UNIVERSITÉ PARIS SUD

Comprendre le monde, construire l'avenir®

MODEL CHECKING PARAMÉTRÉ : INVARIANTS ET CERTIFICATION

GDR-GPL LTP 2013, 18 Novembre 2013

Alain Mebsout
En collaboration avec Sylvain Conchon, Amit Goel, Sava Kristić, et Fatiha Zaïdi

LRI, Université Paris-Sud
Strategic CAD Labs, Intel Corporation

Défi

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

Défi

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

- automatiquement

Le protocole FLASH

Architecture multiprocesseur Stanford FLASH (1994)

- Mémoire partagée avec cohérence de cache
- Passage de messages
- Taille industrielle : 67 million d'états pour 4 processus (28 000 états pour German)

Le protocole FLASH

Architecture multiprocesseur Stanford FLASH (1994)

- Mémoire partagée avec cohérence de cache
- Passage de messages
- Taille industrielle : 67 million d'états pour 4 processus (28 000 états pour German)

Qui a prouvé le protocole ?

- Park et Dill, 1996, preuve PVS
- Das, Dill et Park, 1999, predicate abstraction
- McMillan, 2001, compositional model checking
- Chou, Mannava, Park, 2004, CMP méthode inspirée des travaux de McMillan
- Talapur et Tuttle, 2008, Extension flots de messages de CMP

Aucune de ces preuve n'est purement automatique

Solutions

- Model checking de systèmes paramétrés
- Fragment décidable
- Cubicle implémente une atteignabilité par chaînage arrière (backward reachability)

Solutions

- Model checking de systèmes paramétrés
- Fragment décidable
- Cubicle implémente une atteignabilité par chaînage arrière (backward reachability)

Est-ce que ça marche ?

Quelques benchmarks

	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)

Quelques benchmarks

	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

Comment passer à l'échelle ?

- Réduire l'espace d'état à explorer
- Invariants pour le cas paramétré
- Comportements intéressants souvent déjà observables sur des petites instances

Inférence d'invariants

Problème : Les invariants sont souvent plus difficiles à prouver que la propriété originelle

Inférence d'invariants

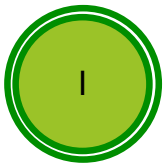
Problème : Les invariants sont souvent plus difficiles à prouver que la propriété originelle

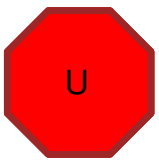
Idée : utiliser des instances finies pour inférer des invariants du cas paramétrés

- Insérer et vérifier à la volée dans la boucle d'atteignabilité arrière
- Retour en arrière (backtrack) si nécessaire

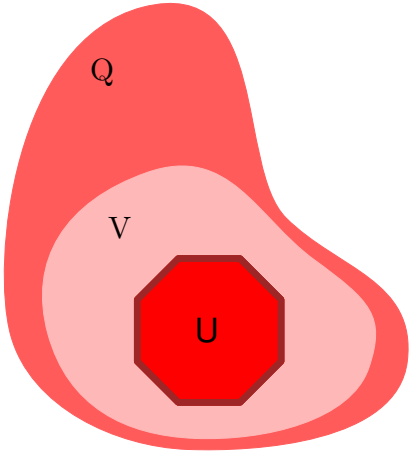
BRAB : **B**ackward **R**eachability with **A**pproximations and **B**acktracking

