Inférence d'invariants pour le model checking de systèmes paramétrés

Alain Mebsout

LRI, Université Paris-Sud

Soutenance de thèse

29 septembre 2014

Direction : Sylvain Conchon et Fatiha Zaïdi

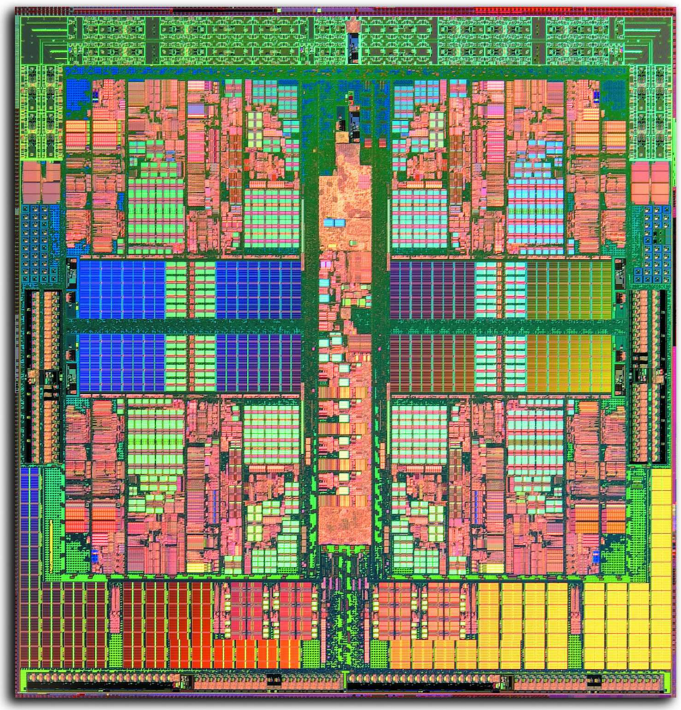
Contexte

Microprocesseurs actuels : objets extrêmement complexes

- mémoire distribuée avec caches
- modèles de mémoire faible
- pipelining d'instructions
- prédiction de branchements
- systèmes sur puce (SoC)
- ...

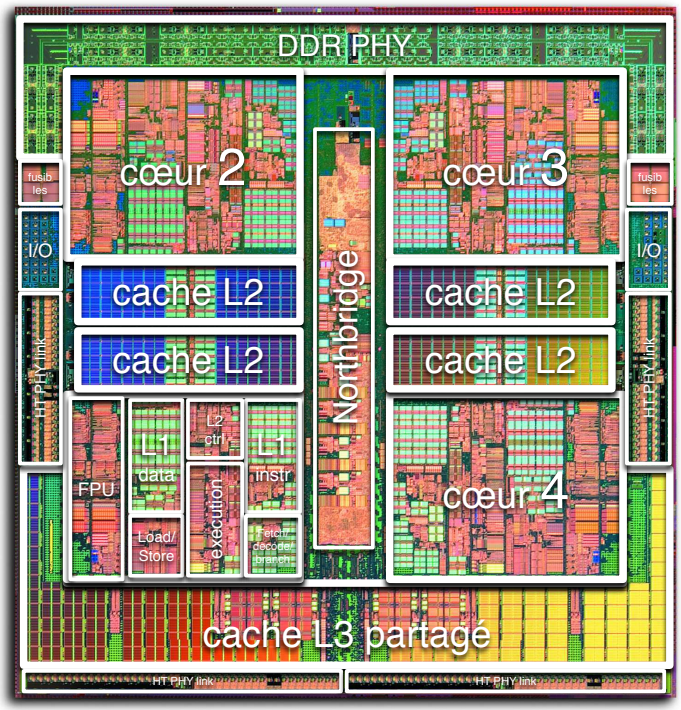
Leur sûreté et leur correction est un enjeu majeur.

Modèles



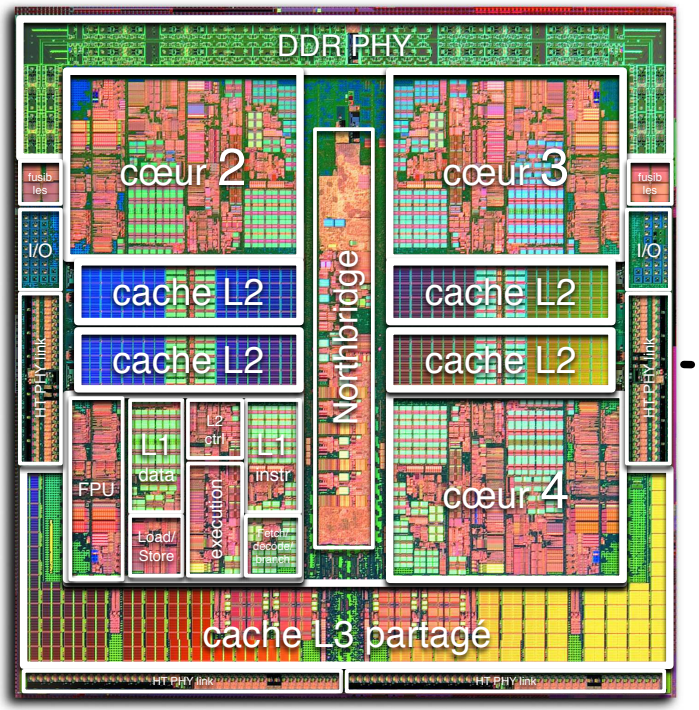
(Quadri-cœurs AMD Opteron « Barcelona » , source AMD)

Modèles



(Quadri-cœurs AMD Opteron « Barcelona » , source AMD)

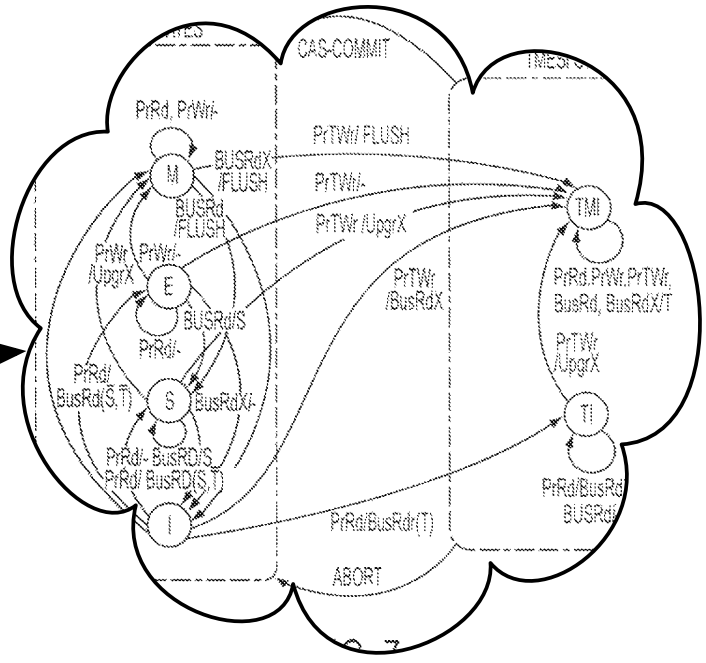
Modèles



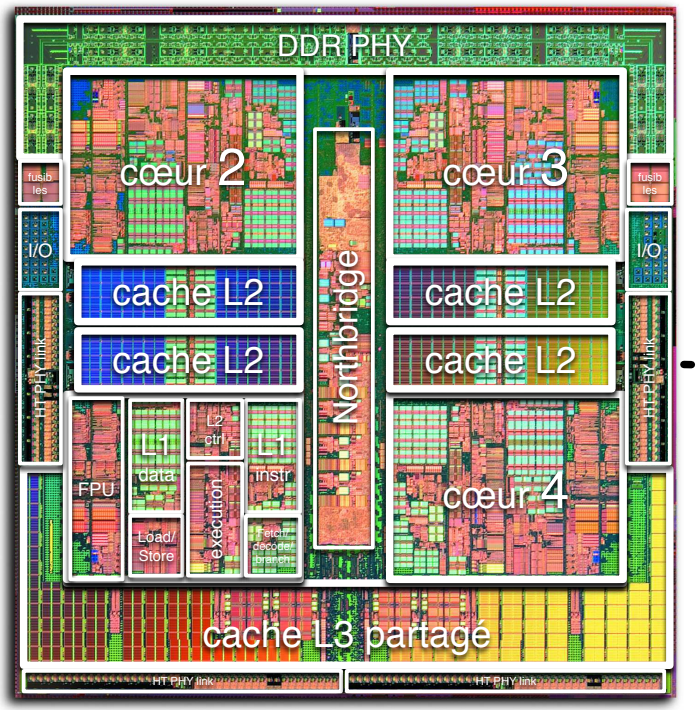
(Quadri-cœurs AMD Opteron « Barcelona » , source AMD)

abstraction

modèle



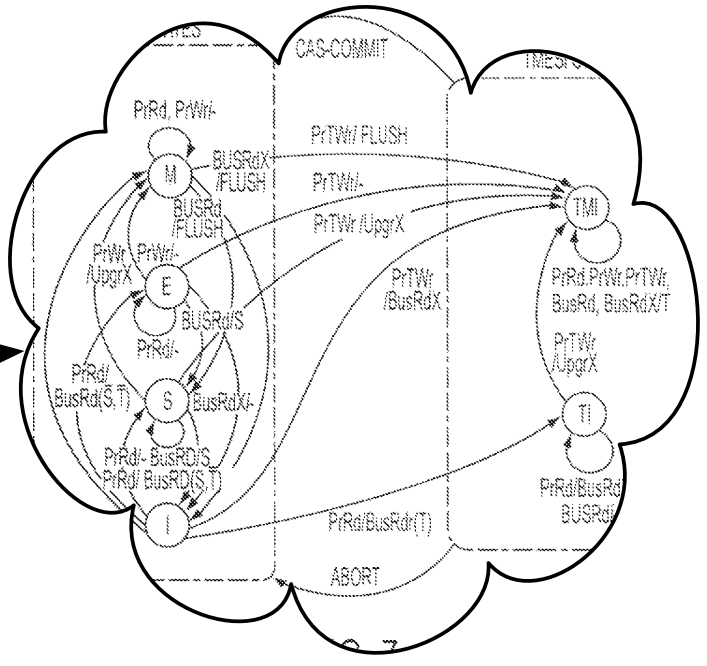
Modèles



(Quadri-cœurs AMD Opteron « Barcelona » , source AMD)

abstraction

modèle

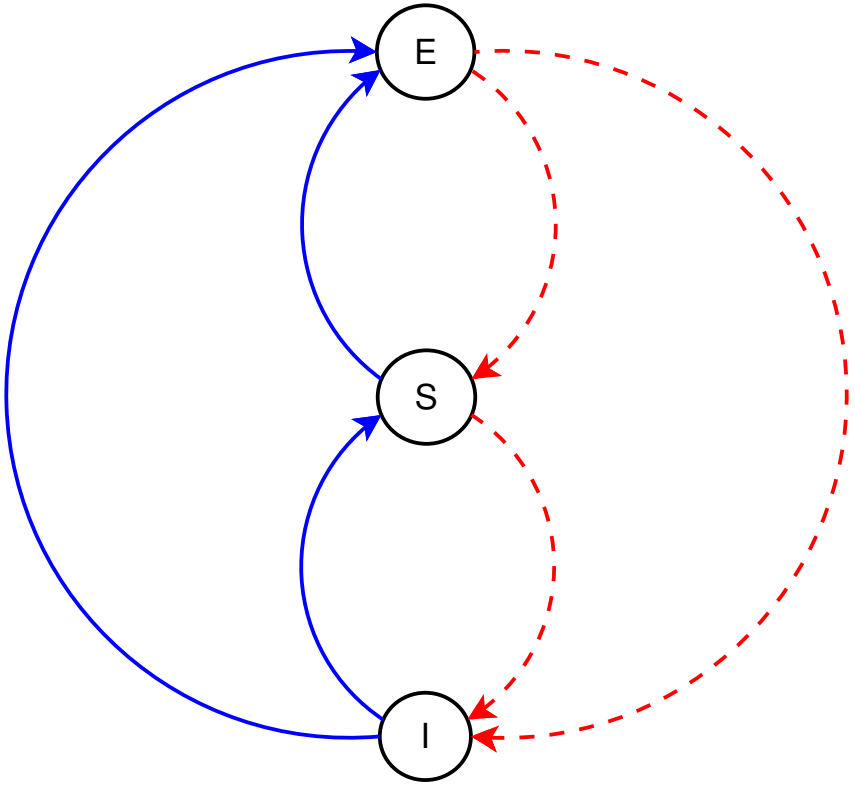


réel — abstrait

expressivité
décidabilité
efficacité

Exemple : protocole de cohérence de cache German-*esque*

- Steven German, IBM
- cohérence de cache par répertoire
  - I (Invalid) : pas de copie
  - S (Shared) : copie en lecture
  - E (Exclusive) : copie en écriture



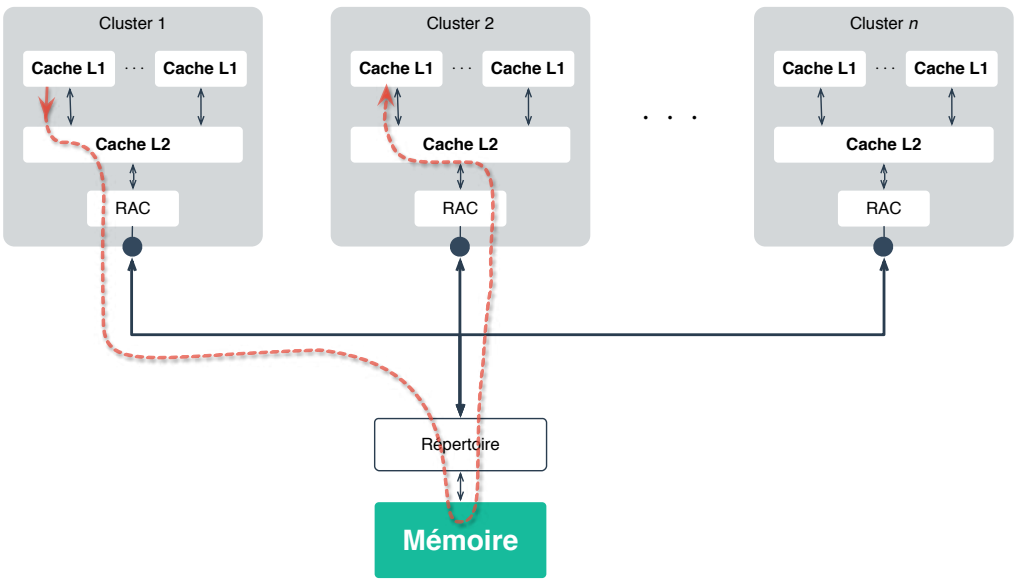
Protocoles de cohérence de cache

Académiques : Xerox Dragon, Futurebus, Firefly, German

$\rightarrow$ milliers d'états

Industriels : Flash, LCP (Larabee), hiérarchiques

$\rightarrow$ millions à milliards d'états



Systèmes paramétrés

Protocoles conçus pour un nombre arbitraire de cœurs

Systèmes paramétrés :

- réplication de composants
- taille pas connue à l'avance
- trop de composants

Model checking

- Décrire le système à l'aide d'un modèle (+/- abstrait)
- Vérifier que le modèle satisfait certaines propriétés
- Avantages :
  - approche complètement automatique
  - très utile pour débuger (traces) même avec modèles très abstraits

Model checking

Modèles

- circuits (RTL)
- réseaux de Petri
- systèmes de transition
  - à états finis
  - à états infinis
  - paramétrés
  - temporisés
  - hybrides
- ...

Propriétés

- sûreté
- absence d'interblocage
- vivacité
- équité
- ...

Model checking

Modèles

- circuits (RTL)
- réseaux de Petri
- systèmes de transition
  - à états finis
  - à états infinis
  - paramétrés
  - temporisés
  - hybrides
- ...

Propriétés

- sûreté
- absence d'interblocage
- vivacité
- équité
- ...

Techniques de model checking

- Énumératif (Murφ, Spin) :
  - exploration exhaustive
  - compactage
  - réductions d'ordres partiels

- Symbolique (NuSMV, TReX, MCMT, Uppaal) :
  - ensembles (potentiellement infinis) d'états
  - structures ad-hoc
  - formules logiques

Techniques de model checking

- Énumératif (Murφ, Spin) :
  - exploration exhaustive
  - compactage
  - réductions d'ordres partiels

- Symbolique (NuSMV, TReX, MCMT, Uppaal) :
  - ensembles (potentiellement infinis) d'états
  - structures ad-hoc
  - formules logiques

Cubicle (UPSud / Intel)

- Model checker symbolique pour systèmes paramétrés
- Manipule des formules logiques
- Solveur SMT Alt-Ergo
- Université Paris-Sud / Intel Corporation

Plan

1. Model checking de systèmes paramétrés avec Cubicle

2. Inférence d'invariants : BRAB

3. Certification

Model checking de systèmes paramétrés avec Cubicle

Challenge

Comment prouver la sûreté d'un système avec un nombre arbitraire de composants ?

= système paramétré

Challenge

Comment prouver la sûreté d'un système avec un nombre arbitraire de composants ?

» automatiquement

Challenge

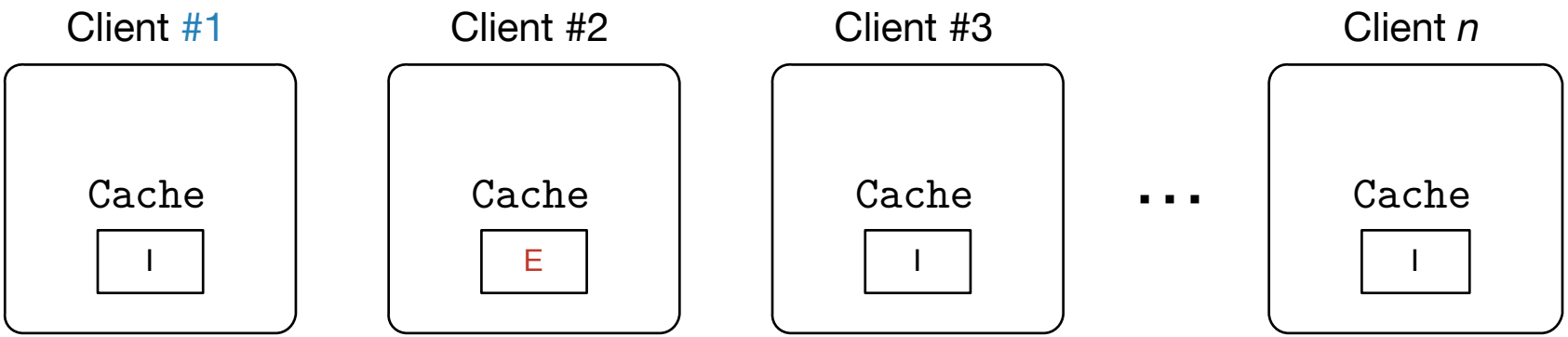
Comment prouver la sûreté d'un système avec un nombre arbitraire de composants ?

» automatiquement

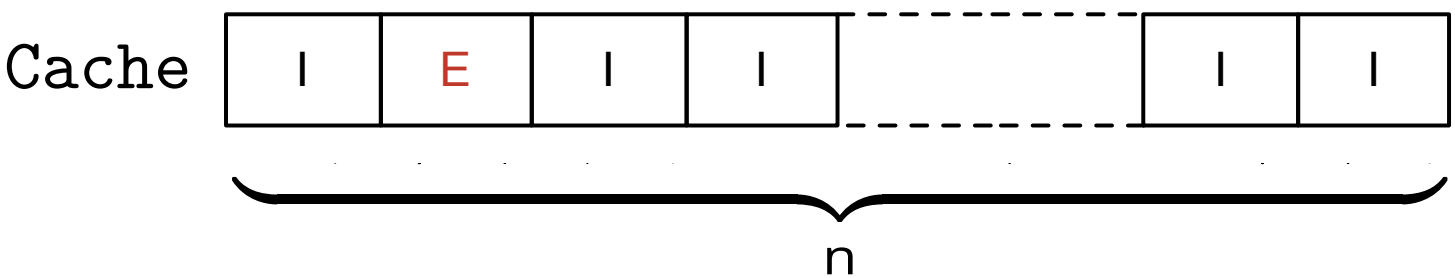
Cadre restreint : systèmes de transition à tableaux

[Ghilardi & Ranise, 2009]

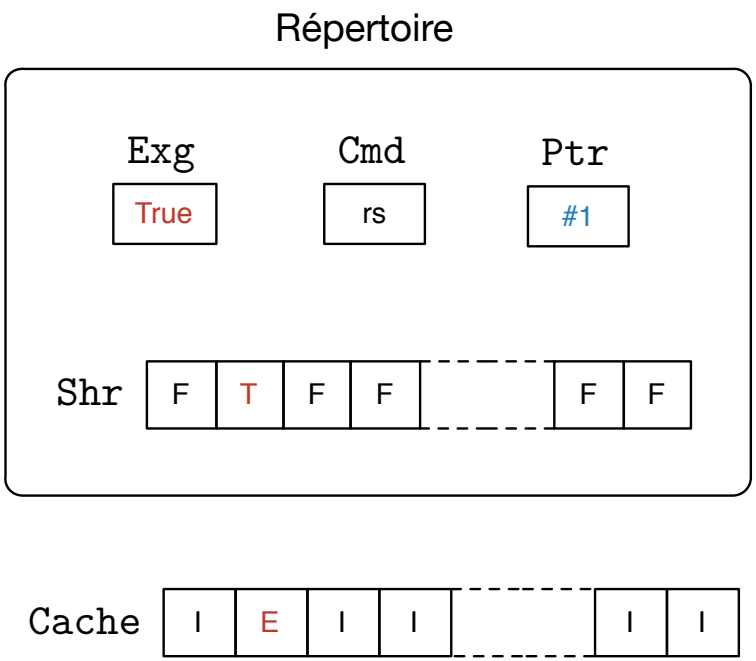
German-*esque* : système à tableaux



German-*esque* : système à tableaux

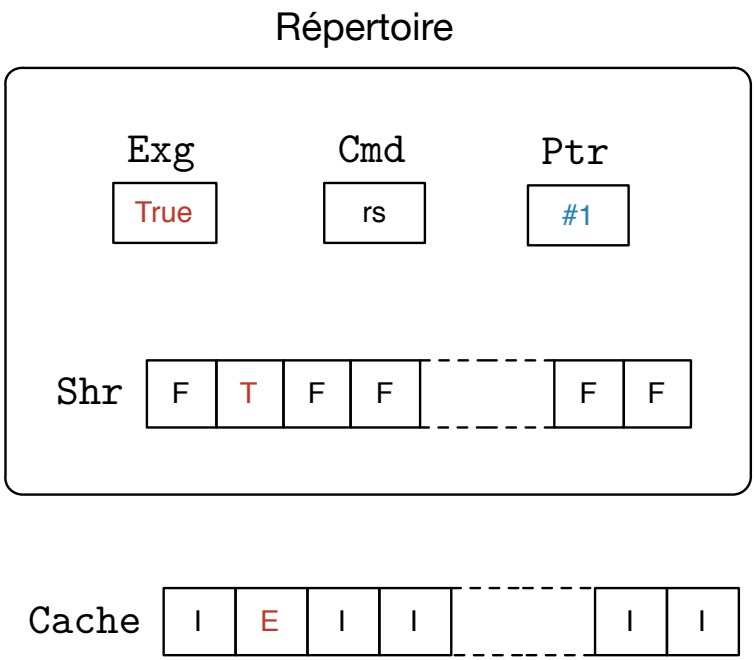


Exemple : protocole de cohérence de cache German-*esque*




```
type msg = Epsilon | RS | RE
type state = I | S | E

(* Directory *)
var Exg : bool
var Cmd : msg
var Ptr : proc
array Shr[proc] : bool

(* Client *)
array Cache[proc] : state
```

Exemple : protocole de cohérence de cache German-*esque*



États initiaux :

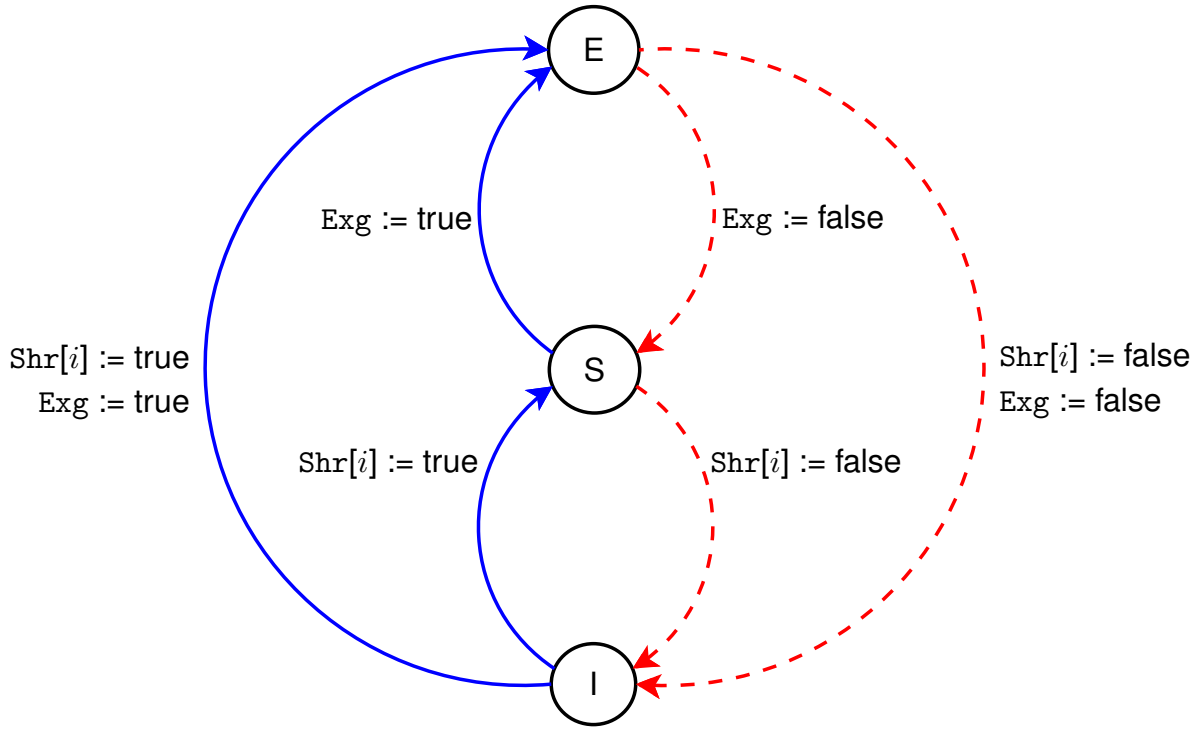
$$\forall i.\ \texttt{Cache}[i] = \texttt{I} \wedge \neg \texttt{Shr}[i] \wedge \neg \texttt{Exg} \wedge \texttt{Cmd} = \epsilon$$

États dangereux : (cubes)

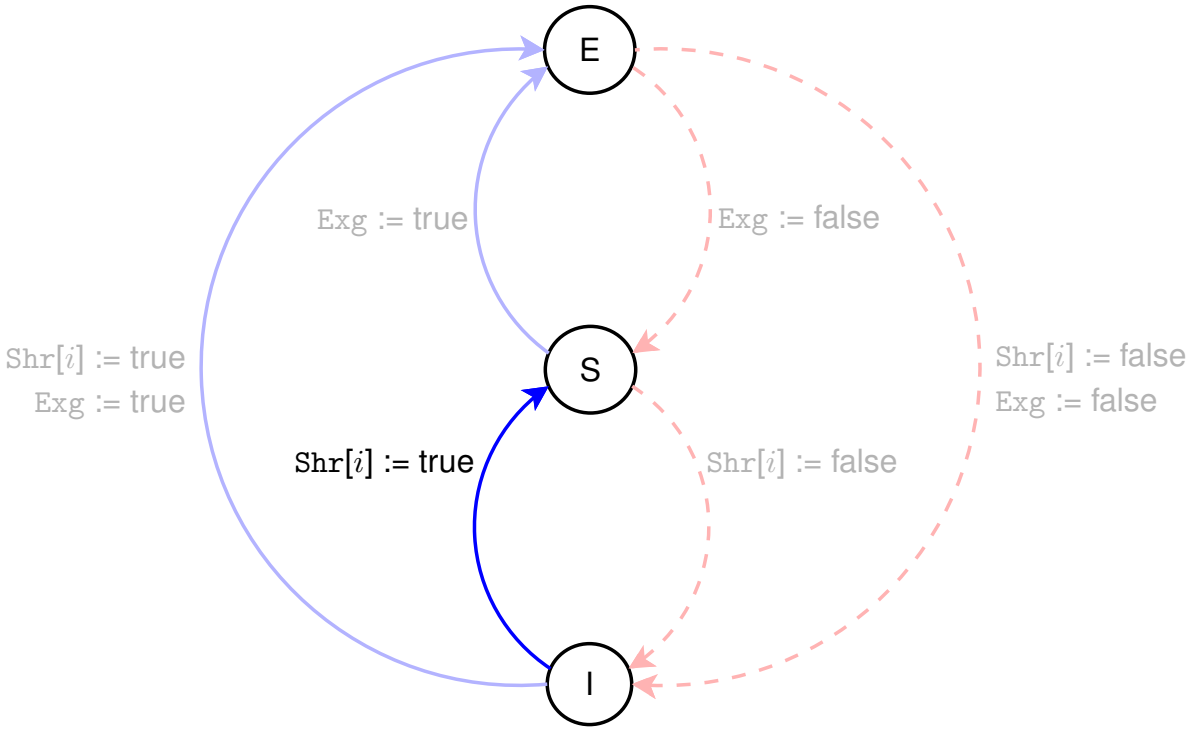
$$\exists i, j.\ i \neq j \wedge \texttt{Cache}[i] = \texttt{E} \wedge \texttt{Cache}[j] \neq \texttt{I} \ ?$$

```
type msg = Epsilon | RS | RE
type state = I | S | E

(* Directory *)
var Exg : bool
var Cmd : msg
var Ptr : proc
array Shr[proc] : bool

(* Client *)
array Cache[proc] : state

init (i) {
    Cache[i] = I && Shr[i] = False &&
    Exg = False && Cmd = Epsilon
}


unsafe (i j) {
    Cache[i] = E && Cache[j] <> I
}
```

