtyping analysis

SMT : incremental calls, efficient set operations

Backward simplifications

efficient instantiation mechanism

Fixpoint computation

```
V := $\emptyset$
push(Q, U)
while not_empty(Q) do
  $\varphi$ := pop(Q)
  if $\varphi \land$ I sat then return(unsafe)
  if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then
    V := $V \cup \{\varphi\}$
    push(Q, $\mathrm{pre}_{\mathcal{T}}(\varphi)$)
return(safe)
```

Fixpoint computation: a challenge

$$\varphi \models \bigvee_{\psi \in \mathtt{V}} \psi$$

Fixpoint computation: a challenge

$$\exists \bar{x}.F \models \bigvee_{\psi \in \mathtt{V}} \exists \bar{y}.G_{\psi}$$

Fixpoint computation: a challenge

$$\exists \bar{x}.F \wedge \bigwedge_{\psi \in \mathtt{V}} \forall \bar{y}.\neg G_\psi \qquad \text{satisfiable ?}$$

Fixpoint computation: a challenge

$$\exists \bar{x}. F \wedge \bigwedge_{\psi \in \mathtt{V}} \forall \bar{y}. \neg G_\psi \qquad \text{satisfiable ?}$$

F and G_ψ are conjunctions of literals involving several theories (uninterpreted function symbols, linear arithmetic, enumerations)

Fixpoint computation: a challenge

$$\exists \bar{x}.F \wedge \bigwedge_{\psi \in \mathtt{V}} \bigwedge_{\sigma \in \Sigma} (\neg G_{\psi})\sigma \qquad \text{satisfiable ?}$$

Fixpoint computation: a challenge

$$\exists \bar{x}.F \wedge \bigwedge_{\psi \in \mathtt{V}} \bigwedge_{\sigma \in \Sigma} (\neg G_\psi)\sigma \qquad \text{satisfiable ?}$$

Suppose $|\mathtt{V}| = 20000$ and $|\Sigma| = 120$ (e.g. $|\bar{x}| = |\bar{y}| = 5$) then

$$|\bigwedge_{\psi \in \mathtt{V}} \bigwedge_{\sigma \in \Sigma} (\neg G_\psi)\sigma| \sim 2.10^6 \text{clauses}$$

Fixpoint computation: optimizations

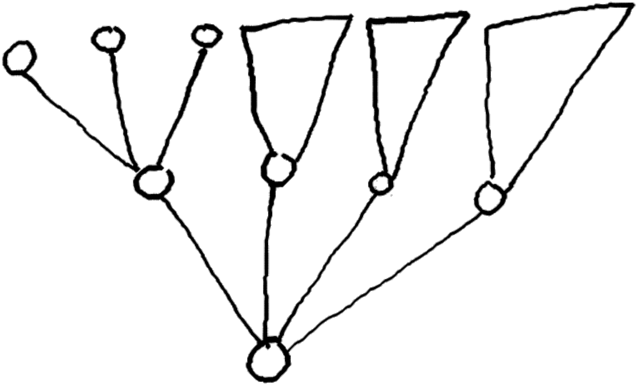
- Fast checks: $G_\psi\sigma \subseteq F$
- Irrelevant permutations:

  $L \in G_\psi\sigma$ and $L' \in F$ and $\neg(L \wedge L')$ is immediate
- A single SMT-context is used for each fixpoint check; it just gets incremented and repeatedly verified

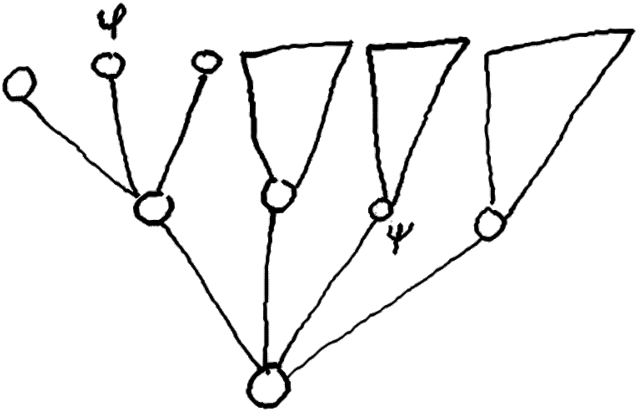
Node deletion

```
V := $\emptyset$
push(Q, U)
while not_empty(Q)
  $\varphi$ := pop(Q)
  if $\varphi \wedge$ I sat then return(unsafe)
  if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then
    V := V $\cup \{\varphi\}$
    push(Q, pre($\varphi$))
return(safe)
```

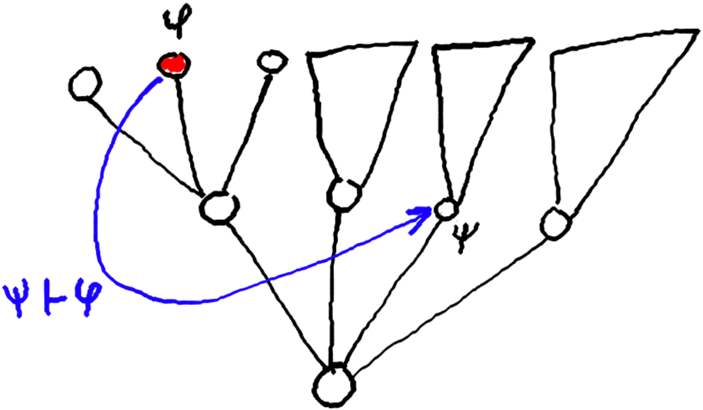
Node deletion : backward simplification



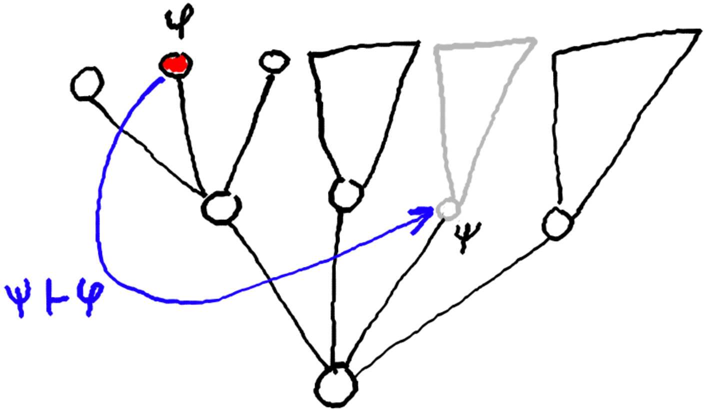
Node deletion : backward simplification



Node deletion : backward simplification



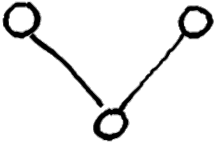
Node deletion : backward simplification


φ
ψ
$\psi \vdash \varphi$

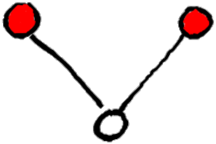
Towards a concurrent implementation

- Based on the Functory Library [Filliâtre, Krishnamani] : master / worker architecture
- Search can be parallelized:
  - Expensive tasks = fixpoint checks
  - Synchronization to keep a precise guidance (BFS)
  - Deletion becomes dangerous

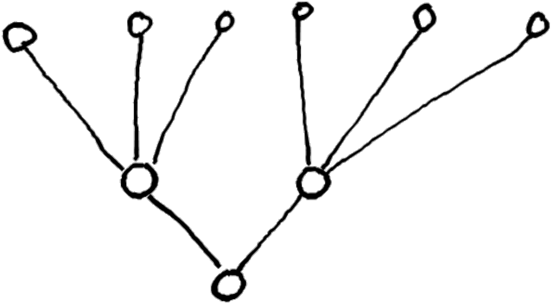
Node deletion in parallel BFS



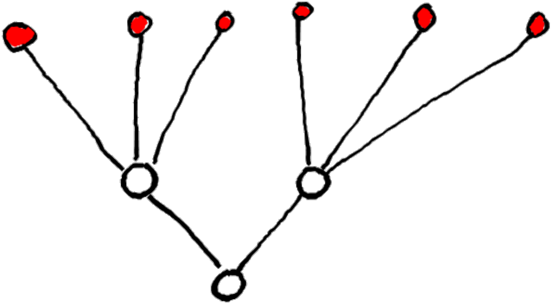
Node deletion in parallel BFS



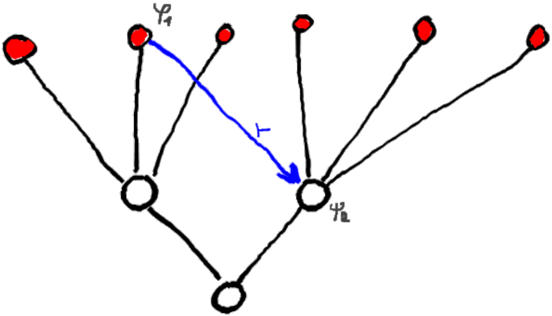
Node deletion in parallel BFS



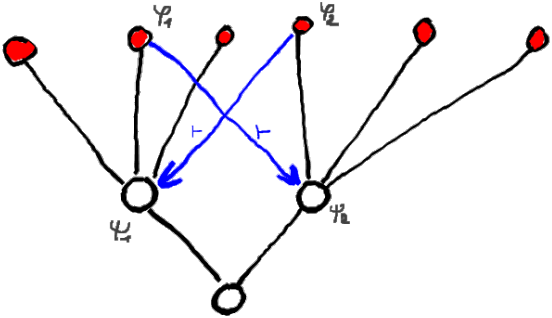
Node deletion in parallel BFS



Node deletion in parallel BFS


φ_1
$\vdash$
φ_2

Node deletion in parallel BFS


φ_1
φ_2
$\vdash$
$\vdash$
ψ_1
ψ_2

Other topics around Cubicle

- Invariant synthesis (BRAB, IC3)
- Industrial size hierarchical protocols
- Certification (traces, Why3)
- X86TSO
- C11

Some papers

The Model Checking Modulo Theories paradigm has been originally proposed by Silvio Ghilardi et Silvio Ranise

- `http://users.mat.unimi.it/users/ghilardi/mcmt/`

In particular, I recommend

Backward Reachability of Array-based Systems by SMT Solving: Termination and Invariant Synthesis [LMCS, vol.6, n.4, 2010]

More information on Cubicle

- Certificates for Parameterized Model Checking [FM 2015]
- Invariants for Finite Instances and Beyond [FMCAD 2013]
- Cubicle: A Parallel SMT-based Model Checker for Parameterized Systems [CAV 2012]

Thank you !

Terminaison de l'analyse

La terminaison de l'analyse arrière n'est pas garantie en général, mais elle l'est sous certaines conditions

- étudions l'évolution de l'ensemble V et considérons la séquence
  $$V_0 \subseteq V_1 \subseteq V_1 \subseteq \ldots \subseteq V_n \subseteq \ldots$$
  où V_n représente l'ensemble V à la $n^{\text{ième}}$ itération de la boucle `while`
- pour que la boucle ne termine pas, il faut nécessairement que de nouveaux cubes soient ajoutés régulièrement dans V, *i.e.* qu'il existe une infinité d'ensembles de nœuds visités V_{k_i} tels que $V_{k_1} \subset V_{k_2} \subset \cdots$

On va définir les condiditions suffisantes pour que cette sous-séquence d'inclusions strictes soit finie

Terminaison: configurations

Soit $\mathcal{S}$ un système paramétré à tableaux

Une configuration de $\mathcal{S}$ est un état concret du système, *i.e.* un modèle pour

- les types
- les variables globales
- les tableaux du système

En particulier, une configuration doit fixer le nombre de processus du système, *i.e.* la cardinalité du type `proc`.

Un système à tableaux $\mathcal{S}$ a un nombre potentiellement infini de configurations.

Terminaison: pré-ordres et idéaux

Définition. un pré-ordre bien fondé $\preceq$ est une relation binaire réflexive et transitive sans suite infinie strictement décroissante

On note $[\![\varphi]\!]$ l'ensemble des configurations qui satisfont un cube φ

Si l'ensemble des configurations est muni d'un pré-ordre bien fondé $\preceq$, alors on peut montrer [Ghilardi&Ranise, LMCS 2010] que $[\![\varphi]\!]$ est un idéal, c'est-à-dire que

si $s \in [\![\varphi]\!]$ et $s \preceq s'$ alors $s' \in [\![\varphi]\!]$

Par extension, il est immédiat de montrer que chaque ensemble V_n de nœuds visités est également un idéal.