Backward Reachability

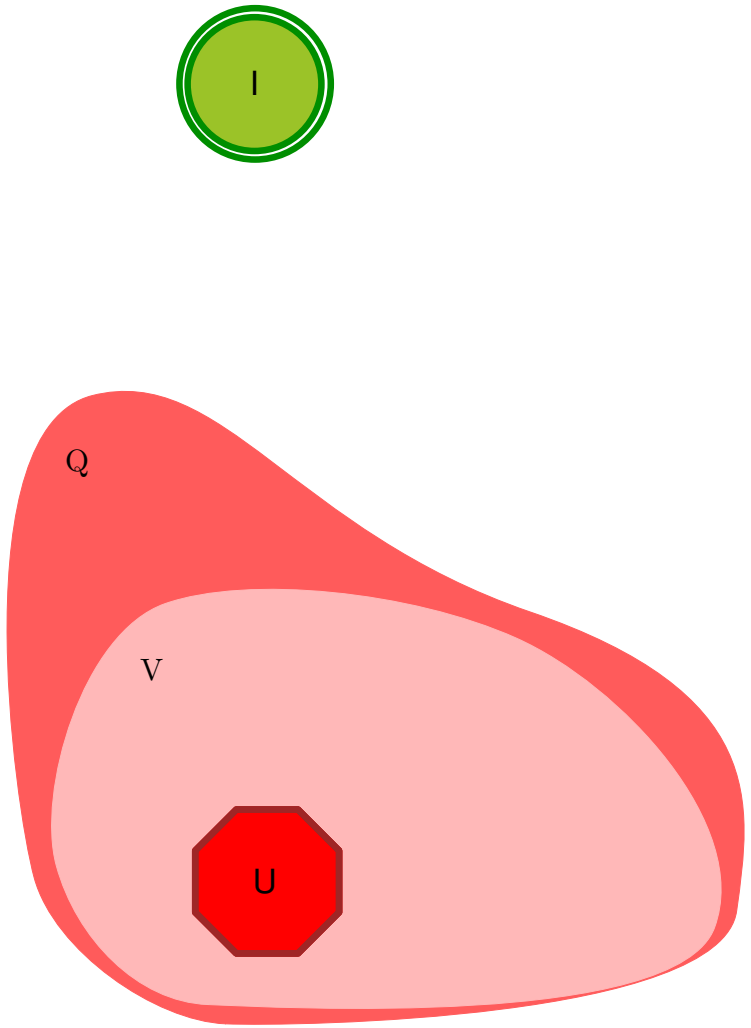




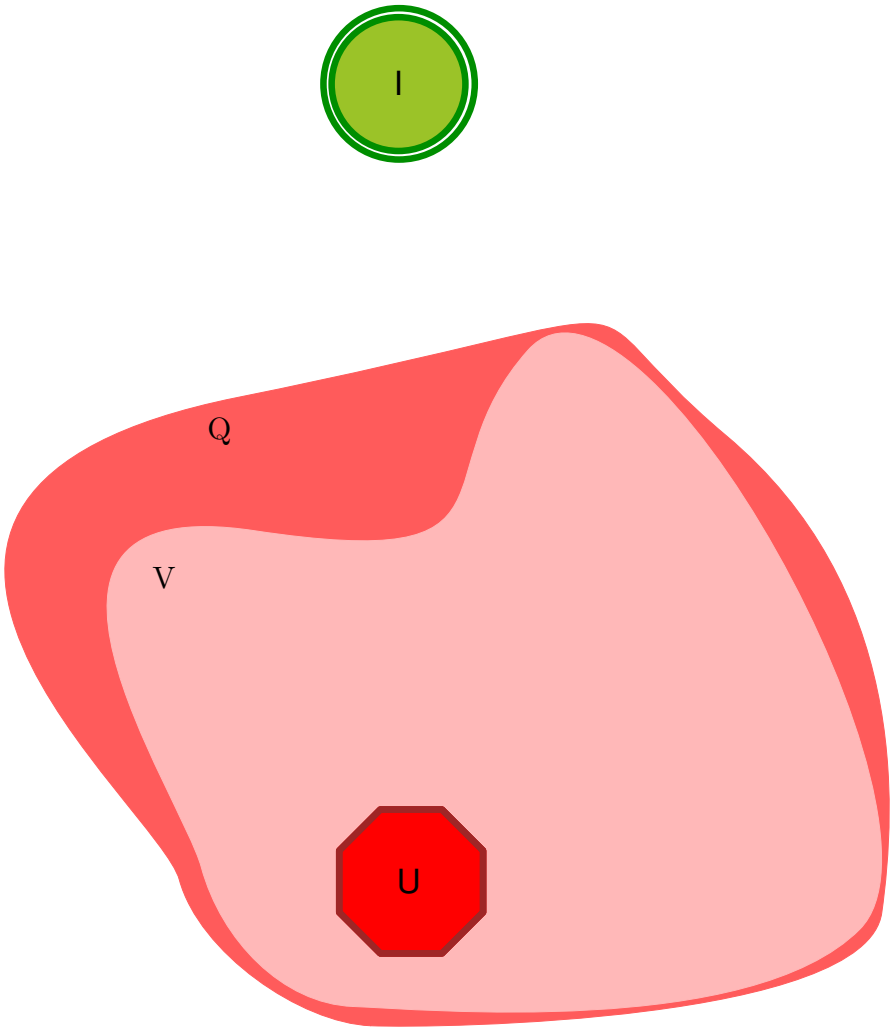
Backward Reachability



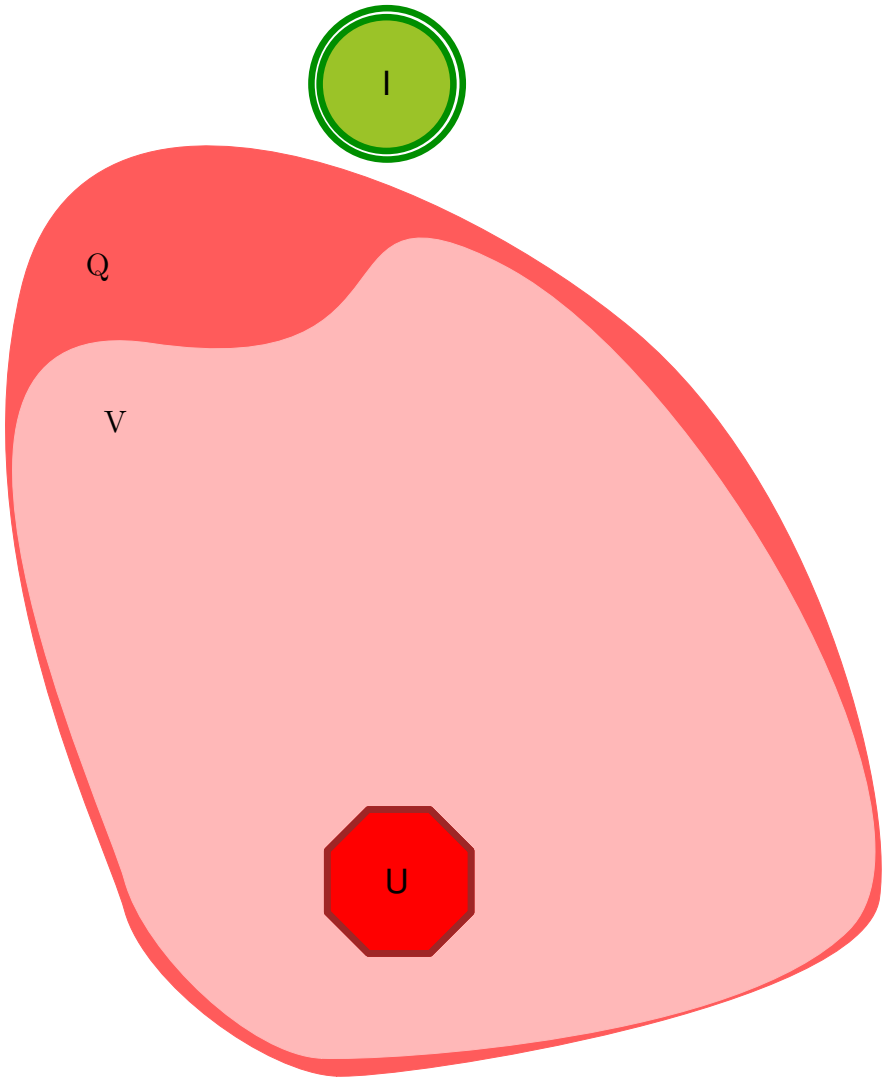
Backward Reachability



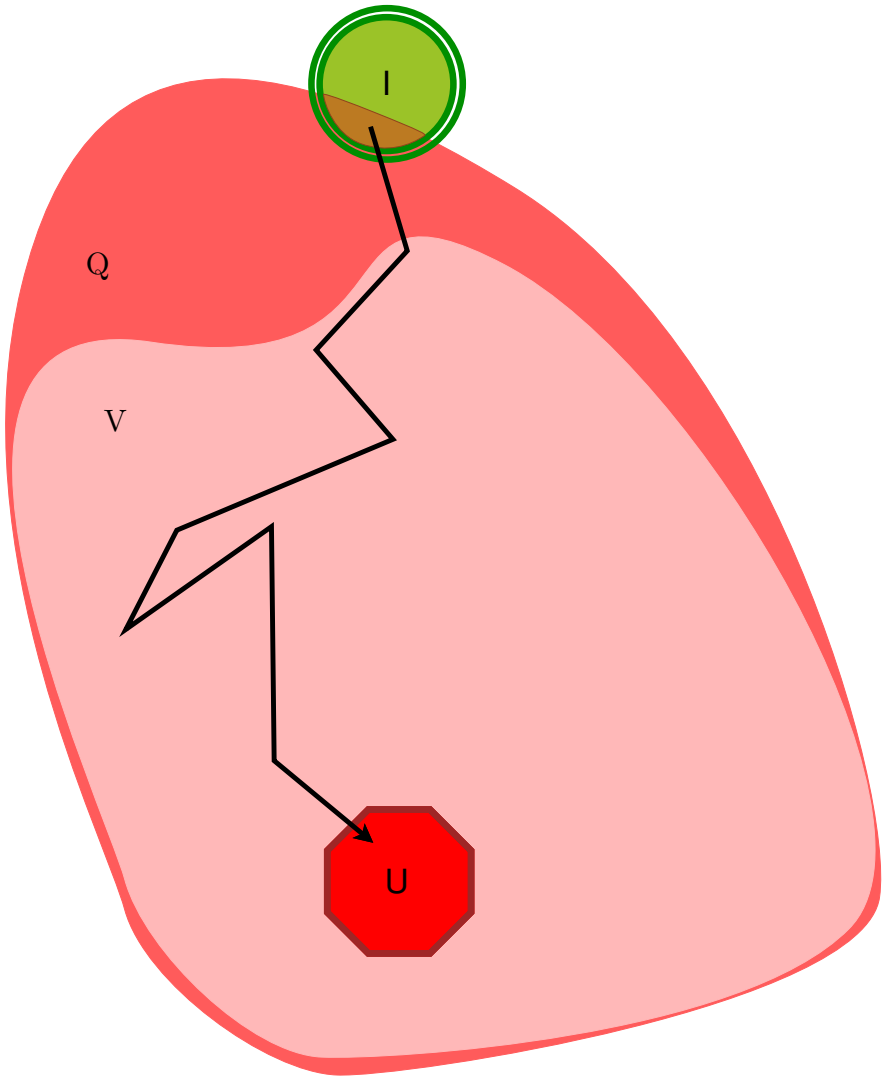
Backward Reachability