Exemple : protocole de cohérence de cache German-*esque*

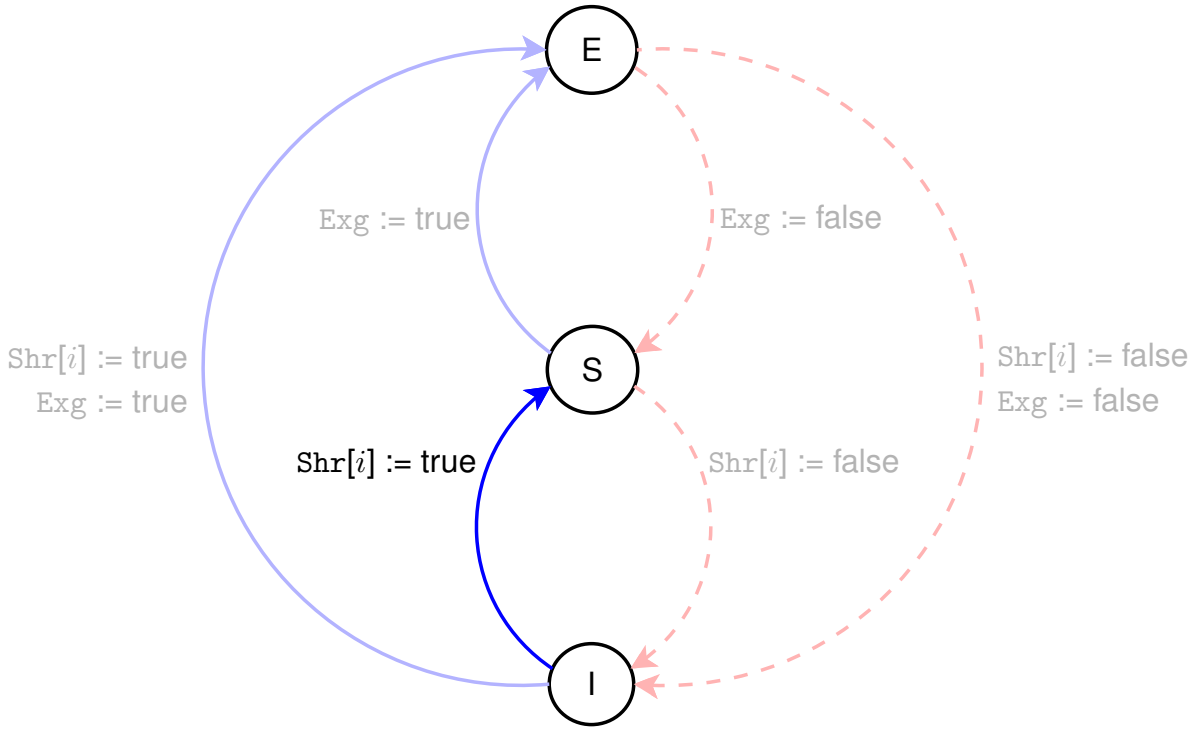


Exemple : protocole de cohérence de cache German-*esque*




```
transition t5 (i)
requires { Ptr = i &&
           Cmd = RS && Exg = False }
{
  Cmd := Epsilon;
  Shr[i] := True;
  Cache[i] := S;
}
```

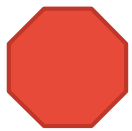
Exemple : protocole de cohérence de cache German-*esque*



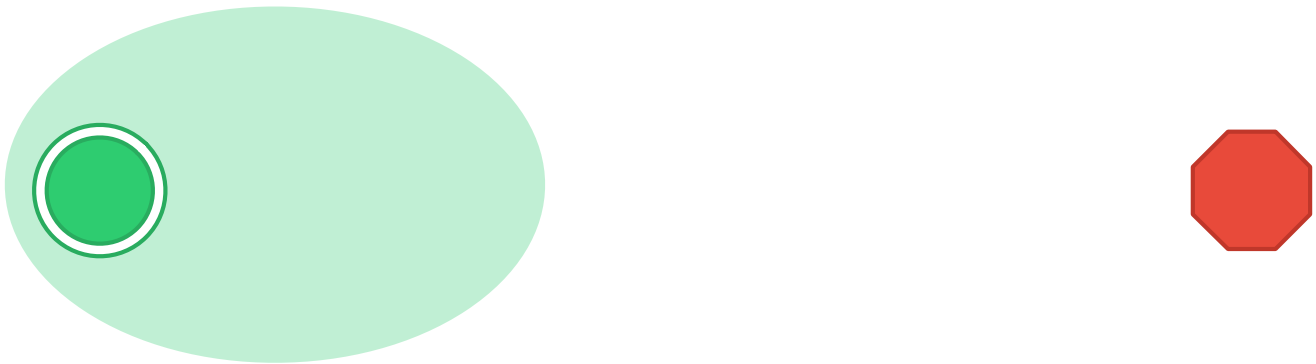
$$t_5 : \exists i.\texttt{Ptr} = i \ \wedge \ \texttt{Cmd} = \texttt{rs} \wedge \neg\texttt{Exg} \ \wedge \texttt{Cmd}' = \epsilon \wedge \texttt{Shr}'[i] \ \wedge \ \texttt{Cache}'[i] = \texttt{S}$$

```
transition t5 (i)
requires { Ptr = i &&
           Cmd = RS && Exg = False }
{
  Cmd := Epsilon;
  Shr[i] := True;
  Cache[i] := S;
}
```

Analyse d’atteignabilité avant

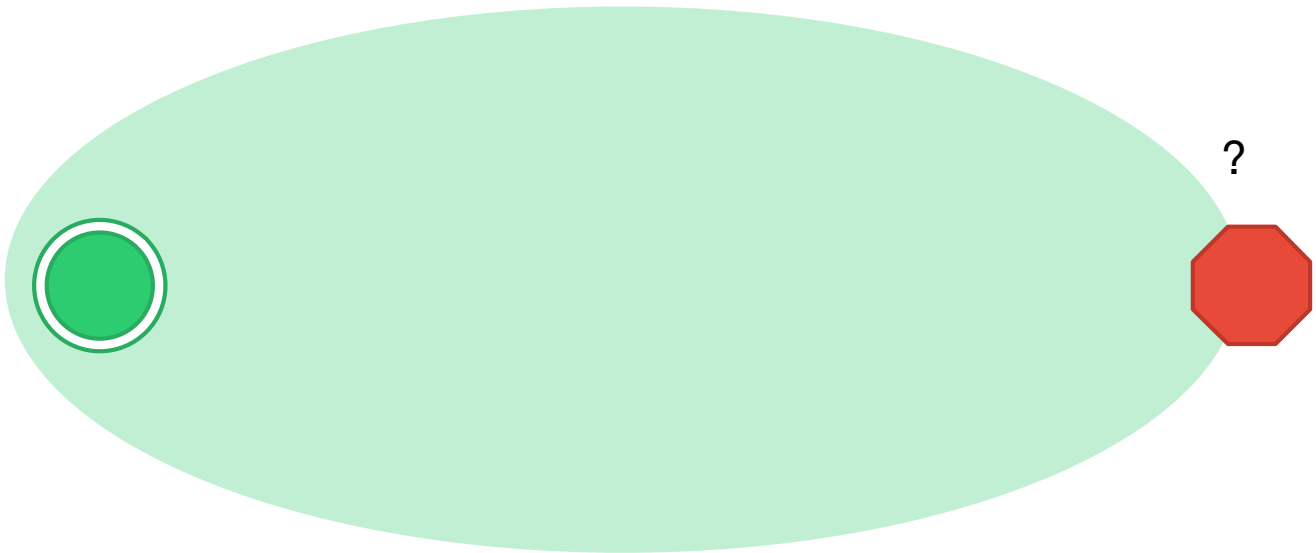


Analyse d’atteignabilité avant

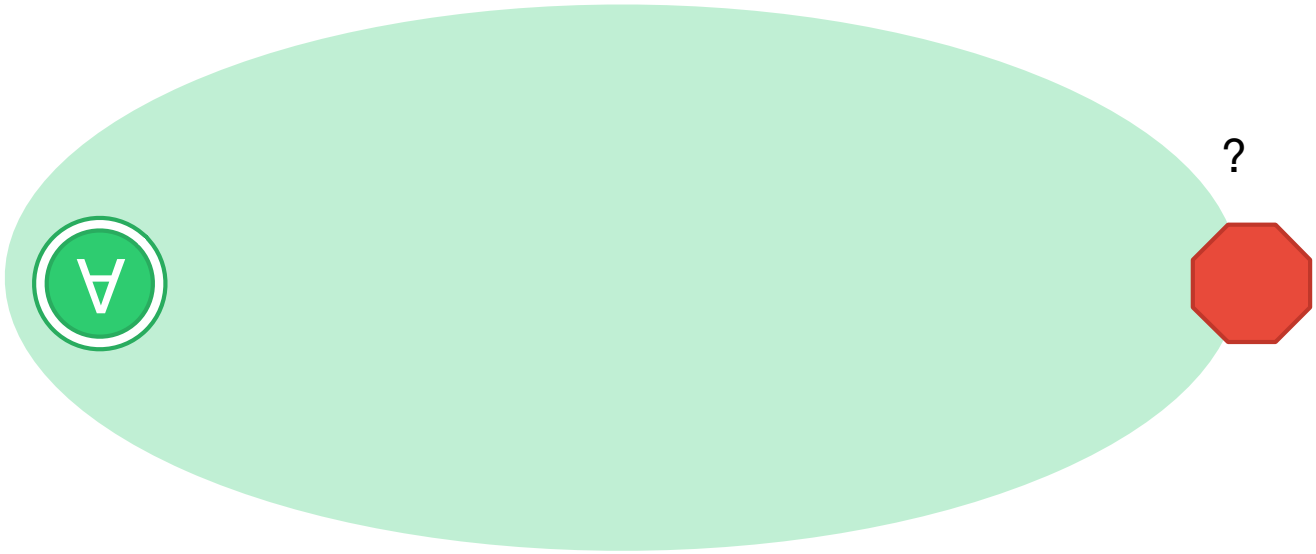


Analyse d’atteignabilité avant

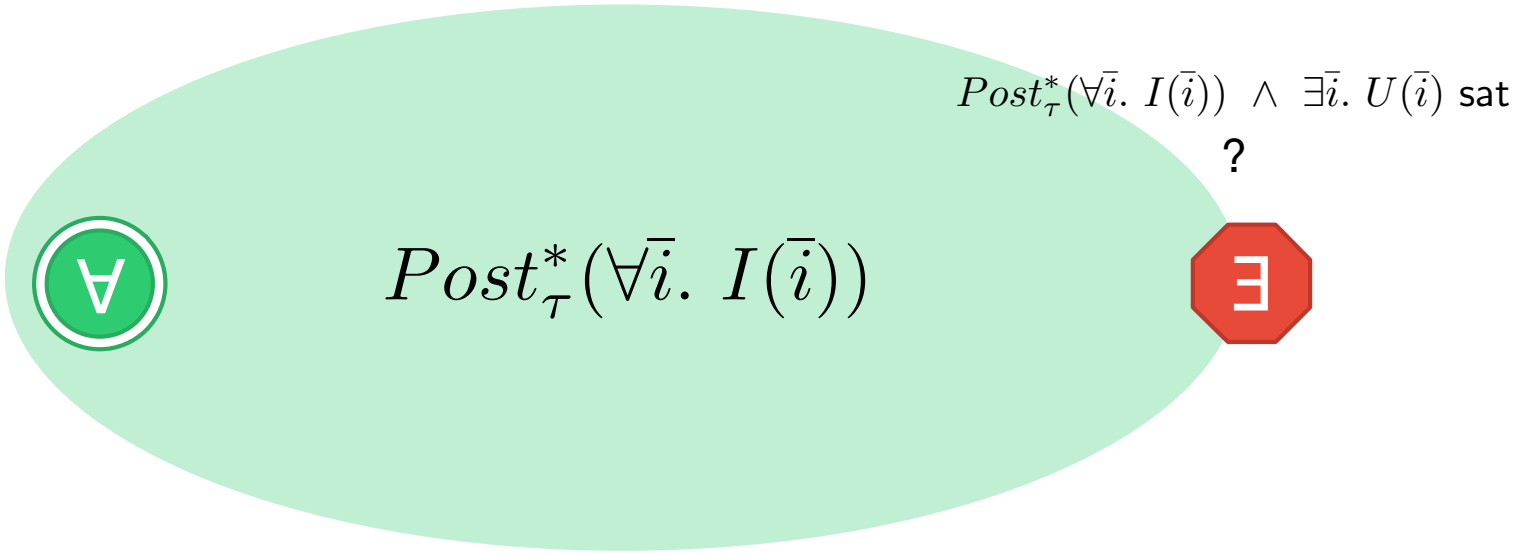
Analyse d’atteignabilité avant



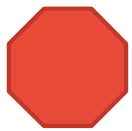
Analyse d’atteignabilité avant



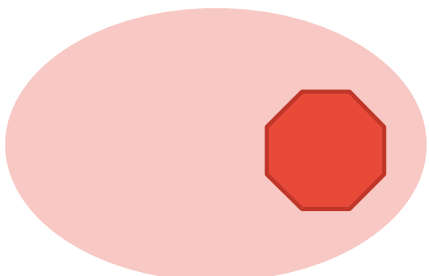
Analyse d’atteignabilité avant



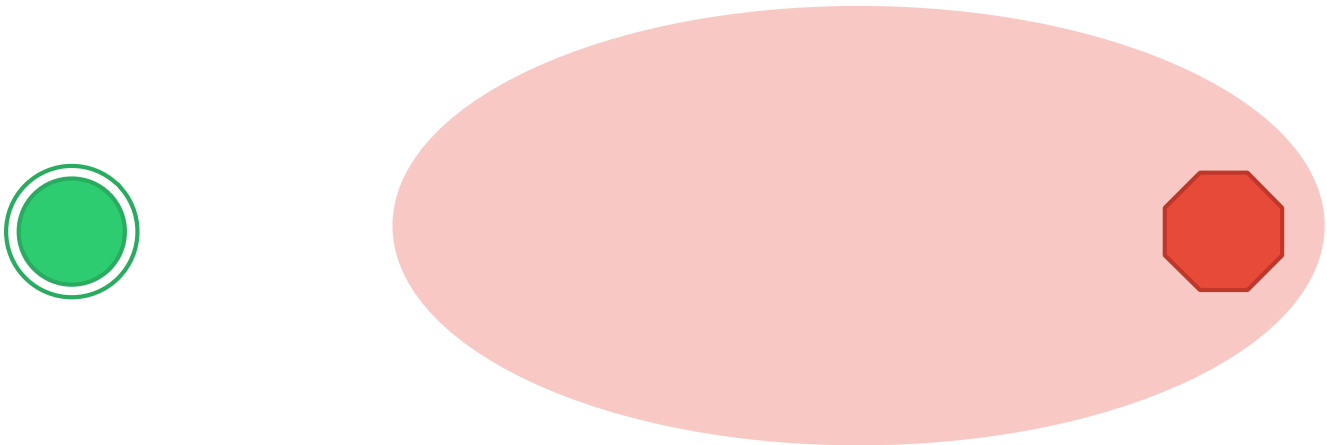
Analyse d’atteignabilité arrière



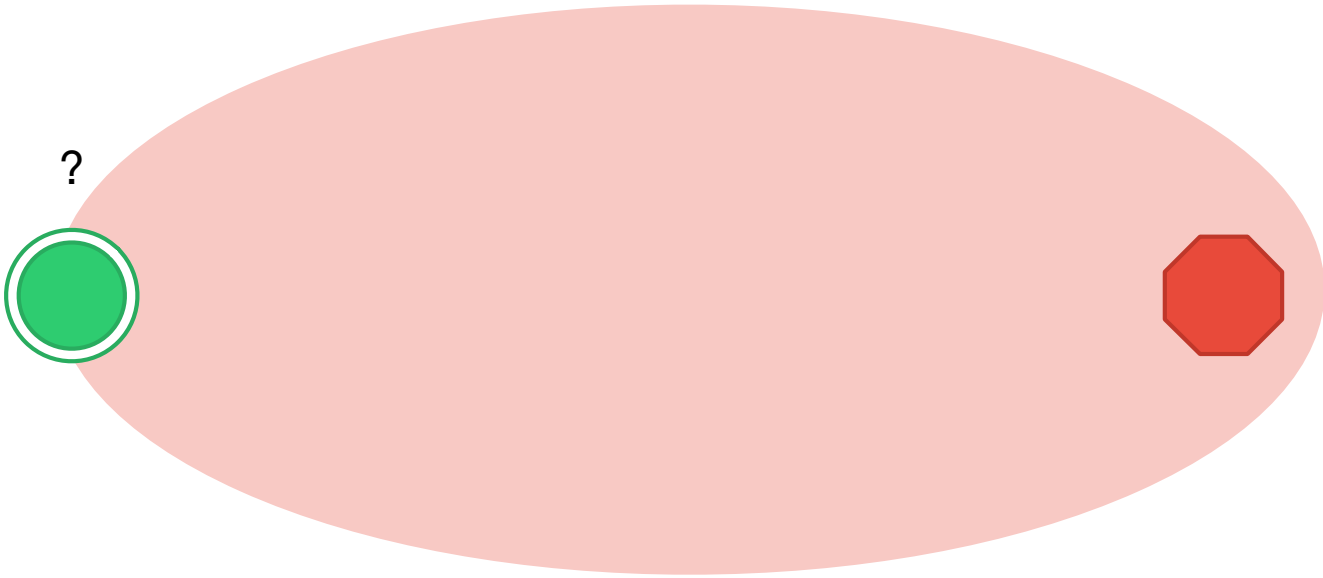
Analyse d’atteignabilité arrière



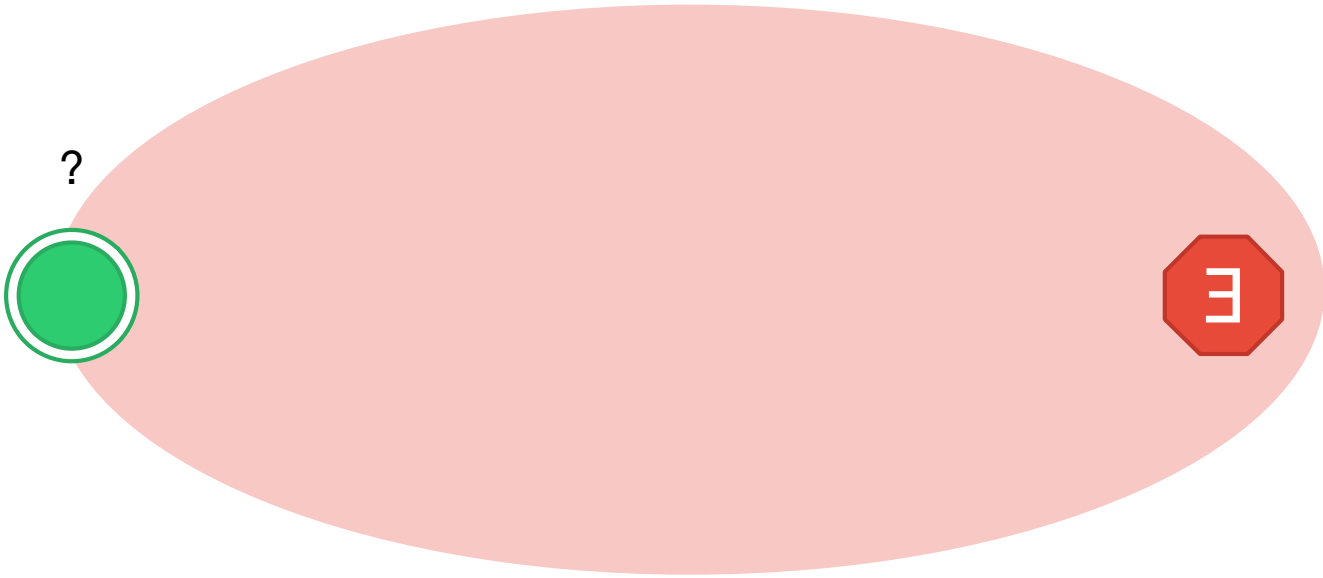
Analyse d’atteignabilité arrière



Analyse d’atteignabilité arrière



Analyse d’atteignabilité arrière



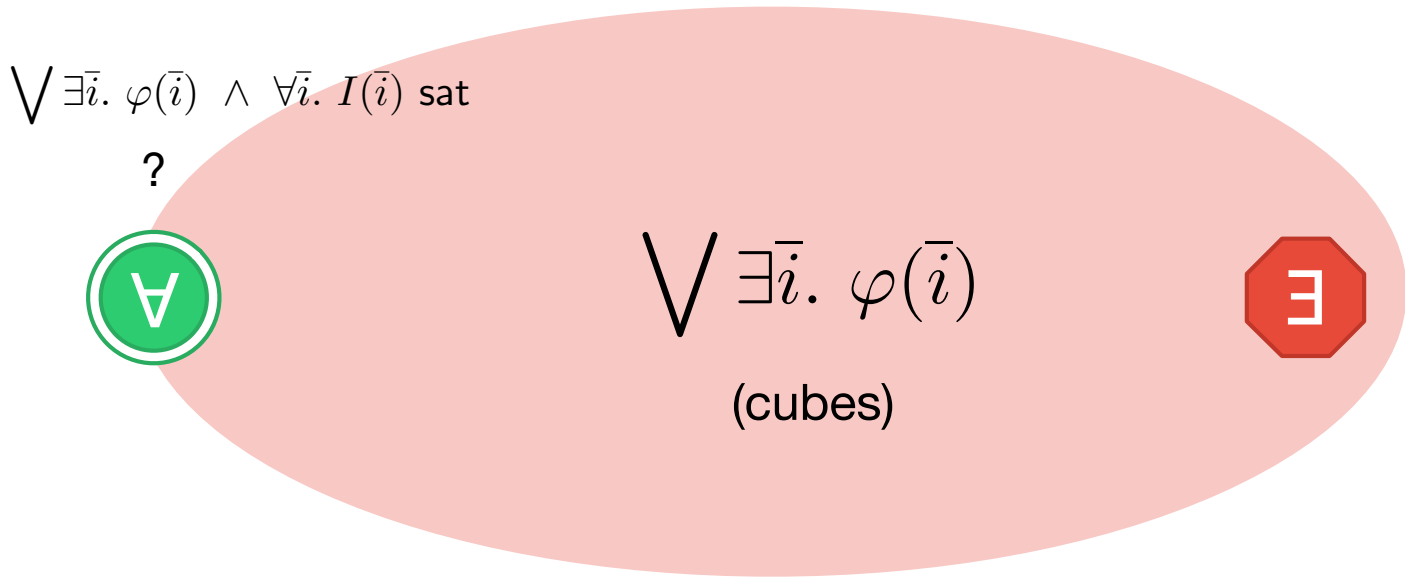
Analyse d’atteignabilité arrière

$Pre^*_\tau(\exists \bar{i}.\ U(\bar{i}))\ \wedge\ \forall \bar{i}.\ I(\bar{i})$ sat

?

$$Pre^*_\tau(\exists \bar{i}.\ U(\bar{i}))$$

Analyse d’atteignabilité arrière



? = solveur SMT

Solution

Cadre restreint : systèmes de transition à tableaux

Technique : Model Checking Modulo Théories (MCMT)
[Ghilardi & Ranise, 2009]

Implémentée dans MCMT et Cubicle

Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD () :

V := $\emptyset$;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

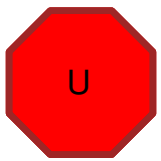
if $\varphi \wedge$ I sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V $\cup \{\varphi\}$;

push(Q, $\mathtt{pre}_\tau(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD () :

V := $\emptyset$;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

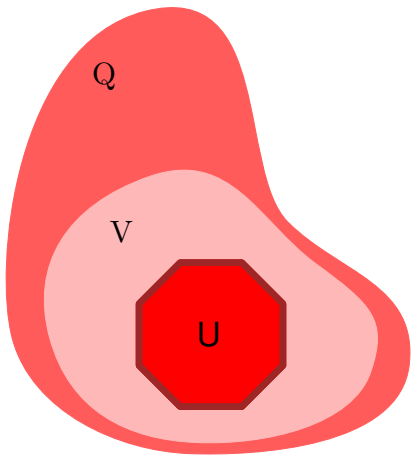
if $\varphi \wedge$ I sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V $\cup \{\varphi\}$;

push(Q, $\mathtt{pre}_\tau(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD ():

V := ∅;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q);

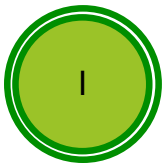
if $\varphi \wedge I$ sat **then return** unsafe

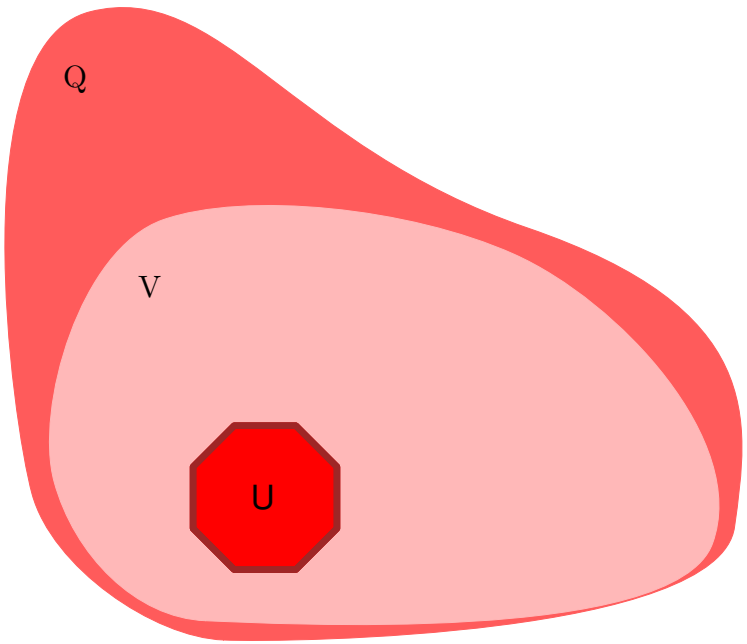
if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V ∪ {φ} ;

push(Q, $pre_\tau(\varphi)$) ;

return safe





Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD () :

V := $\emptyset$;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

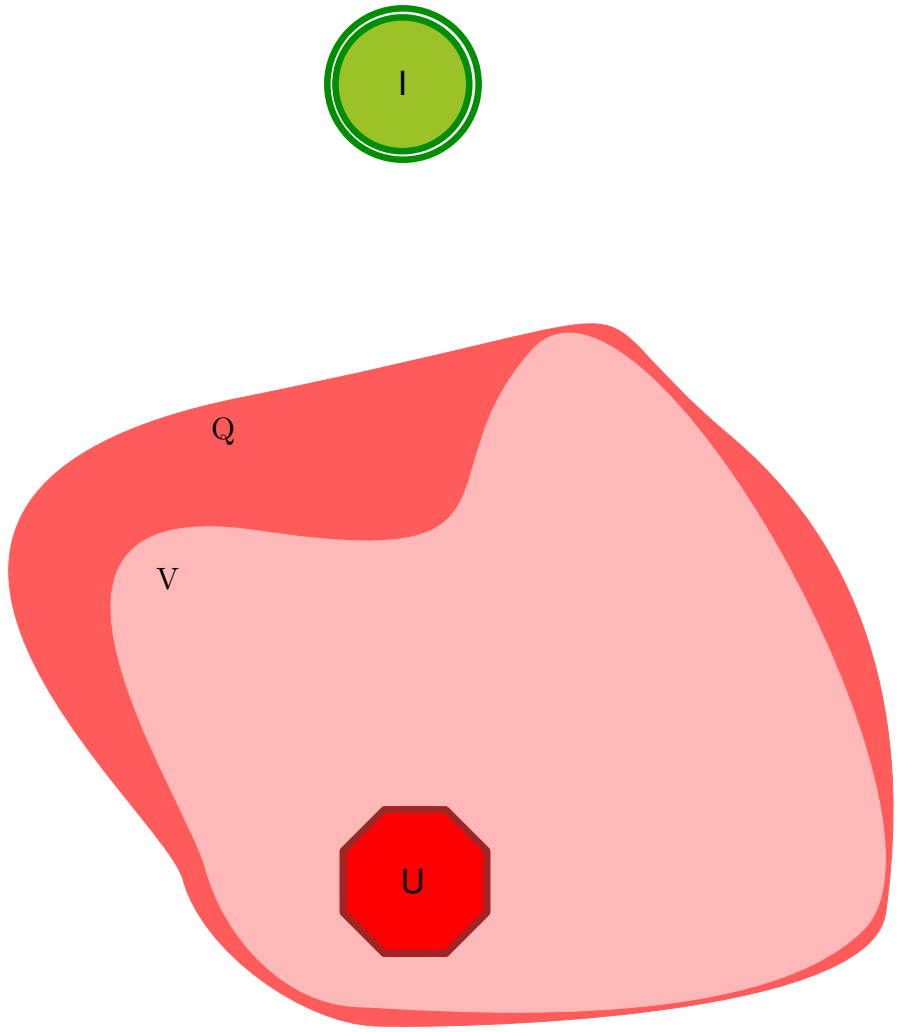
if $\varphi \wedge$ I sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V $\cup \{\varphi\}$;

push(Q, $\mathtt{pre}_\tau(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux

U : états dangereux (cubes)

τ : transitions

BWD () :

V := ∅ ;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

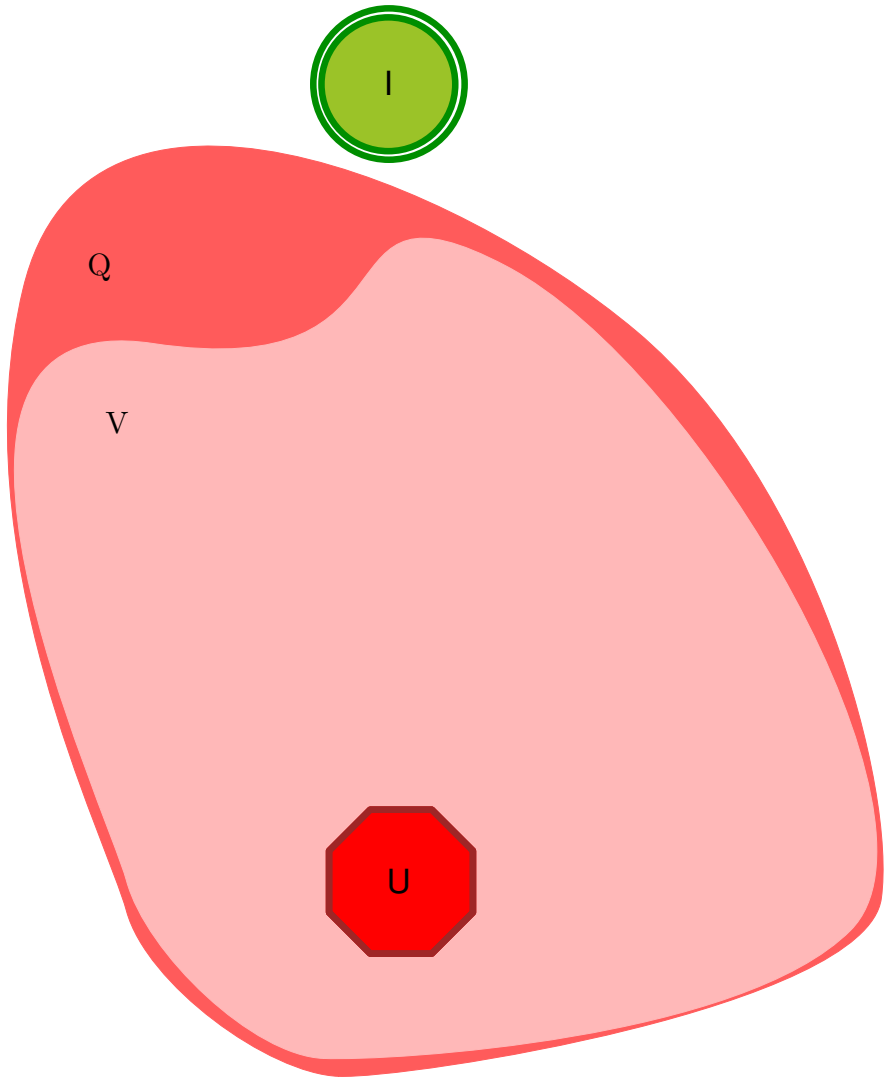
if $\varphi \wedge I$ sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V ∪ {φ} ;

push(Q, $pre_\tau(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD () :

V := $\emptyset$;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

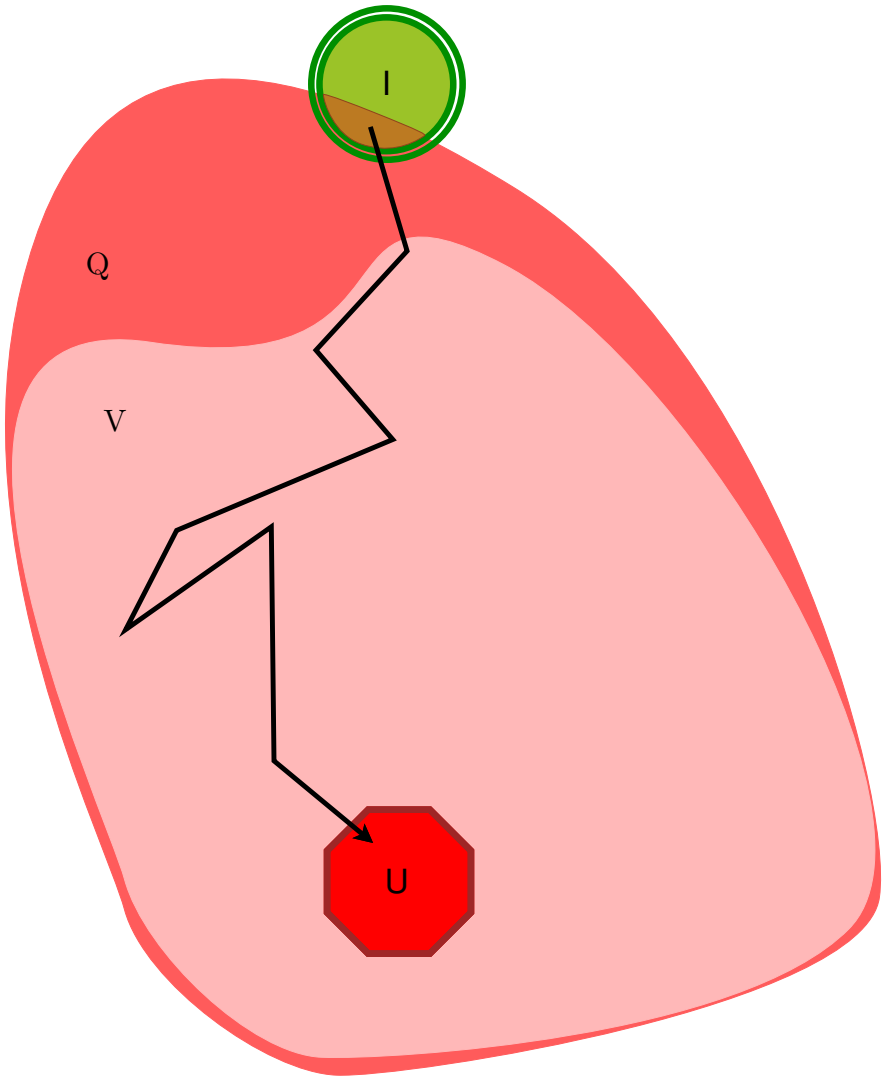
if $\varphi \wedge$ I sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V $\cup \{\varphi\}$;

push(Q, $\mathtt{pre}_\tau(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD () :

V := ∅ ;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

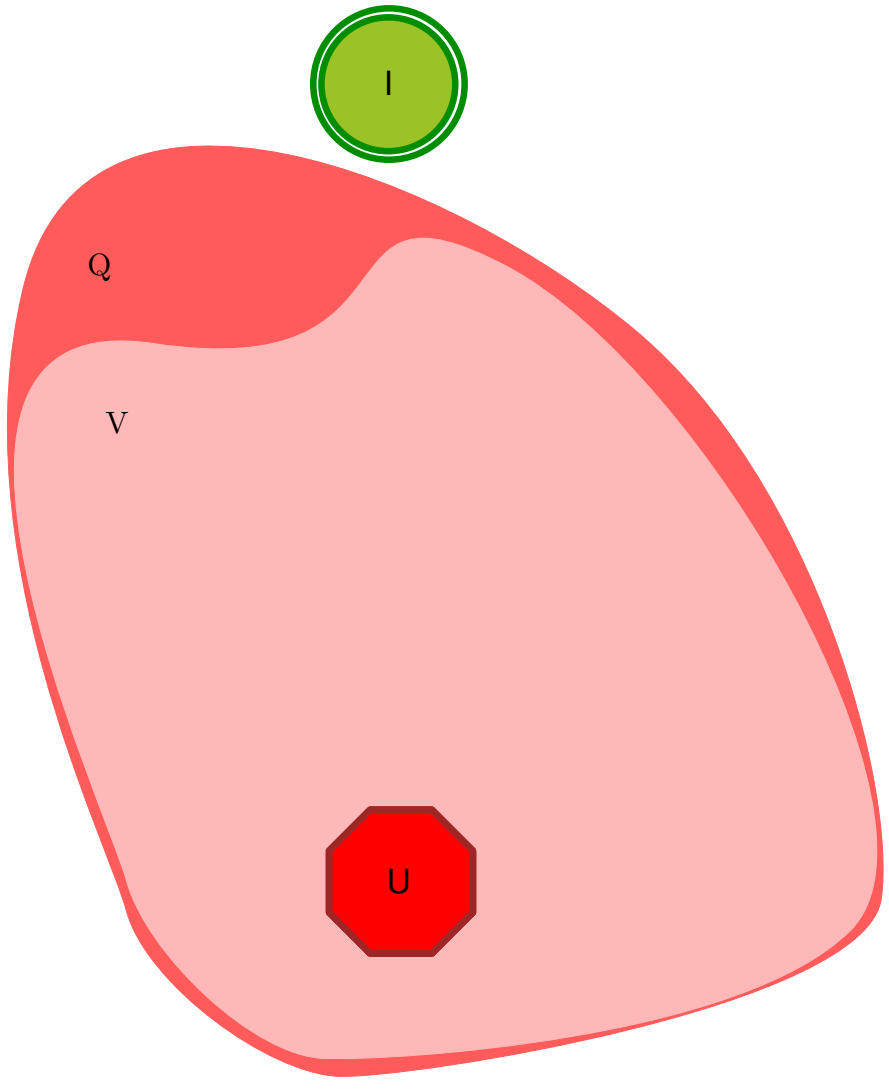
if $\varphi \wedge I$ sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V ∪ {φ} ;

push(Q, $\mathtt{pre}_{\tau}(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux

U : états dangereux (cubes)

τ : transitions

BWD () :

V := ∅ ;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

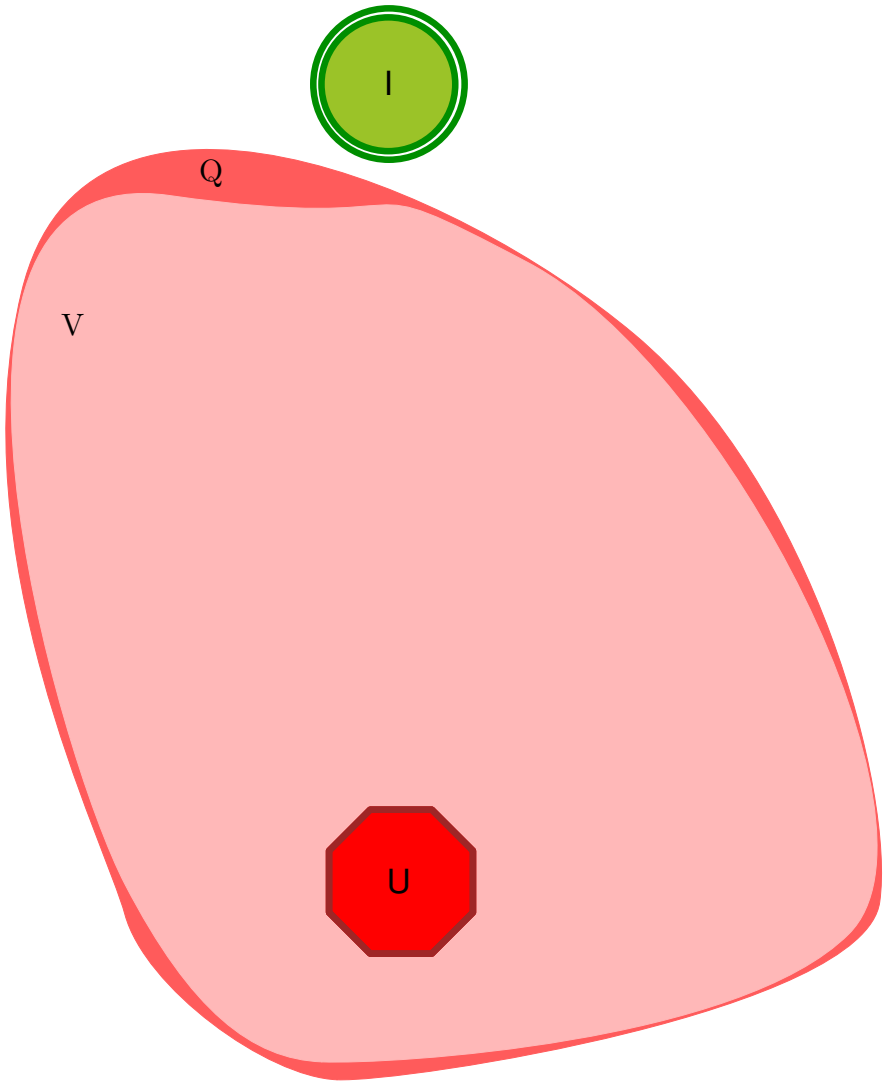
if $\varphi \wedge I$ sat **then return** unsafe

if $\neg(\ \varphi \models \bigvee_{\psi \in V} \psi\ )$ **then**

V := V ∪ {φ} ;

push(Q, $pre_\tau(\varphi)$) ;

return safe



Atteignabilité arrière

I : états initiaux
U : états dangereux (cubes)
τ : transitions

BWD () :

V := ∅ ;

push(Q, U) ;

while not empty(Q) **do**

φ := pop(Q) ;

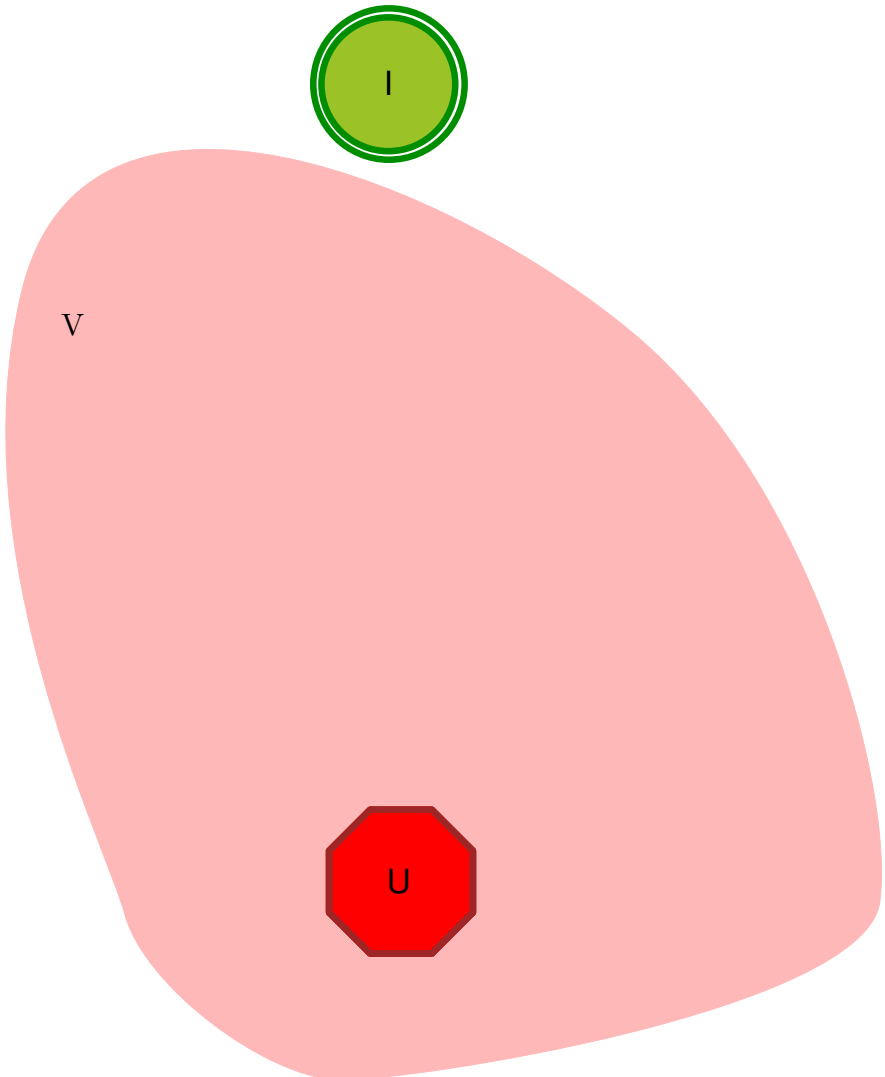
if $\varphi \wedge I$ sat **then return** unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ **then**

V := V ∪ {φ} ;

push(Q, $pre_\tau(\varphi)$) ;

return safe

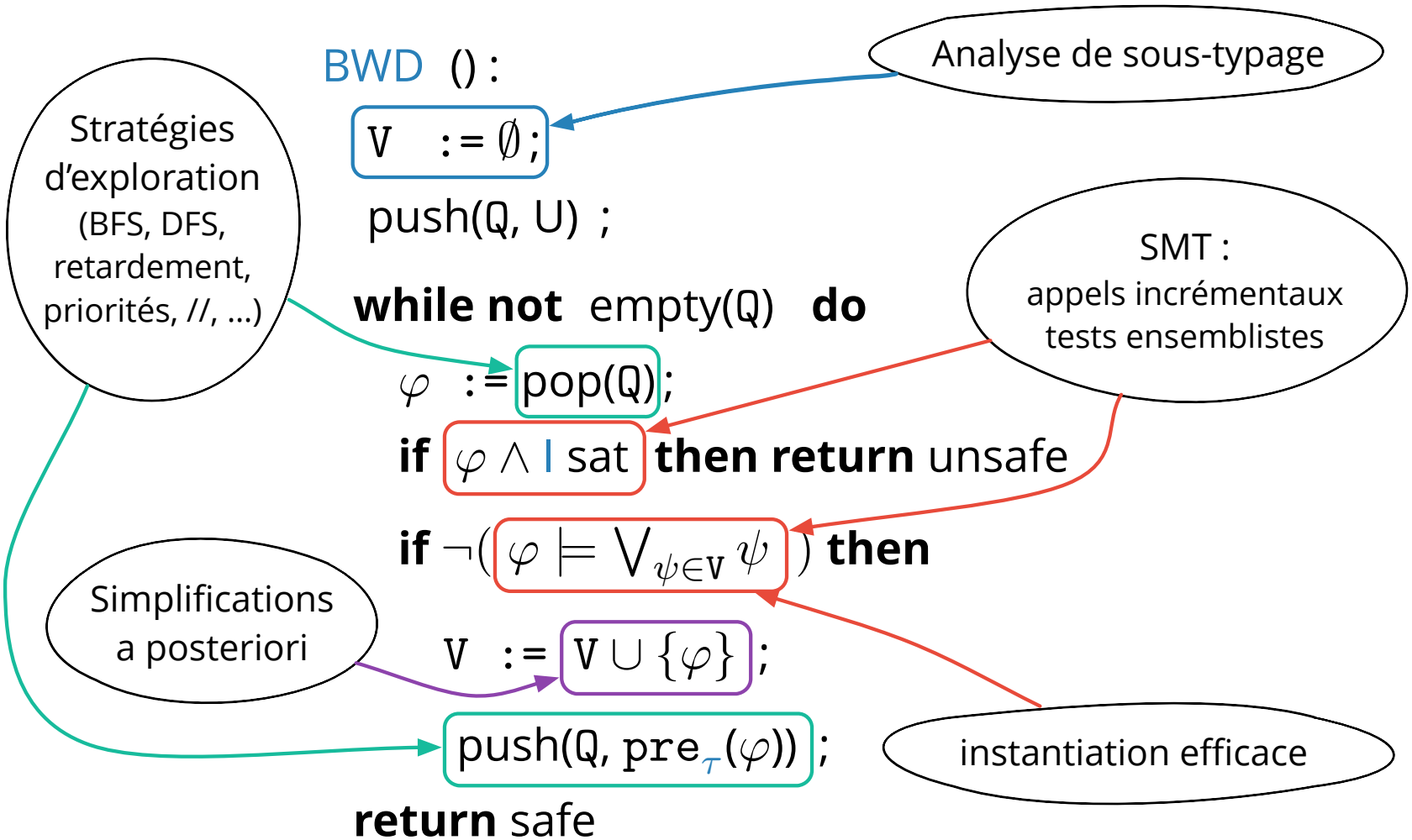


Est-ce que ça marche en pratique ?

Non.

Est-ce que ça marche en pratique ?

Optimisations :



Est-ce que ça marche ?

Benchmark	Cubicle	MCMT	Undip	PFS
exclusion mutuelle				
Bakery	0,01s	0,01s	0,04s	0,01s
Dijkstra	0,12s	0,70s	**0,04s**	0,26s
Java Meta Lock	0,02s	0,04s	0,25s	0,02s
Ricart Agrawala	5,01s	1m10s	**4,17s**	/
Szymanski_at	4m41s	**0,24s**	32,1s	T.O.
cohérence de cache				
Berkeley	0,01s	0,01s	0,01s	0,01s
German_Baukus	**5,06s**	33m15s	9m43s	/
German_pfs	**2m45s**	6m47s	T.O.	36m05s
German_undip	**0,10s**	0,46s	1m32	/
Illinois	0,02s	0,04s	0,06s	0,06s
Moesi	0,01s	0,01s	0,01s	0,01s
tolérance aux pannes				
Chandra-Toueg	2h01m	**4m34s**	/	/

Est-ce que ça marche ?

	Paramétré	*Énumératif*		
	Cubicle	**CMurphi**		
Szymanski_at	4m41s	8,04s (8)	5m12s (10)	2h50m (12)
German_Baukus	5,06s	0,74s (4)	19m35s (8)	4h49m (10)
German.CTC	4,58s	1,83s (4)	43m46s (8)	12h35m (10)
German_pfs	2m45s	0,99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	2h01m	5,68s (4)	2m58s (5)	1h36m (6)

Est-ce que ça marche ?

	Paramétré	*Énumératif*		
	Cubicle	**CMurphi**		
Szymanski_at	4m41s	8,04s (8)	5m12s (10)	2h50m (12)
German_Baukus	5,06s	0,74s (4)	19m35s (8)	4h49m (10)
German.CTC	4,58s	1,83s (4)	43m46s (8)	12h35m (10)
German_pfs	2m45s	0,99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	2h01m	5,68s (4)	2m58s (5)	1h36m (6)

Est-ce que ça marche ?

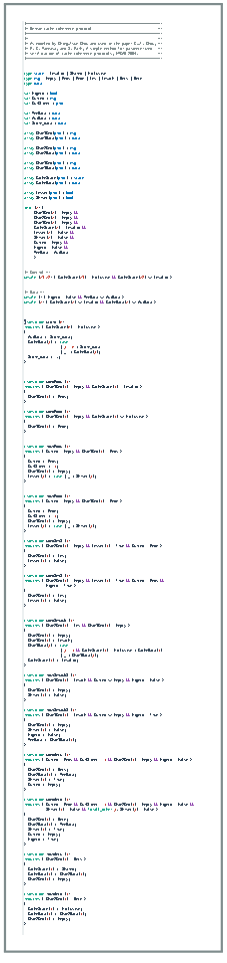
	Paramétré	*Énumératif*		
	Cubicle	**CMurphi**		
Szymanski_at	4m41s	8,04s (8)	5m12s (10)	2h50m (12)
German_Baukus	5,06s	0,74s (4)	19m35s (8)	4h49m (10)
German.CTC	4,58s	1,83s (4)	43m46s (8)	12h35m (10)
German_pfs	2m45s	0,99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	2h01m	5,68s (4)	2m58s (5)	1h36m (6)
Szymanski_na	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

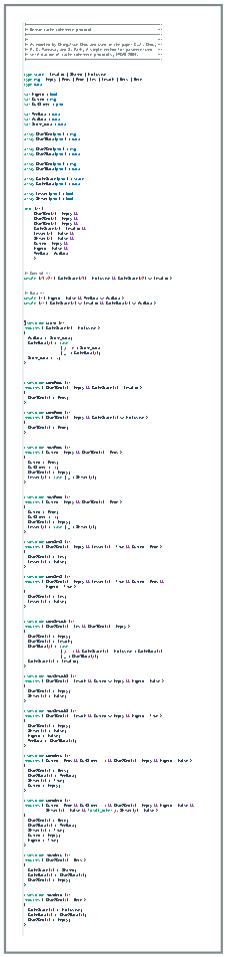
Le protocole FLASH

German



Le protocole FLASH

German

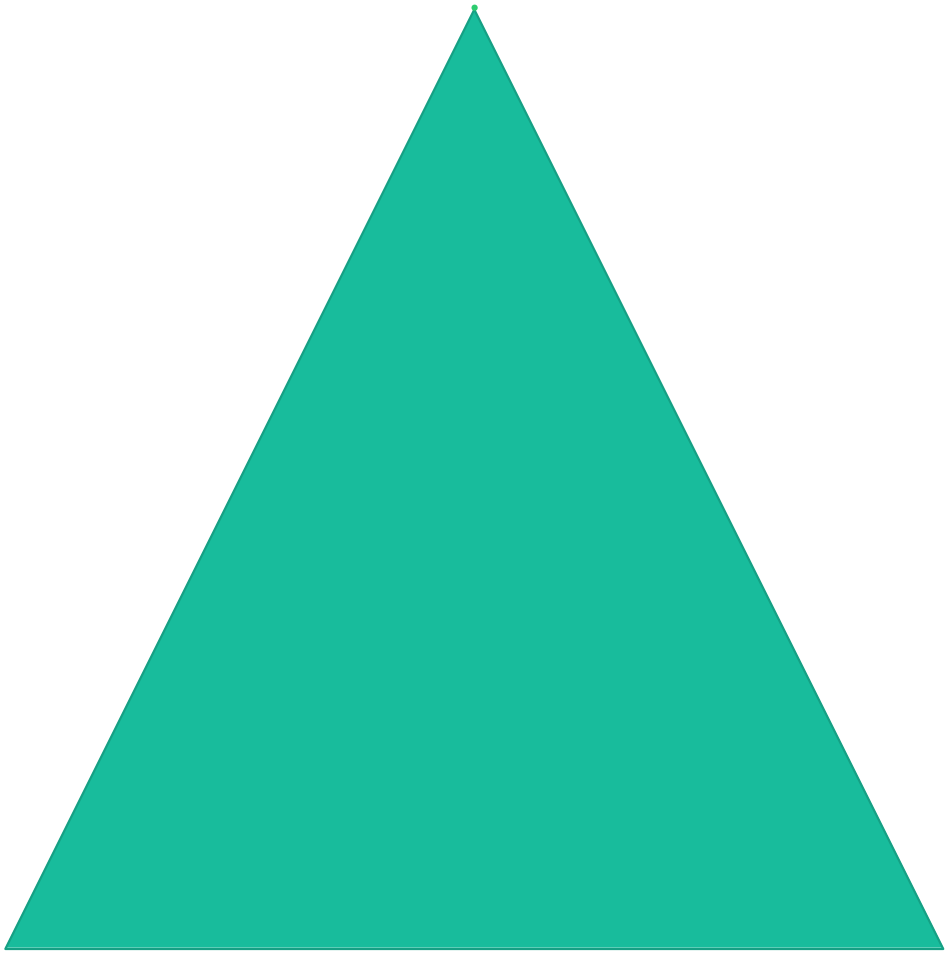


FLASH

Le protocole FLASH

German 4 processus
28 000 états

FLASH 4 processus
67 millions d'états



Le protocole FLASH

Architecture multi-processeur Stanford FLASH (1994)

- Mémoire partagée à cohérence de cache
- Modulaire : plusieurs milliers de cœurs (jusqu'à 4096)
- Taille industrielle

Le protocole FLASH

Architecture multi-processeur Stanford FLASH (1994)

- Mémoire partagée à cohérence de cache
- Modulaire : plusieurs milliers de cœurs (jusqu'à 4096)
- Taille industrielle

Qui a prouvé (cohérence des données) le protocole ?

Park et Dill, 1996	PVS
Das, Dill et Park, 1999	abstraction par prédicats
McMillan, 2001	model checking compositionnel
Chou, Mannava, Park, 2004	méthode CMP, McMillan
Talapur et Tuttle, 2008	CMP + flots de messages

Aucune de ces preuves n'est purement automatique

Challenge

Comment prouver la sûreté de protocoles de taille industrielle comme FLASH pour un nombre arbitraire processeurs ?

Inférence d’invariants : BRAB

Comment passer à l'échelle ?