Terminaison: bel ordre

Définition. Un pré-ordre bien fondé $\preceq$ est un bel ordre si pour toute séquence infinie de configurations $s_1, s_2, \ldots$, il existe nécessairement $i < j$ tels que $s_i \preceq s_j$

Théorème. Si $\preceq$ est un bel ordre, alors toute séquence d'inclusions strictes d'idéaux $[\![V_{k_1}]\!] \subset [\![V_{k_2}]\!] \subset \cdots$ est finie.

De plus, $[\![.]\!]$ est monotone, *i.e.* si $V \subset V'$ alors $[\![V]\!] \subset [\![V']\!]$ donc la finitude de la séquence d'inclusion $[\![V_{k_1}]\!] \subset [\![V_{k_2}]\!] \subset \cdots$ implique bien la finitude de la séquence $V_0 \subseteq V_1 \subseteq \cdots \subseteq V_n \subseteq \cdots$

Terminaison: preuve du théorème

On raisonne par contradiction

- Si la séquence $[\![V_{k_1}]\!] \subset [\![V_{k_2}]\!] \subset \cdots$ est infinie, alors il existe également une séquence infinie de configurations $s_1, s_2, \ldots$ telle que $s_i \in [\![V_{k_i}]\!]$ et $s_i \notin [\![V_{k_j}]\!]$, pour tout $j < i$.
- De plus, $s_j \not\preceq s_i$, sinon $s_i \in [\![V_{k_j}]\!]$ puisque chaque $[\![V_{k_j}]\!]$ est un idéal.
- L'existence de cette suite infinie contredit donc l'hypothèse que $\preceq$ est un bel ordre

Terminaison: conditions suffisantes

Les conditions suffisantes pour que l'ensemble des configurations soit muni d'un bel ordre dépendent essentiellement du choix des opérations de comparaison sur le type `proc` et sur les types des éléments des tableaux

```
type t = A | B | C
array V[proc] : t
```

On peut exhiber un bel ordre grâce au lemme de Dickson :

$(\mathbb{N}^k, \leq_k)$ est muni d'un bel ordre pour tout k

Ainsi, la configuration `V[#1]=A, V[#2]=B, V[#3]=A, V[#4]=C` est encodée par le triplet d'entiers $(2, 1, 1)$

How to scale?

Cubicle's benchmarks on academic problems are promising

	Cubicle	CMurphi		
Szymanski_at	0.30s	8.04s (8)	5m12s (10)	2h50m (12)
German_Baukus	7.03s	0.74s (4)	19m35s (8)	4h49m (10)
German.CTC	3m23s	1.83s (4)	43m46s (8)	12h35m (10)
German_pfs	3m58s	0.99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	2h01m	5.68s (4)	2m58s (5)	1h36m (6)

How to scale?

But how to scale up on industrial-like problems?

	Cubicle	CMurphi		
Szymanski_at	0.30s	8.04s (8)	5m12s (10)	2h50m (12)
German_Baukus	7.03s	0.74s (4)	19m35s (8)	4h49m (10)
German.CTC	3m23s	1.83s (4)	43m46s (8)	12h35m (10)
German_pfs	3m58s	0.99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	2h01m	5.68s (4)	2m58s (5)	1h36m (6)
Szymanski_na	T.O.	0.88s (4)	8m25s (6)	7h08m (8)
Flash_nodata	O.M.	4.86s (3)	3m33s (4)	2h46m (5)
Flash	O.M.	1m27s (3)	2h15m (4)	O.M. (5)

O.M. $>$ 20 GB

T.O. $>$ 20 h

What Industrial-Like Means (for us)

A well-known candidate : the FLASH protocol (stanford multiprocessor architecture — 1994)

- Cache-coherence shared memory
- High-performance message passing
- 67 million states for 4 processes ($\sim$40 variables, $\sim$75 transitions)

Our Solution to Scale Up

We designed a new core algorithm for Cubicle that

- infers invariants for parameterized case using finite instances
- inserts and checks them on the fly in a backward reachability loop
- backtracks if necessary

BRAB (Backward Reachability Algorithm with Approximations) [FMCAD 2013]

Example: German-*ish* cache coherence protocol

Client i:

$\texttt{Cache}[i] \in \{\mathsf{E}, \mathsf{S}, \mathsf{I}\}$

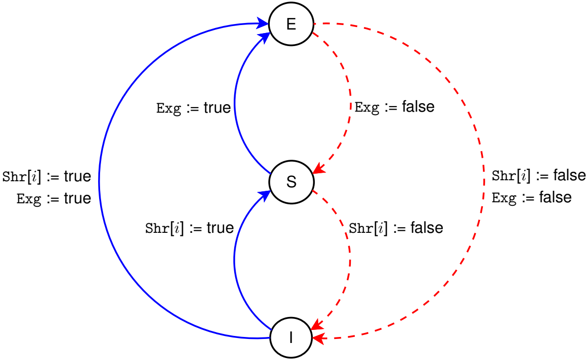
Directory:

$\texttt{Cmd} \in \{\mathsf{rs}, \mathsf{re}, \epsilon\}$

$\texttt{Ptr} \in proc$

$\texttt{Shr}[i] \in \{\mathsf{true}, \mathsf{false}\}$

$\texttt{Exg} \in \{\mathsf{true}, \mathsf{false}\}$


E
S
I
Exg := true
Exg := false
Shr[i] := true
Exg := true
Shr[i] := true
Shr[i] := false
Shr[i] := false
Exg := false

Initial states: $\forall i.\ \texttt{Cache}[i] = \mathsf{I}\ \wedge\ \neg\texttt{Shr}[i]\ \wedge\ \neg\texttt{Exg}\ \wedge\ \texttt{Cmd} = \epsilon$

Unsafe states: $\exists i, j.\ i \neq j\ \wedge\ \texttt{Cache}[i] = \mathsf{E}\ \wedge\ \texttt{Cache}[j] \neq \mathsf{I}$ **?**

(cubes)

Example: German-*ish* cache coherence protocol

Client i:

$\texttt{Cache}[i] \in \{\mathsf{E}, \mathsf{S}, \mathsf{I}\}$

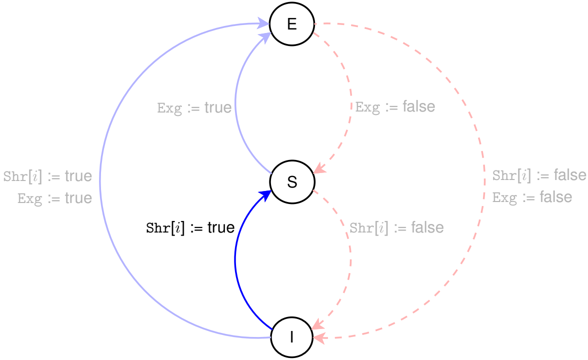
Directory:

$\texttt{Cmd} \in \{\mathsf{rs}, \mathsf{re}, \epsilon\}$

$\texttt{Ptr} \in \mathit{proc}$

$\texttt{Shr}[i] \in \{\mathsf{true}, \mathsf{false}\}$

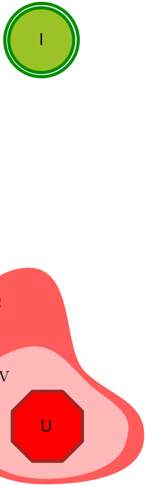
$\texttt{Exg} \in \{\mathsf{true}, \mathsf{false}\}$


E
S
I
Exg := true
Exg := false
Shr[i] := true
Exg := true
Shr[i] := false
Exg := false
Shr[i] := true
Shr[i] := false

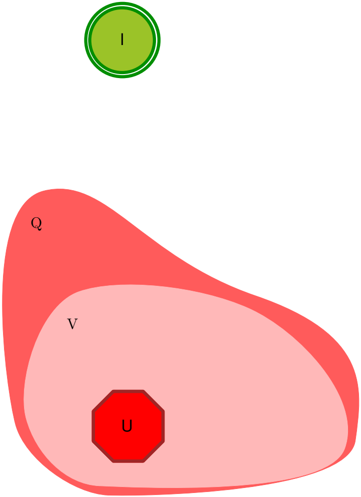
$$\begin{aligned} t_5 : \exists i. \quad & \texttt{Ptr} = i \ \wedge\ \texttt{Cmd} = \mathsf{rs} \ \wedge\ \neg\texttt{Exg} \wedge \\ & \texttt{Cmd}' = \epsilon \ \wedge\ \texttt{Shr}'[i] \ \wedge\ \texttt{Cache}'[i] = \mathsf{S} \end{aligned}$$