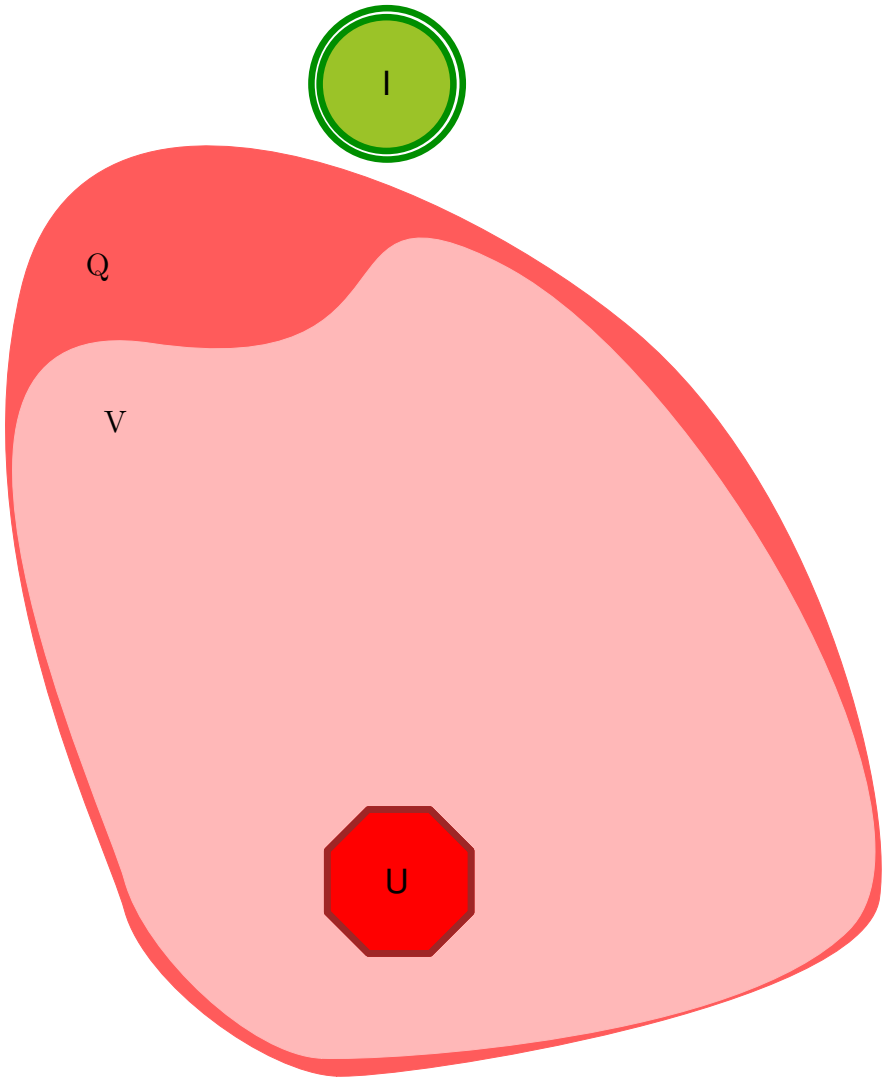
Backward Reachability



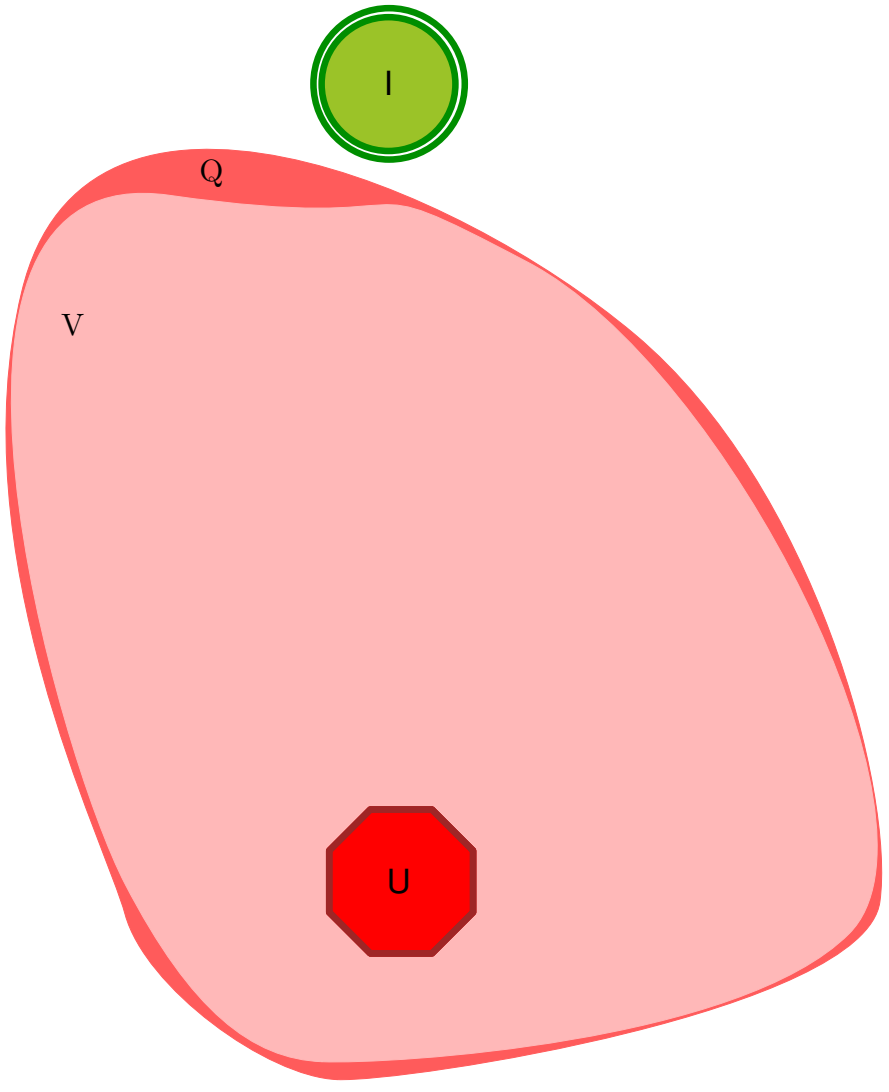
Backward Reachability



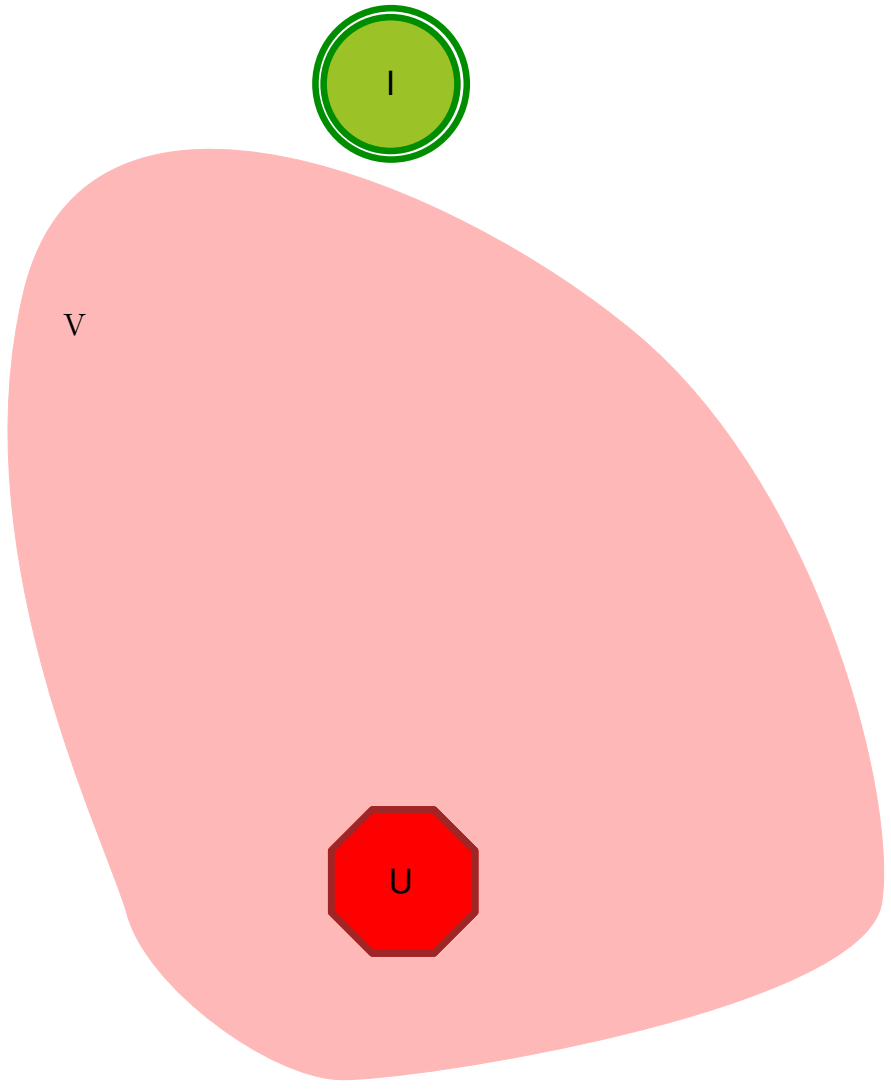
Backward Reachability



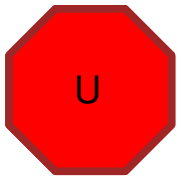
Backward Reachability