- Réduire l'espace d'états à explorer
  - Invariants pour le cas paramétré
    - invariants fournis
    - découvrir des invariants

Comment passer à l'échelle ?

- Réduire l'espace d'états à explorer
  - Invariants pour le cas paramétré
    - invariants fournis
    - découvrir des invariants

Inférence d'invariants

Remarque : comportements intéressants déjà observables sur les petites instances

Idée : instances finies pour inférer des invariants du cas paramétré

Problème : invariants souvent plus difficiles à prouver que la propriété originale

Inférence d'invariants

Remarque : comportements intéressants déjà observables sur les petites instances

Idée : instances finies pour inférer des invariants du cas paramétré

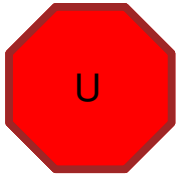
Problème : invariants souvent plus difficiles à prouver que la propriété originale

---

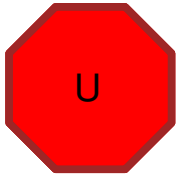
BRAB :

Backward **R**eachability with **A**pproximations and **B**acktracking

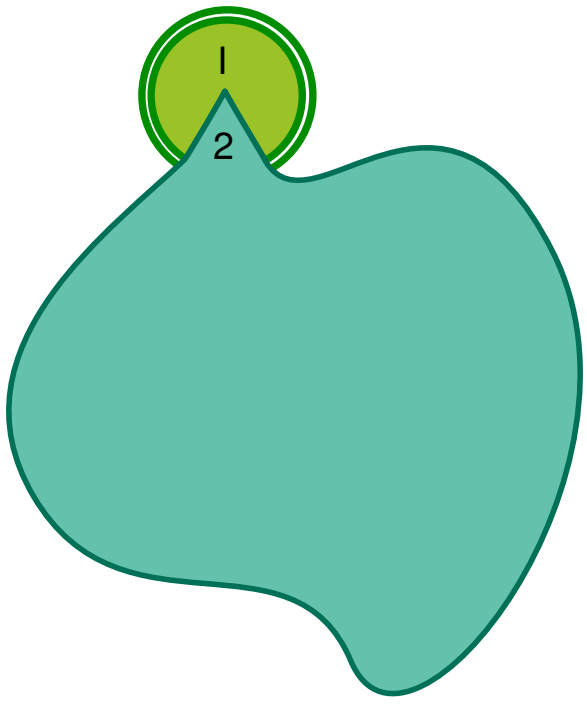
BRAB : intuition

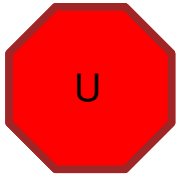


BRAB : intuition

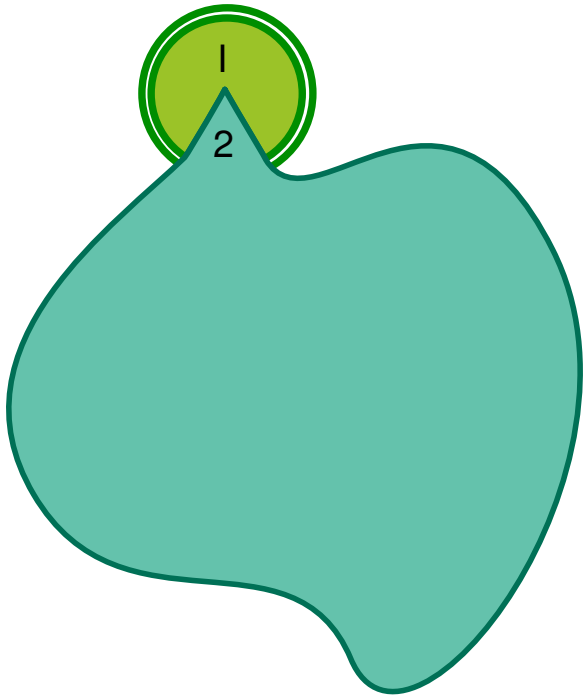


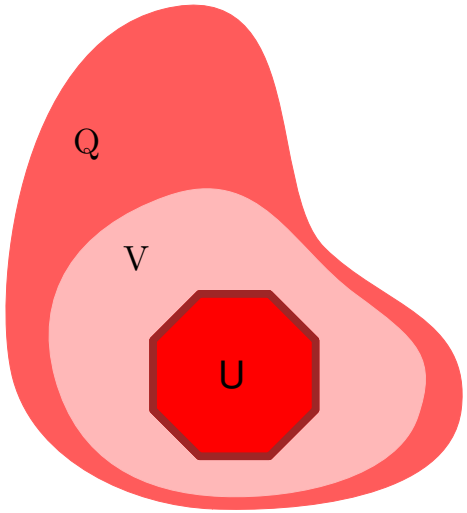
BRAB : intuition



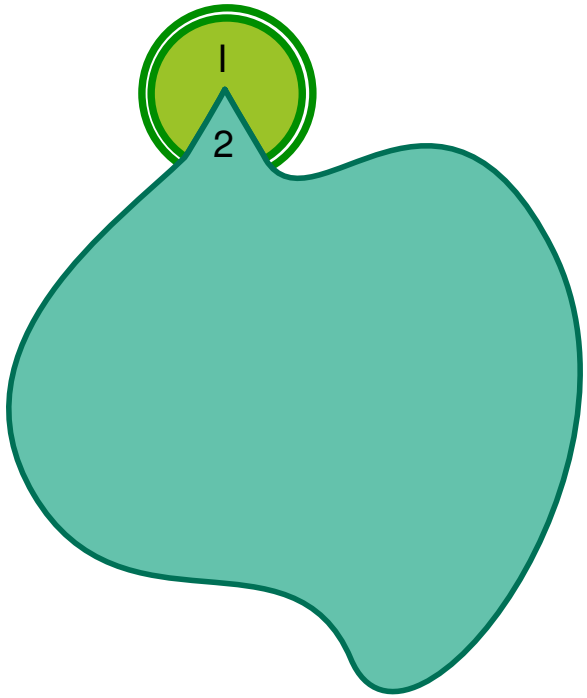


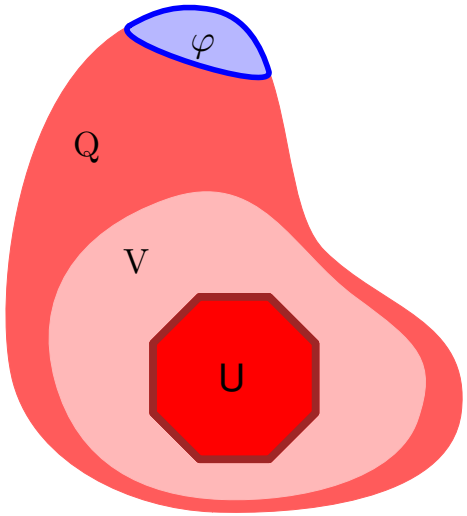
BRAB : intuition



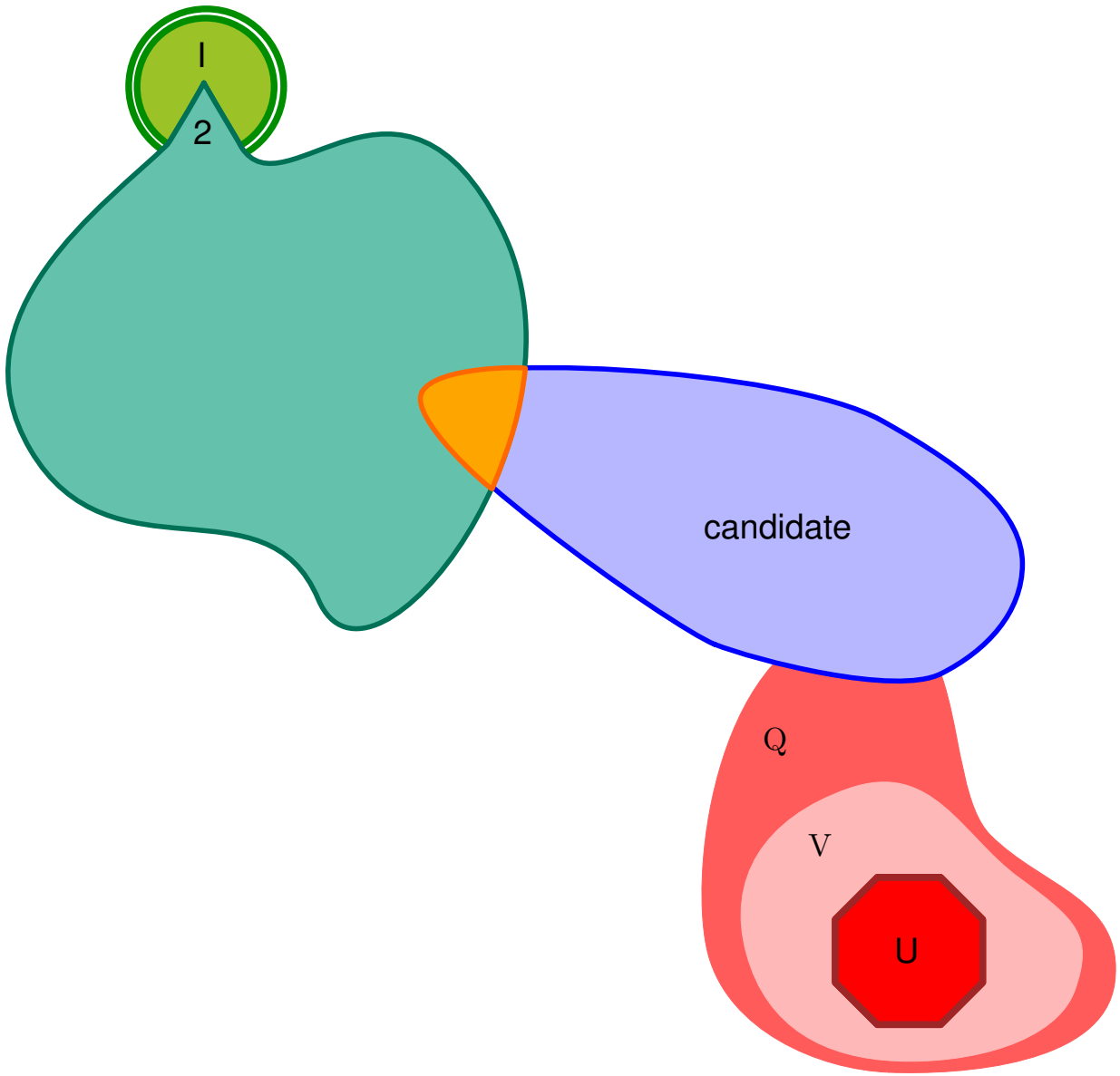


BRAB : intuition

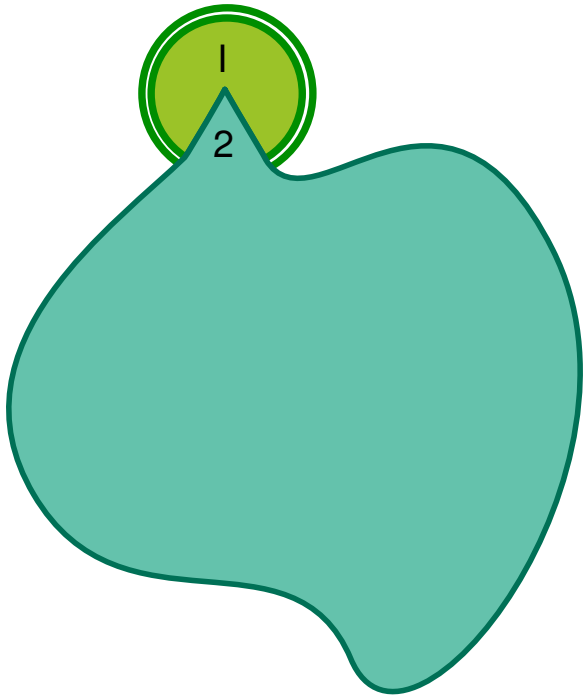


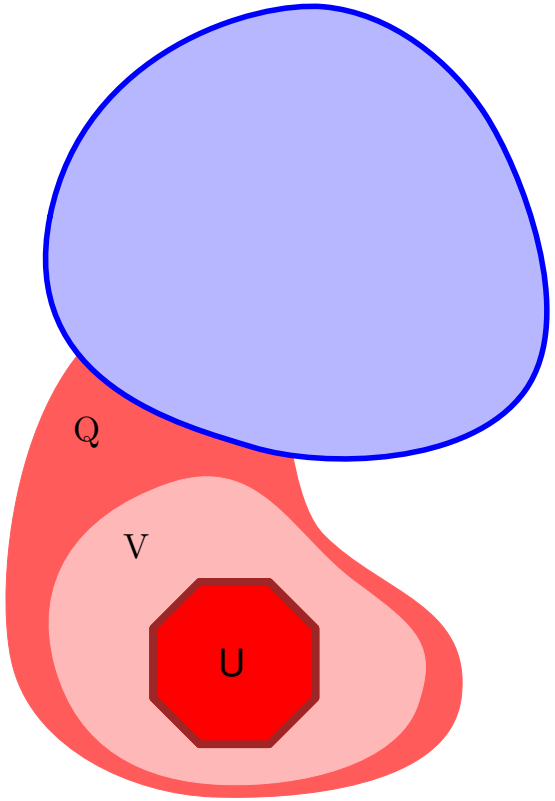


BRAB : intuition

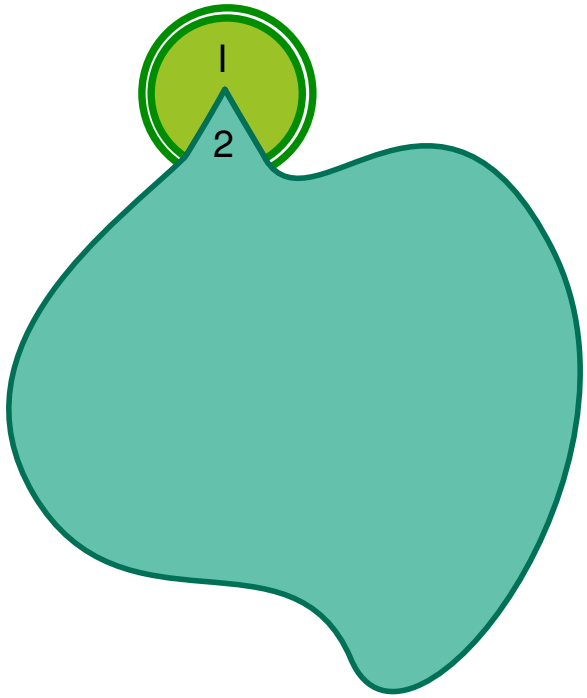


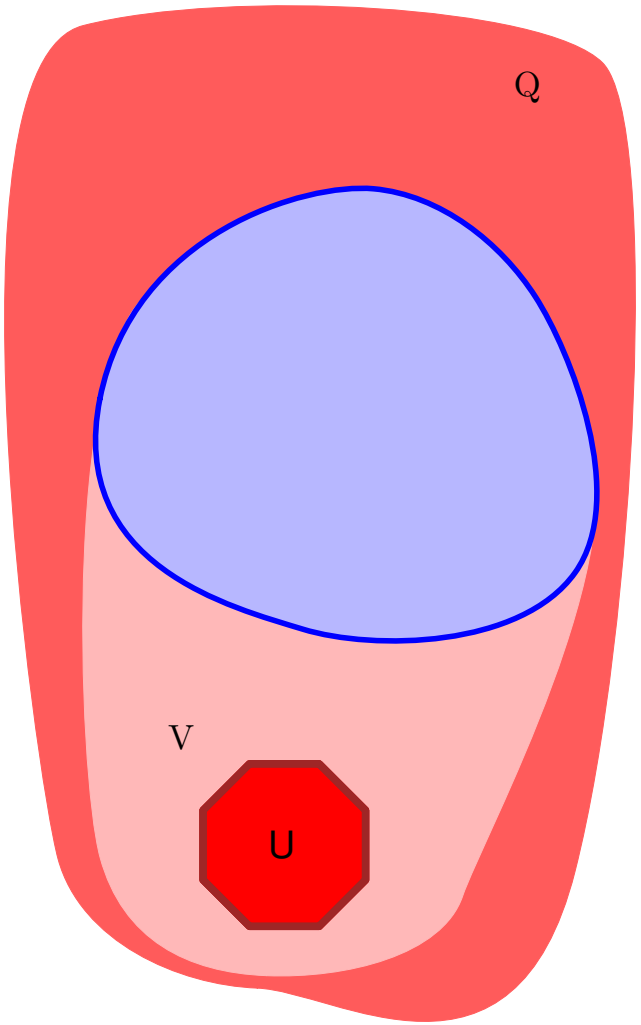
BRAB : intuition



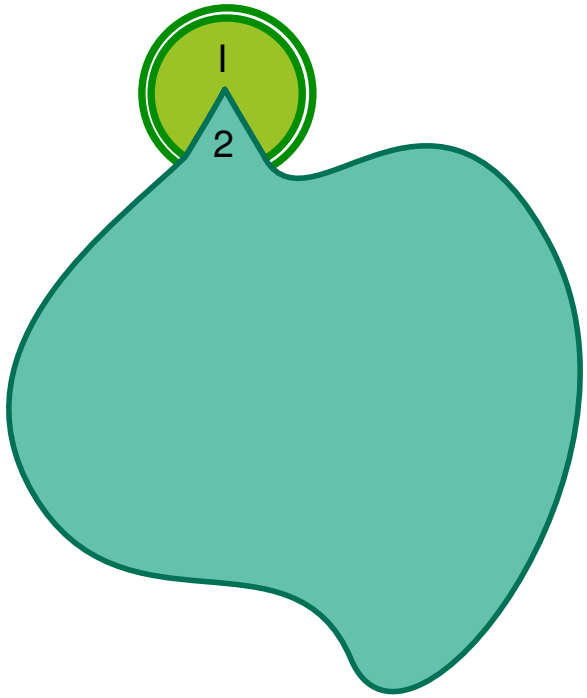


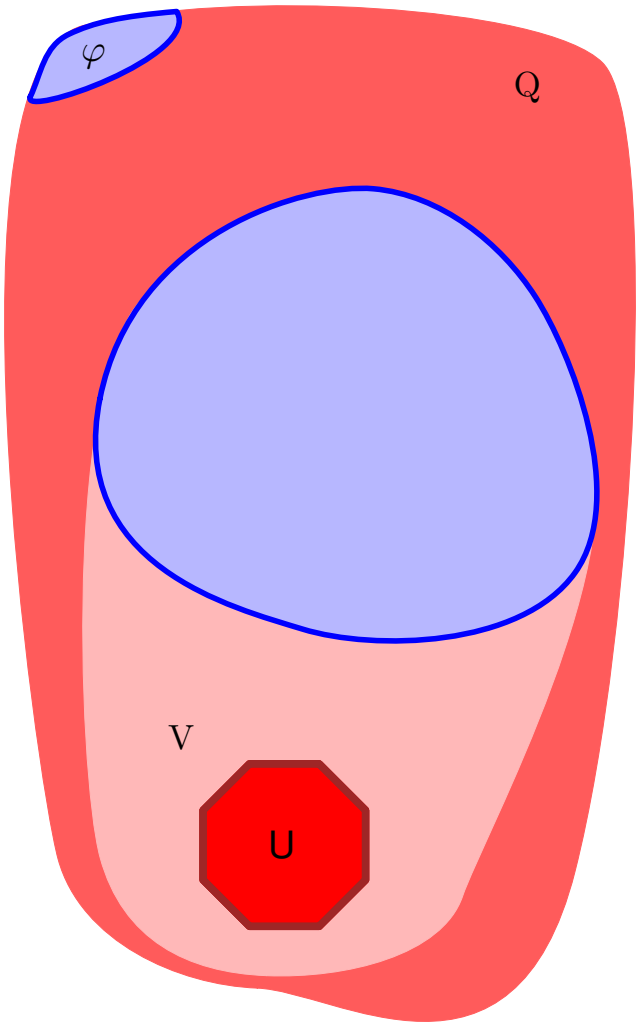
BRAB : intuition



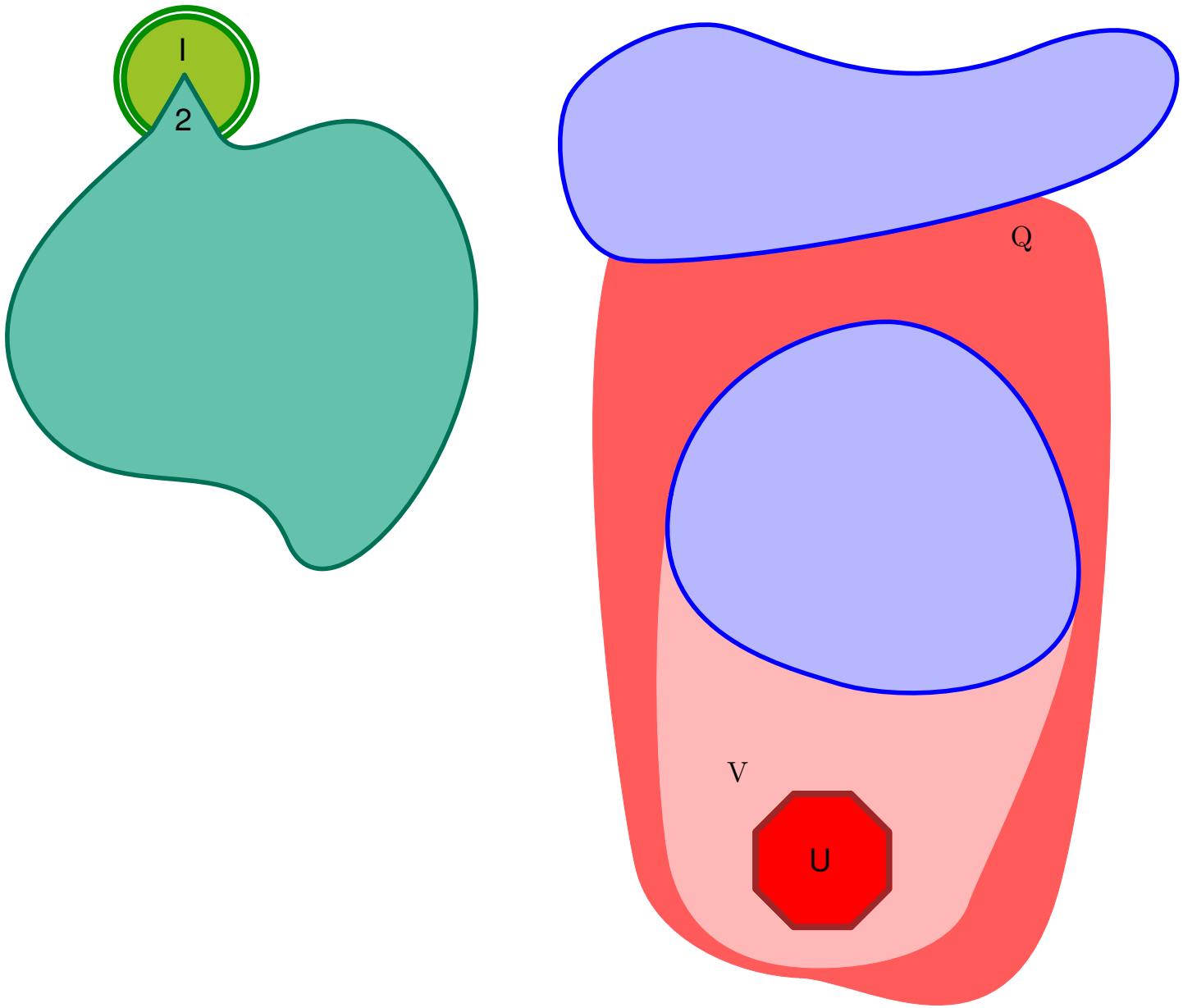


BRAB : intuition

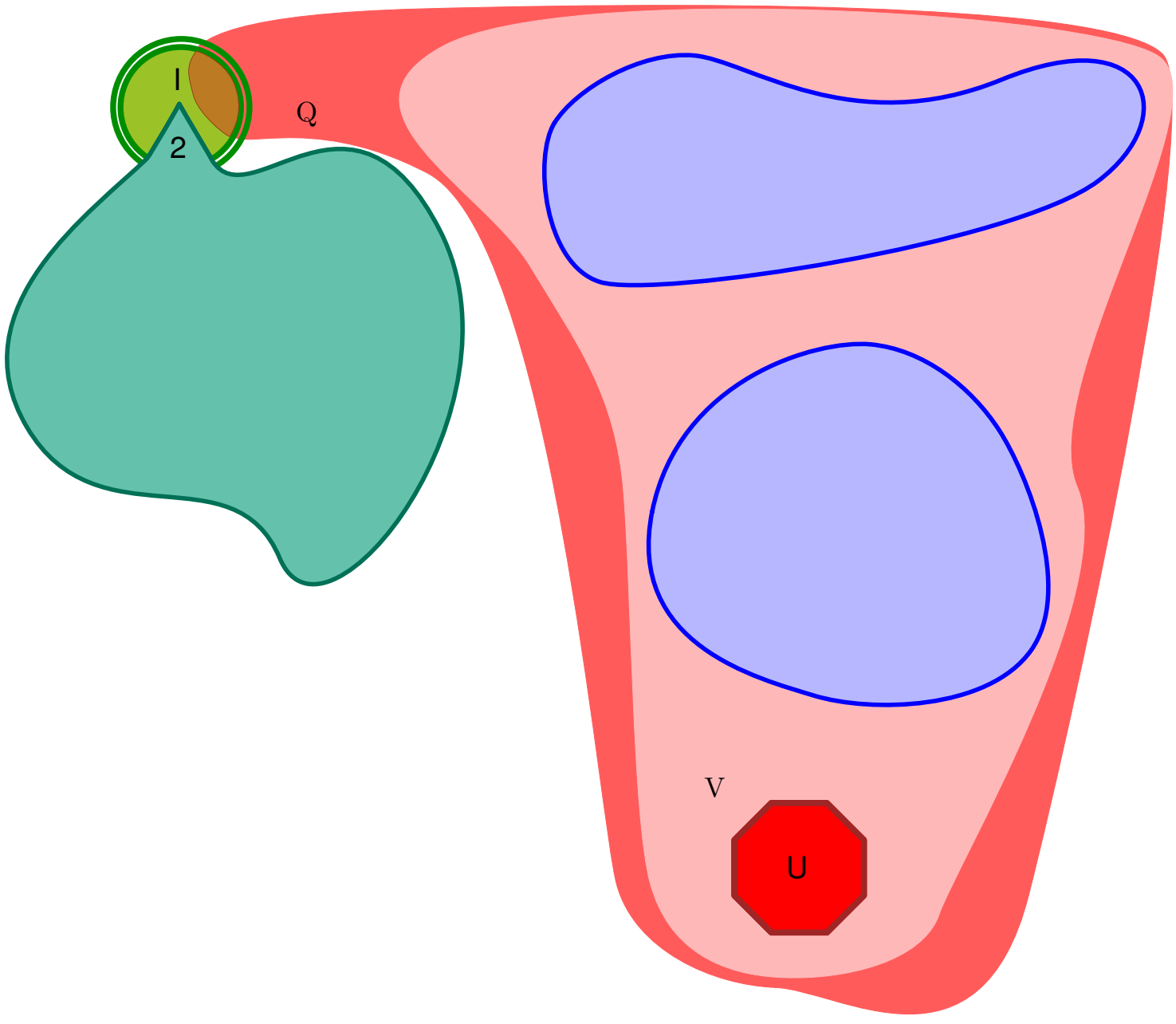




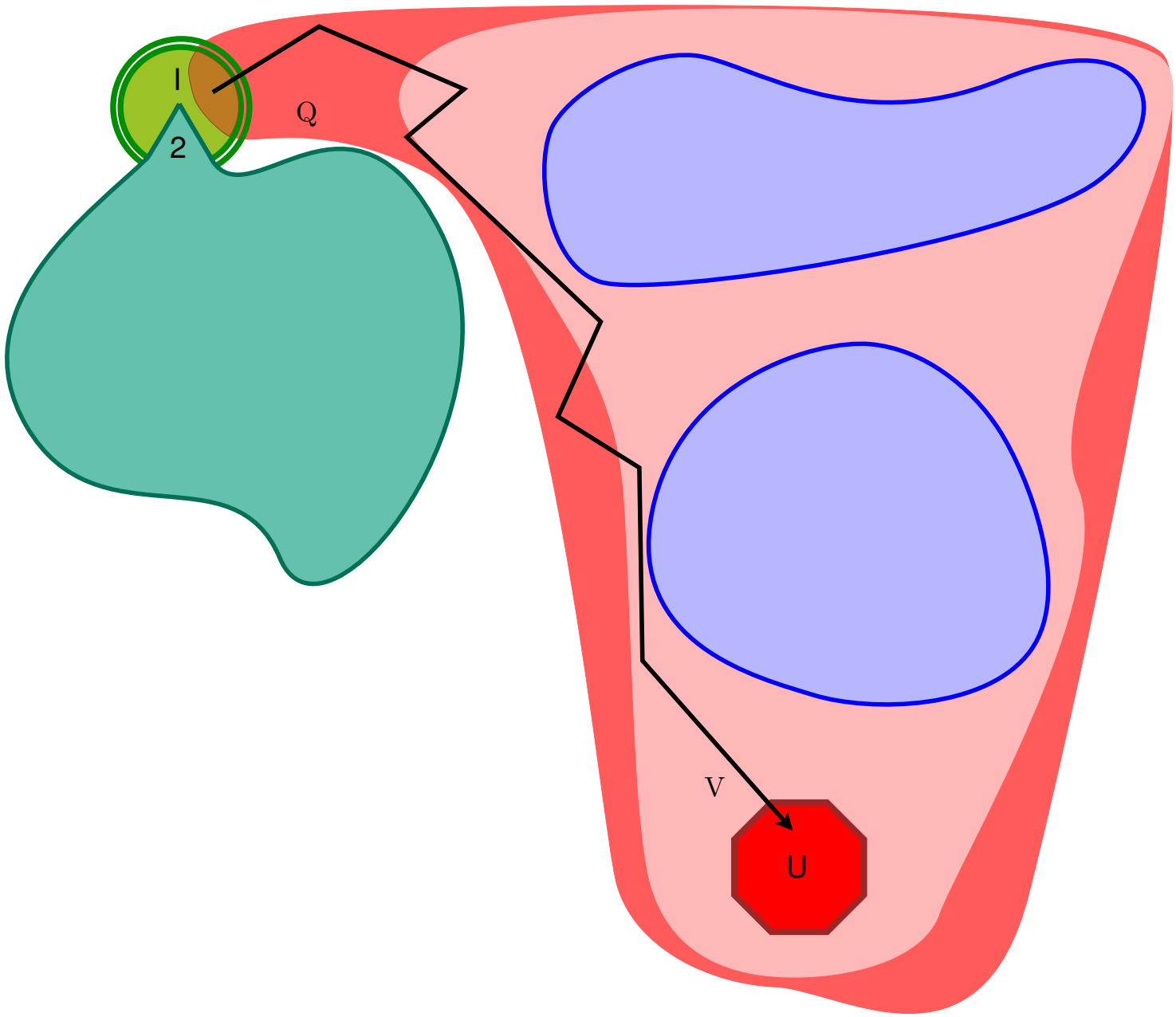
BRAB : intuition



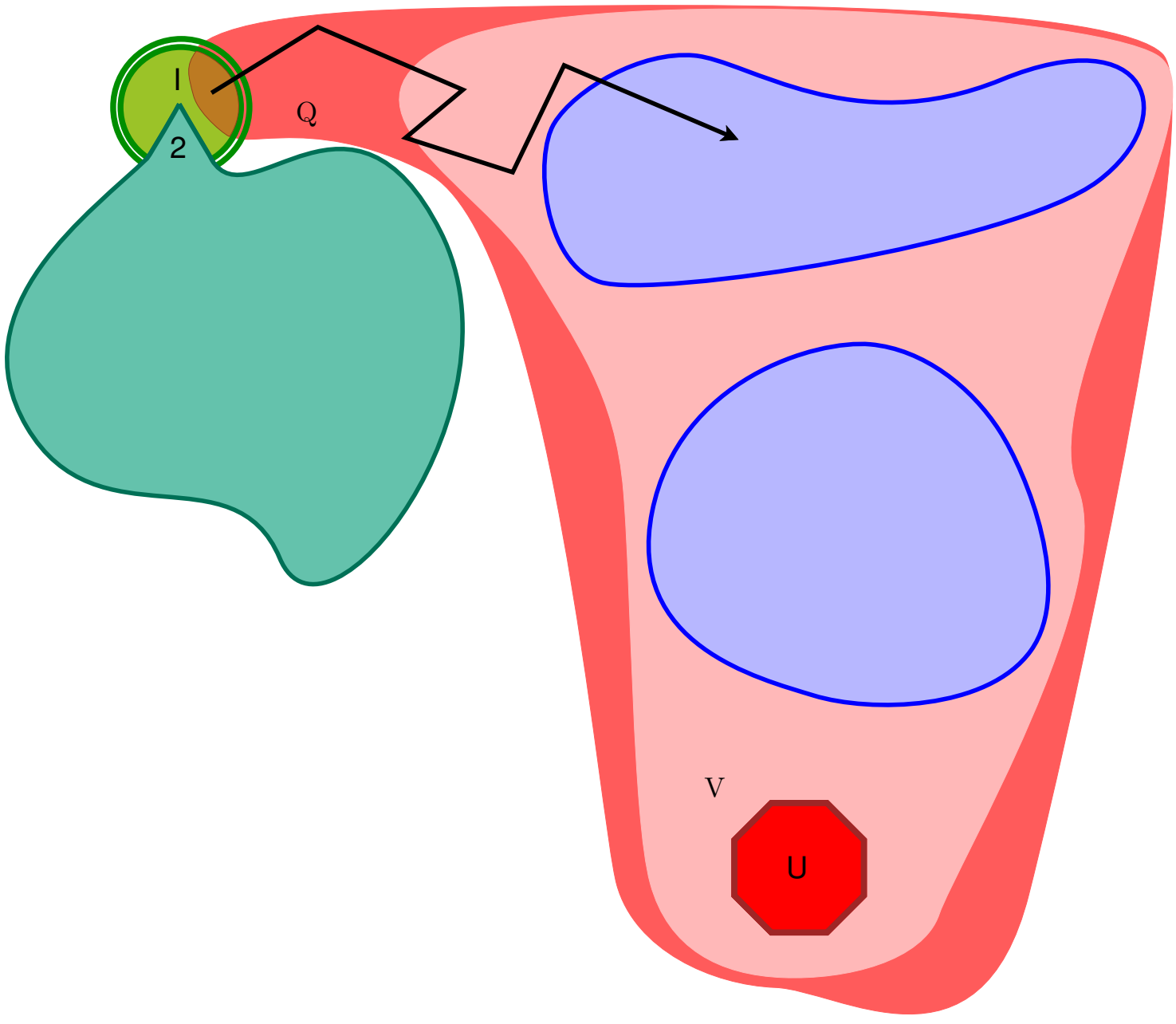
BRAB : intuition



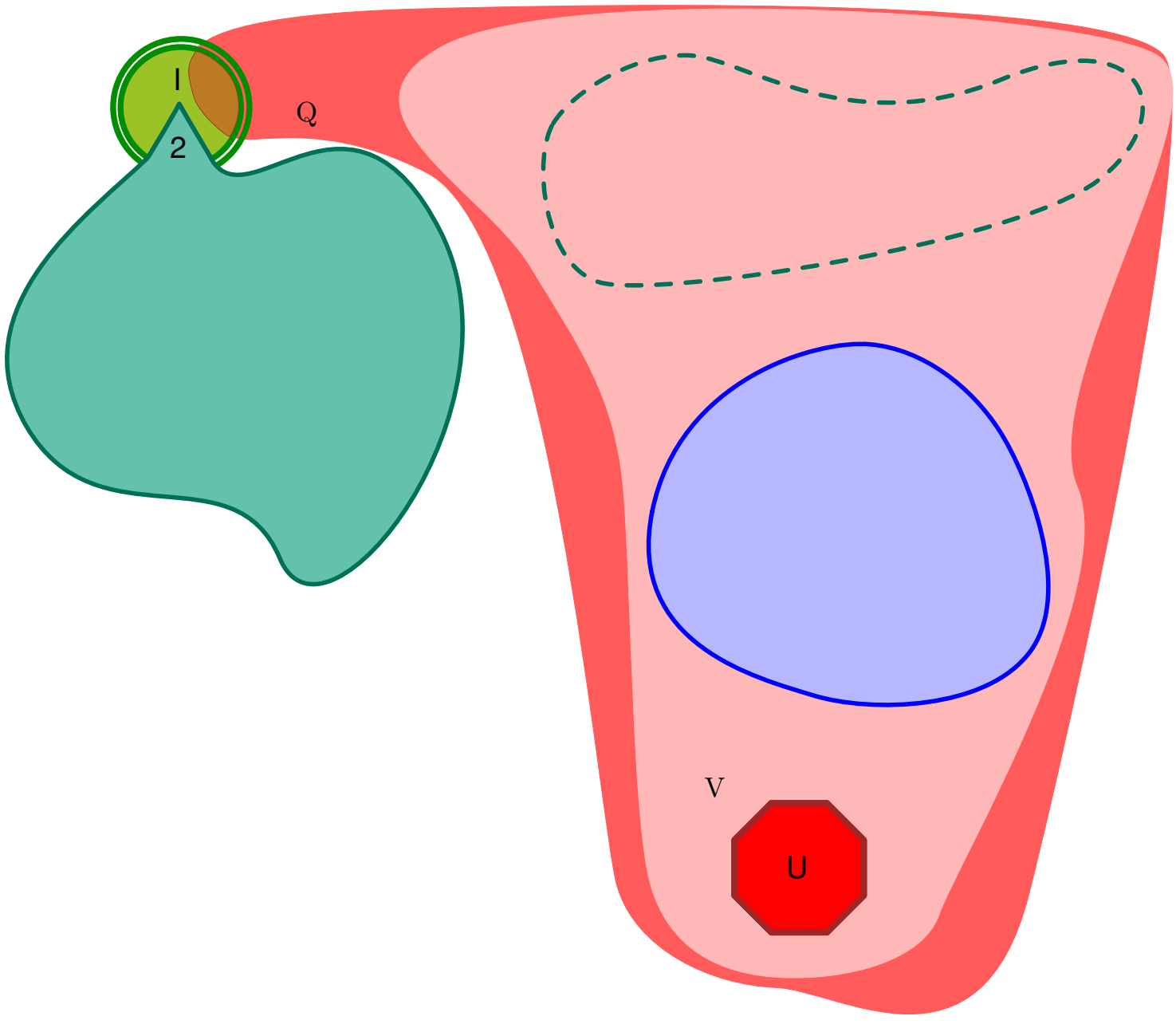
BRAB : intuition



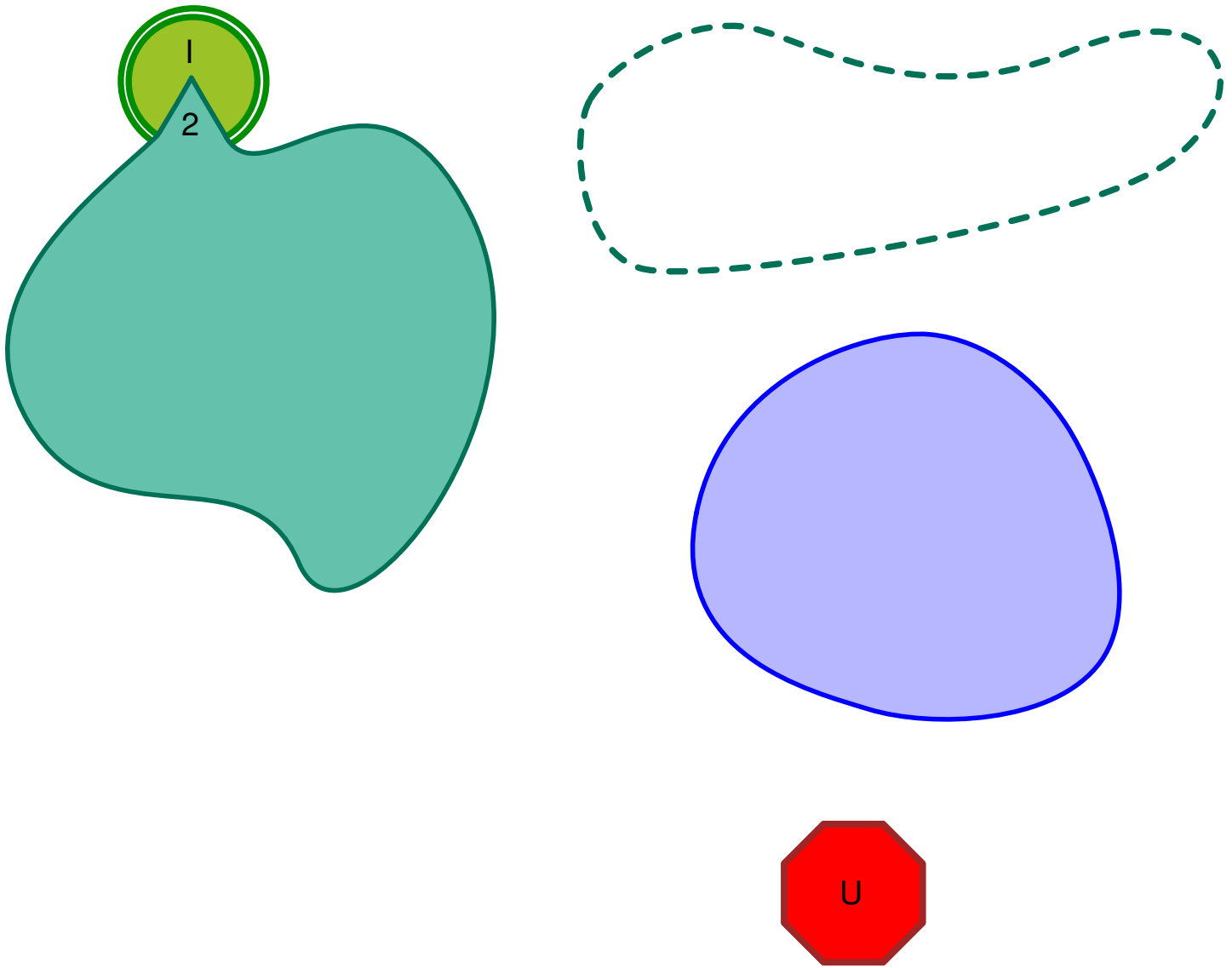
BRAB : intuition



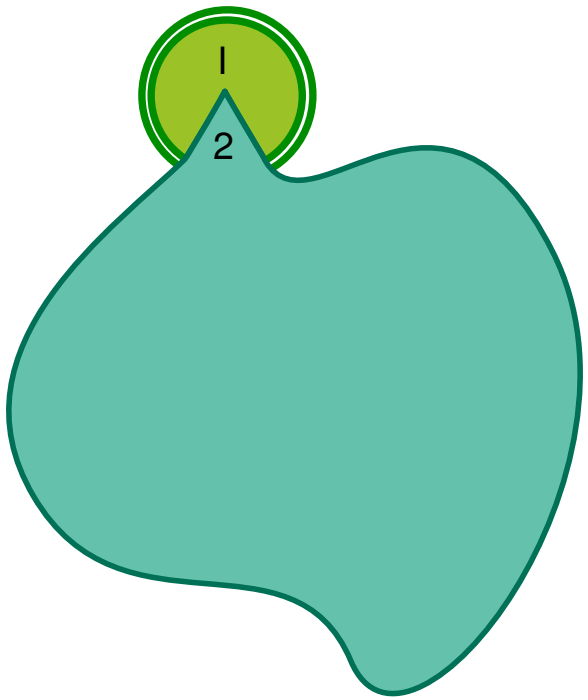
BRAB : intuition