Backward reachability algorithm

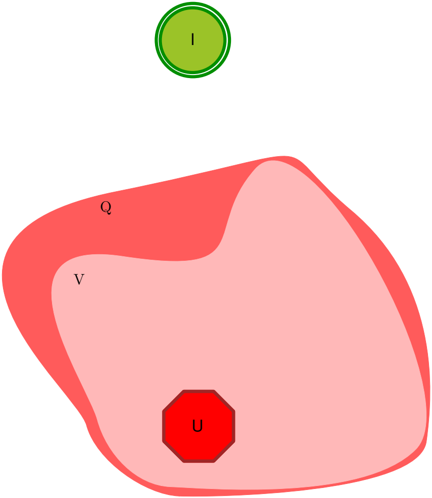
Backward reachability algorithm



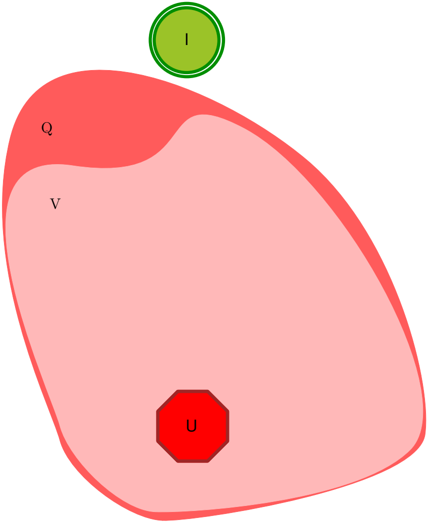
Backward reachability algorithm



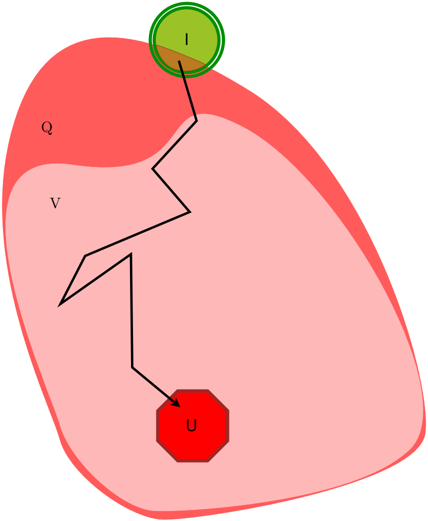
Backward reachability algorithm



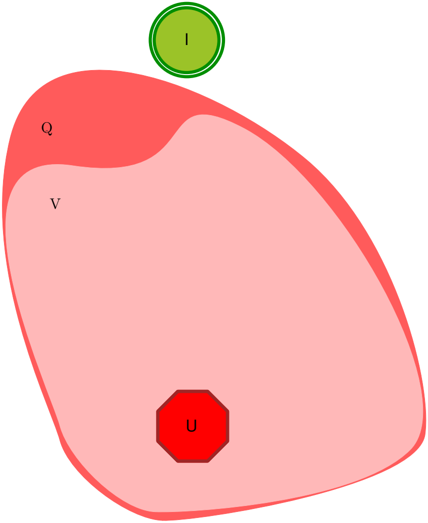
Backward reachability algorithm



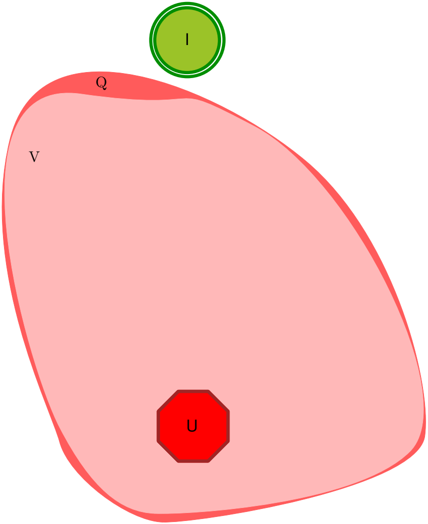
Backward reachability algorithm



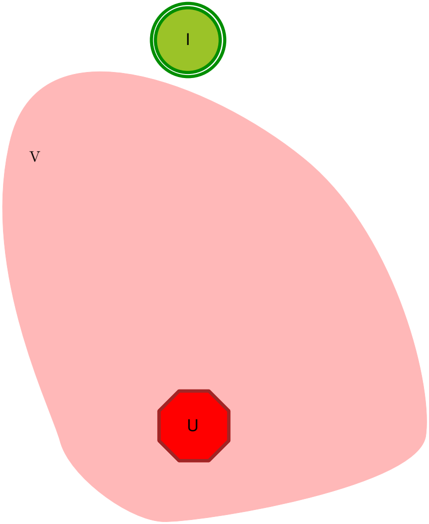
Backward reachability algorithm



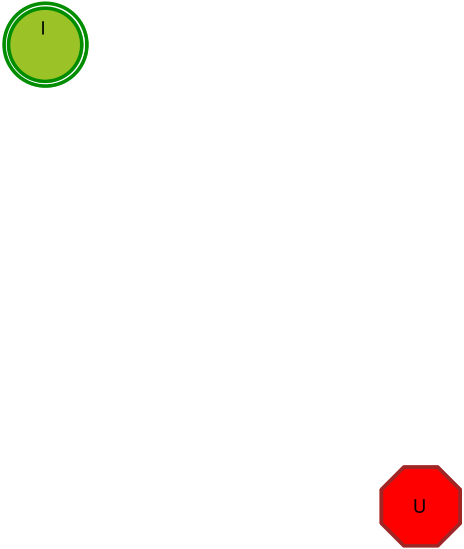
Backward reachability algorithm



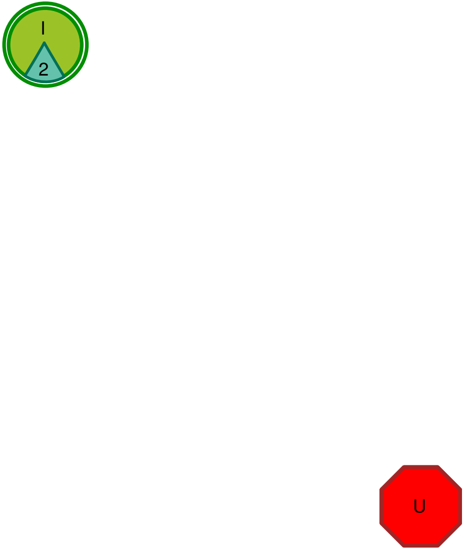
Backward reachability algorithm



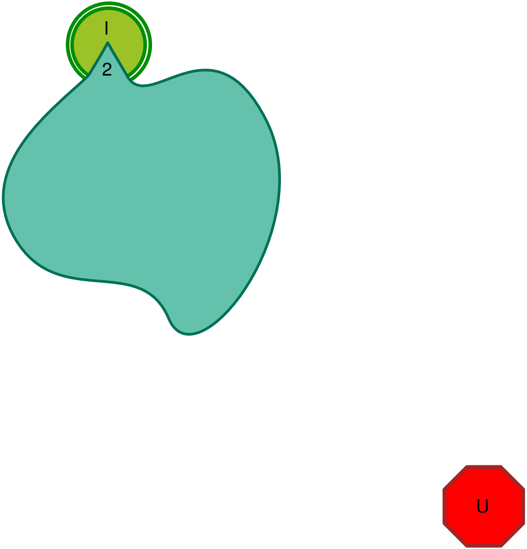
BRAB: intuition



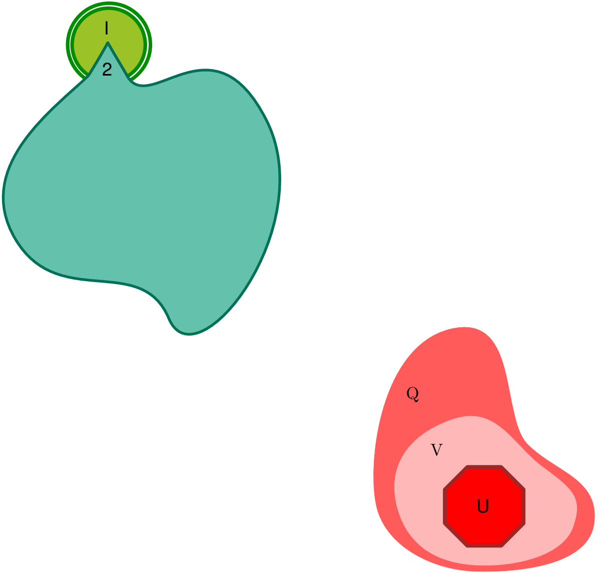
BRAB: intuition



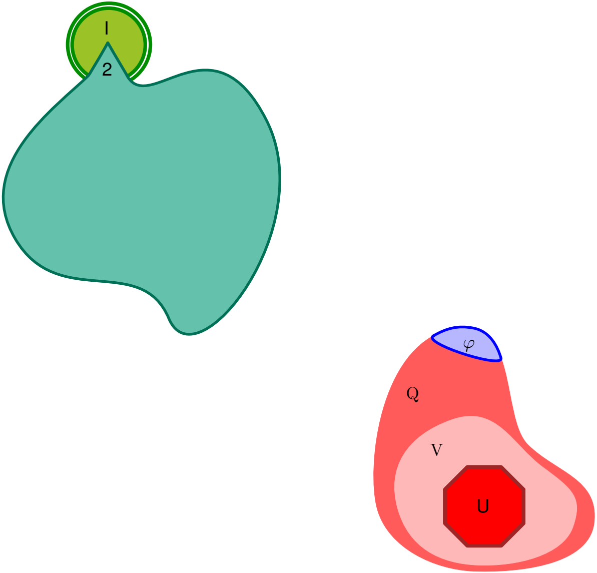
BRAB: intuition



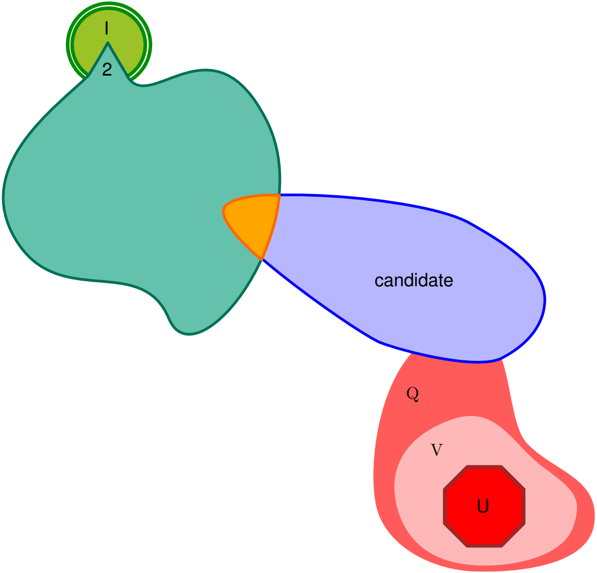
BRAB: intuition



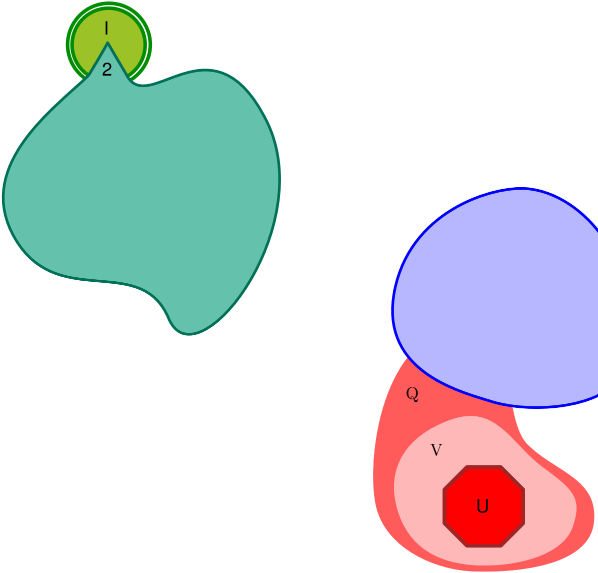
BRAB: intuition


I
2
φ
Q
V
U

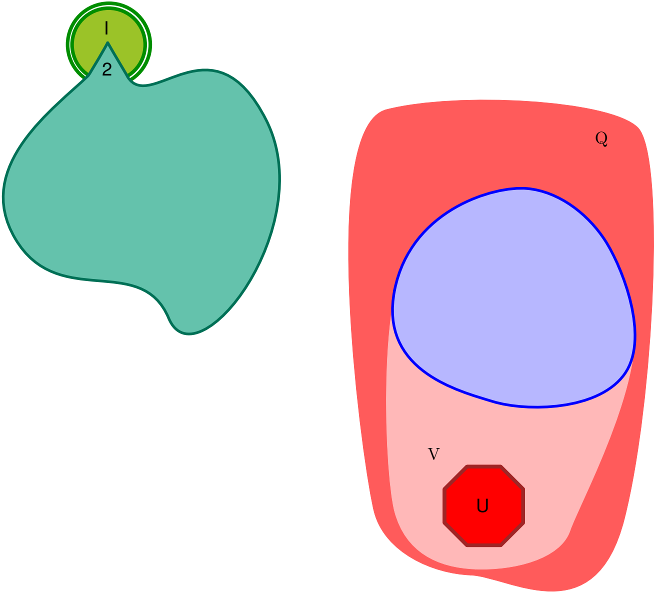
BRAB: intuition


I
2
candidate
Q
V
U

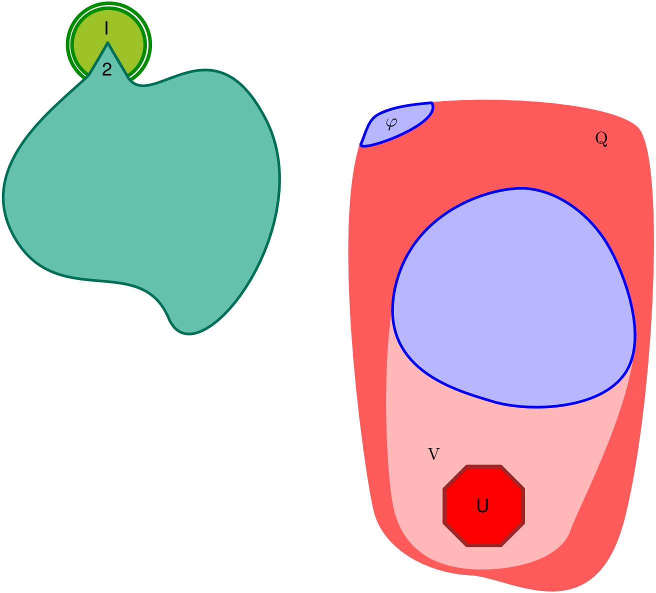