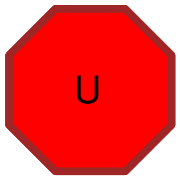
Backward Reachability



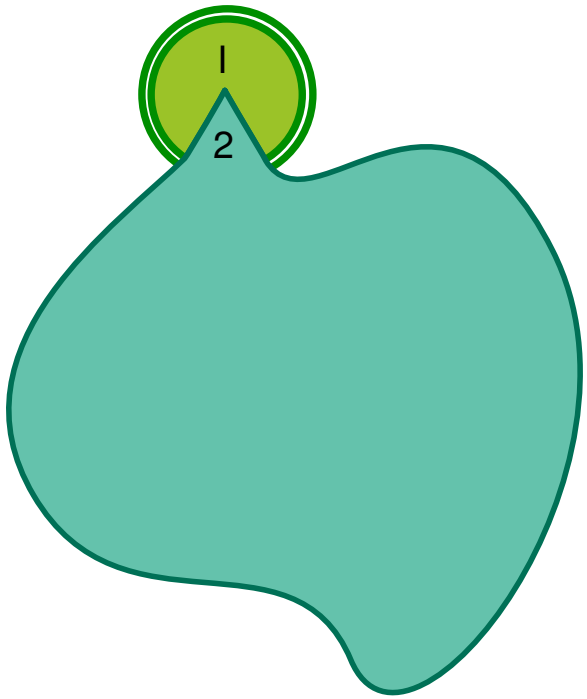
BRAB : intuition

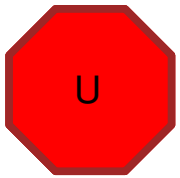


BRAB : intuition

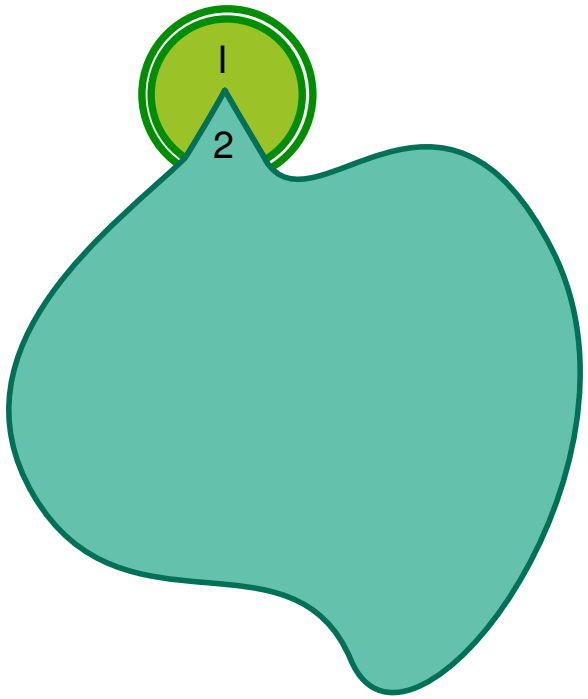


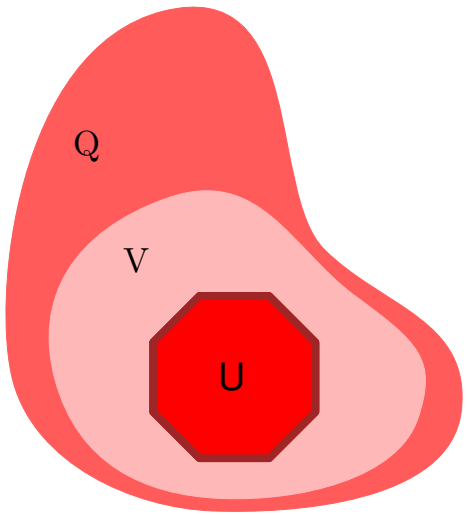