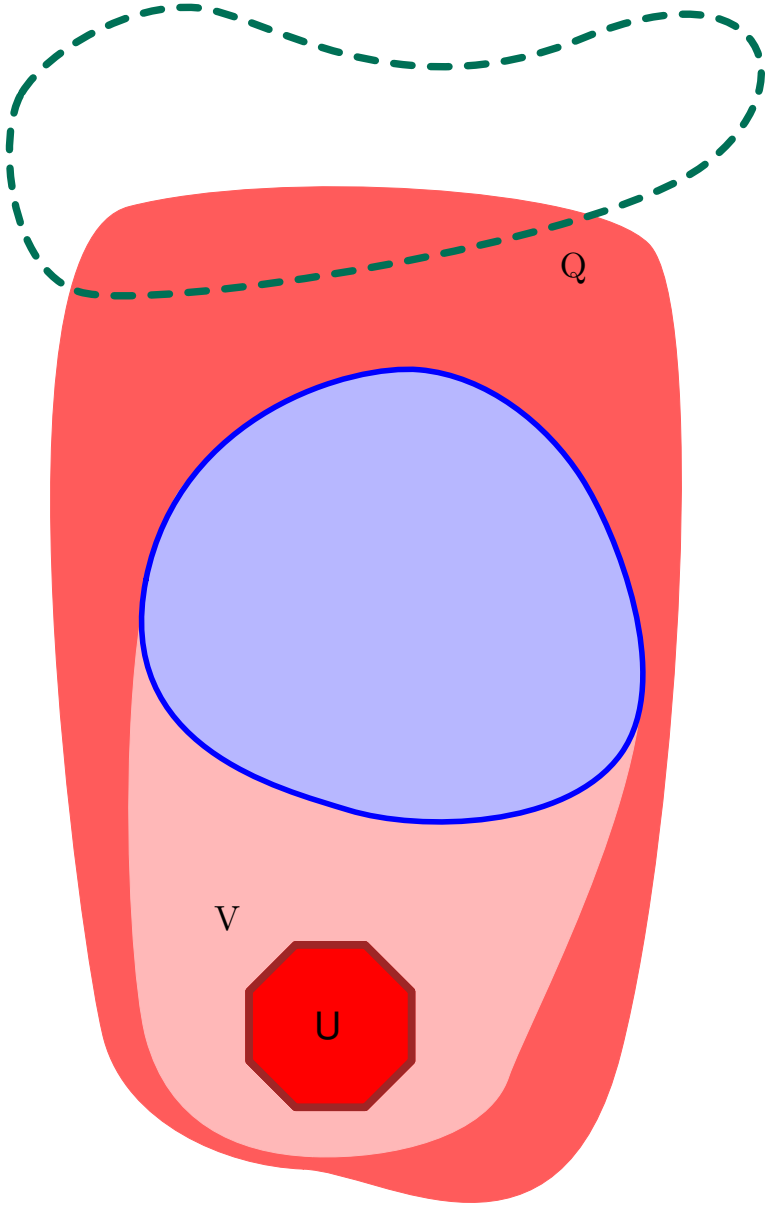


BRAB : intuition

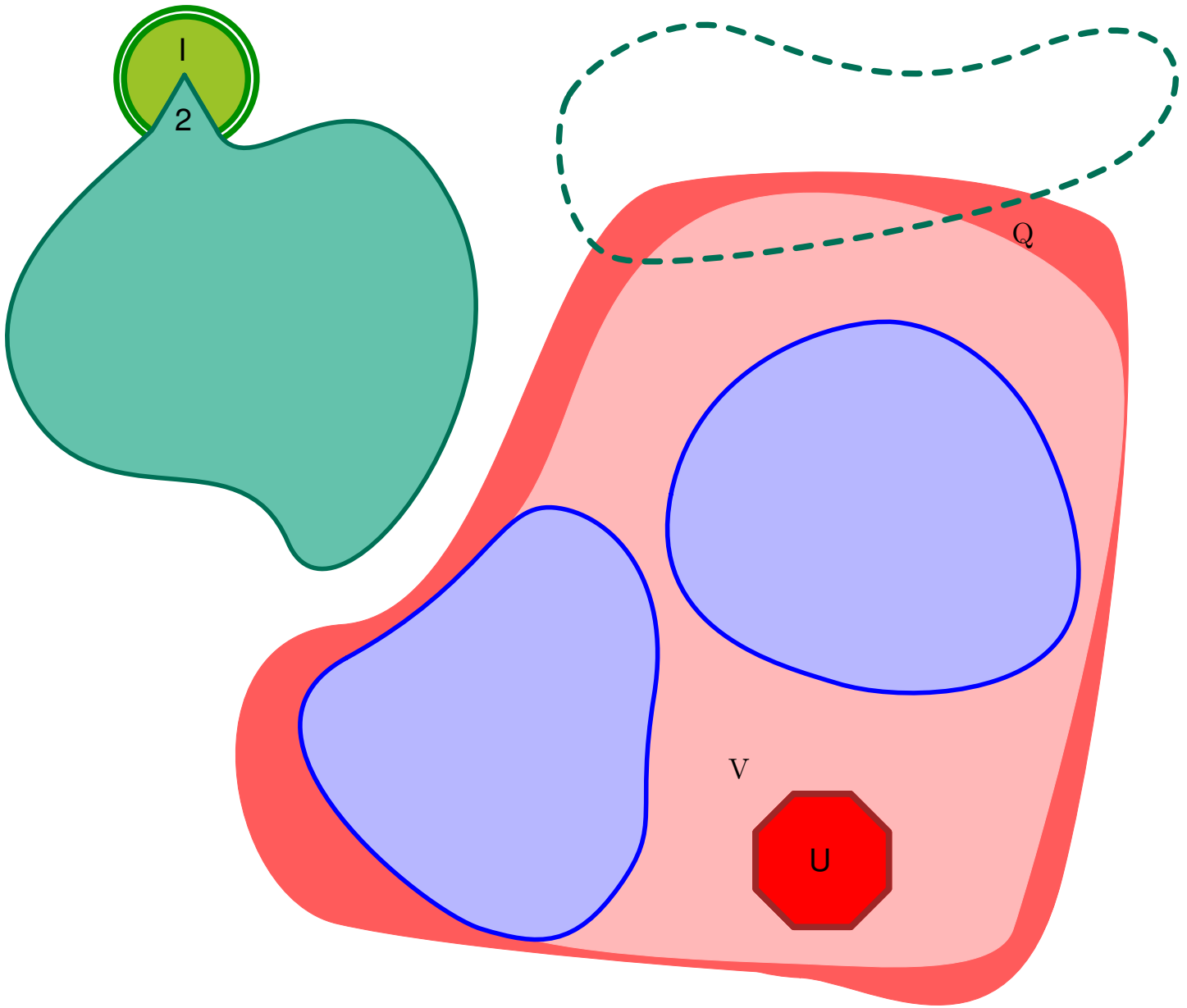


BRAB : intuition

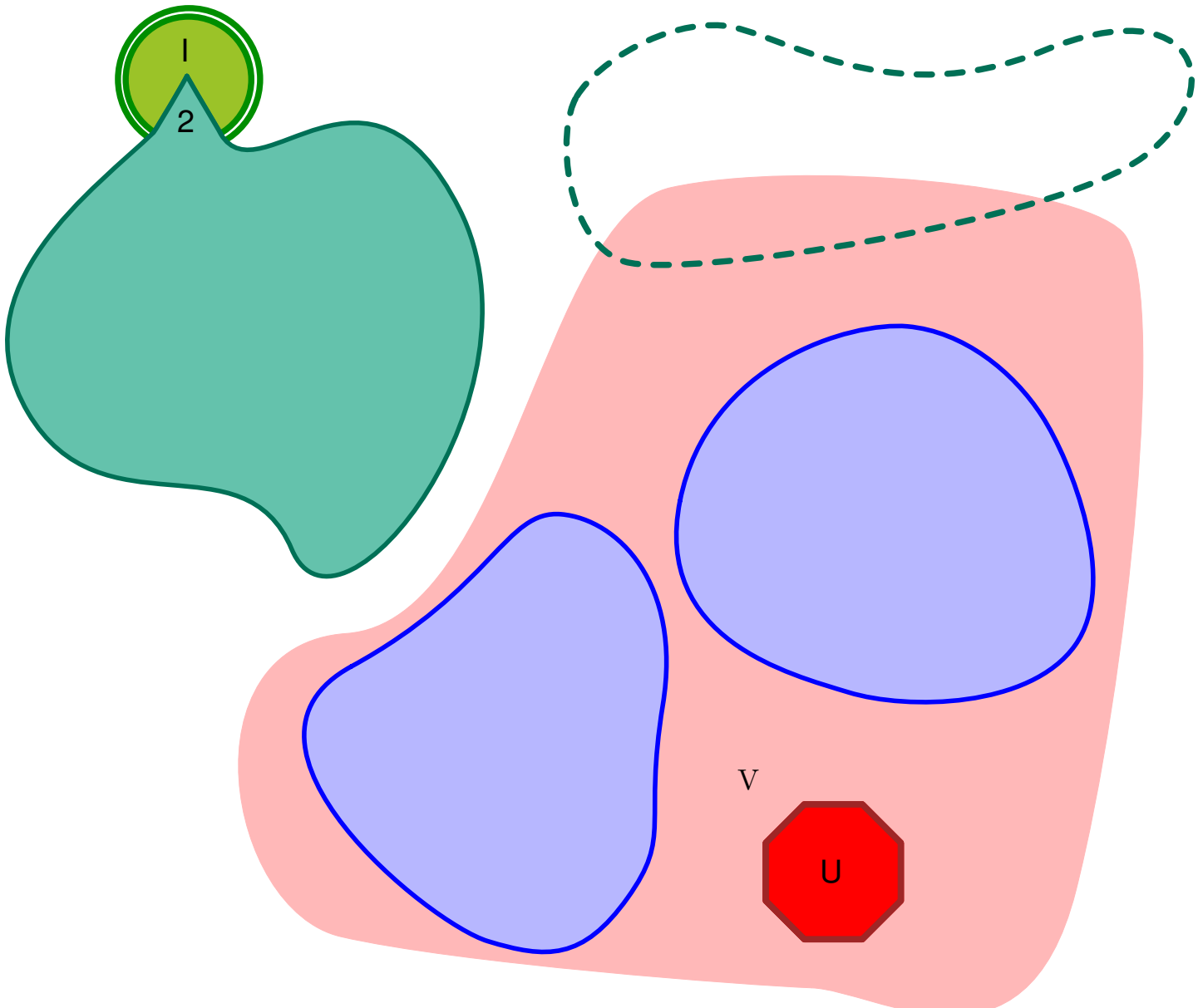




BRAB : intuition



BRAB : intuition



BRAB

I : états initiaux U : états dangereux (cubes) τ : transitions

---

BRAB () :

```
B := ∅;   Kind(U) := Orig;   From(U) := U;
while BWDA() = unsafe do
    if Kind(F) = Orig then return unsafe
    B := B ∪ { From(F) };
return safe
```

BRAB

I : états initiaux U : états dangereux (cubes) τ : transitions

---

BWD ():

  V := $\emptyset$;

  push(Q, U);

  while not empty(Q) **do**

    φ := 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, $\mathtt{pre}_\tau(\varphi)$);

  return safe

BRAB

I : états initiaux U : états dangereux (cubes) τ : transitions

---

BWDA () :

  $\mathtt{V} := \emptyset$;

  push($\mathtt{Q}$, U);

  while not empty($\mathtt{Q}$) **do**

    $\varphi := \text{pop}(\mathtt{Q})$;

    if $\varphi \wedge$ I sat **then**

      $\mathtt{F} := \varphi$;

      return unsafe

    if $\neg(\varphi \models \bigvee_{\psi \in \mathtt{V}} \psi)$ **then**

      $\mathtt{V} := \mathtt{V} \cup \{\varphi\}$;

      push($\mathtt{Q}$, $\text{Approx}_\tau(\varphi)$);

  return safe

BRAB

I : états initiaux U : états dangereux (cubes) τ : transitions

---

Approx_τ (φ) :

foreach ψ in `candidates`(φ) **do**

if $\psi \notin$ B $\wedge$ Oracle(ψ) = Good **then**

`Kind`(ψ) := Appr ;

...

return ψ

...

return $\texttt{pre}_\tau(\varphi)$

Oracle : formula $\rightarrow \{\text{Good}, \text{Bad}\}$

BRAB

I : états initiaux U : états dangereux (cubes) τ : transitions

---

Approx$_\tau$ (φ) :

foreach ψ in `candidates`(φ) **do**

if $\psi \notin$ B $\wedge$ Oracle(ψ) = Good **then**

`Kind`(ψ) := Appr ;

...

return ψ

...

return `pre`$_\tau$(φ)

Oracle (ψ) :

if $\mathcal{M} \not\models \psi$ **then return** Good

else return Bad

$\mathcal{M}$:= `FWD`(d_{max}, k)

Exemple : protocole de cohérence de cache German-*esque*

Client i :

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

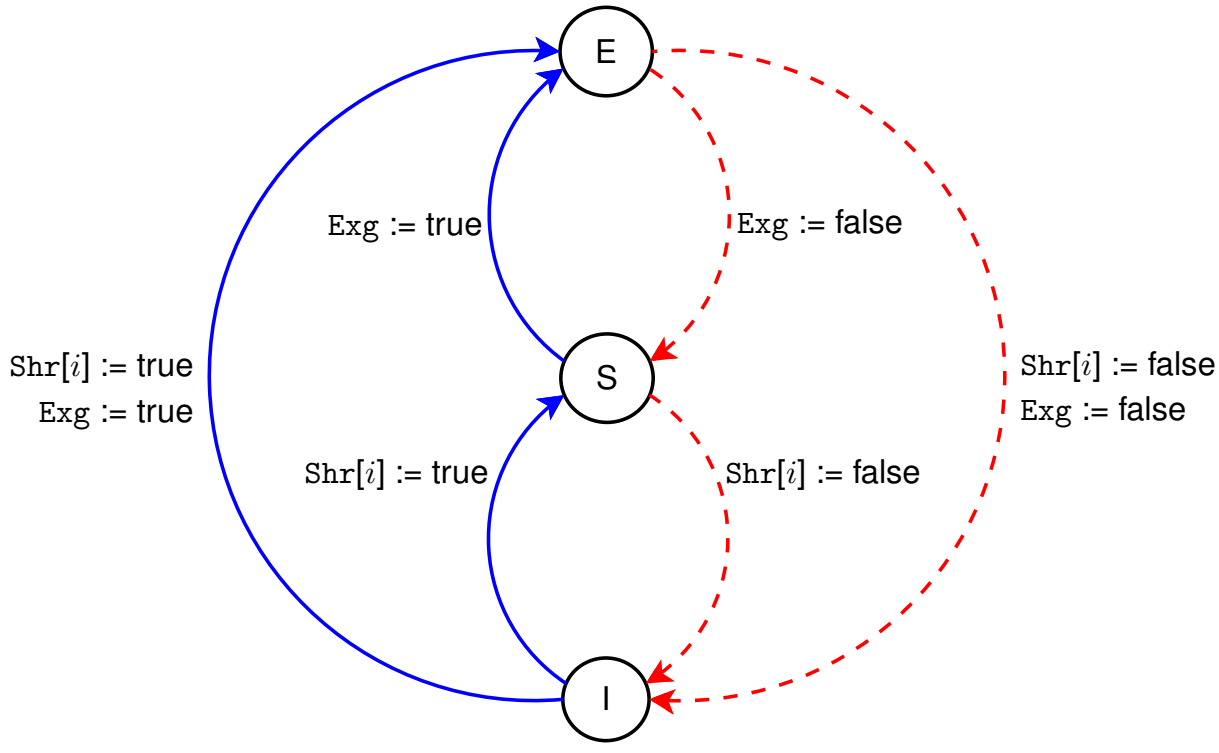
Répertoire :