BRAB: intuition



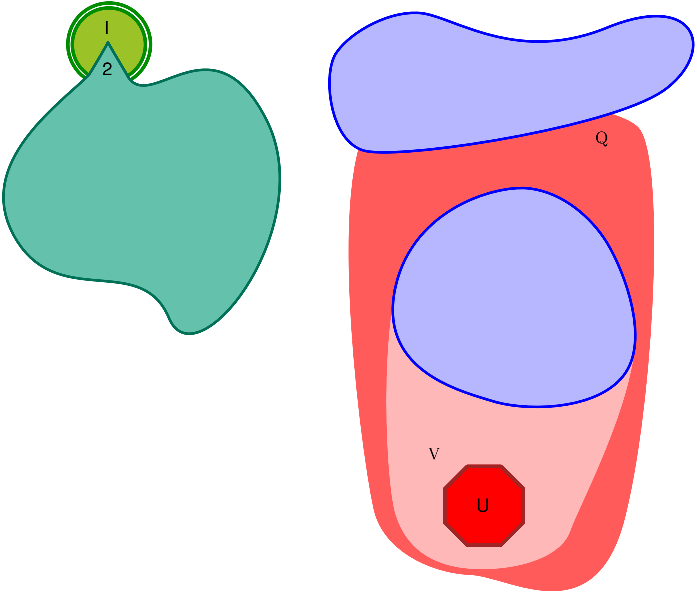
BRAB: intuition



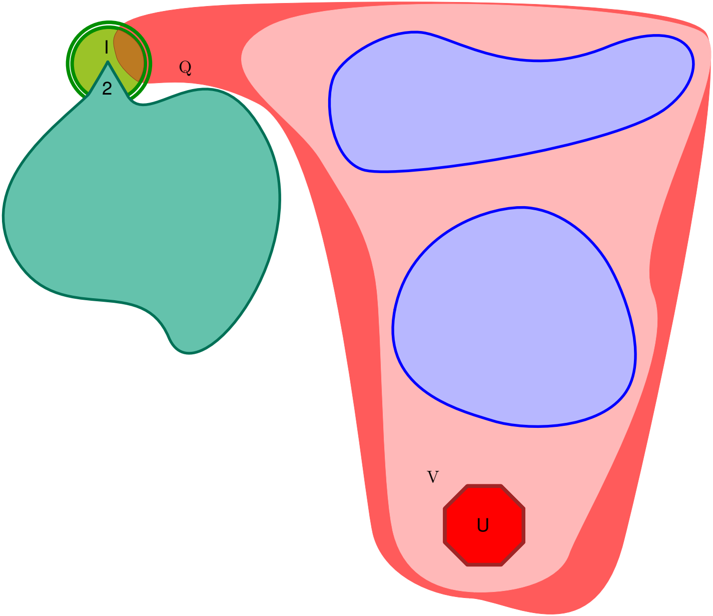
BRAB: intuition


I
2
φ
Q
V
U

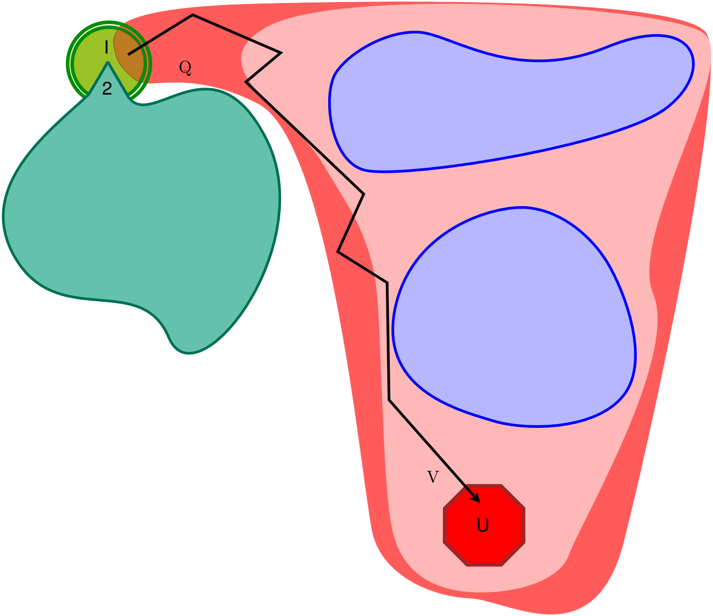
BRAB: intuition



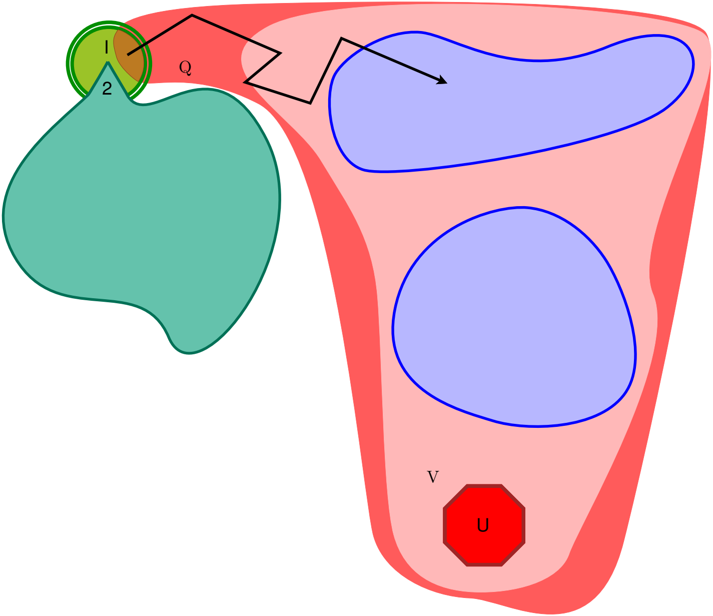
BRAB: intuition



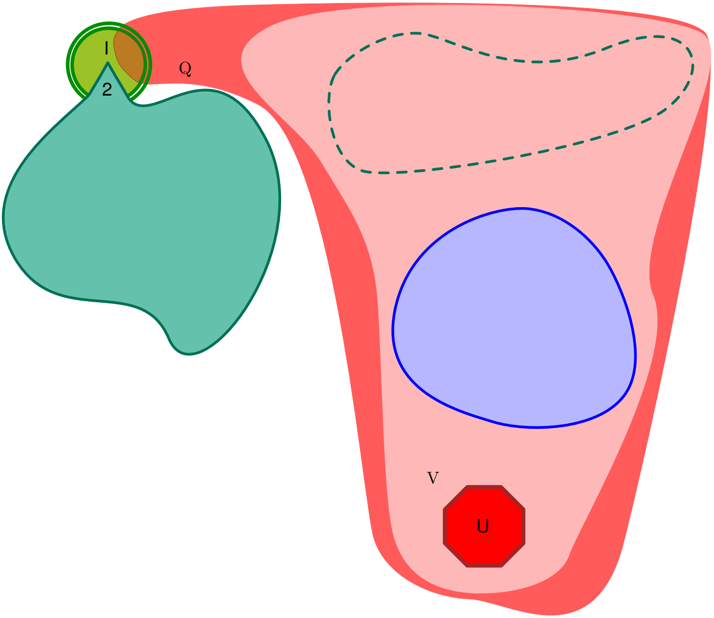
BRAB: intuition



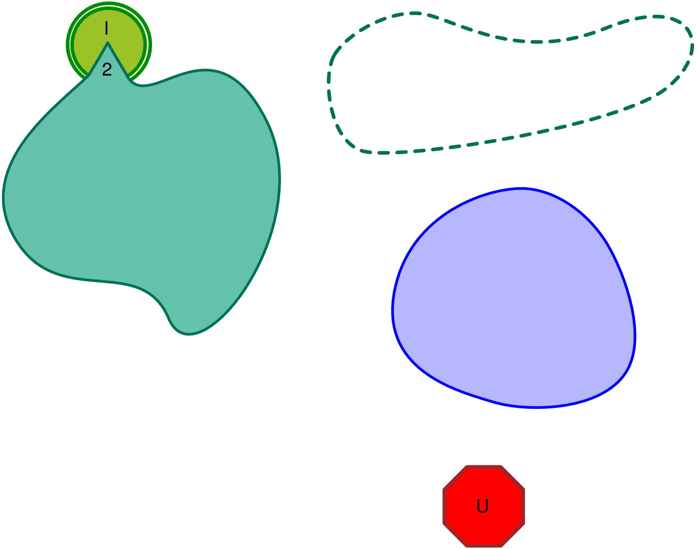
BRAB: intuition



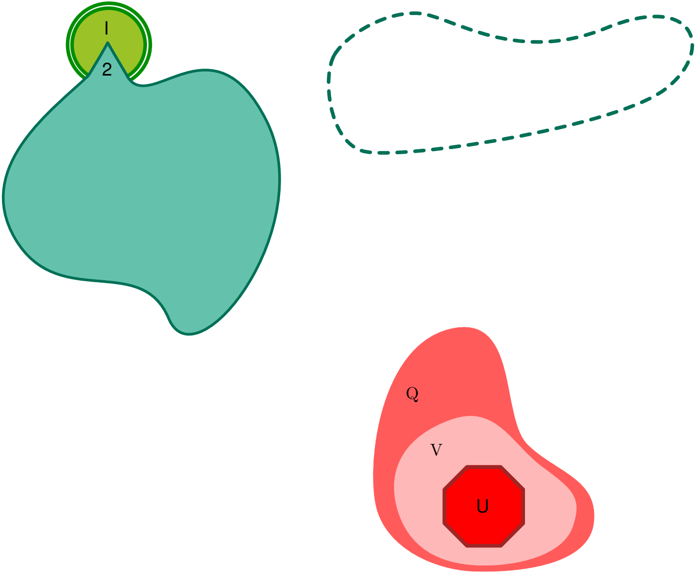
BRAB: intuition



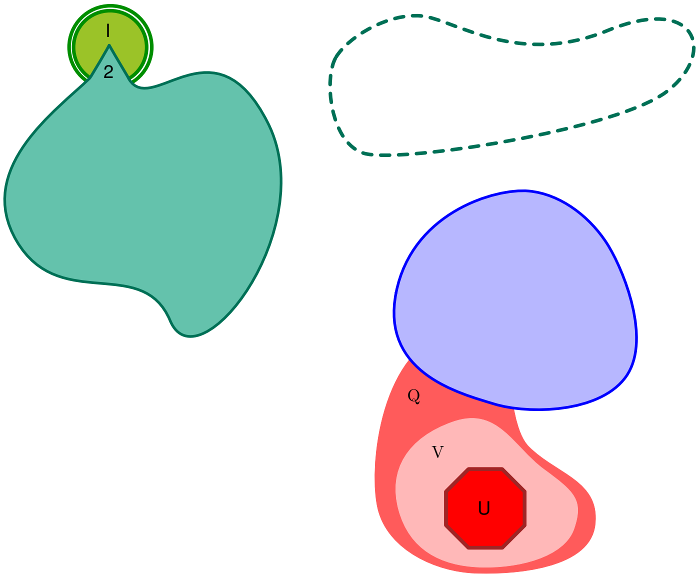
BRAB: intuition



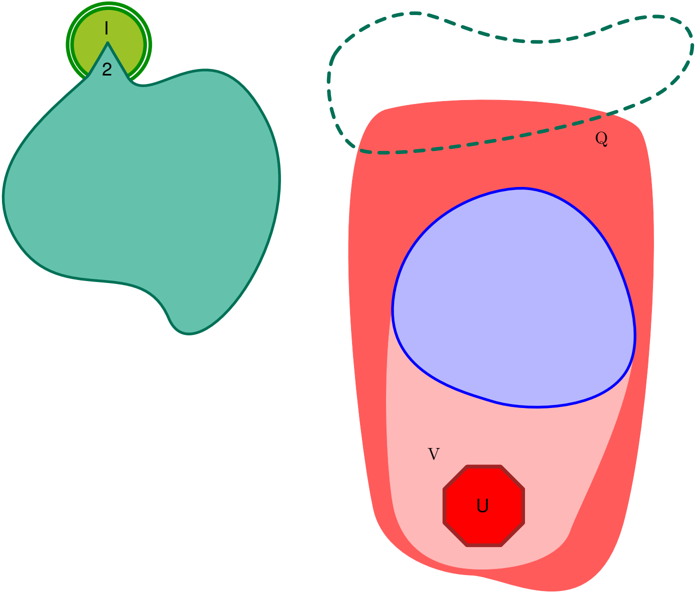
BRAB: intuition



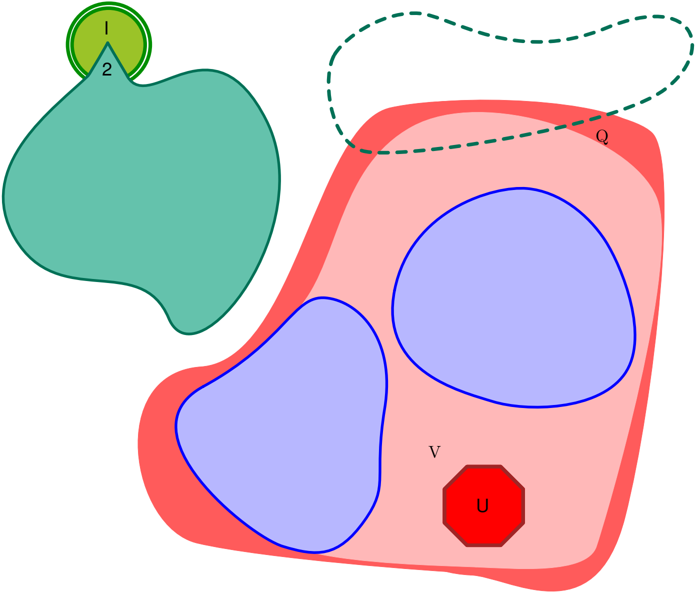
BRAB: intuition



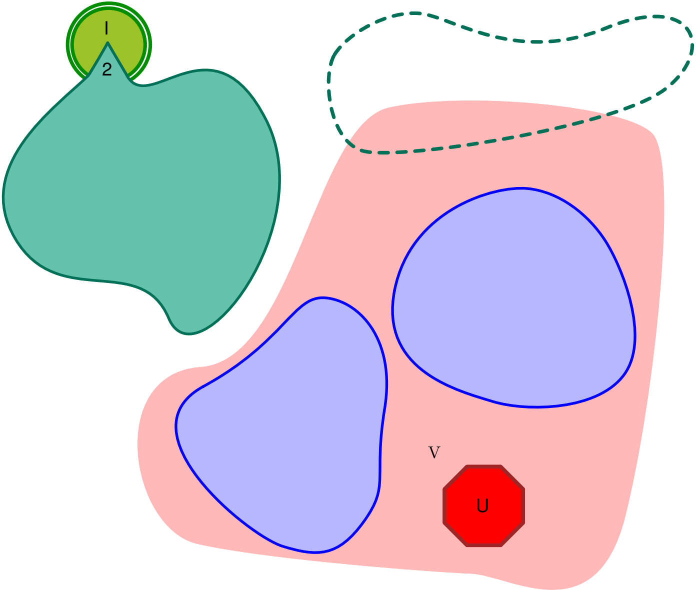
BRAB: intuition



BRAB: intuition



BRAB: intuition



BRAB algorithm

I : inital states U : unsafe states (cubes) $\mathcal{T}$: transitions

---

```
BRAB ():
    B := ∅;    Kind(U) := Orig;    From(U) := U;
    $\mathcal{M} := \texttt{FWD}(d_{max}, k)$;
    while BWDA() = unsafe do
        if Kind(F) = Orig then return unsafe
        B := B ∪ { From(F) };
    return safe
```