BRAB : intuition



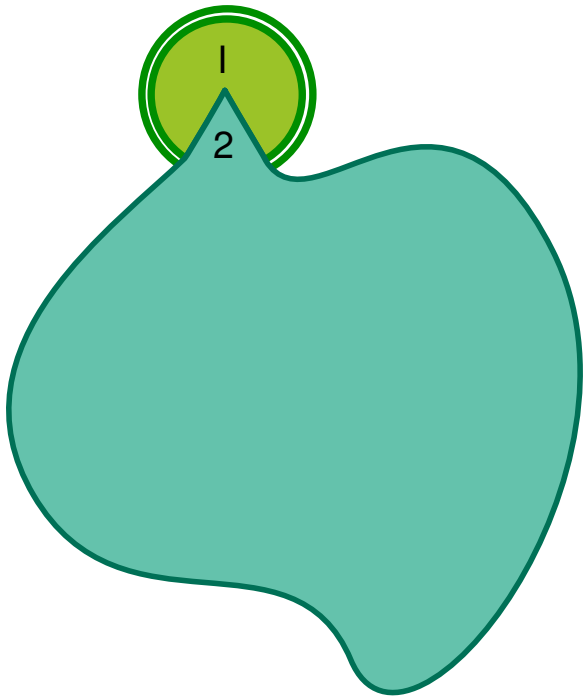


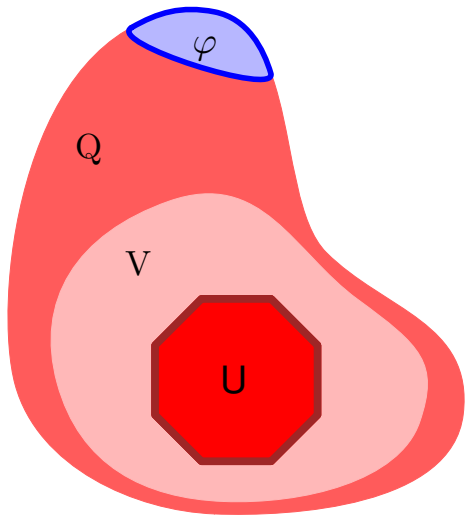
BRAB : intuition



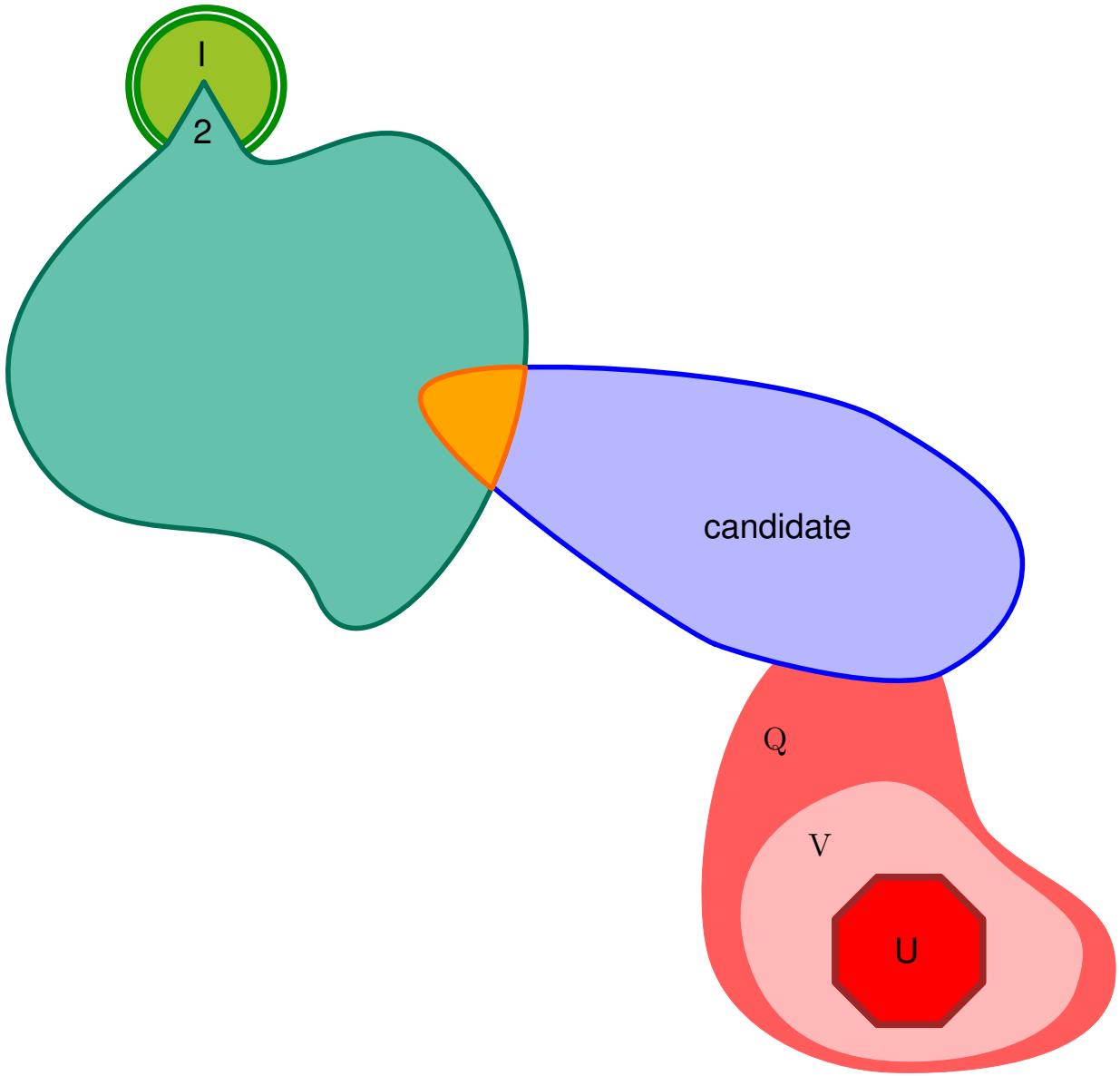


BRAB : intuition

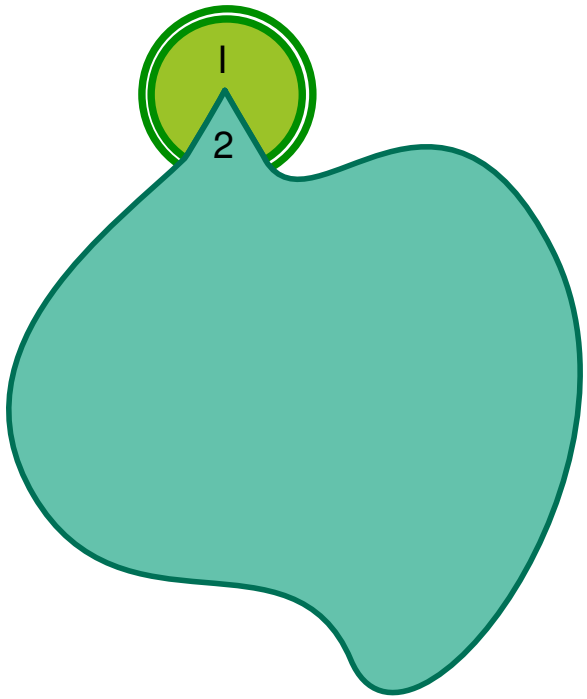