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

$\texttt{Ptr} \in proc$

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

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

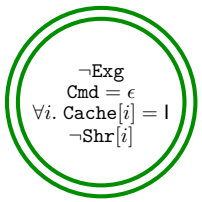


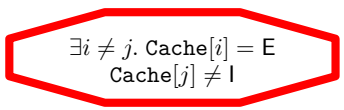
États initiaux : $\forall i.\, \texttt{Cache}[i] = \mathsf{I} \wedge \neg\texttt{Shr}[i] \wedge \neg\texttt{Exg} \wedge \texttt{Cmd} = \epsilon$

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

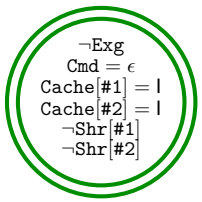
(cubes)

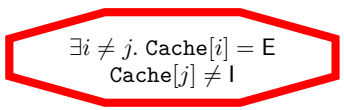
Exemple : BRAB sur German-*esque*



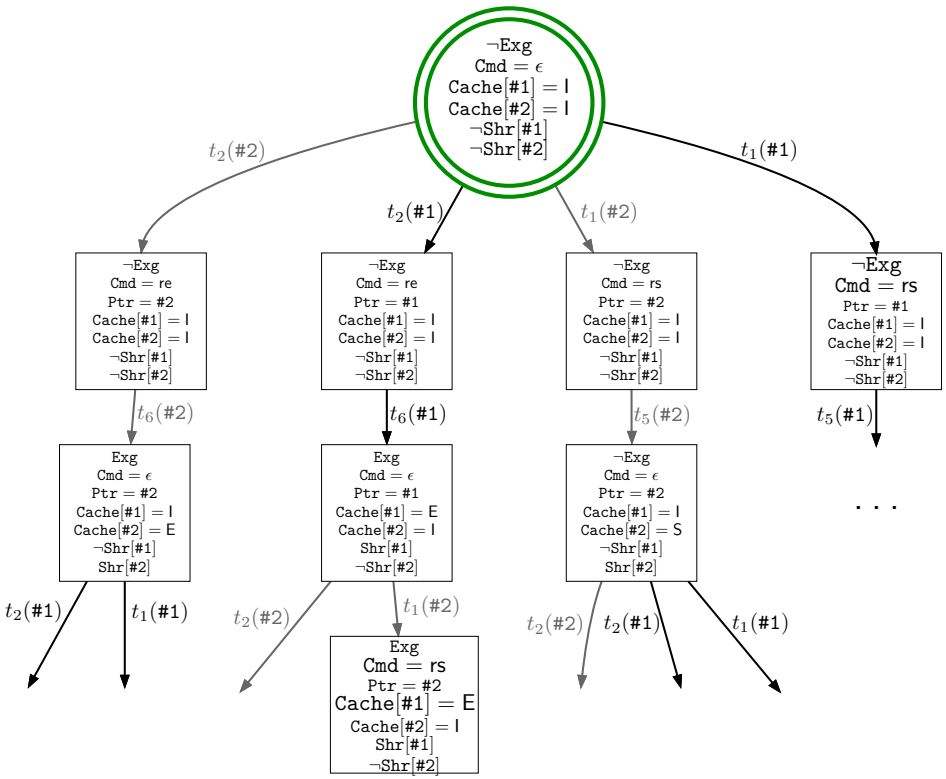


Exemple : BRAB sur German-*esque*



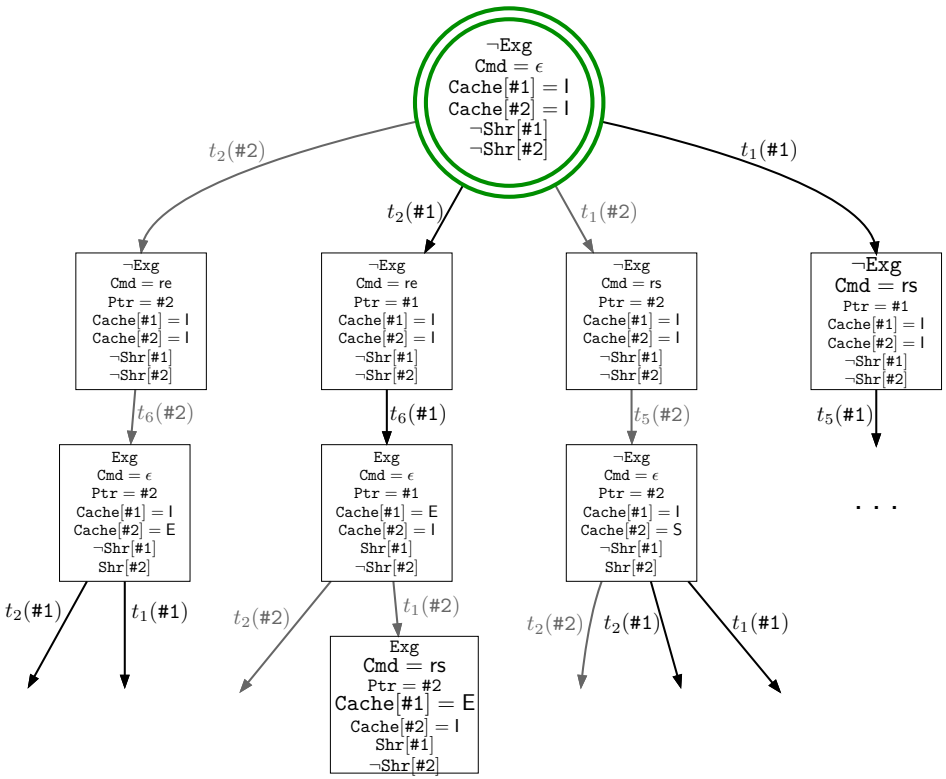


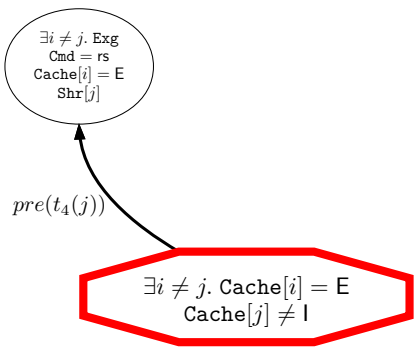
Exemple : BRAB sur German-*esque*



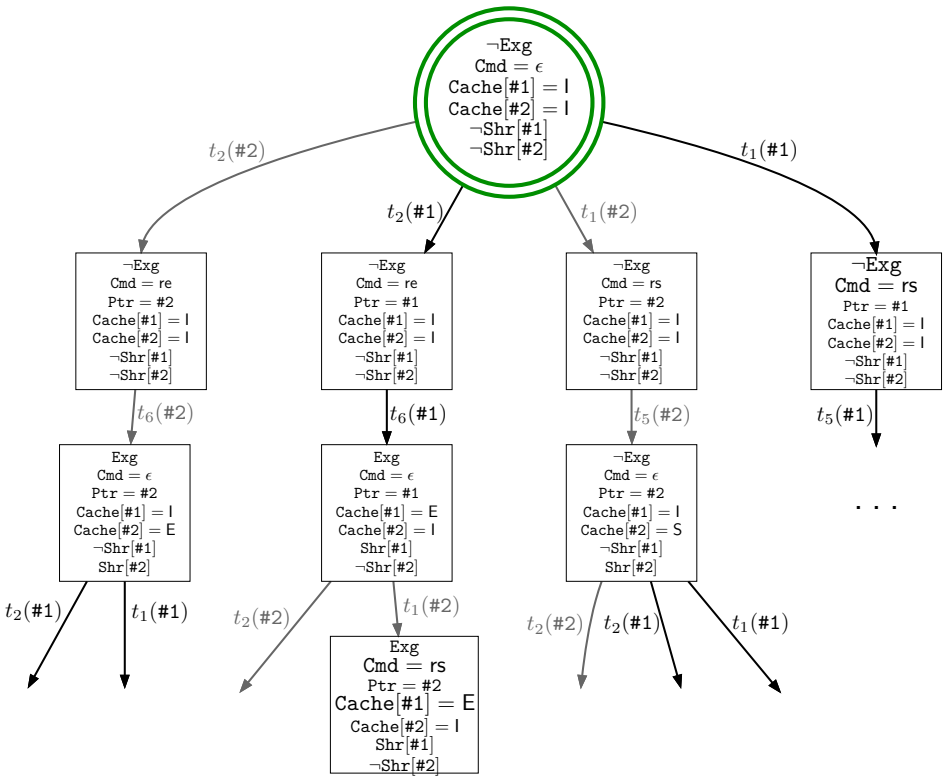
$\exists i \neq j.\ \texttt{Cache}[i] = \texttt{E}$
$\texttt{Cache}[j] \neq \texttt{I}$

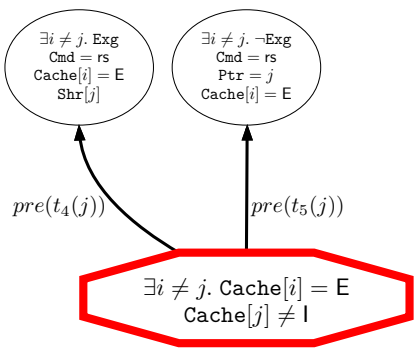
Exemple : BRAB sur German-*esque*



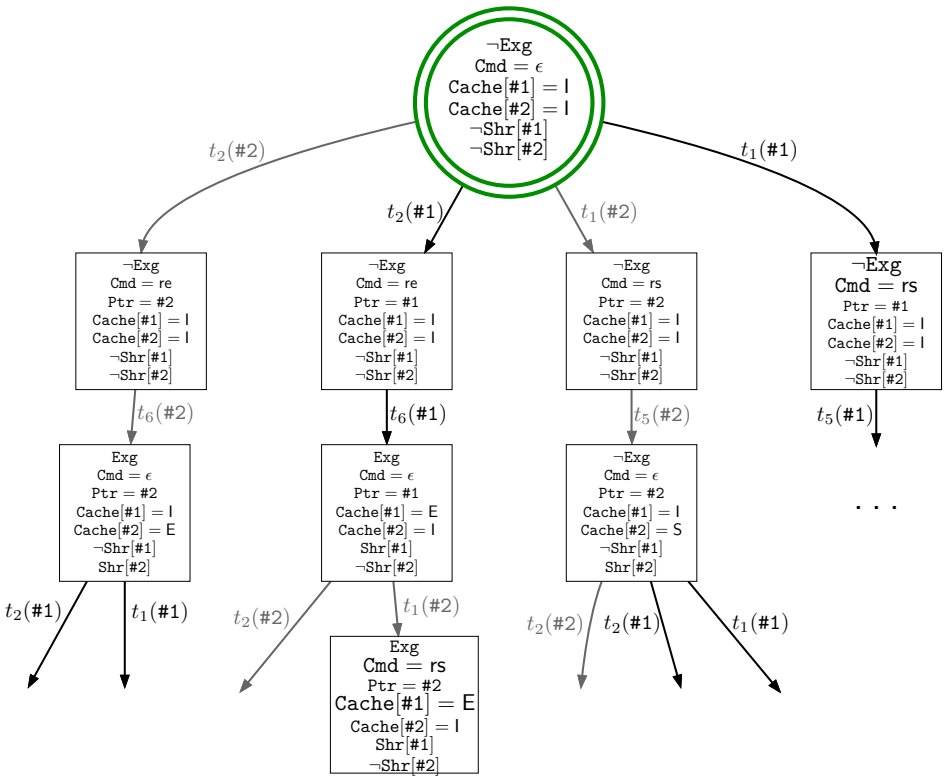


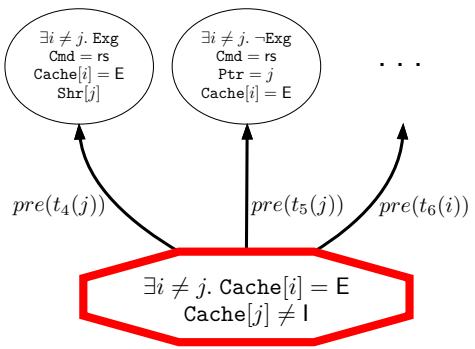
Exemple : BRAB sur German-*esque*



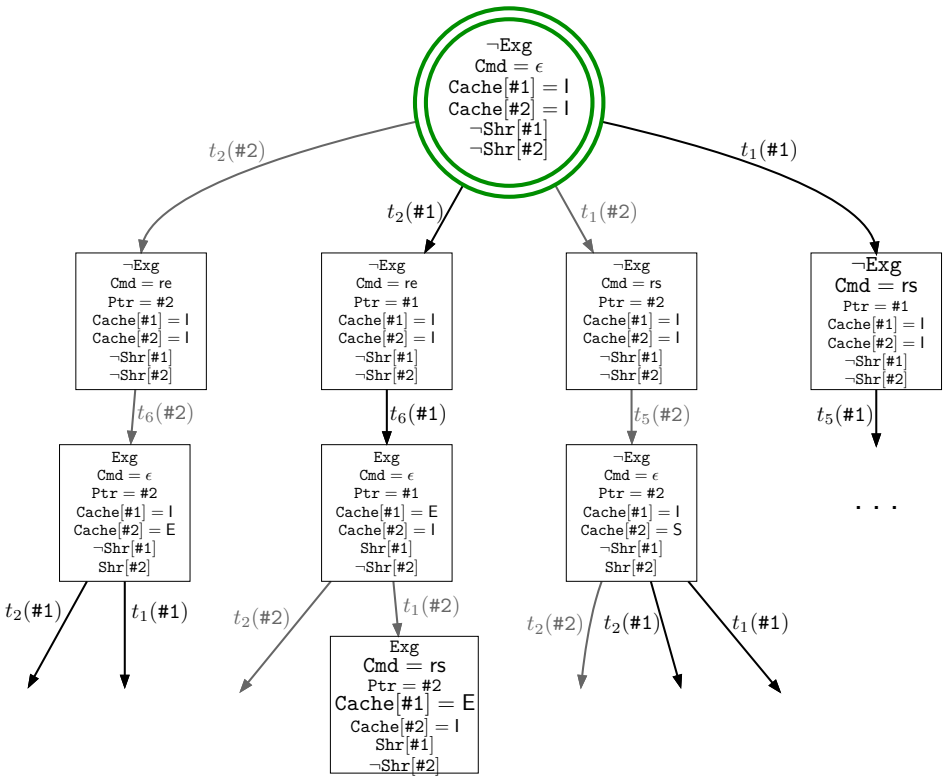


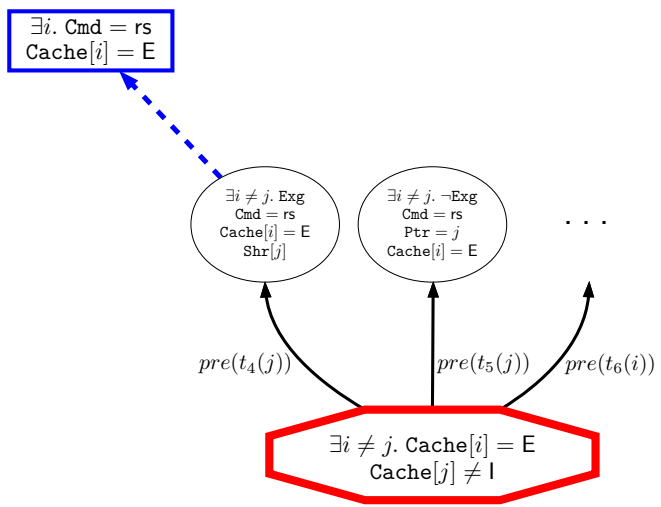
Exemple : BRAB sur German-*esque*



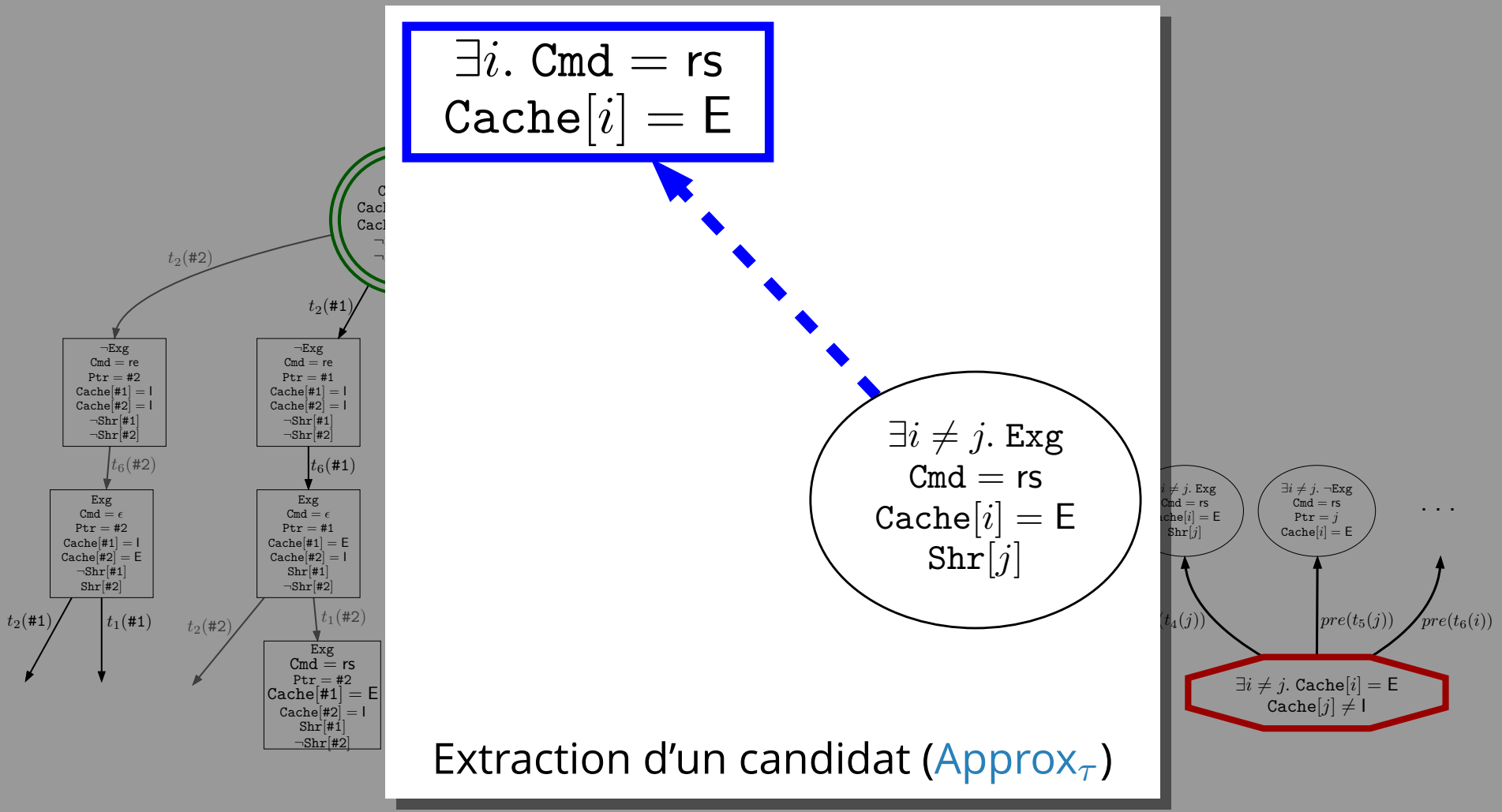


Exemple : BRAB sur German-*esque*



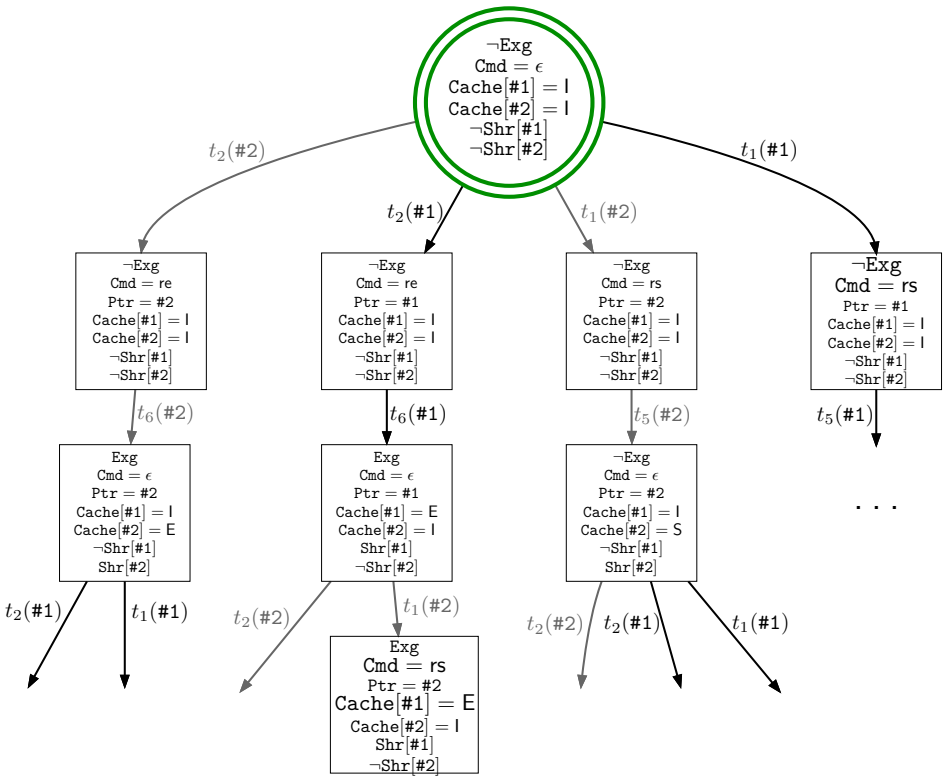


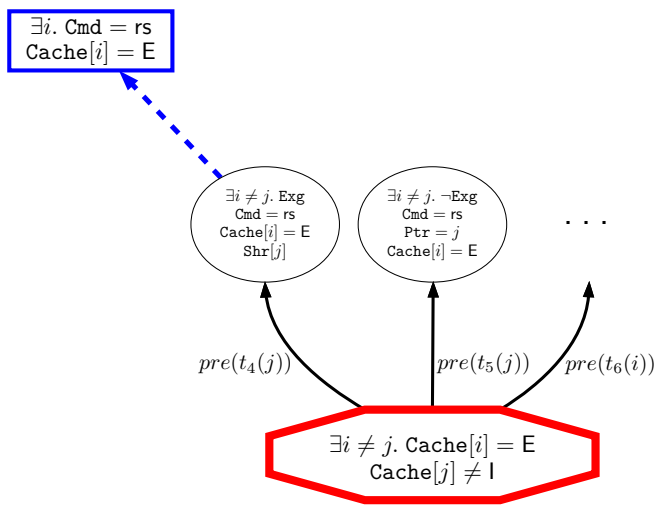
Exemple : BRAB sur German-*esque*



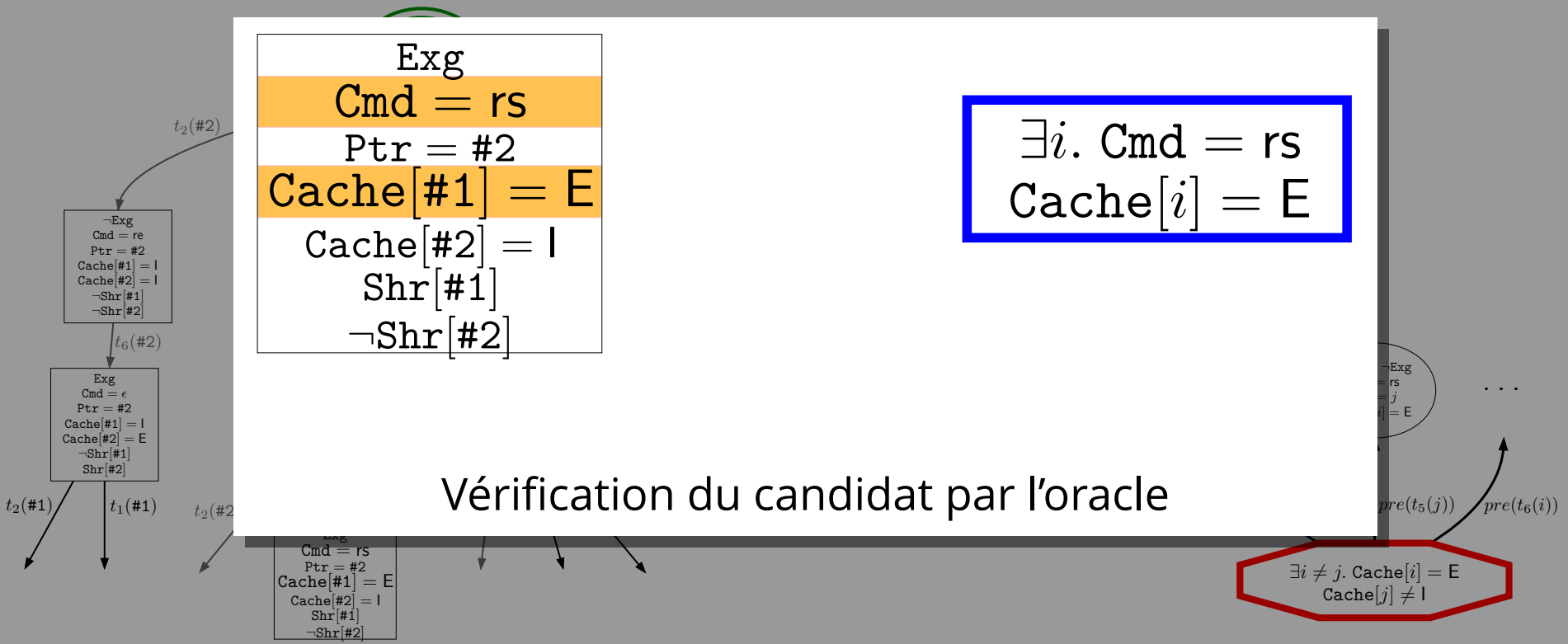
Extraction d'un candidat (Approx_{τ})