BRAB algorithm

I : inital states   U : unsafe states (cubes) $\mathcal{T}$: transitions

```
BWD ():
  V := ∅;
  push(Q, U);
  while not empty(Q) do
    $\varphi$ := pop(Q);
    if $\varphi \wedge I$ sat then return unsafe
    if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then
      V := $V \cup \{\varphi\}$;
      push(Q, $\mathtt{pre}_{\mathcal{T}}(\varphi)$);
  return safe
```

BRAB algorithm

I : inital states U : unsafe states (cubes) $\mathcal{T}$: transitions

```
BWDA ():
    V := ∅;
    push(Q, U);
    while not empty(Q) do
        $\varphi$ := pop(Q);
        if $\varphi \land I$ sat then return unsafe
        if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then
            V := $V \cup \{\varphi\}$;
            push(Q, $\mathrm{Approx}_{\mathcal{T}}(\varphi)$);
    return safe
```

BRAB algorithm

I : inital states U : unsafe states (cubes) $\mathcal{T}$: transitions

---

```
Approx_T (φ):
    foreach $\psi$ in candidates($\varphi$) do
        if $\boxed{\psi \notin \mathrm{B}} \wedge \boxed{\mathcal{M} \not\models \psi}$ then
            Kind($\psi$) := Appr;
            ...
            return $\psi$
        ...
    return pre_T($\varphi$)
```

Example: BRAB on German-*ish*

$\neg\texttt{Exg}$
$\texttt{Cmd} = \epsilon$
$\forall i.\ \texttt{Cache}[i] = \mathsf{I}$
$\neg\texttt{Shr}[i]$

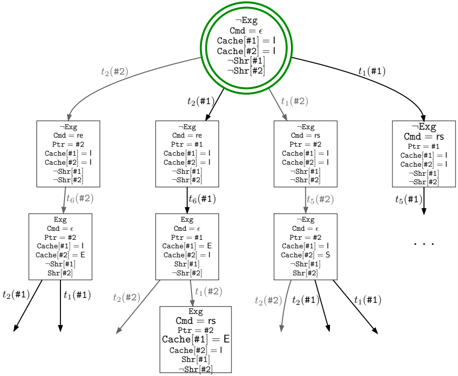
$\exists i \neq j.\ \texttt{Cache}[i] = \mathsf{E}$
$\texttt{Cache}[j] \neq \mathsf{I}$

Example: BRAB on German-*ish*

$\neg$Exg
Cmd $= \epsilon$
Cache$[\#1] = \mathsf{I}$
Cache$[\#2] = \mathsf{I}$
$\neg$Shr$[\#1]$
$\neg$Shr$[\#2]$

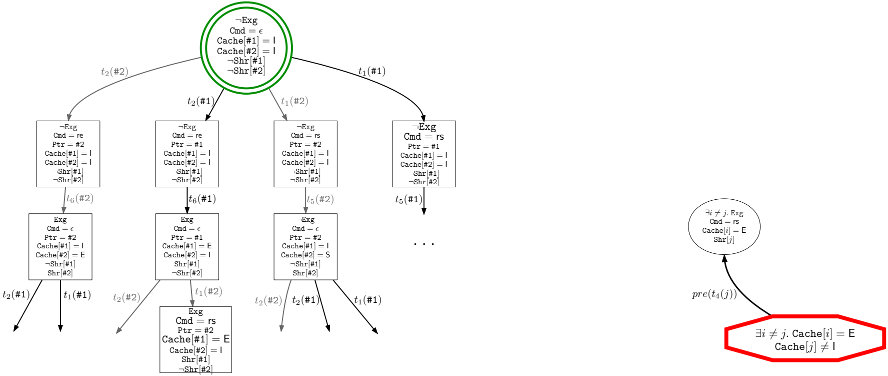
$\exists i \neq j.$ Cache$[i] = \mathsf{E}$
Cache$[j] \neq \mathsf{I}$

Example: BRAB on German-*ish*

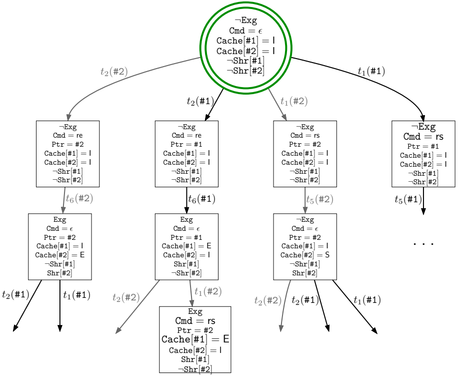

$\neg$Exg
Cmd $= \epsilon$
Cache[#1] $=$ I
Cache[#2] $=$ I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_2(#2)
t_2(#1)
t_1(#2)
t_1(#1)
$\neg$Exg
Cmd $=$ re
Ptr $=$ #2
Cache[#1] $=$ I
Cache[#2] $=$ I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $=$ re
Ptr $=$ #1
Cache[#1] $=$ I
Cache[#2] $=$ I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $=$ rs
Ptr $=$ #2
Cache[#1] $=$ I
Cache[#2] $=$ I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $=$ rs
Ptr $=$ #1
Cache[#1] $=$ I
Cache[#2] $=$ I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_6(#2)
t_6(#1)
t_5(#2)
t_5(#1)
Exg
Cmd $= \epsilon$
Ptr $=$ #2
Cache[#1] $=$ I
Cache[#2] $=$ E
$\neg$Shr[#1]
Shr[#2]
Exg
Cmd $= \epsilon$
Ptr $=$ #1
Cache[#1] $=$ E
Cache[#2] $=$ I
Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $= \epsilon$
Ptr $=$ #2
Cache[#1] $=$ I
Cache[#2] $=$ S
$\neg$Shr[#1]
Shr[#2]
. . .
t_2(#1)
t_1(#1)
t_2(#2)
t_1(#2)
t_2(#2)
t_2(#1)
t_1(#1)
Exg
Cmd $=$ rs
Ptr $=$ #2
Cache[#1] $=$ E
Cache[#2] $=$ I
Shr[#1]
$\neg$Shr[#2]

$\exists i \neq j.$ Cache$[i] =$ E
Cache$[j] \neq$ I

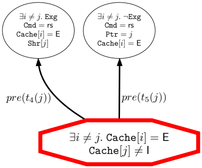
Example: BRAB on German-*ish*


¬Exg
Cmd = ϵ
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t₂(#2)
t₂(#1)
t₁(#2)
t₁(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t₆(#2)
t₆(#1)
t₅(#2)
t₅(#1)
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ϵ
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t₂(#1)
t₁(#1)
t₂(#2)
t₁(#2)
t₂(#2)
t₂(#1)
t₁(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
pre(t₄(j))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

Example: BRAB on German-*ish*

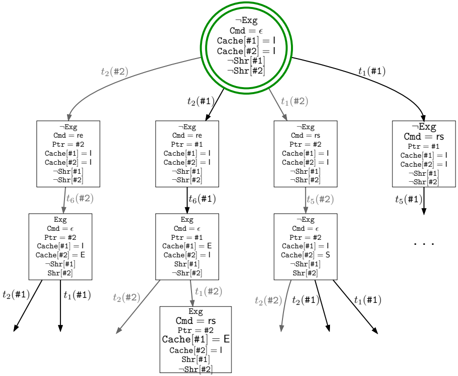


$\neg$Exg
Cmd $= \epsilon$
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_2(#2)
t_2(#1)
t_1(#2)
t_1(#1)
$\neg$Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_6(#2)
t_6(#1)
t_5(#2)
t_5(#1)
Exg
Cmd $= \epsilon$
Ptr = #2
Cache[#1] = I
Cache[#2] = E
$\neg$Shr[#1]
Shr[#2]
Exg
Cmd $= \epsilon$
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $= \epsilon$
Ptr = #2
Cache[#1] = I
Cache[#2] = S
$\neg$Shr[#1]
Shr[#2]
. . .
t_2(#1)
t_1(#1)
t_2(#2)
t_1(#2)
t_2(#2)
t_2(#1)
t_1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]

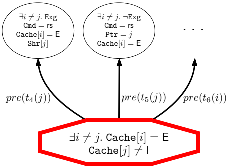


$\exists i \neq j$. Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j$. $\neg$Exg
Cmd = rs
Ptr = j
Cache[i] = E
$pre(t_4(j))$
$pre(t_5(j))$
$\exists i \neq j$. Cache[i] = E
Cache[j] $\neq$ I

Example: BRAB on German-*ish*

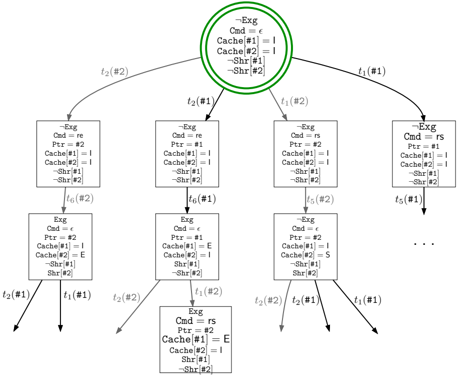


$\neg$Exg
Cmd $= \epsilon$
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_2(#2)
t_2(#1)
t_1(#2)
t_1(#1)
$\neg$Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_6(#2)
t_6(#1)
t_5(#2)
t_5(#1)
Exg
Cmd $= \epsilon$
Ptr = #2
Cache[#1] = I
Cache[#2] = E
$\neg$Shr[#1]
Shr[#2]
Exg
Cmd $= \epsilon$
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $= \epsilon$
Ptr = #2
Cache[#1] = I
Cache[#2] = S
$\neg$Shr[#1]
Shr[#2]
. . .
t_2(#1)
t_1(#1)
t_2(#2)
t_1(#2)
t_2(#2)
t_2(#1)
t_1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]

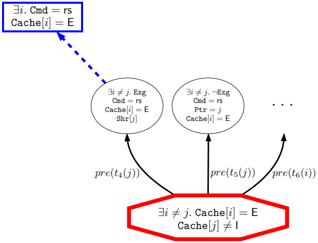


$\exists i \neq j.$ Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j.$ $\neg$Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] $\neq$ I

Example: BRAB on German-*ish*

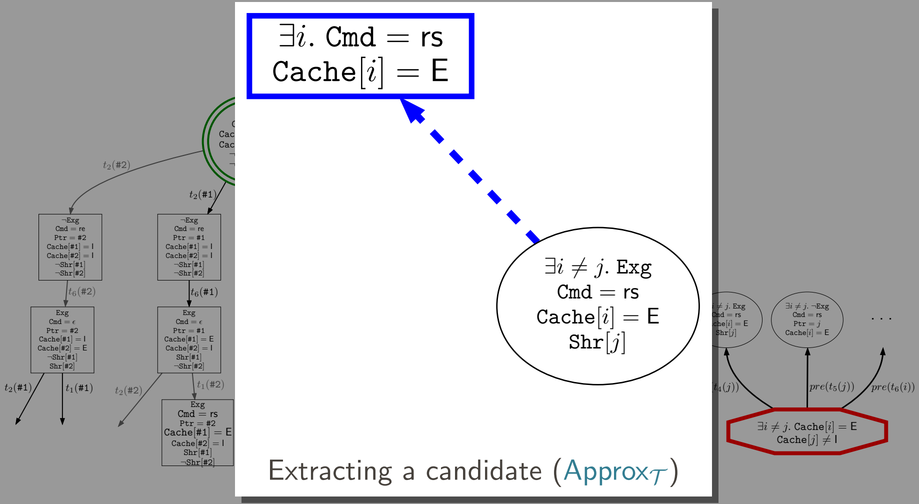


$\neg$Exg
Cmd $= \epsilon$
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_2(#2)
t_2(#1)
t_1(#2)
t_1(#1)
$\neg$Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
t_6(#2)
t_6(#1)
t_5(#2)
t_5(#1)
. . .
Exg
Cmd $= \epsilon$
Ptr = #2
Cache[#1] = I
Cache[#2] = E
$\neg$Shr[#1]
Shr[#2]
Exg
Cmd $= \epsilon$
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
Cmd $= \epsilon$
Ptr = #2
Cache[#1] = I
Cache[#2] = S
$\neg$Shr[#1]
Shr[#2]
t_2(#1)
t_1(#1)
t_2(#2)
t_1(#2)
t_2(#2)
t_2(#1)
t_1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]