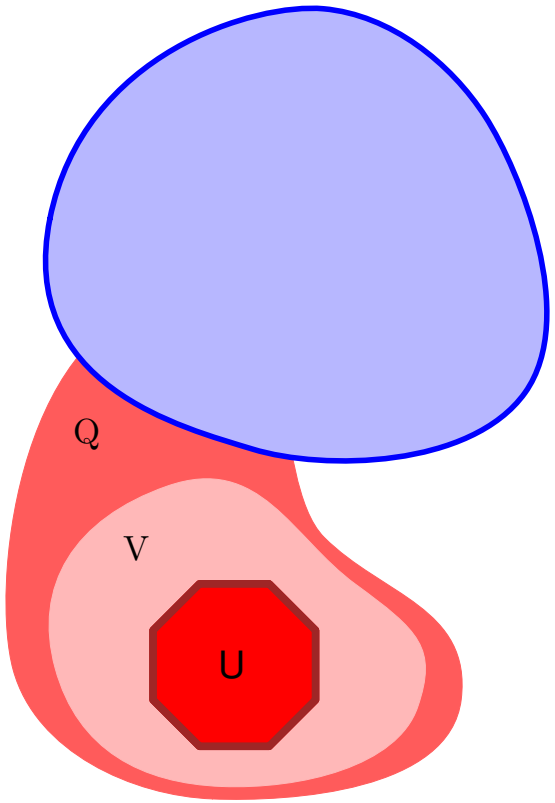


BRAB : intuition

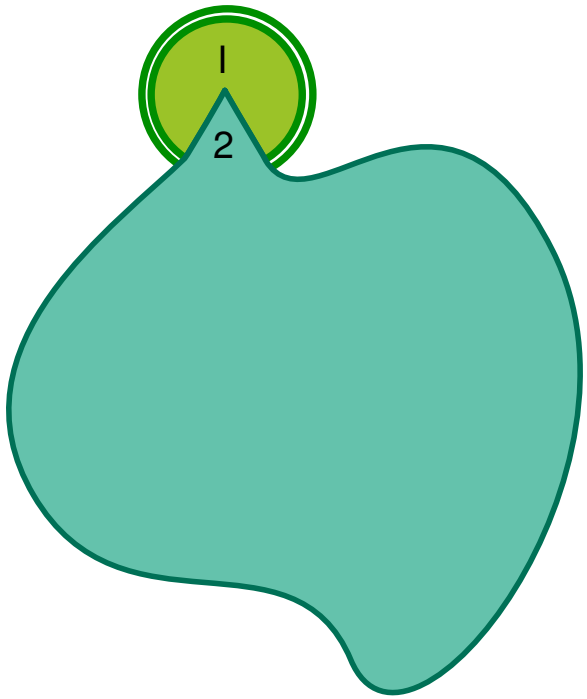


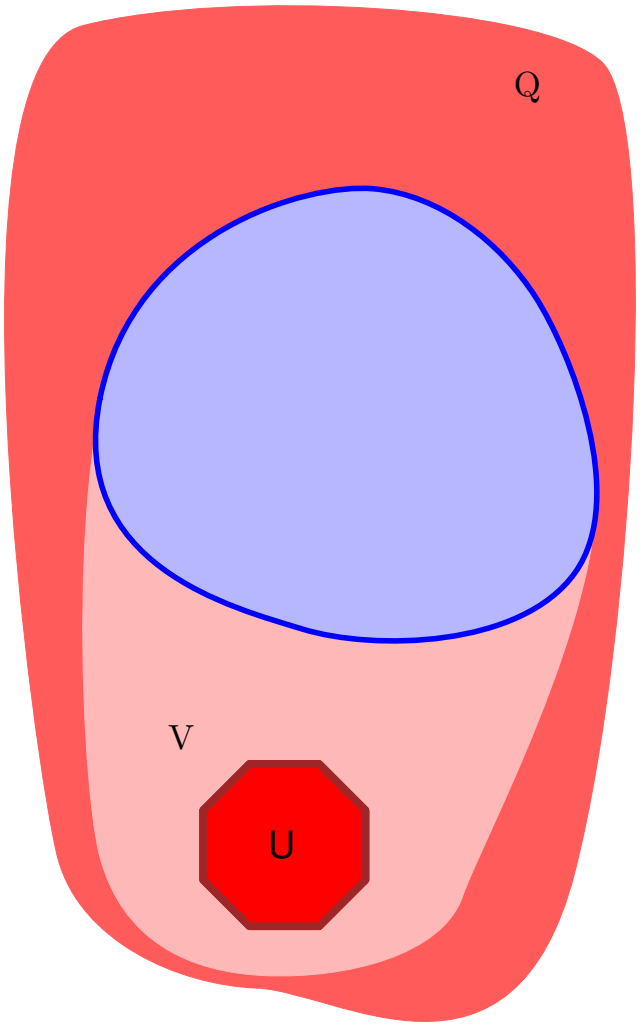
BRAB : intuition



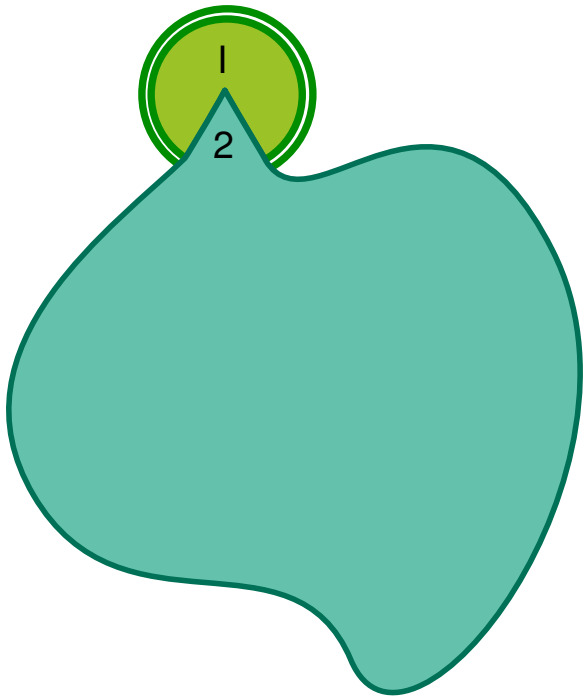


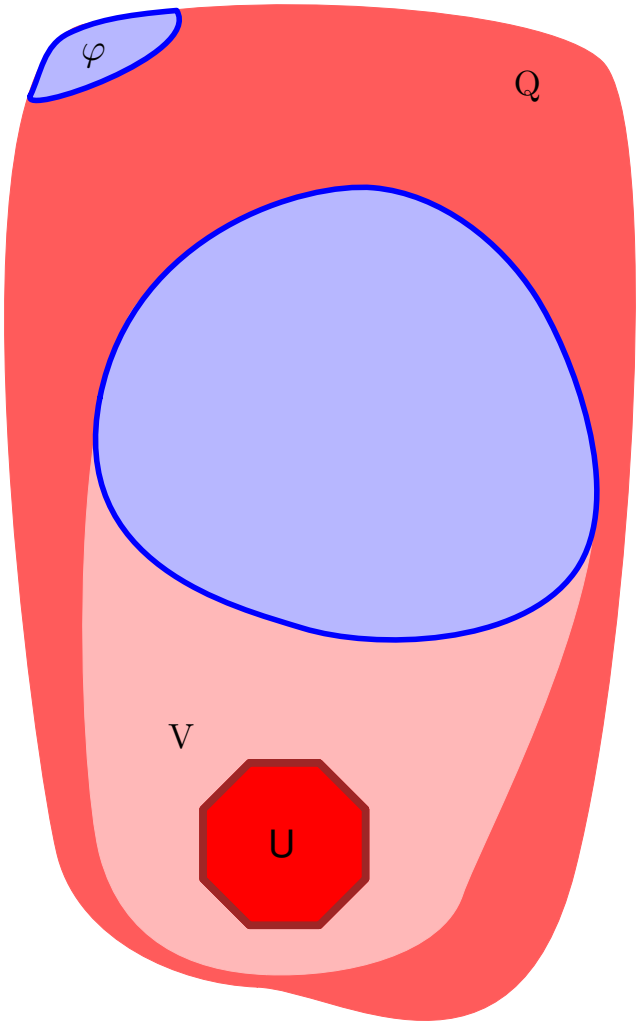
BRAB : intuition