Exemple : BRAB sur German-*esque*

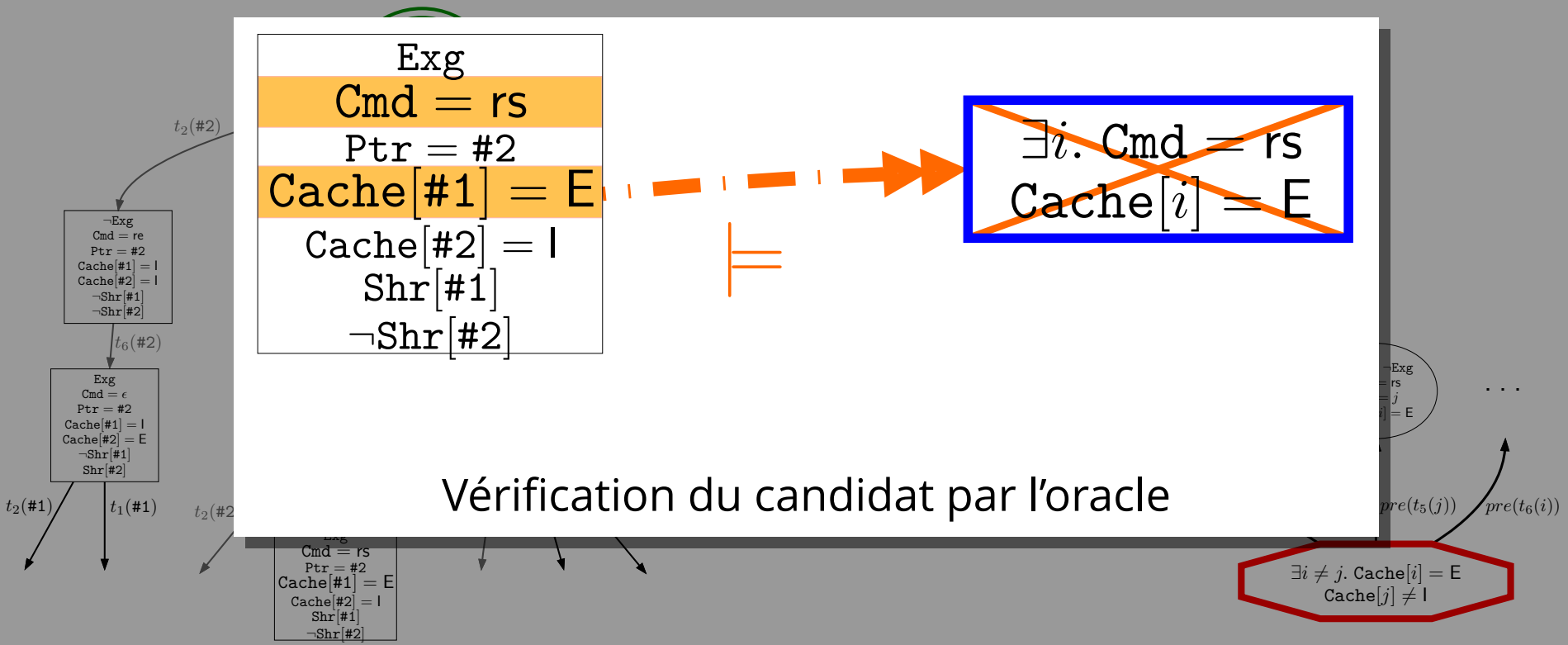




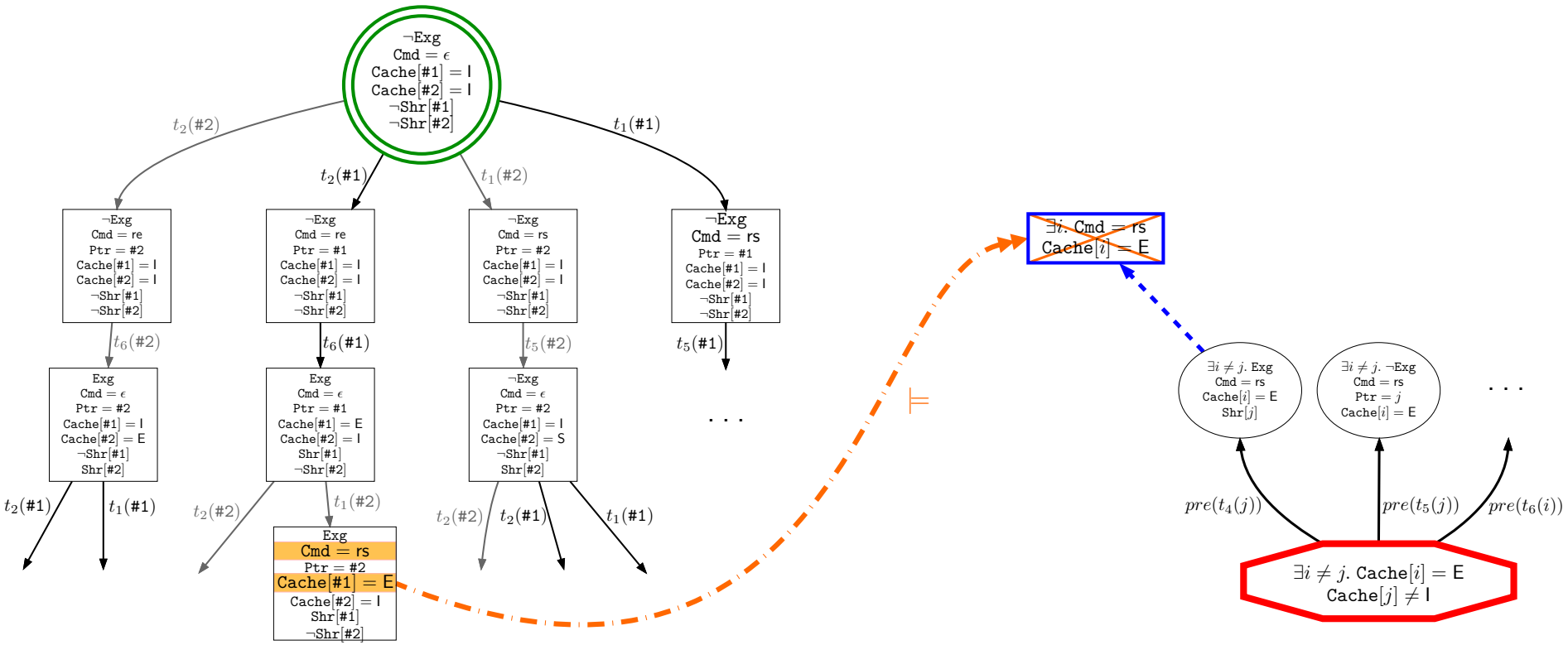
Exemple : BRAB sur German-*esque*



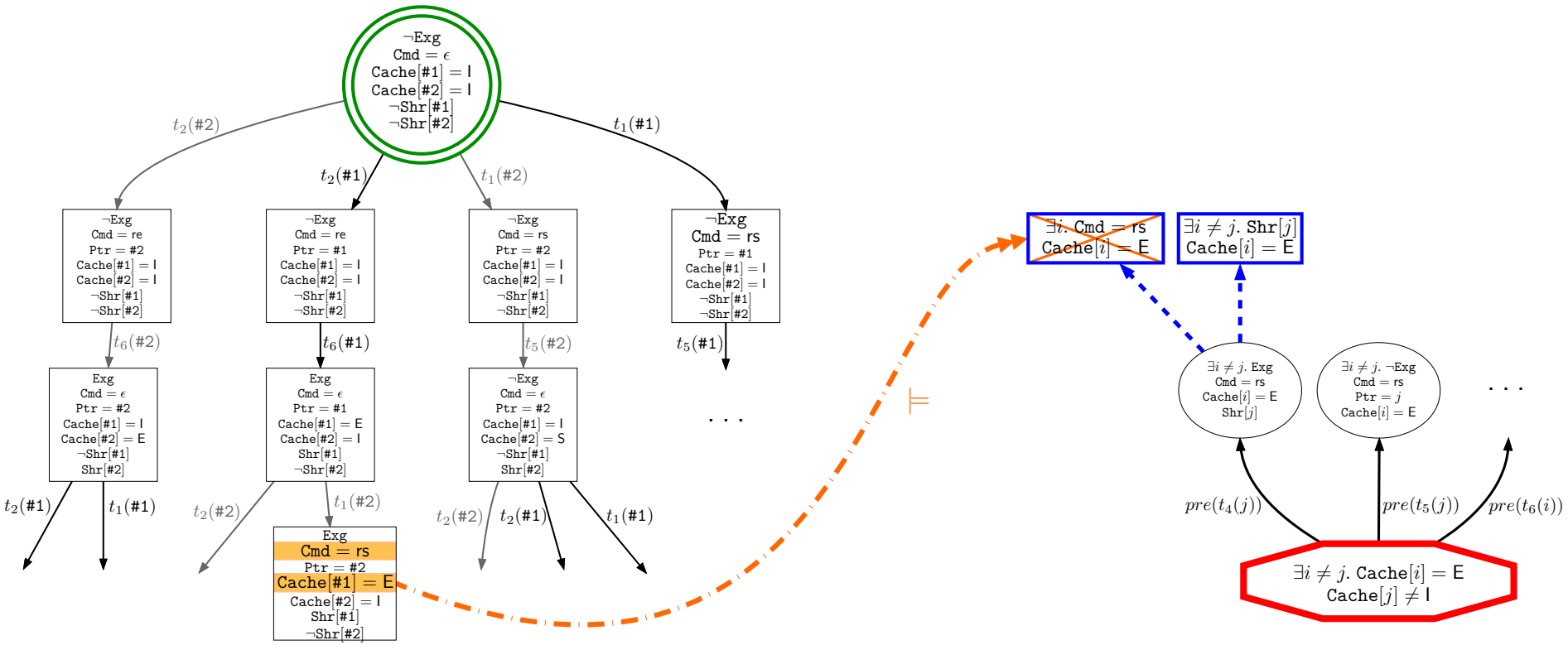
Exemple : BRAB sur German-*esque*



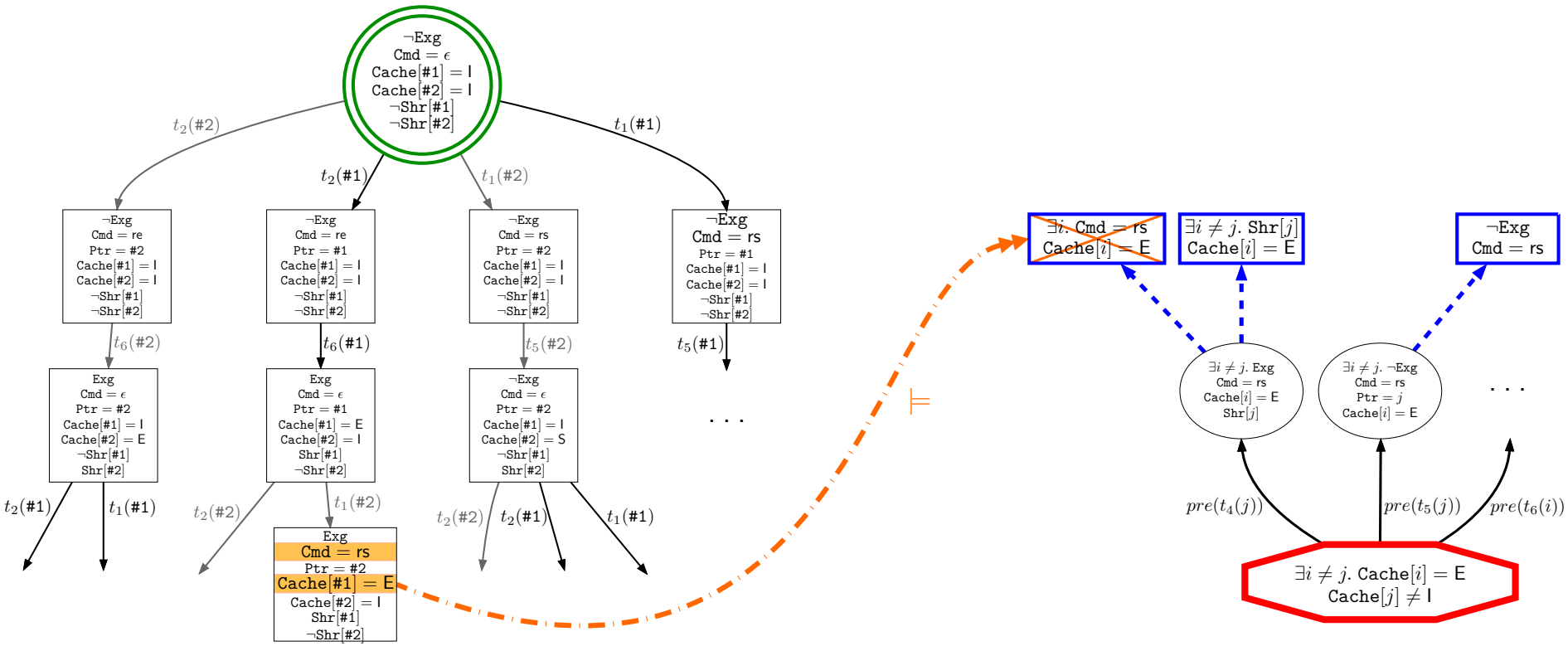
Exemple : BRAB sur German-*esque*



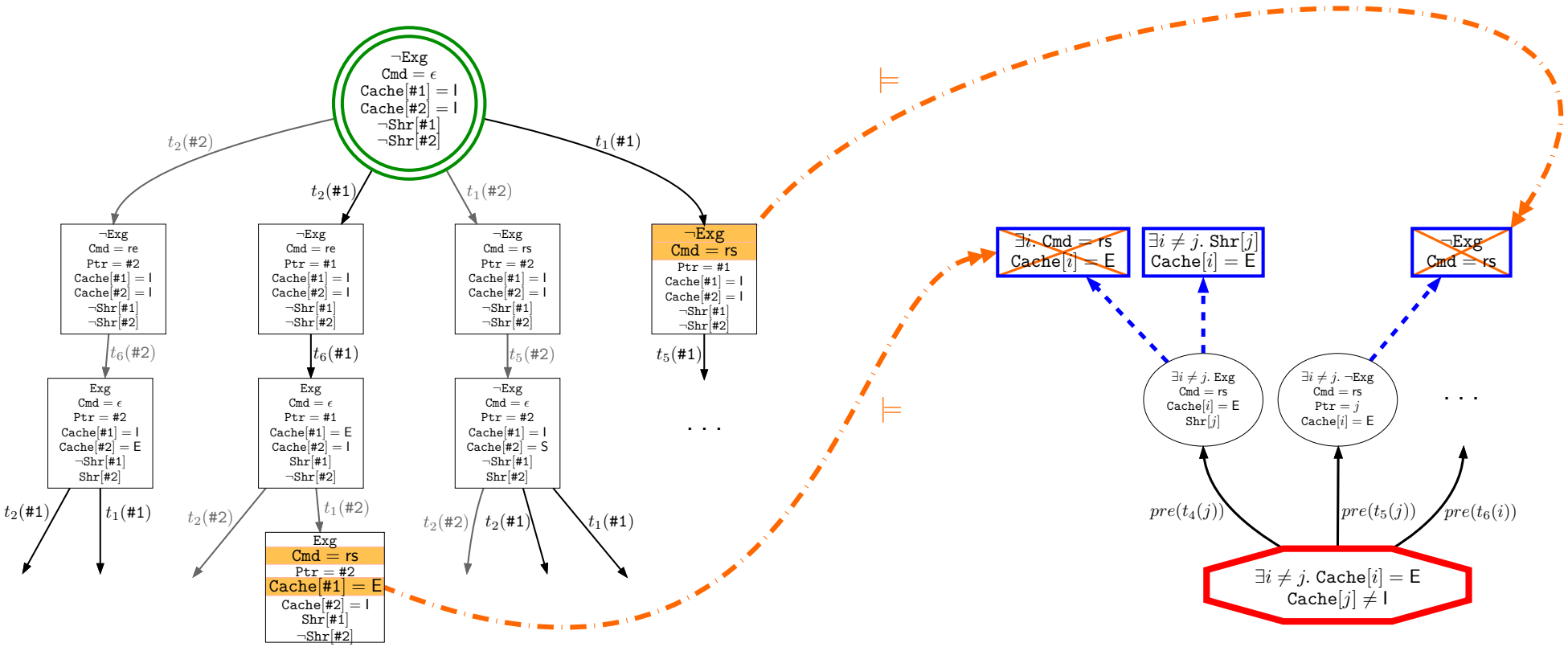
Exemple : BRAB sur German-*esque*



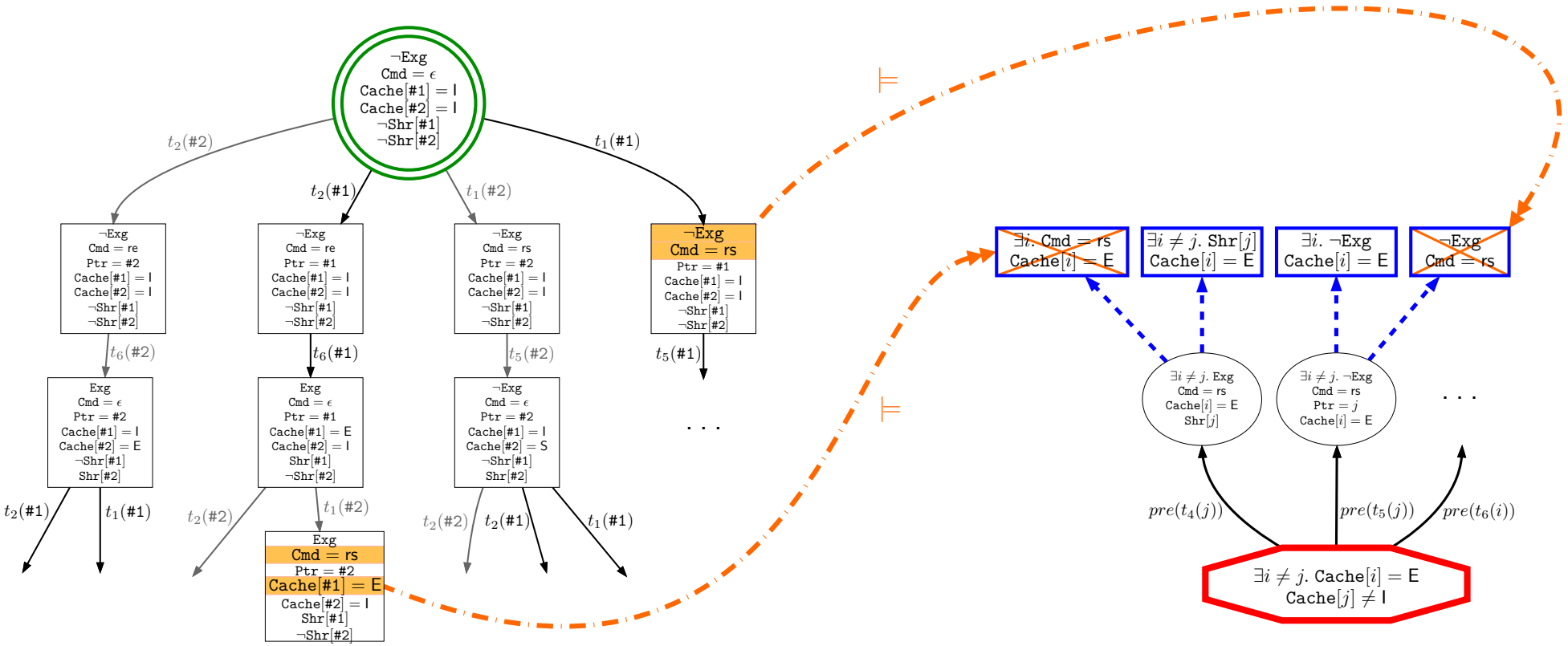
Exemple : BRAB sur German-*esque*



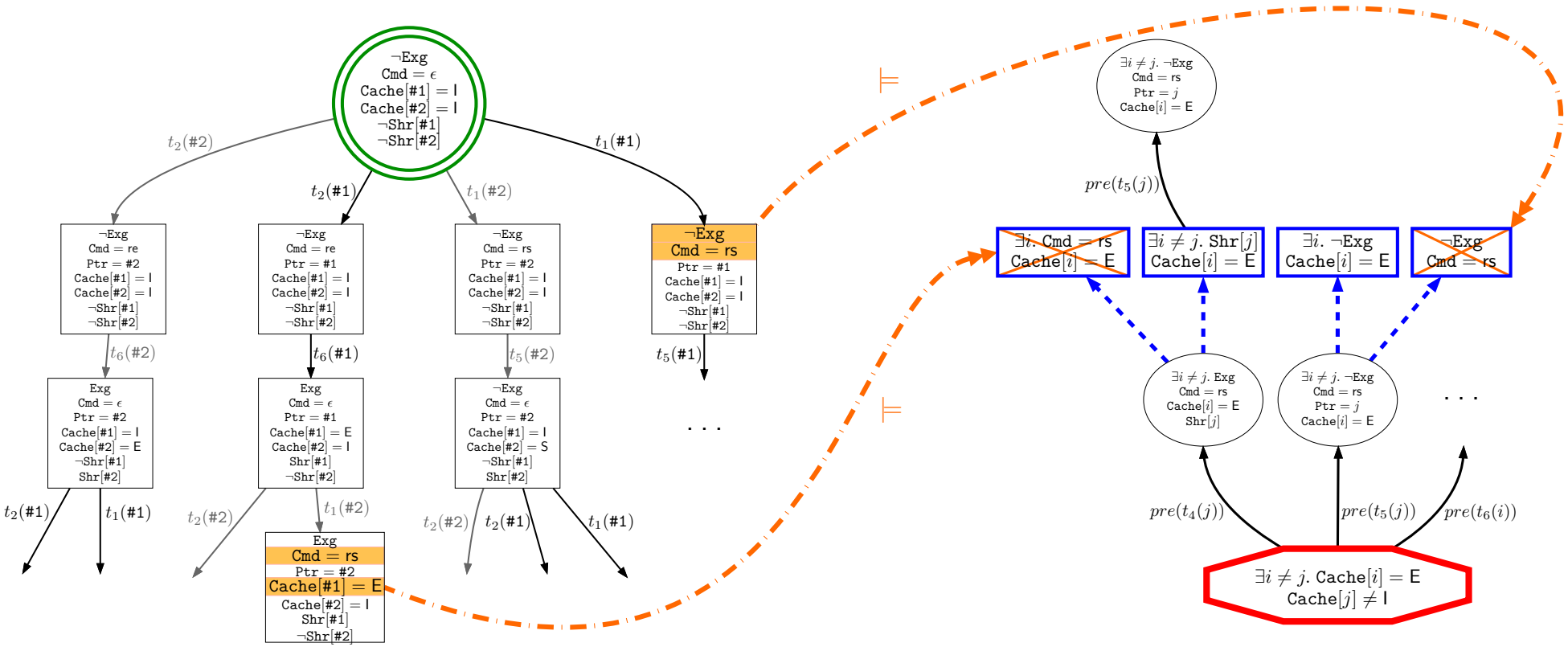
Exemple : BRAB sur German-*esque*



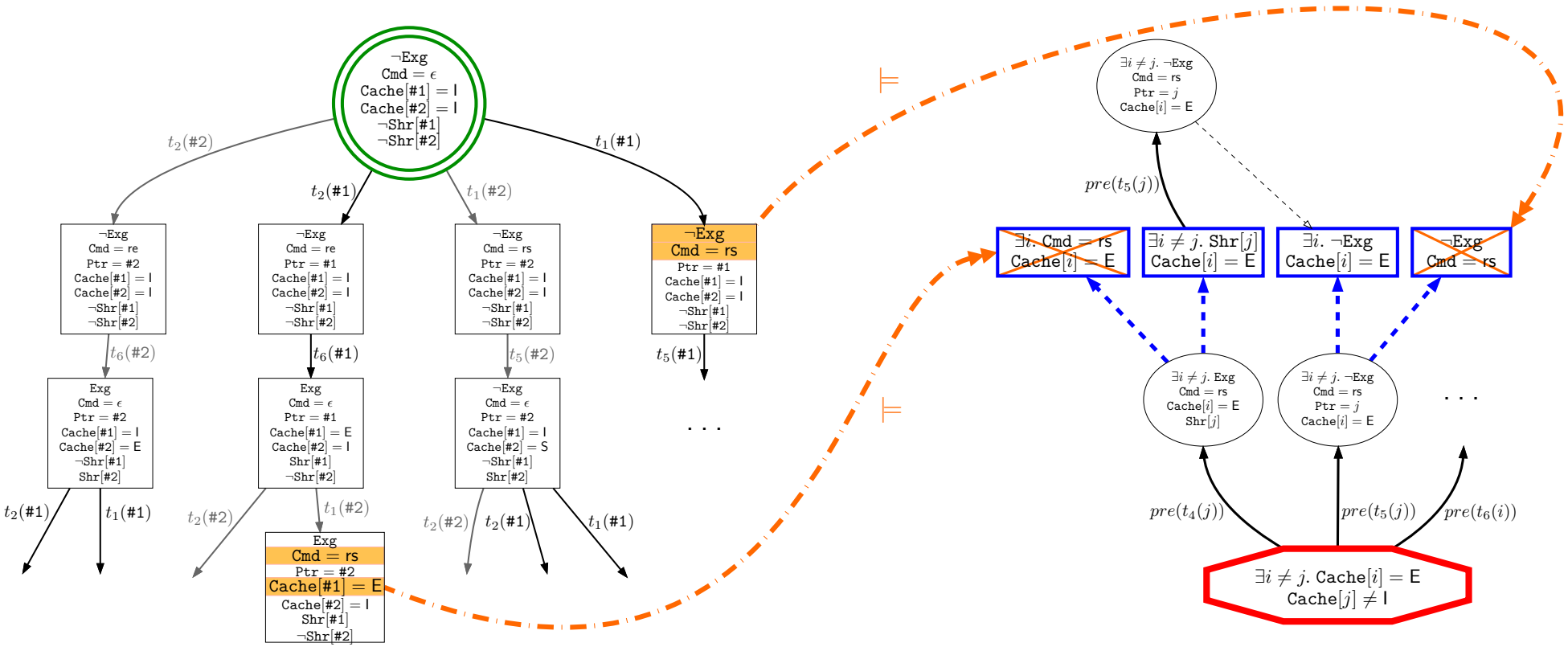
Exemple : BRAB sur German-*esque*



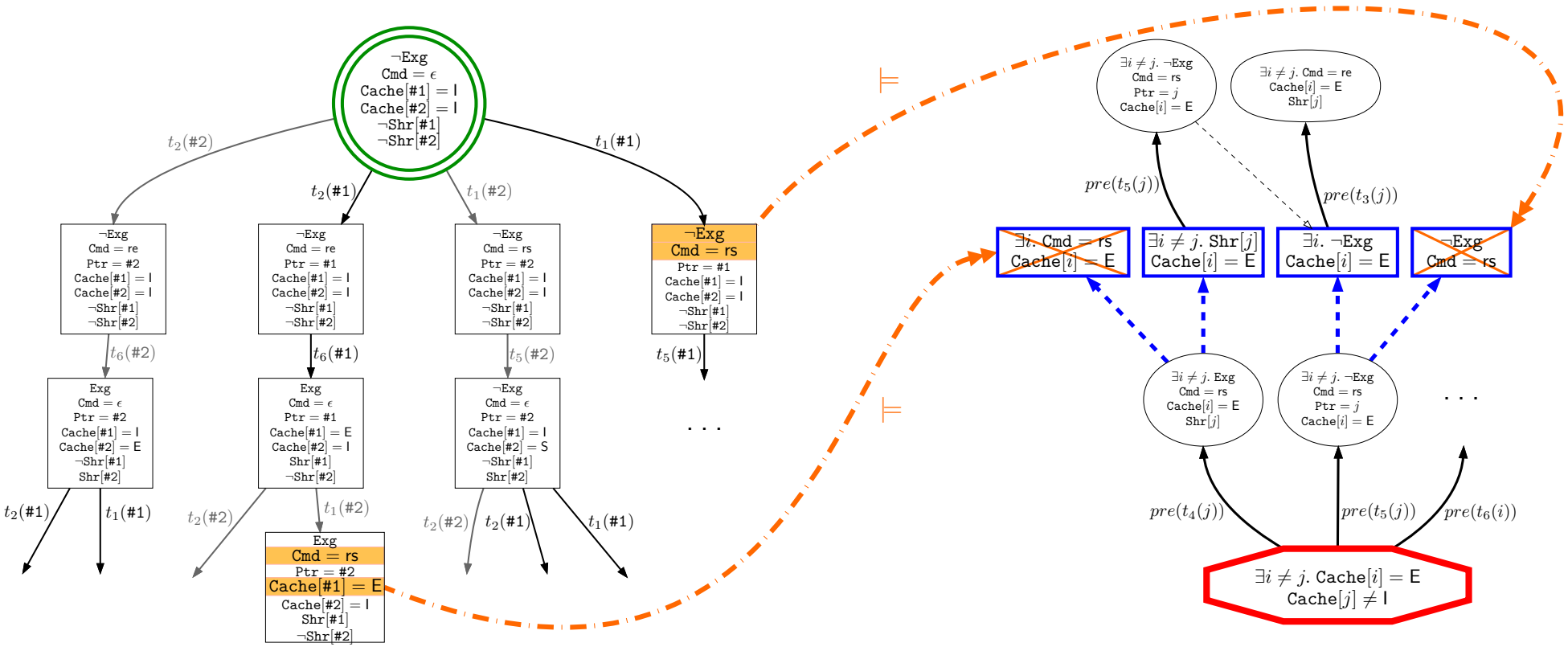
Exemple : BRAB sur German-*esque*



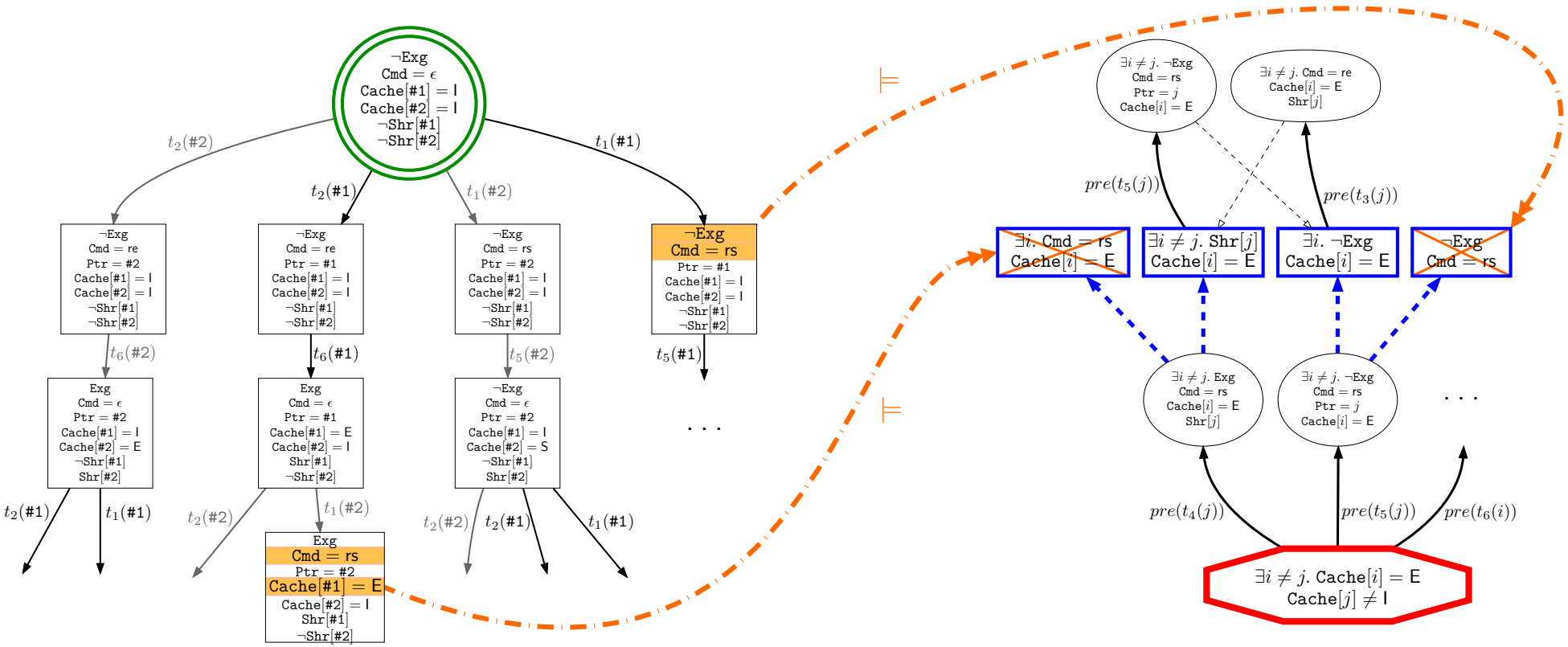
Exemple : BRAB sur German-*esque*



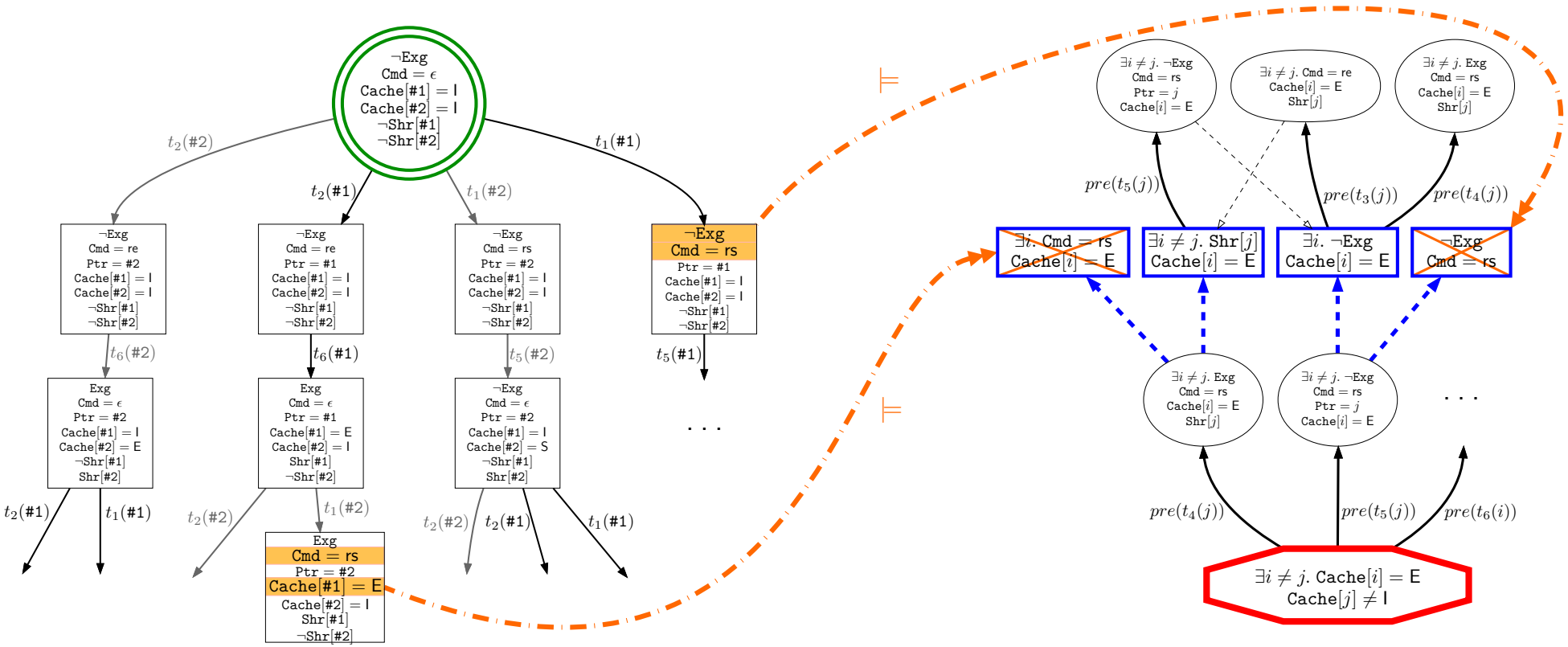
Exemple : BRAB sur German-*esque*



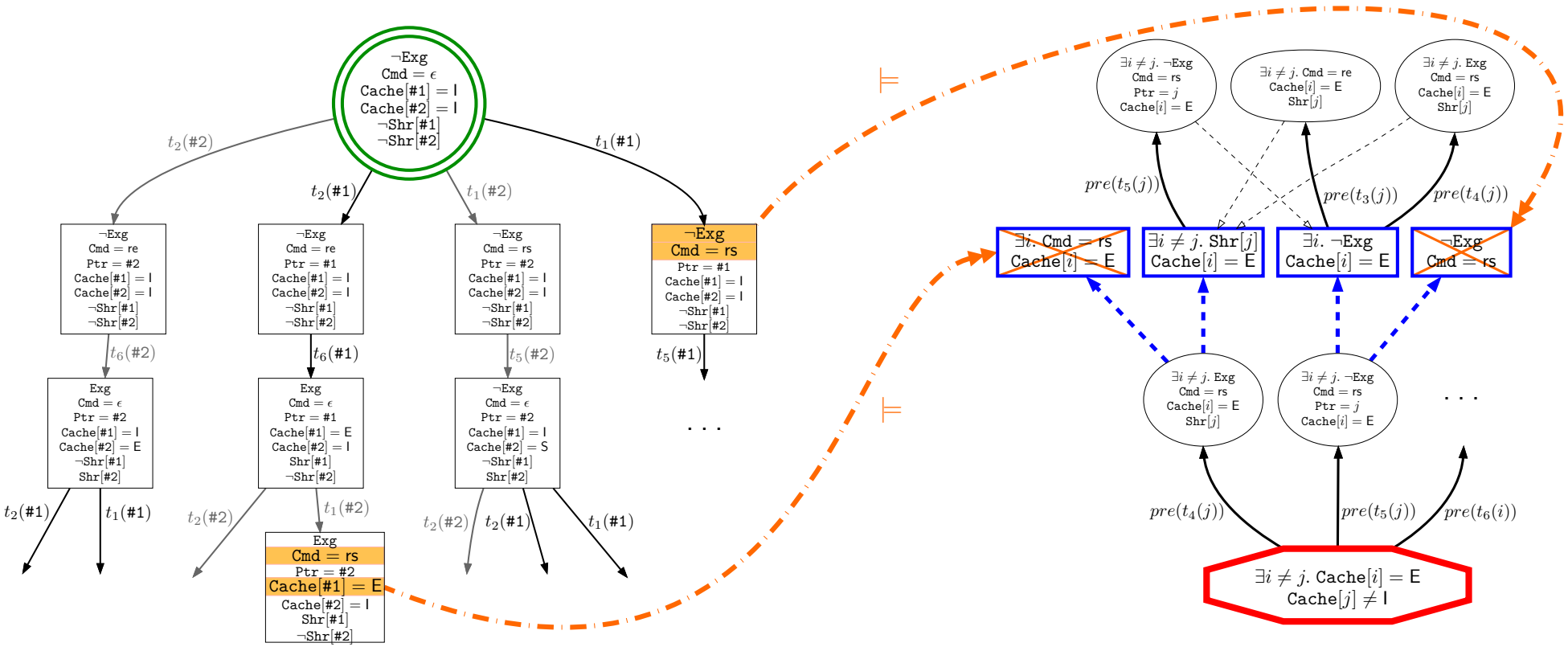
Exemple : BRAB sur German-*esque*



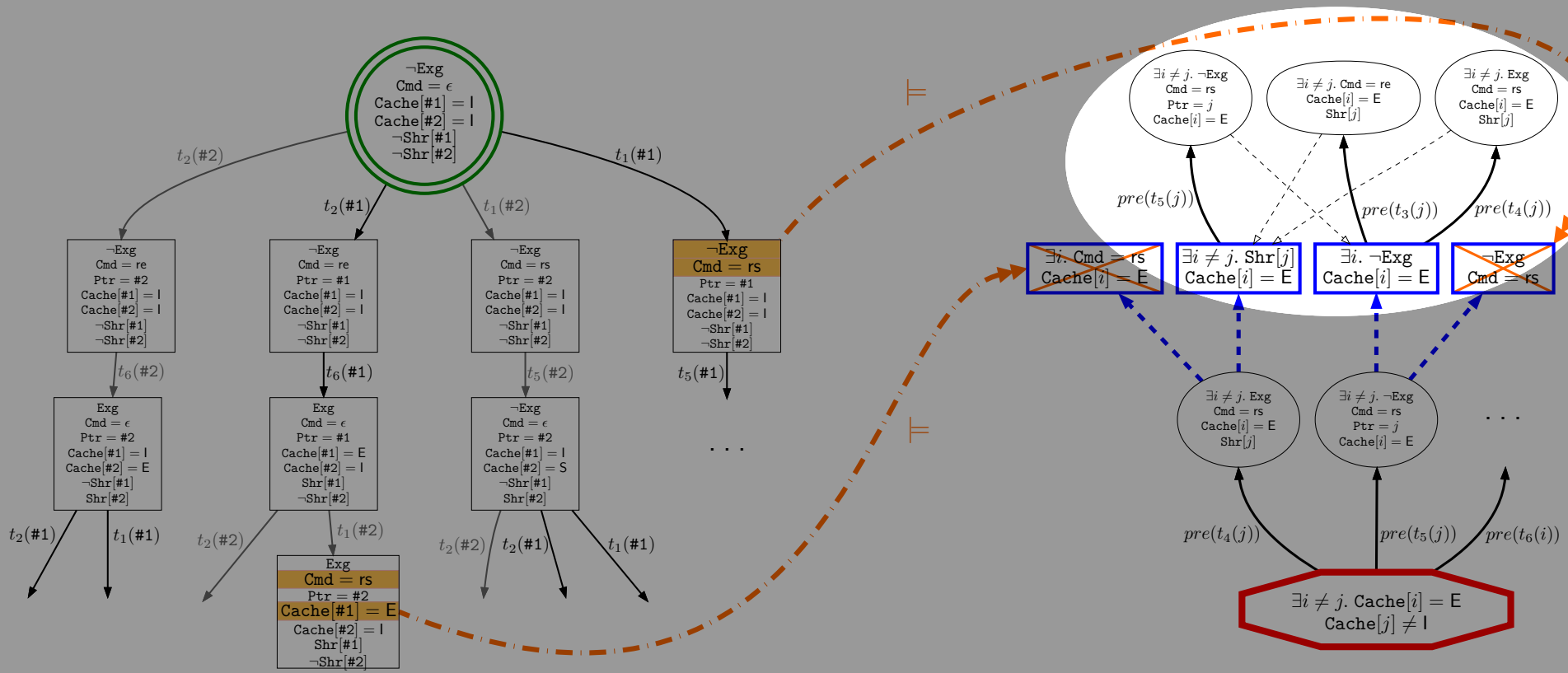
Exemple : BRAB sur German-*esque*



Exemple : BRAB sur German-*esque*



Exemple : BRAB sur German-*esque*



Quelques benchmarks

	BRAB	Cubicle	CMurphi		
Szymanski_at	**0,04s** 12	4m41s	8,04s (8)	5m12s (10)	2h50m (12)
German_Baukus	**0,08s** 22	5,01s	0,74s (4)	19m35s (8)	4h49m (10)
German.CTC	**0,11s** 23	4,58s	1,83s (4)	43m46s (8)	12h35m (10)
German_pfs	**0,09s** 37	2m45s	0,99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	**54,1s** (1) 4	2h01m	5,68s (4)	2m58s (5)	1h36m (6)
Szymanski_na	**0,19s** 13	T.O.	0,88s (4)	8m25s (6)	7h08m (8)
Flash_nodata	**0,10s** 31	O.M.	4,86s (3)	3m33s (4)	2h46m (5)
Flash	**1m30s** 109	O.M.	1m27s (3)	2h15m (4)	O.M. (5)

O.M. $>$ 20 GB

T.O. $>$ 20 h

Quelques benchmarks

	BRAB	Cubicle	CMurphi		
Szymanski_at	**0,04s** 12	4m41s	8,04s (8)	5m12s (10)	2h50m (12)
German_Baukus	**0,08s** 22	5,01s	0,74s (4)	19m35s (8)	4h49m (10)
German.CTC	**0,11s** 23	4,58s	1,83s (4)	43m46s (8)	12h35m (10)
German_pfs	**0,09s** 37	2m45s	0,99s (4)	22m56s (8)	5h30m (10)
Chandra-Toueg	**54,1s** (1) 4	2h01m	5,68s (4)	2m58s (5)	1h36m (6)
Szymanski_na	**0,19s** 13	T.O.	0,88s (4)	8m25s (6)	7h08m (8)
Flash_nodata	**0,10s** 31	O.M.	4,86s (3)	3m33s (4)	2h46m (5)
Flash	**1m30s** 109	O.M.	1m27s (3)	2h15m (4)	O.M. (5)

O.M. $>$ 20 GB

T.O. $>$ 20 h

Remarques

- correction de BRAB indépendante de l'oracle
- efficacité de BRAB dépend de la précision de l'oracle

Certification

Certification

Comment faire confiance à la réponse du model checker ?

Certification

Comment faire confiance à la réponse du model checker ?

Plusieurs possibilités :

- Vérifier que le model checker est correct par construction
- Produire un certificat à vérifier de manière indépendante

Certification

Comment faire confiance à la réponse du model checker ?

Plusieurs possibilités :

- Vérifier que le model checker est correct par construction
- Produire un certificat à vérifier de manière indépendante

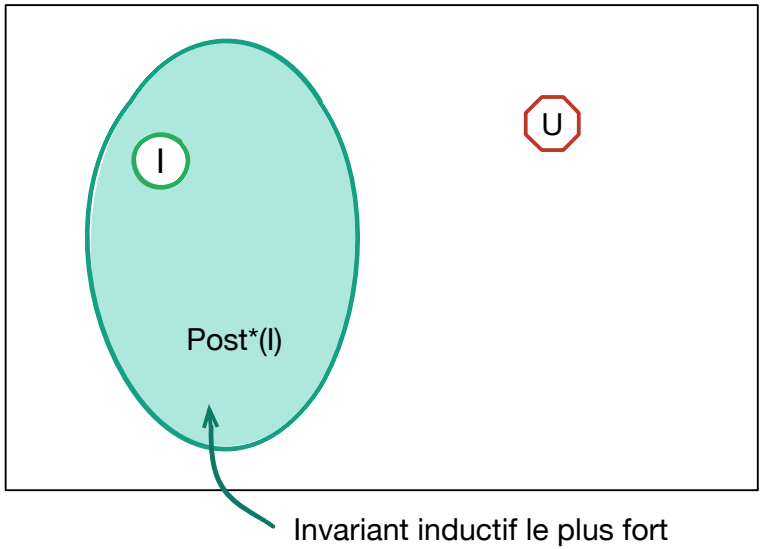
Certificats

Un bon certificat

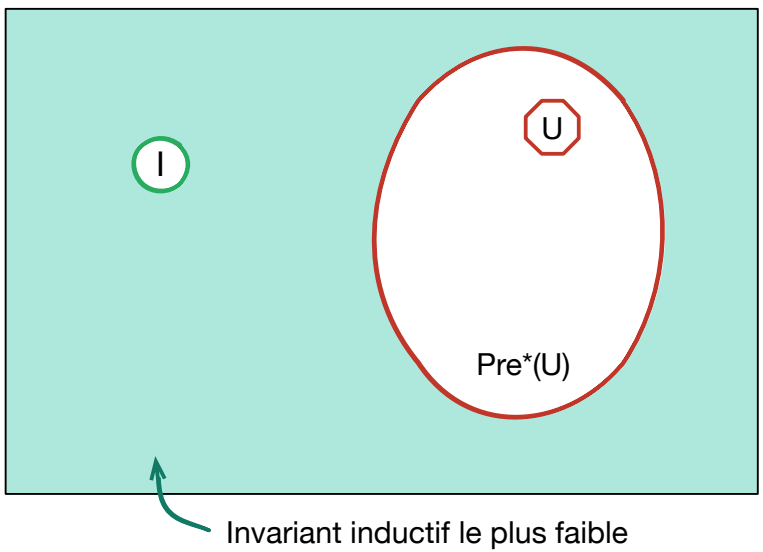
- contient une « preuve » du résultat annoncé
- doit être vérifiable indépendamment
- doit être petit (on doit vérifier le certificat à chaque exécution)

Invariants inductifs comme certificats

Ensemble des états **atteignables** à partir de I
= invariant inductif le plus fort



Ensemble des états **ne pouvant pas atteindre** U
= invariant inductif le plus faible