$\exists i.$ Cmd = rs
Cache[i] = E
$\exists i \neq j.$ Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j. \neg$Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] $\neq$ I

Example: BRAB on German-*ish*



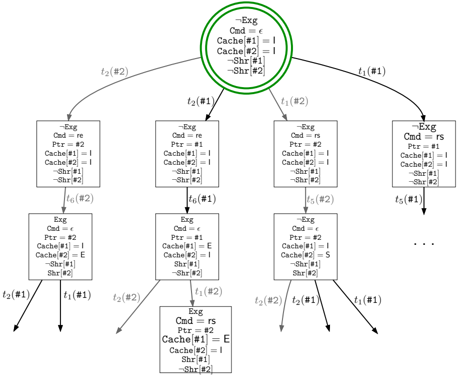
$\exists i.\ \texttt{Cmd} = \textsf{rs}$
$\texttt{Cache}[i] = \textsf{E}$

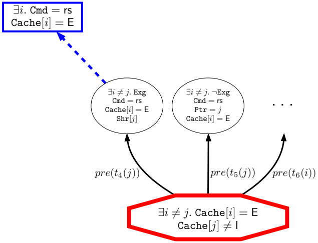
$\exists i \neq j.\ \texttt{Exg}$
$\texttt{Cmd} = \textsf{rs}$
$\texttt{Cache}[i] = \textsf{E}$
$\texttt{Shr}[j]$

$t_2(\#2)$
$t_2(\#1)$
$\neg$Exg Cmd = re Ptr = #2 Cache[#1] = I Cache[#2] = I $\neg$Shr[#1] $\neg$Shr[#2]
$\neg$Exg Cmd = re Ptr = #1 Cache[#1] = I Cache[#2] = I $\neg$Shr[#1] $\neg$Shr[#2]
$t_6(\#2)$
$t_6(\#1)$
Exg Cmd = ϵ Ptr = #2 Cache[#1] = I Cache[#2] = E $\neg$Shr[#1] Shr[#2]
Exg Cmd = ϵ Ptr = #1 Cache[#1] = E Cache[#2] = I Shr[#1] $\neg$Shr[#2]
$t_2(\#1)$
$t_1(\#1)$
$t_2(\#2)$
$t_1(\#2)$
Exg Cmd = rs Ptr = #2 Cache[#1] = E Cache[#2] = I Shr[#1] $\neg$Shr[#2]
$\exists i \neq j.$ Exg Cmd = rs Cache[i] = E Shr[j]
$\exists i \neq j.\ \neg$Exg Cmd = rs Ptr = j Cache[i] = E
. . .
$t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E Cache[j] $\neq$ I

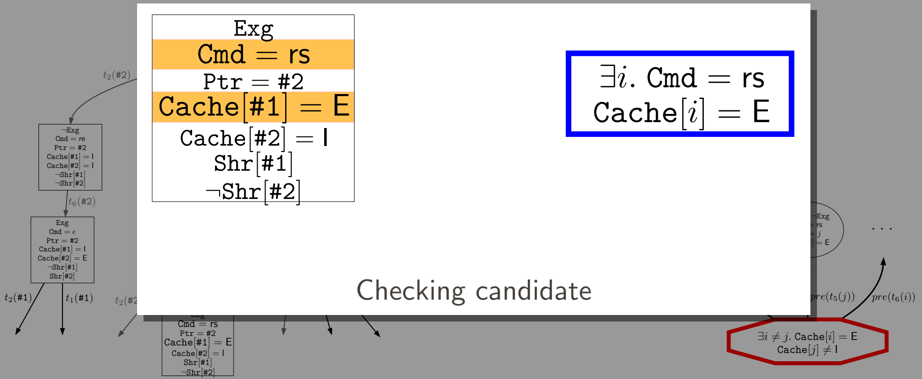
Extracting a candidate ($\text{Approx}_{\mathcal{T}}$)

Example: BRAB on German-*ish*


¬Exg
Cmd = ε
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
$t_2(\#2)$
$t_2(\#1)$
$t_1(\#2)$
$t_1(\#1)$
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
$t_6(\#2)$
$t_6(\#1)$
$t_5(\#2)$
$t_5(\#1)$
Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ε
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
$t_2(\#1)$
$t_1(\#1)$
$t_2(\#2)$
$t_1(\#2)$
$t_2(\#2)$
$t_2(\#1)$
$t_1(\#1)$
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]


$\exists i.$ Cmd = rs
Cache[i] = E
$\exists i \neq j.$ Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j.$ ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] ≠ I

Example: BRAB on German-*ish*



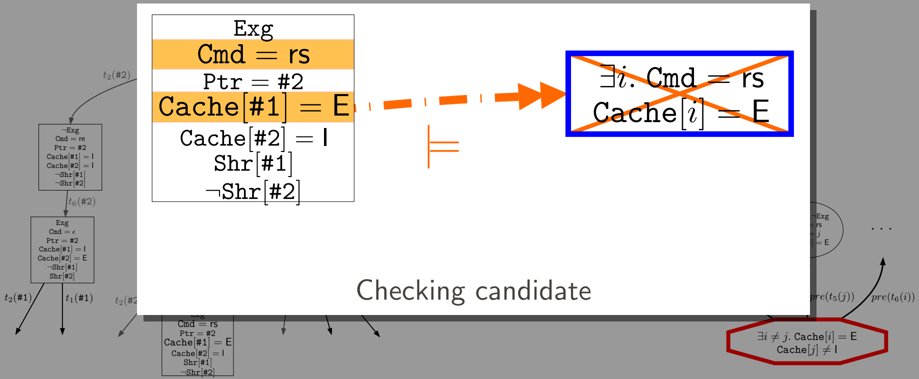
$t_2(\#2)$
$\neg$Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
$\neg$Shr[#1]
$\neg$Shr[#2]
$t_6(\#2)$
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
$\neg$Shr[#1]
Shr[#2]
$t_2(\#1)$
$t_1(\#1)$
$t_2(\#2$
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]
$\neg$Exg
= rs
= j
] = E
. . .
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] $\neq$ I

Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
$\neg$Shr[#2]

$\exists i.$ Cmd = rs
Cache[i] = E

Checking candidate

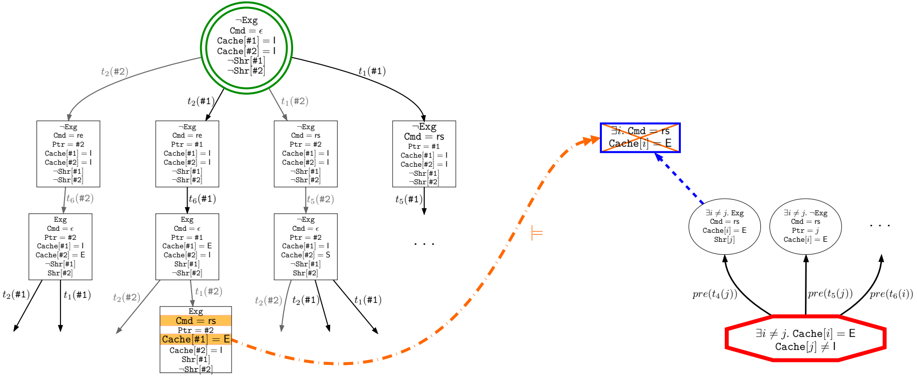
Example: BRAB on German-*ish*


$t_2(\#2)$
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
$t_6(\#2)$
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
$t_2(\#1)$
$t_1(\#1)$
$t_2(\#2)$
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
. . .
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] $\neq$ I

$$\begin{array}{c} \mathtt{Exg} \\ \mathtt{Cmd} = \mathsf{rs} \\ \mathtt{Ptr} = \#2 \\ \mathtt{Cache}[\#1] = \mathsf{E} \\ \mathtt{Cache}[\#2] = \mathsf{I} \\ \mathtt{Shr}[\#1] \\ \neg\mathtt{Shr}[\#2] \end{array} \models \boxed{\xcancel{\begin{array}{c} \exists i.\ \mathtt{Cmd} = \mathsf{rs} \\ \mathtt{Cache}[i] = \mathsf{E} \end{array}}}$$

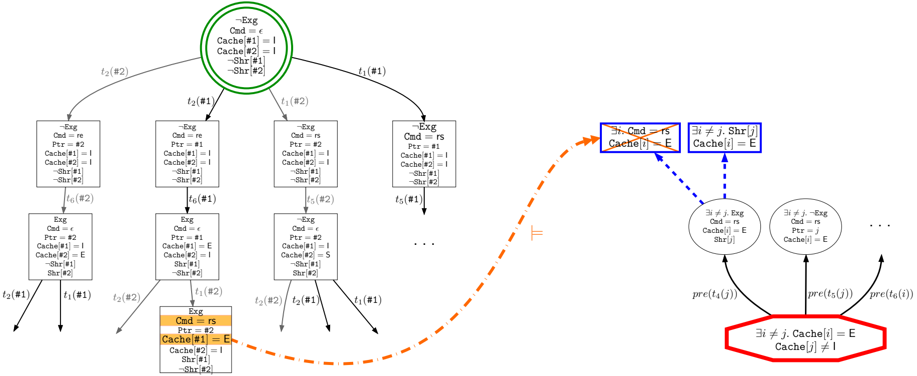
Checking candidate

Example: BRAB on German-*ish*

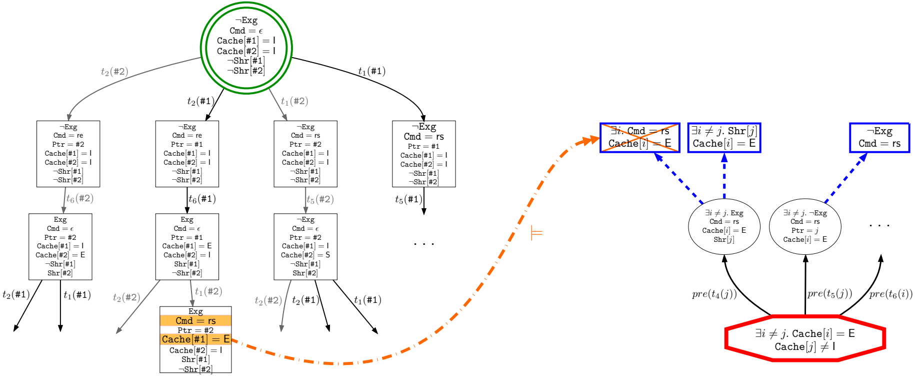


¬Exg
Cmd = ϵ
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t_2(#2)
t_2(#1)
t_1(#2)
t_1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t_6(#2)
t_6(#1)
t_5(#2)
t_5(#1)
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ϵ
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t_2(#1)
t_1(#1)
t_2(#2)
t_1(#2)
t_2(#2)
t_2(#1)
t_1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
$\models$
~~$\exists i$. Cmd = rs~~
~~Cache[i] = E~~
$\exists i \neq j$. Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j$. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j$. Cache[i] = E
Cache[j] $\neq$ I