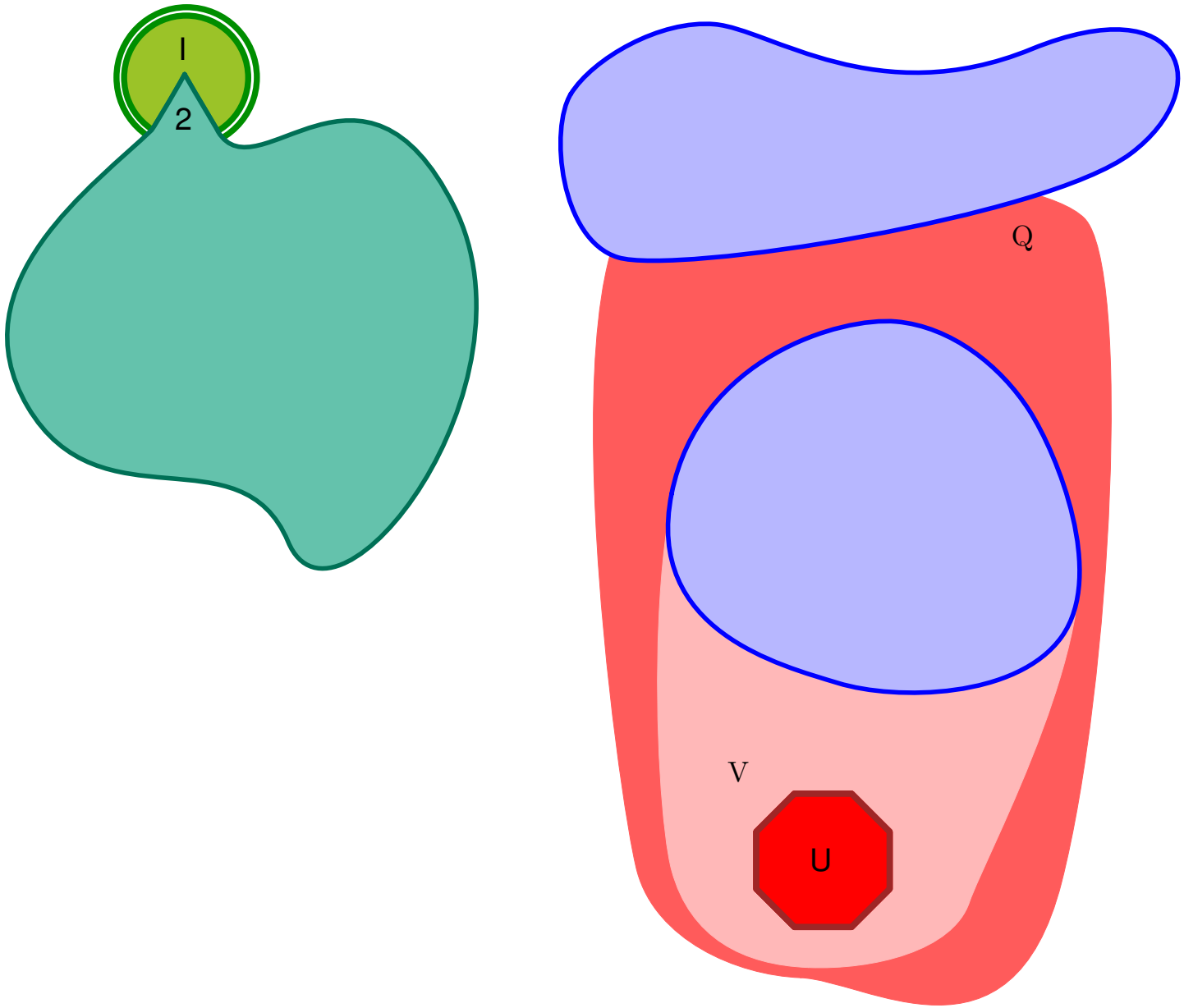


BRAB : intuition

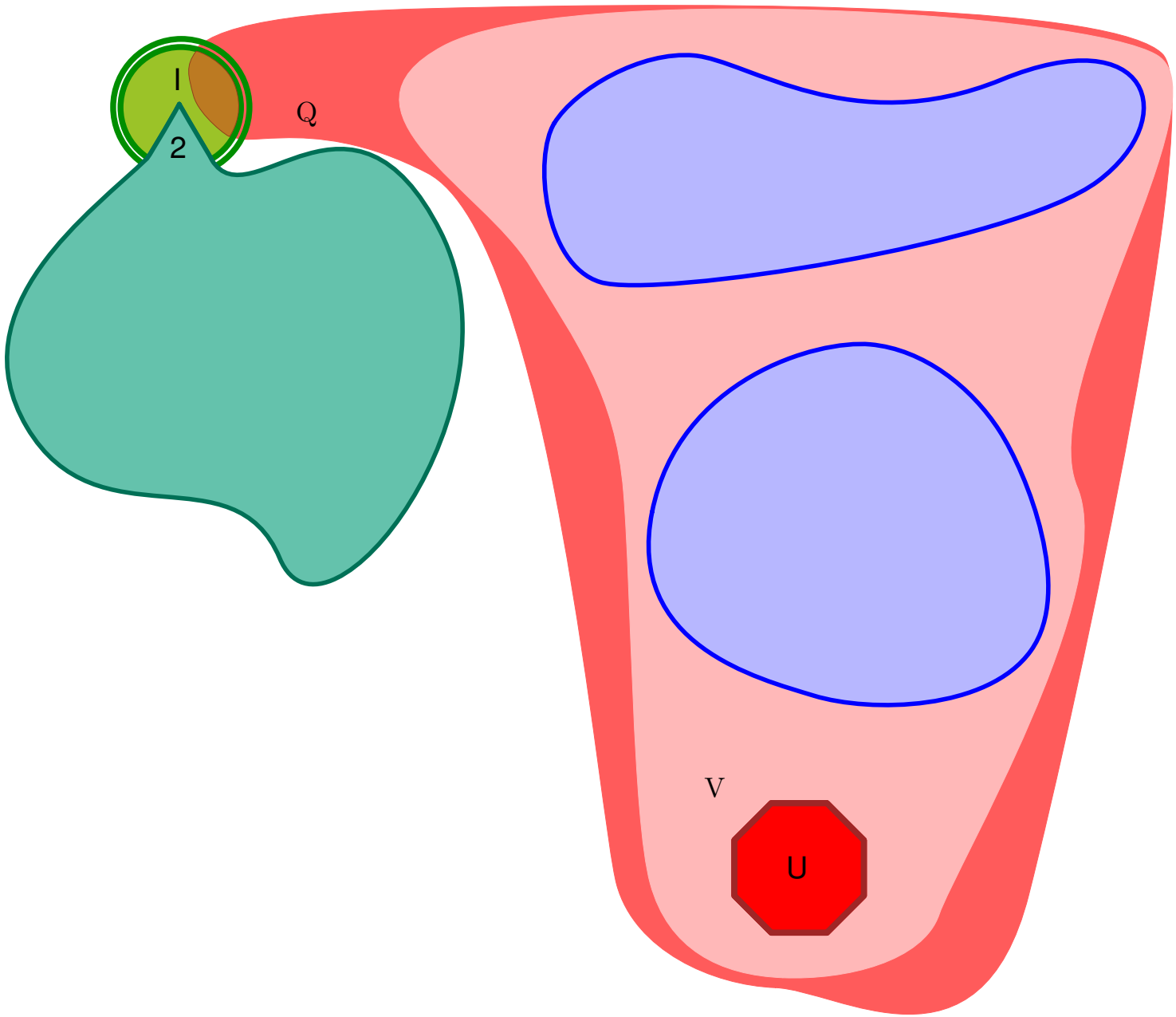




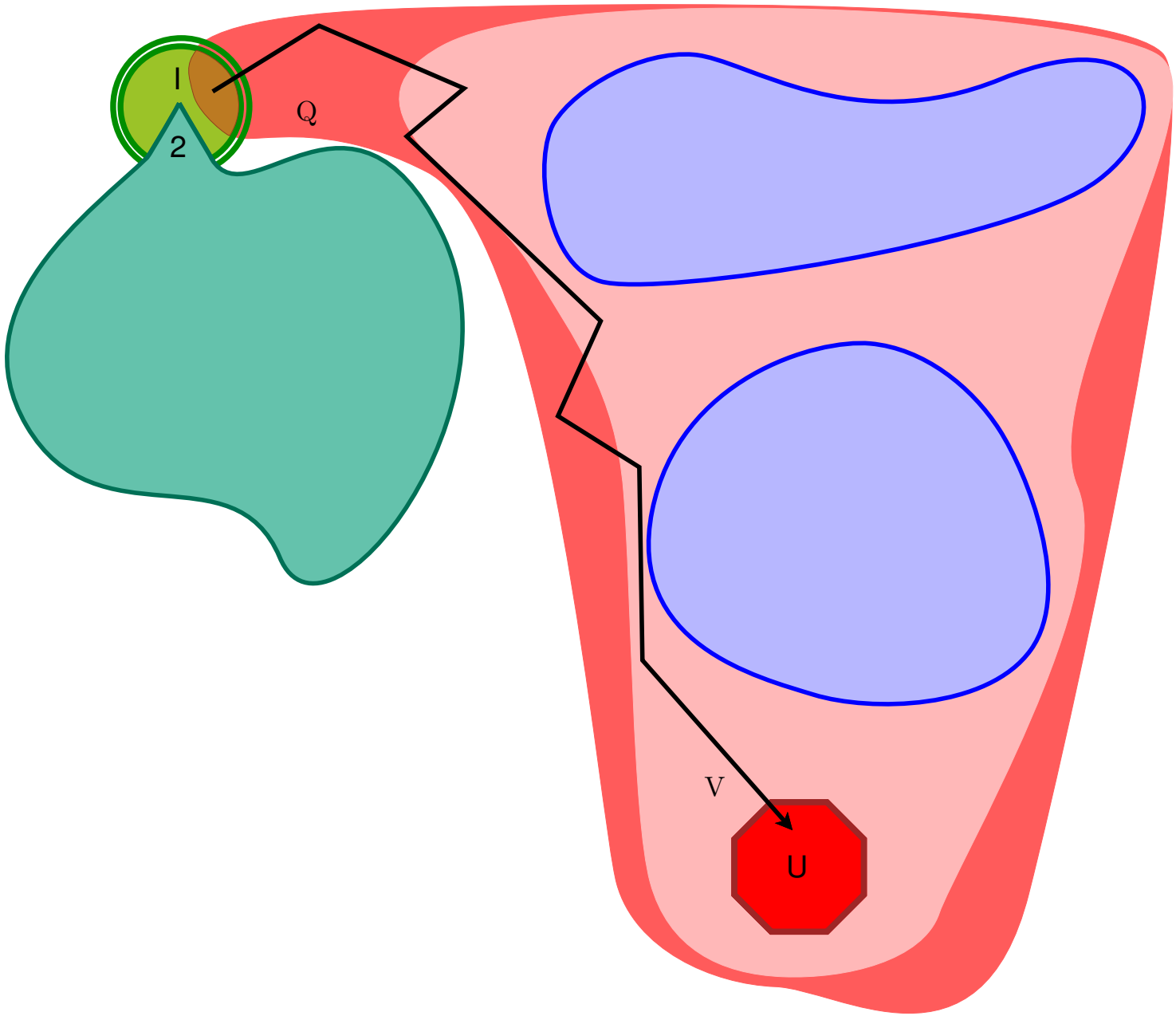
BRAB : intuition



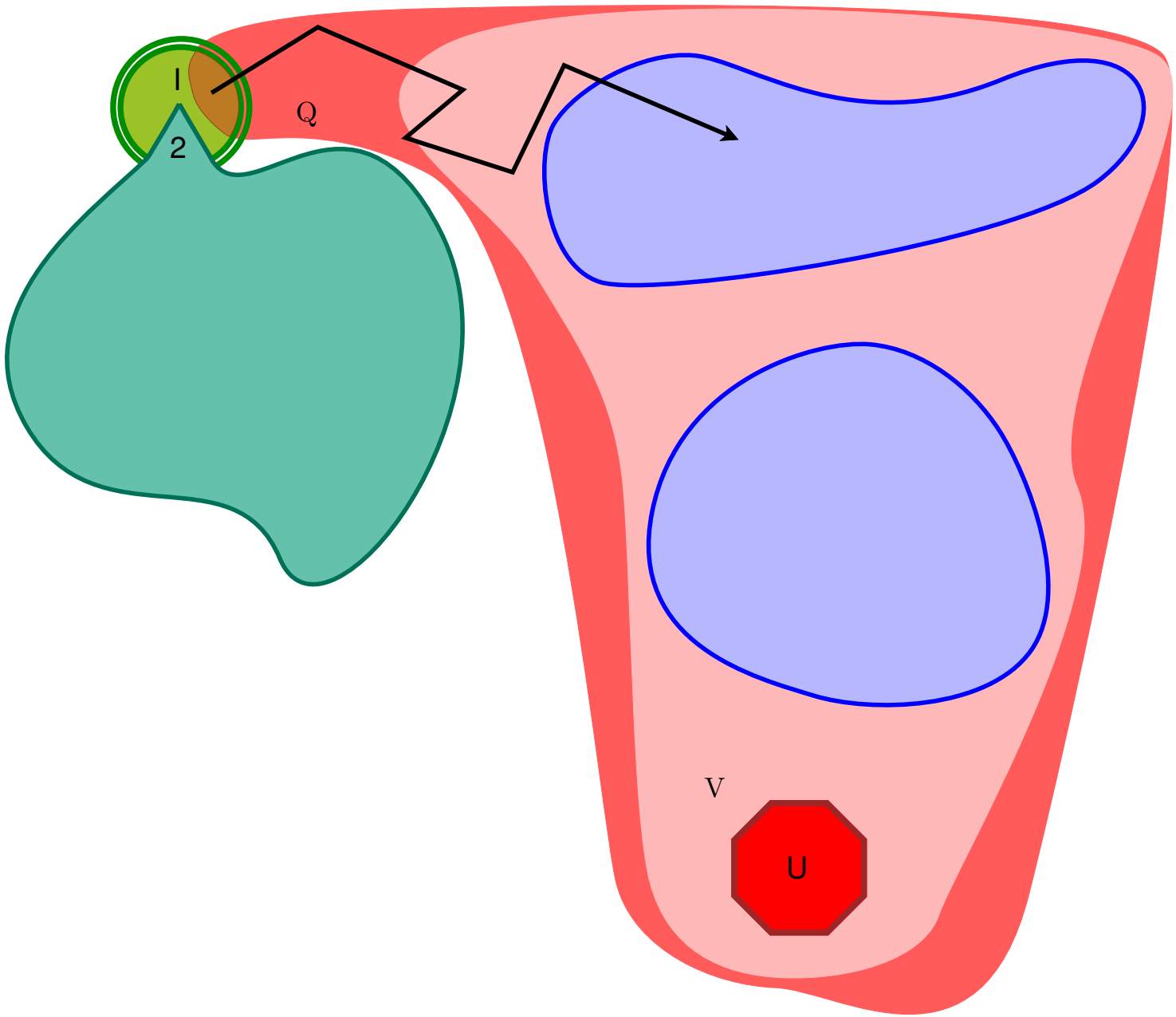
BRAB : intuition



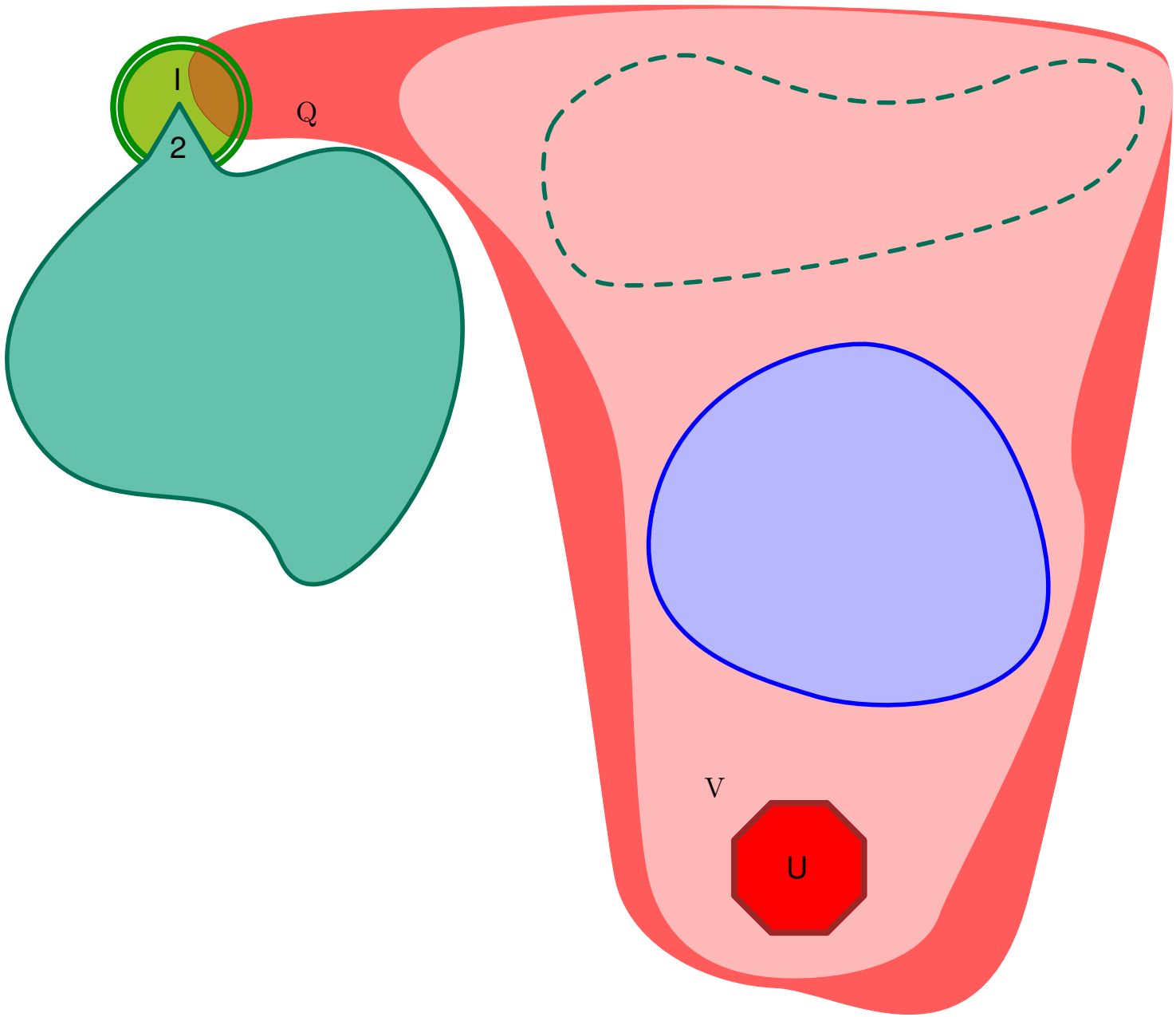