Invariants inductifs comme certificats

Vérification d'un invariant inductif ϕ comme preuve que U n'est pas atteignable dans le système $\mathcal{S} = (Q, \mathsf{I}, \tau)$:

- Initialisation : $\mathsf{I} \models \phi$
- Préservation : $\phi \wedge \tau \models \phi'$
- Propriété : $\phi \models \neg \mathsf{U}$

Invariants inductifs comme certificats

Vérification d'un invariant inductif ϕ comme preuve que U n'est pas atteignable dans le système $\mathcal{S} = (Q, \mathsf{I}, \tau)$:

- Initialisation : $\mathsf{I} \models \phi$
- Préservation : $\phi \wedge \tau \models \phi'$
- Propriété : $\phi \models \neg \mathsf{U}$

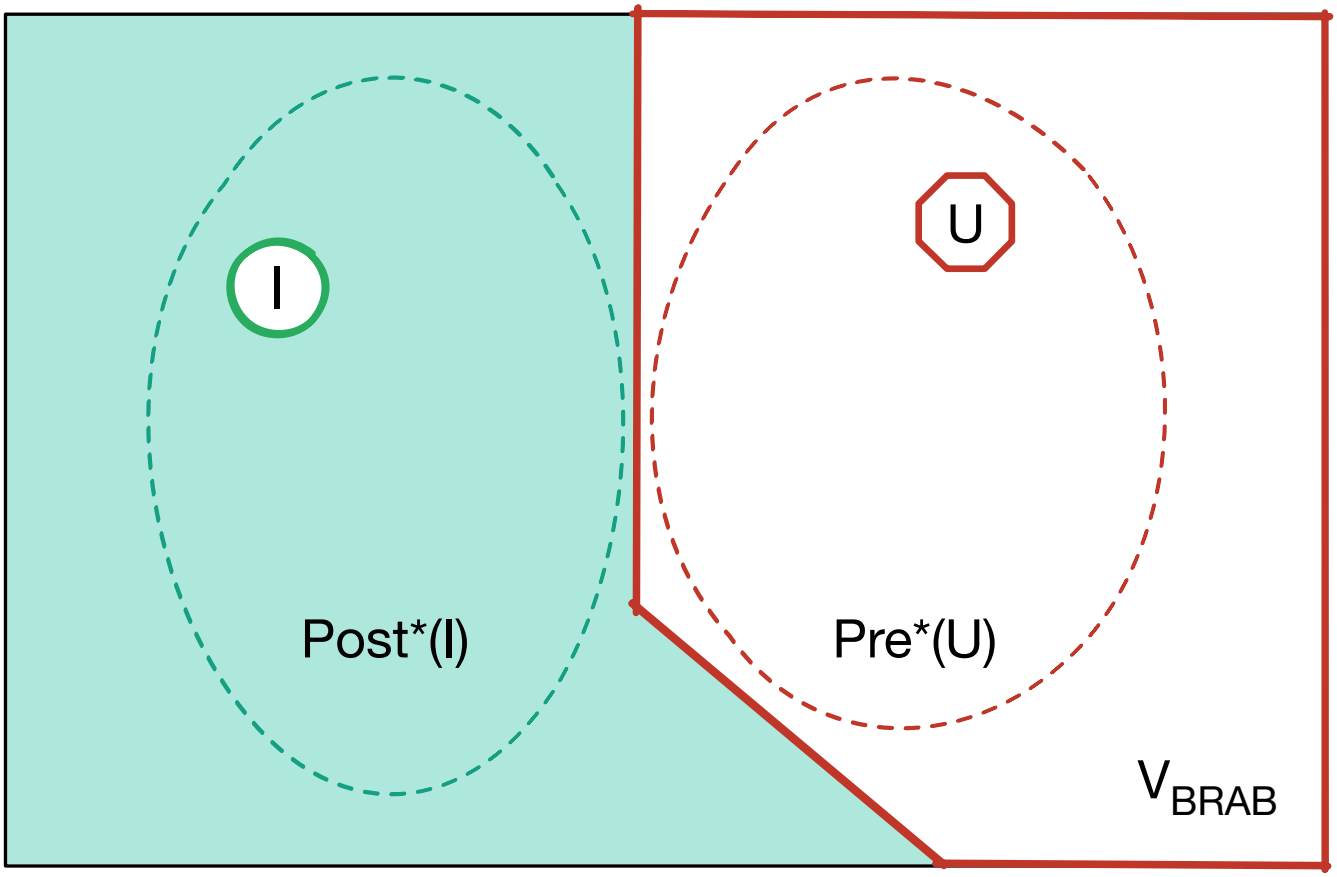
$$\phi \equiv \neg \mathsf{Pre}^*(\mathsf{U}) \equiv \neg \mathsf{V}$$

Invariants inductifs comme certificats : exemple

Protocole de cohérence de cache German-*esque* :

- 17 clauses quantifiées
- 18 ko en Why3
- Vérifié entièrement par 3 prouveurs automatiques de manière indépendante (CVC3, E, Z3)
- Vérifié en 0,44s

Invariants inductifs et BRAB



$$\phi \equiv \neg V_{BRAB}$$

Exemple de certificat avec BRAB

Protocole de cohérence de cache German-*esque* :

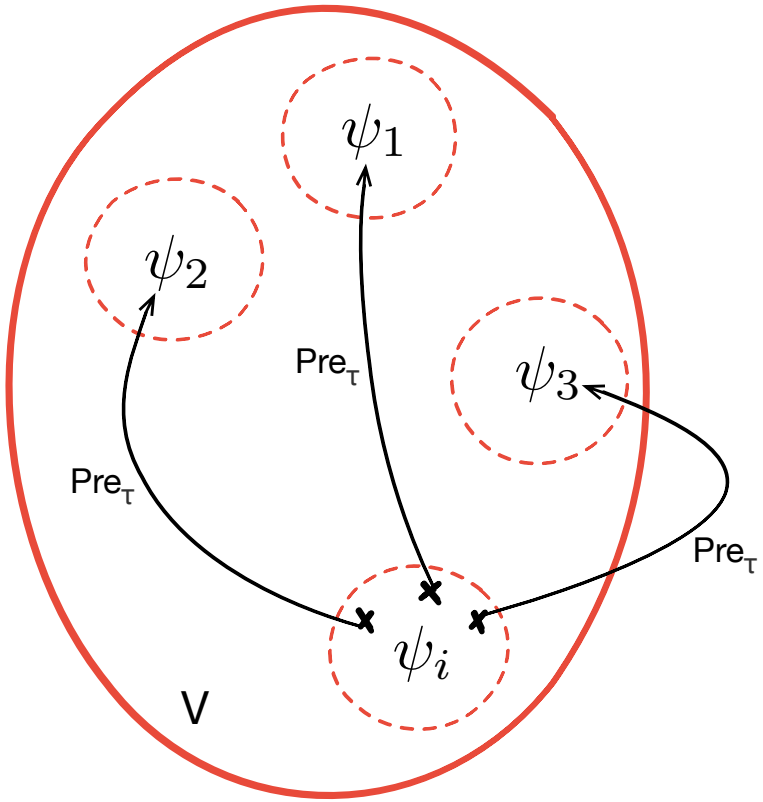
- 4 clauses quantifiées (contre 17)
- 5 ko en Why3 (contre 18 ko)
- Vérifié entièrement par 6 prouveurs automatiques de manière indépendante (Alt-Ergo, CVC3, CVC4, E, Spass, Z3)
- Vérifié en 0,16s (contre 0,44s)

Quelques certificats avec BRAB

Benchmark	**$\forall$-clauses**	**Taille**	**Vérifié**	**Niveau**	**Temps**
Szymanski_at	31	18 ko	Oui	3	0,96s
Szymanski_na	38	28 ko	Oui	2	1,45s
Ricart_Agrawala	30	39 ko	Oui	2	1,26s
German_Baukus	48	44 ko	Oui	2	1,58s
German.CTC	69	83 ko	Oui	2	2,73s
German_pfs	51	50 ko	Oui	3	1,79s
Flash_nodata	41	123 ko	Oui	2	2,99s
Flash	742	1,7 Mo	1462/1475	0	35m

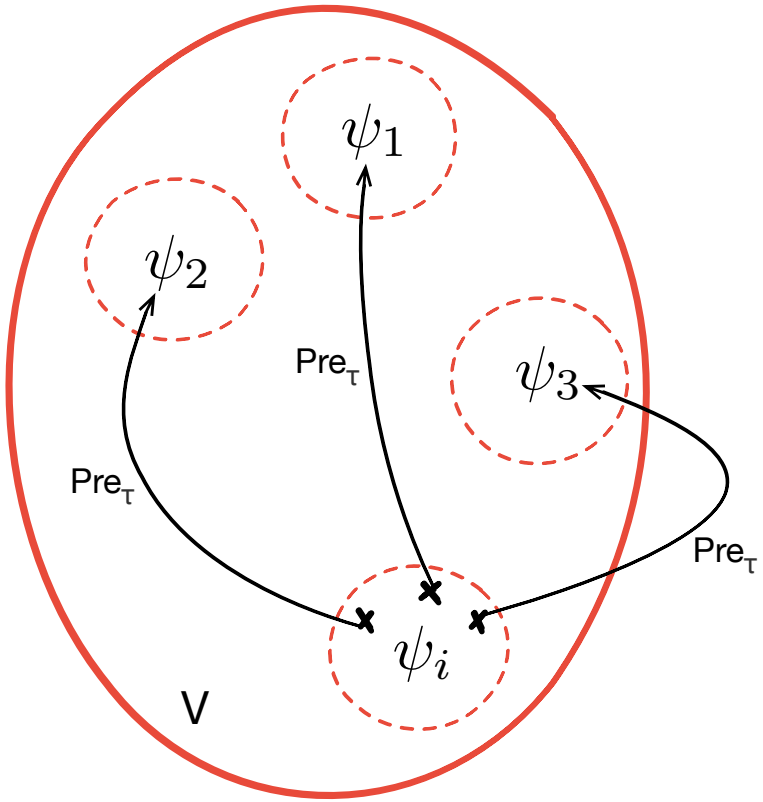
Lemmes intermédiaires

- utiliser les informations calculées par Cubicle lors de l'analyse d'atteignabilité
- calculer les lemmes
  - à la volée
  - après construction de V



Lemmes intermédiaires

- utiliser les informations calculées par Cubicle lors de l'analyse d'atteignabilité
- calculer les lemmes
  - à la volée
  - après construction de V



Quelques certificats avec BRAB

Benchmark	P.O.	Taille	Vérifié	Niveau	Temps
Szymanski_at	34	21 ko	Oui	3	0,66s
Szymanski_na	41	30 ko	Oui	2	1,79s
Ricart_Agrawala	19	36 ko	Oui	2	0,52s
German_Baukus	51	40 ko	Oui	**3**	1,16s
German.CTC	72	62 ko	Oui	**4**	1,98s
German_pfs	51	43 ko	Oui	3	1,42s
Flash_nodata	44	133 ko	Oui	**3**	2,68s
Flash	736	1,1 Mo	**Oui**	**1**	4m7s

Quelques certificats avec BRAB

Benchmark	P.O.	Taille	Vérifié	Niveau	Temps
Szymanski_at	34	21 ko	Oui	3	0,66s
Szymanski_na	41	30 ko	Oui	2	1,79s
Ricart_Agrawala	19	36 ko	Oui	2	0,52s
German_Baukus	51	40 ko	Oui	**3**	1,16s
German.CTC	72	62 ko	Oui	**4**	1,98s
German_pfs	51	43 ko	Oui	3	1,42s
Flash_nodata	44	133 ko	Oui	**3**	2,68s
Flash	736	1,1 Mo	**Oui**	**1**	4m7s

Flash : 718/736 prouvés par 2+ solveurs

Conclusion

- Model checker pour systèmes paramétrés Cubicle
- BRAB : inférence d'invariants
- Première vérification automatique de sûreté de FLASH
- Certification de Cubicle avec Why3

Perspectives

BRAB :

- Oracles :
  - profiter d'outils comme Murφ ou Spin
  - recherche de bugs
- Améliorations backtracking
- Utilisation des idées sous-jacentes à BRAB dans d'autres cadres (IC3)

Certification :

- Simplification des certificats
- Réécriture de Cubicle en Why3