Example: BRAB on German-*ish*

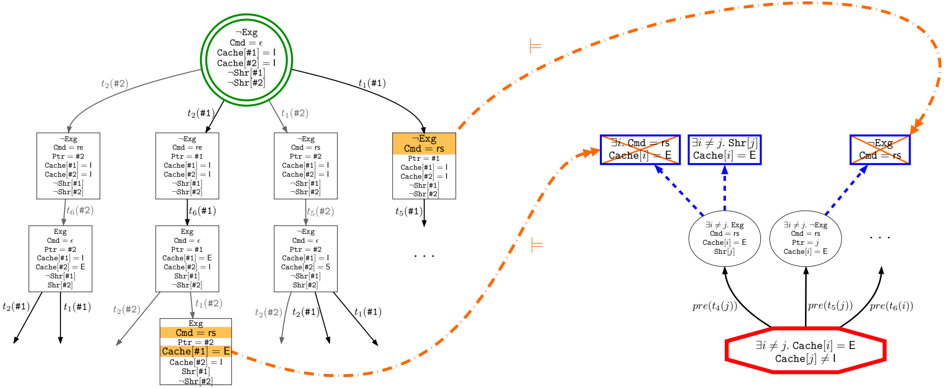

¬Exg
Cmd = ε
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t2(#2)
t2(#1)
t1(#2)
t1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t6(#2)
t6(#1)
t5(#2)
t5(#1)
Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ε
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t2(#1)
t1(#1)
t2(#2)
t1(#2)
t2(#2)
t2(#1)
t1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
~~∃i. Cmd = rs~~
~~Cache[i] = E~~
∃i ≠ j. Shr[j]
Cache[i] = E
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
pre(t4(j))
pre(t5(j))
pre(t6(i))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

Example: BRAB on German-*ish*

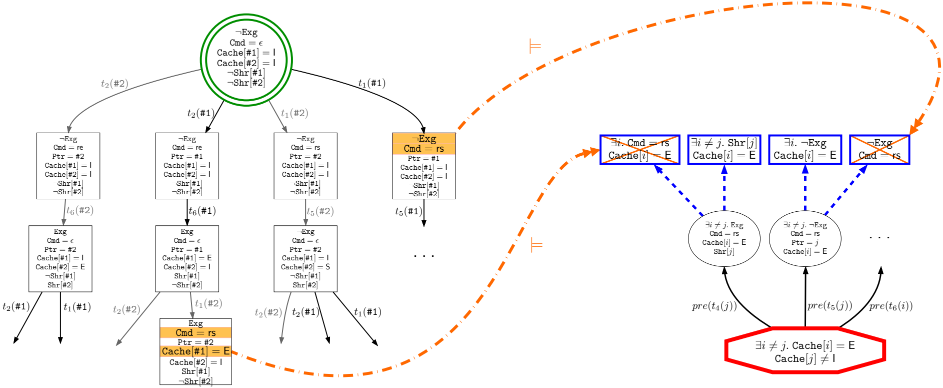


¬Exg
Cmd = ϵ
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t_2(#2)
t_2(#1)
t_1(#2)
t_1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t_6(#2)
t_6(#1)
t_5(#2)
t_5(#1)
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ϵ
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t_2(#1)
t_1(#1)
t_2(#2)
t_1(#2)
t_2(#2)
t_2(#1)
t_1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
$\models$
~~$\exists i.$ Cmd = rs~~
~~Cache[i] = E~~
$\exists i \neq j.$ Shr[j]
Cache[i] = E
¬Exg
Cmd = rs
$\exists i \neq j.$ Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j.$ ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] $\neq$ I

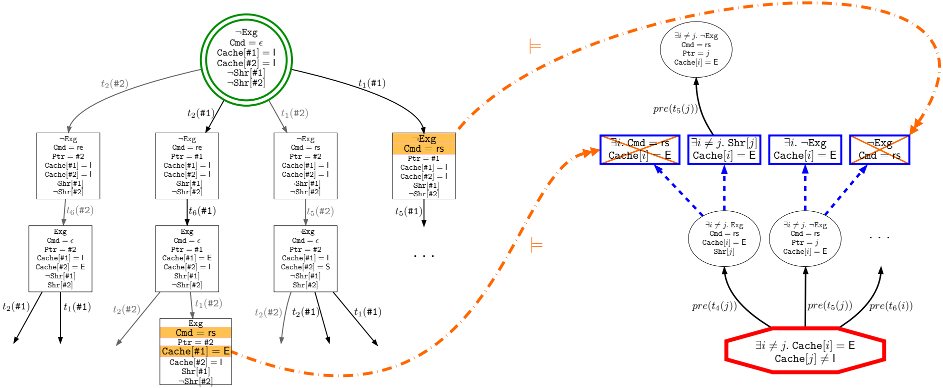
Example: BRAB on German-*ish*


¬Exg
Cmd = ϵ
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t2(#2)
t2(#1)
t1(#2)
t1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t6(#2)
t6(#1)
t5(#2)
t5(#1)
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ϵ
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t2(#1)
t1(#1)
t2(#2)
t1(#2)
t2(#2)
t2(#1)
t1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
⊨
~~∃i. Cmd = rs~~
~~Cache[i] = E~~
∃i ≠ j. Shr[j]
Cache[i] = E
~~¬Exg~~
~~Cmd = rs~~
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
pre(t4(j))
pre(t5(j))
pre(t6(i))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

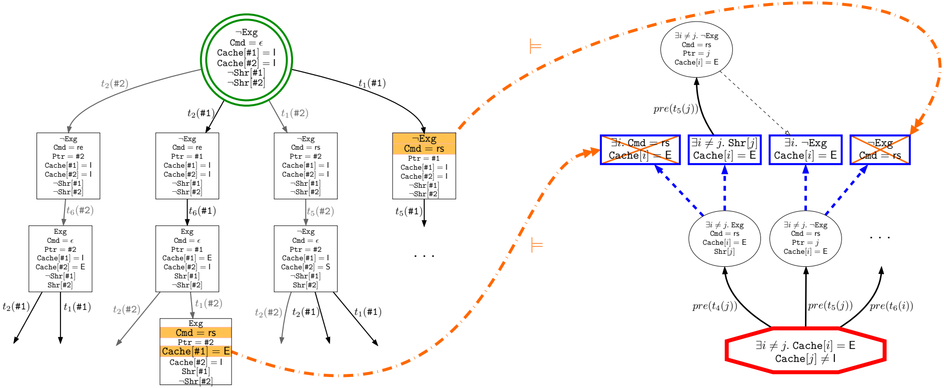
Example: BRAB on German-*ish*


¬Exg
Cmd = ϵ
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t2(#2)
t2(#1)
t1(#2)
t1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t6(#2)
t6(#1)
t5(#2)
t5(#1)
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ϵ
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t2(#1)
t1(#1)
t2(#2)
t1(#2)
t2(#2)
t2(#1)
t1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
⊨
∃i. Cmd = rs
Cache[i] = E
∃i ≠ j. Shr[j]
Cache[i] = E
∃i. ¬Exg
Cache[i] = E
¬Exg
Cmd = rs
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
pre(t4(j))
pre(t5(j))
pre(t6(i))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

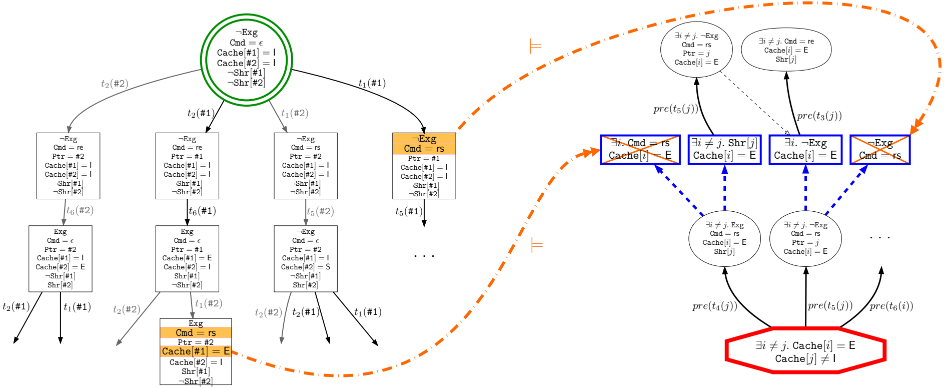
Example: BRAB on German-*ish*


¬Exg Cmd = ε Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
$t_2(\#2)$
$t_2(\#1)$
$t_1(\#2)$
$t_1(\#1)$
¬Exg Cmd = re Ptr = #2 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
¬Exg Cmd = re Ptr = #1 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
¬Exg Cmd = rs Ptr = #2 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
¬Exg Cmd = rs Ptr = #1 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
$t_6(\#2)$
$t_6(\#1)$
$t_5(\#2)$
$t_5(\#1)$
Exg Cmd = ε Ptr = #2 Cache[#1] = I Cache[#2] = E ¬Shr[#1] Shr[#2]
Exg Cmd = ε Ptr = #1 Cache[#1] = E Cache[#2] = I Shr[#1] ¬Shr[#2]
¬Exg Cmd = ε Ptr = #2 Cache[#1] = I Cache[#2] = S ¬Shr[#1] Shr[#2]
. . .
$t_2(\#1)$
$t_1(\#1)$
$t_2(\#2)$
$t_1(\#2)$
$t_2(\#2)$
$t_2(\#1)$
$t_1(\#1)$
Exg Cmd = rs Ptr = #2 Cache[#1] = E Cache[#2] = I Shr[#1] ¬Shr[#2]
⊨
⊨
∃i ≠ j. ¬Exg Cmd = rs Ptr = j Cache[i] = E
$pre(t_5(j))$
~~∃i. Cmd = rs Cache[i] = E~~
∃i ≠ j. Shr[j] Cache[i] = E
∃i. ¬Exg Cache[i] = E
~~¬Exg Cmd = rs~~
∃i ≠ j. Exg Cmd = rs Cache[i] = E Shr[j]
∃i ≠ j. ¬Exg Cmd = rs Ptr = j Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
∃i ≠ j. Cache[i] = E Cache[j] ≠ I

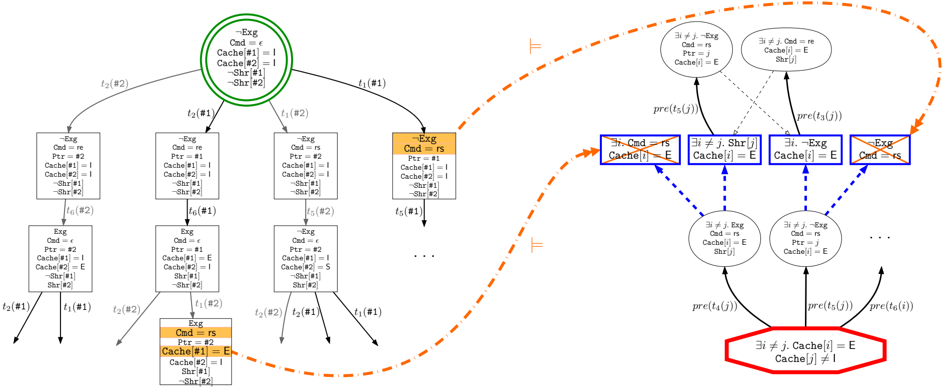
Example: BRAB on German-*ish*


¬Exg
Cmd = ε
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t2(#2)
t2(#1)
t1(#2)
t1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t6(#2)
t6(#1)
t5(#2)
t5(#1)
Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ε
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t2(#1)
t1(#1)
t2(#2)
t1(#2)
t2(#2)
t2(#1)
t1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
⊨
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
pre(t5(j))
∃i. Cmd = rs
Cache[i] = E
∃i ≠ j. Shr[j]
Cache[i] = E
∃i. ¬Exg
Cache[i] = E
¬Exg
Cmd = rs
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
pre(t4(j))
pre(t5(j))
pre(t6(i))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

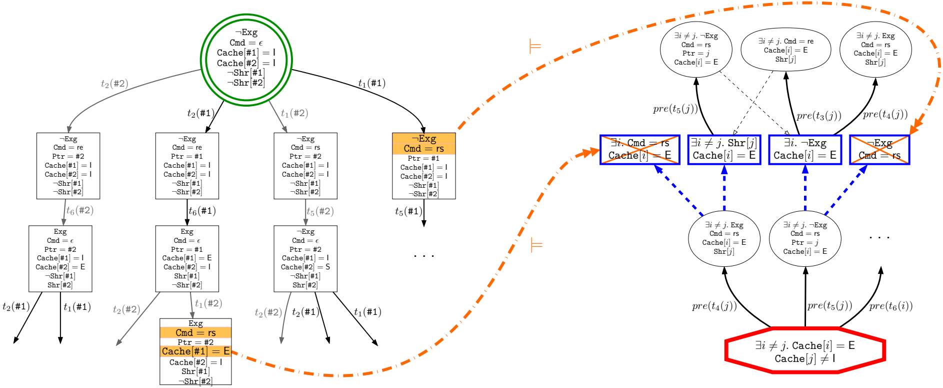
Example: BRAB on German-*ish*


¬Exg Cmd = ε Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
t₂(#2)
t₂(#1)
t₁(#2)
t₁(#1)
¬Exg Cmd = re Ptr = #2 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
¬Exg Cmd = re Ptr = #1 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
¬Exg Cmd = rs Ptr = #2 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
¬Exg Cmd = rs Ptr = #1 Cache[#1] = I Cache[#2] = I ¬Shr[#1] ¬Shr[#2]
t₆(#2)
t₆(#1)
t₅(#2)
t₅(#1)
Exg Cmd = ε Ptr = #2 Cache[#1] = I Cache[#2] = E ¬Shr[#1] Shr[#2]
Exg Cmd = ε Ptr = #1 Cache[#1] = E Cache[#2] = I Shr[#1] ¬Shr[#2]
¬Exg Cmd = ε Ptr = #2 Cache[#1] = I Cache[#2] = S ¬Shr[#1] Shr[#2]
...
t₂(#1)
t₁(#1)
t₂(#2)
t₁(#2)
t₂(#2)
t₂(#1)
t₁(#1)
Exg Cmd = rs Ptr = #2 Cache[#1] = E Cache[#2] = I Shr[#1] ¬Shr[#2]
⊨
⊨
∃i ≠ j. ¬Exg Cmd = rs Ptr = j Cache[i] = E
∃i ≠ j. Cmd = re Cache[i] = E Shr[j]
pre(t₅(j))
pre(t₃(j))
~~∃i. Cmd = rs Cache[i] = E~~
∃i ≠ j. Shr[j] Cache[i] = E
∃i. ¬Exg Cache[i] = E
~~¬Exg Cmd = rs~~
∃i ≠ j. Exg Cmd = rs Cache[i] = E Shr[j]
∃i ≠ j. ¬Exg Cmd = rs Ptr = j Cache[i] = E
...
pre(t₄(j))
pre(t₅(j))
pre(t₆(i))
∃i ≠ j. Cache[i] = E Cache[j] ≠ I

Example: BRAB on German-*ish*


¬Exg
Cmd = ϵ
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t2(#2)
t2(#1)
t1(#2)
t1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t6(#2)
t6(#1)
t5(#2)
t5(#1)
Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ϵ
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ϵ
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t2(#1)
t1(#1)
t2(#2)
t1(#2)
t2(#2)
t2(#1)
t1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
⊨
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
∃i ≠ j. Cmd = re
Cache[i] = E
Shr[j]
pre(t5(j))
pre(t3(j))
~~∃i. Cmd = rs~~
~~Cache[i] = E~~
∃i ≠ j. Shr[j]
Cache[i] = E
∃i. ¬Exg
Cache[i] = E
~~¬Exg~~
~~Cmd = rs~~
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
pre(t4(j))
pre(t5(j))
pre(t6(i))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

Example: BRAB on German-*ish*



$\neg$Exg
$\mathtt{Cmd} = \epsilon$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\neg\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$t_2(\#2)$ $t_2(\#1)$ $t_1(\#2)$ $t_1(\#1)$

$\neg$Exg
$\mathtt{Cmd} = \mathsf{re}$
$\mathtt{Ptr} = \#2$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\neg\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$\neg$Exg
$\mathtt{Cmd} = \mathsf{re}$
$\mathtt{Ptr} = \#1$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\neg\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$\neg$Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Ptr} = \#2$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\neg\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$\neg$Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Ptr} = \#1$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\neg\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$t_6(\#2)$ $t_6(\#1)$ $t_5(\#2)$ $t_5(\#1)$

Exg
$\mathtt{Cmd} = \epsilon$
$\mathtt{Ptr} = \#2$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{E}$
$\neg\mathtt{Shr}[\#1]$
$\mathtt{Shr}[\#2]$

Exg
$\mathtt{Cmd} = \epsilon$
$\mathtt{Ptr} = \#1$
$\mathtt{Cache}[\#1] = \mathsf{E}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$\neg$Exg
$\mathtt{Cmd} = \epsilon$
$\mathtt{Ptr} = \#2$
$\mathtt{Cache}[\#1] = \mathsf{I}$
$\mathtt{Cache}[\#2] = \mathsf{S}$
$\neg\mathtt{Shr}[\#1]$
$\mathtt{Shr}[\#2]$

. . .

$t_2(\#1)$ $t_1(\#1)$ $t_2(\#2)$ $t_1(\#2)$ $t_2(\#2)$ $t_2(\#1)$ $t_1(\#1)$

Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Ptr} = \#2$
$\mathtt{Cache}[\#1] = \mathsf{E}$
$\mathtt{Cache}[\#2] = \mathsf{I}$
$\mathtt{Shr}[\#1]$
$\neg\mathtt{Shr}[\#2]$

$\models$

$\models$

$\exists i \neq j.\ \neg$Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Ptr} = j$
$\mathtt{Cache}[i] = \mathsf{E}$

$\exists i \neq j.\ \mathtt{Cmd} = \mathsf{re}$
$\mathtt{Cache}[i] = \mathsf{E}$
$\mathtt{Shr}[j]$

$\exists i \neq j.$ Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Cache}[i] = \mathsf{E}$
$\mathtt{Shr}[j]$

$pre(t_5(j))$ $pre(t_3(j))$ $pre(t_4(j))$

~~$\exists i.\ \mathtt{Cmd} = \mathsf{rs}$~~
~~$\mathtt{Cache}[i] = \mathsf{E}$~~

$\exists i \neq j.\ \mathtt{Shr}[j]$
$\mathtt{Cache}[i] = \mathsf{E}$

$\exists i.\ \neg$Exg
$\mathtt{Cache}[i] = \mathsf{E}$

~~$\neg$Exg~~
~~$\mathtt{Cmd} = \mathsf{rs}$~~

$\exists i \neq j.$ Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Cache}[i] = \mathsf{E}$
$\mathtt{Shr}[j]$

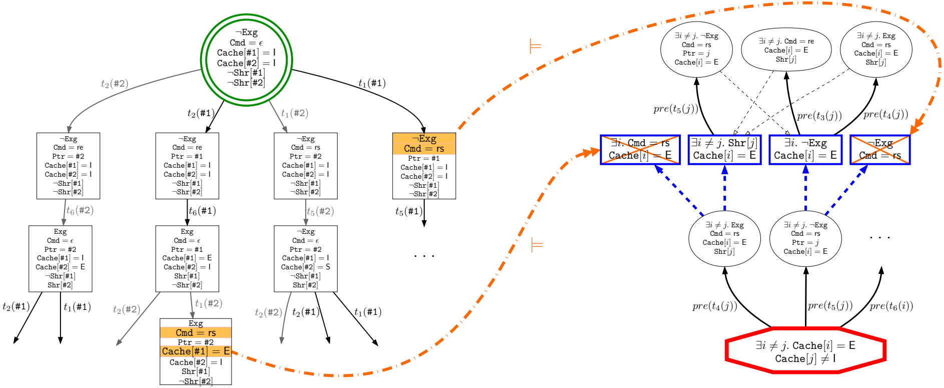
$\exists i \neq j.\ \neg$Exg
$\mathtt{Cmd} = \mathsf{rs}$
$\mathtt{Ptr} = j$
$\mathtt{Cache}[i] = \mathsf{E}$

. . .

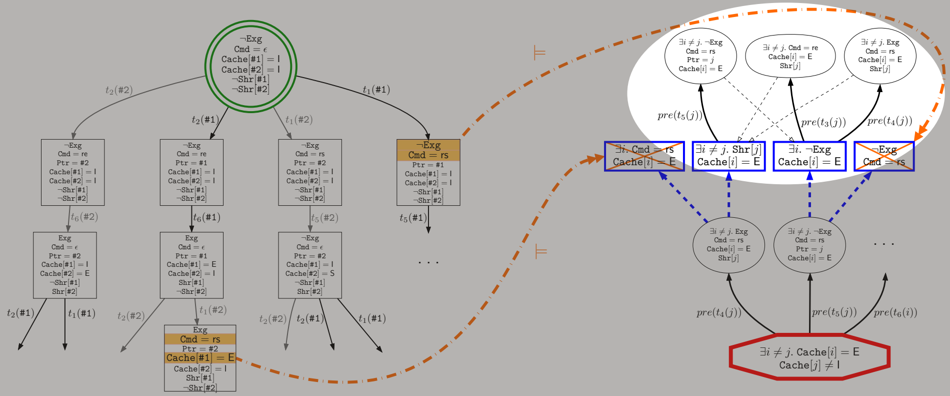
$pre(t_4(j))$ $pre(t_5(j))$ $pre(t_6(i))$

$\exists i \neq j.\ \mathtt{Cache}[i] = \mathsf{E}$
$\mathtt{Cache}[j] \neq \mathsf{I}$

Example: BRAB on German-*ish*


¬Exg
Cmd = ε
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
$t_2(\#2)$
$t_2(\#1)$
$t_1(\#2)$
$t_1(\#1)$
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
$t_6(\#2)$
$t_6(\#1)$
$t_5(\#2)$
$t_5(\#1)$
Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ε
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
$t_2(\#1)$
$t_1(\#1)$
$t_2(\#2)$
$t_1(\#2)$
$t_2(\#2)$
$t_2(\#1)$
$t_1(\#1)$
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
⊨
$\exists i \neq j.\ \neg$Exg
Cmd = rs
Ptr = j
Cache[i] = E
$\exists i \neq j.$ Cmd = re
Cache[i] = E
Shr[j]
$\exists i \neq j.$ Exg
Cmd = rs
Cache[i] = E
Shr[j]
$pre(t_5(j))$
$pre(t_3(j))$
$pre(t_4(j))$
~~$\exists i.$ Cmd = rs~~
~~Cache[i] = E~~
$\exists i \neq j.$ Shr[j]
Cache[i] = E
$\exists i.\ \neg$Exg
Cache[i] = E
~~¬Exg~~
~~Cmd = rs~~
$\exists i \neq j.$ Exg
Cmd = rs
Cache[i] = E
Shr[j]
$\exists i \neq j.\ \neg$Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
$pre(t_4(j))$
$pre(t_5(j))$
$pre(t_6(i))$
$\exists i \neq j.$ Cache[i] = E
Cache[j] ≠ I

Example: BRAB on German-*ish*


¬Exg
Cmd = ε
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t2(#2)
t2(#1)
t1(#2)
t1(#1)
¬Exg
Cmd = re
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = re
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #2
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
¬Exg
Cmd = rs
Ptr = #1
Cache[#1] = I
Cache[#2] = I
¬Shr[#1]
¬Shr[#2]
t6(#2)
t6(#1)
t5(#2)
t5(#1)
Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = E
¬Shr[#1]
Shr[#2]
Exg
Cmd = ε
Ptr = #1
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
¬Exg
Cmd = ε
Ptr = #2
Cache[#1] = I
Cache[#2] = S
¬Shr[#1]
Shr[#2]
. . .
t2(#1)
t1(#1)
t2(#2)
t1(#2)
t2(#2)
t2(#1)
t1(#1)
Exg
Cmd = rs
Ptr = #2
Cache[#1] = E
Cache[#2] = I
Shr[#1]
¬Shr[#2]
⊨
⊨
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
∃i ≠ j. Cmd = re
Cache[i] = E
Shr[j]
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
pre(t5(j))
pre(t3(j))
pre(t4(j))
∃i. Cmd = rs
Cache[i] = E
∃i ≠ j. Shr[j]
Cache[i] = E
∃i. ¬Exg
Cache[i] = E
¬Exg
Cmd = rs
∃i ≠ j. Exg
Cmd = rs
Cache[i] = E
Shr[j]
∃i ≠ j. ¬Exg
Cmd = rs
Ptr = j
Cache[i] = E
. . .
pre(t4(j))
pre(t5(j))
pre(t6(i))
∃i ≠ j. Cache[i] = E
Cache[j] ≠ I

Some benchmarks

	BRAB	Cubicle	CMurphi		
Szymanski_at	**0.14s**	0.30s	8.04s (8)	5m12s (10)	2h50m (12)
German_Baukus	**0.25s**	7.03s	0.74s (4)	19m35s (8)	4h49m (10)
German.CTC	**0.29s**	3m23s	1.83s (4)	43m46s (8)	12h35m (10)
German_pfs	**0.34s**	3m58s	0.99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	**2m17s**	2h01m	5.68s (4)	2m58s (5)	1h36m (6)
Szymanski_na	**0.19s**	T.O.	0.88s (4)	8m25s (6)	7h08m (8)
Flash_nodata	**0.36s**	O.M.	4.86s (3)	3m33s (4)	2h46m (5)
Flash	**5m40s**	O.M.	1m27s (3)	2h15m (4)	O.M. (5)

O.M. > 20 GB

T.O. > 20 h