BRAB : intuition



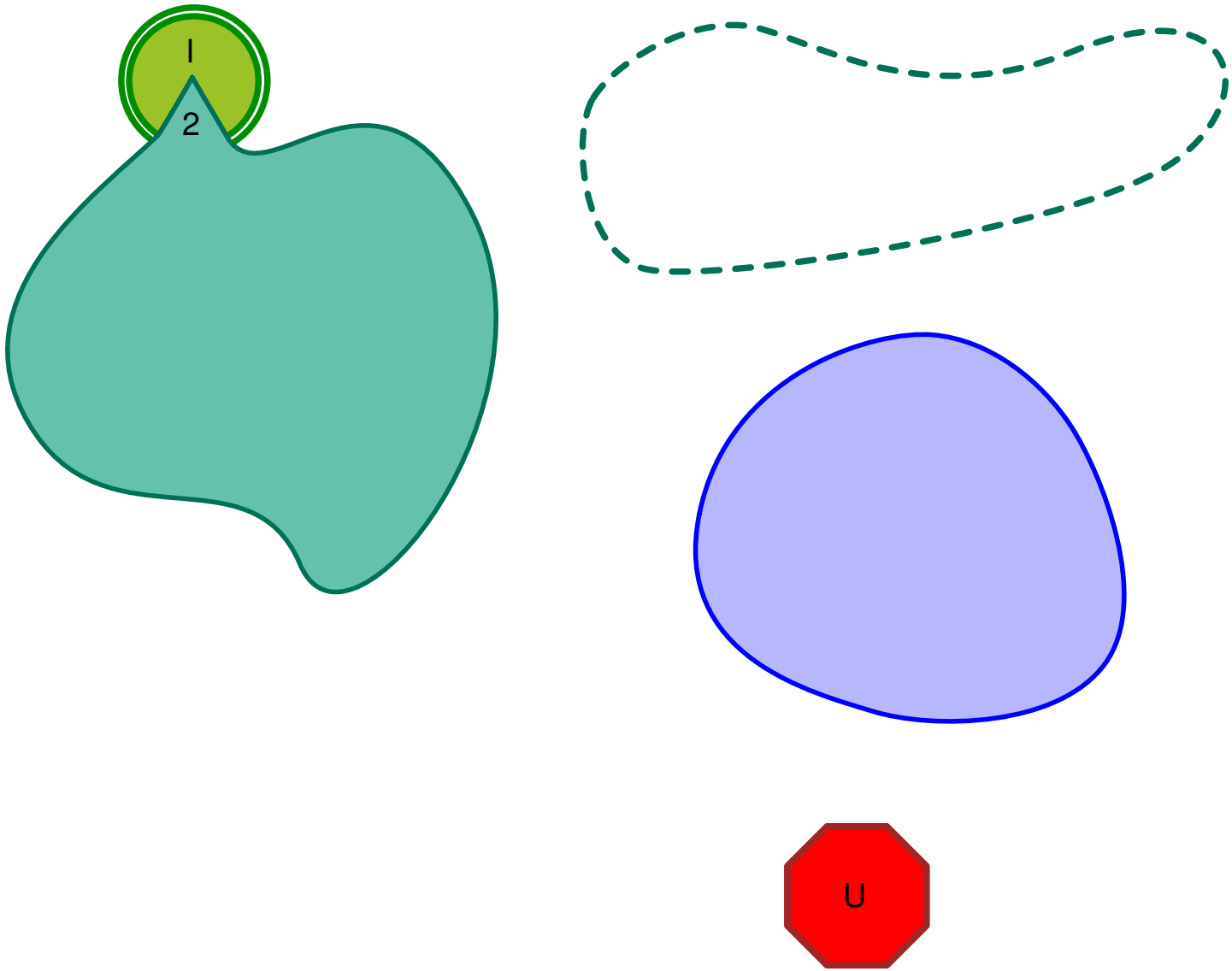
BRAB : intuition



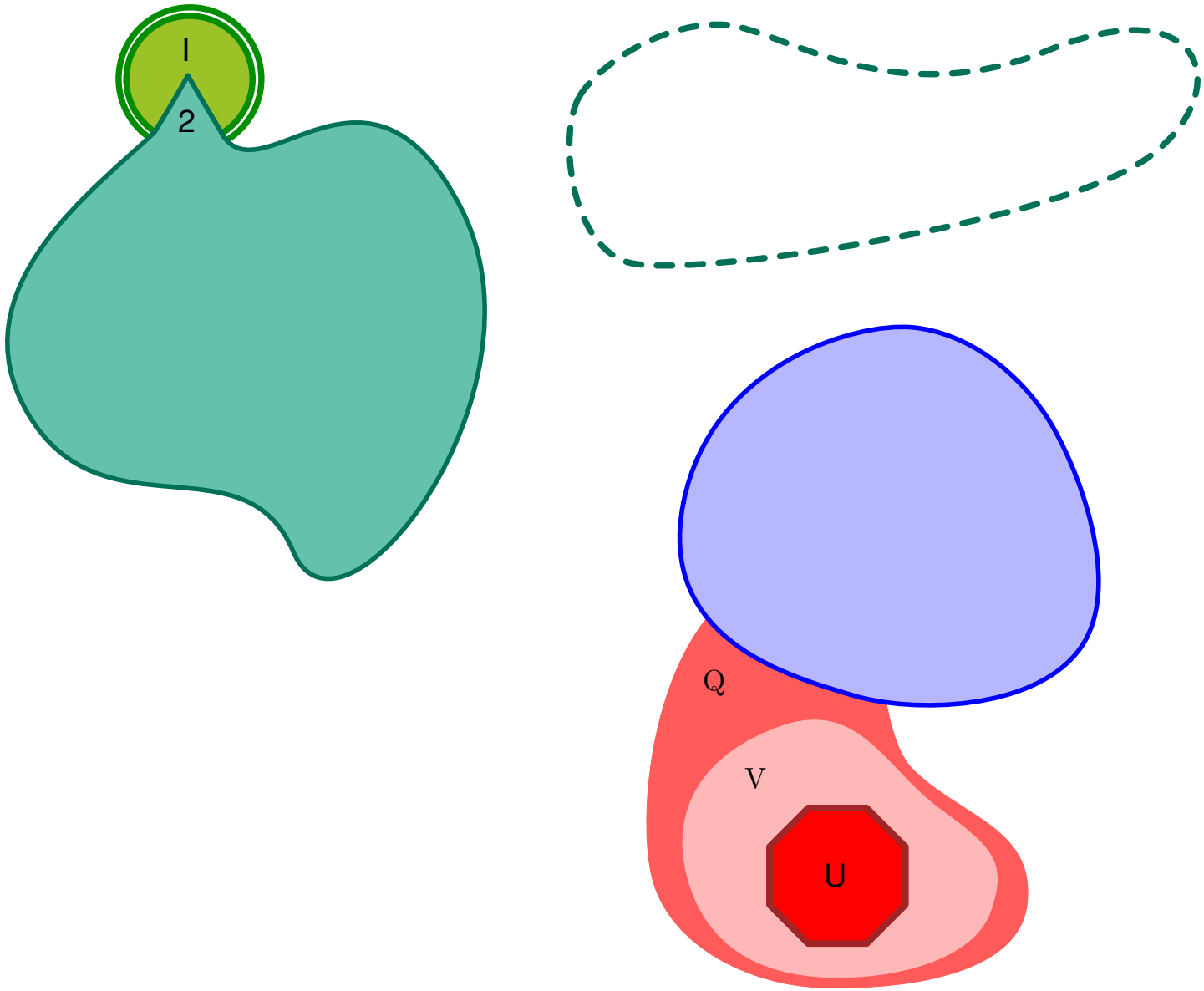
BRAB : intuition



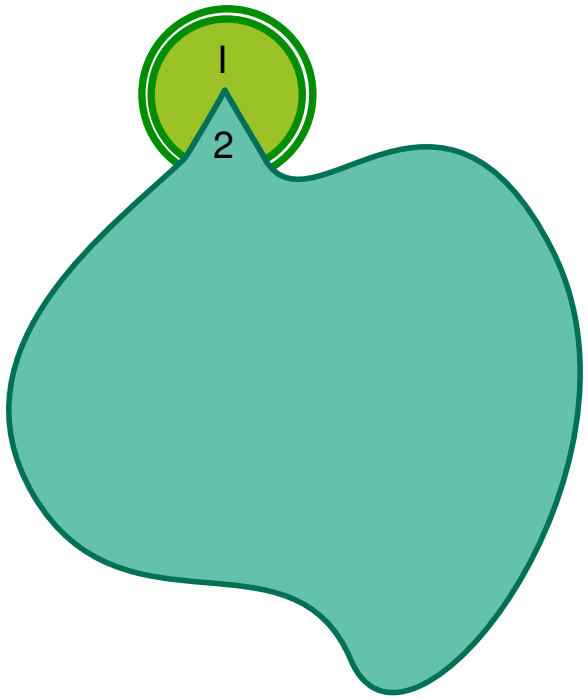
BRAB : intuition

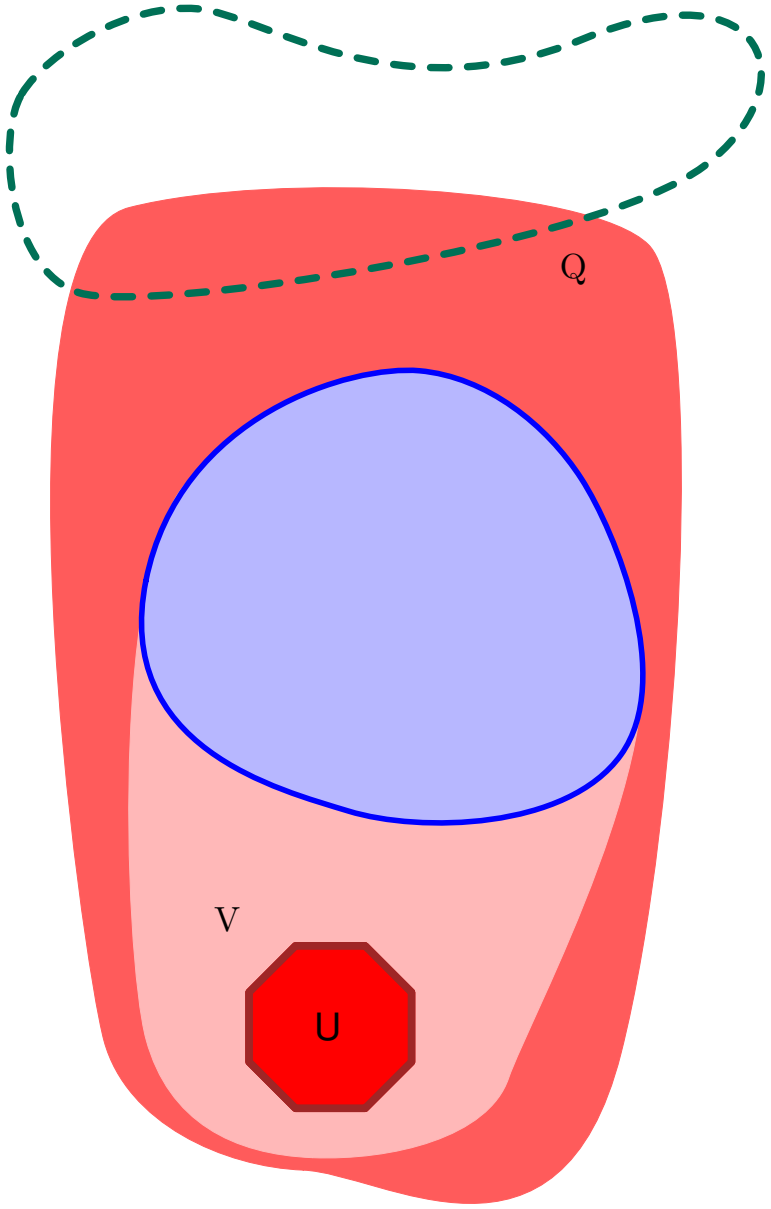


BRAB : intuition

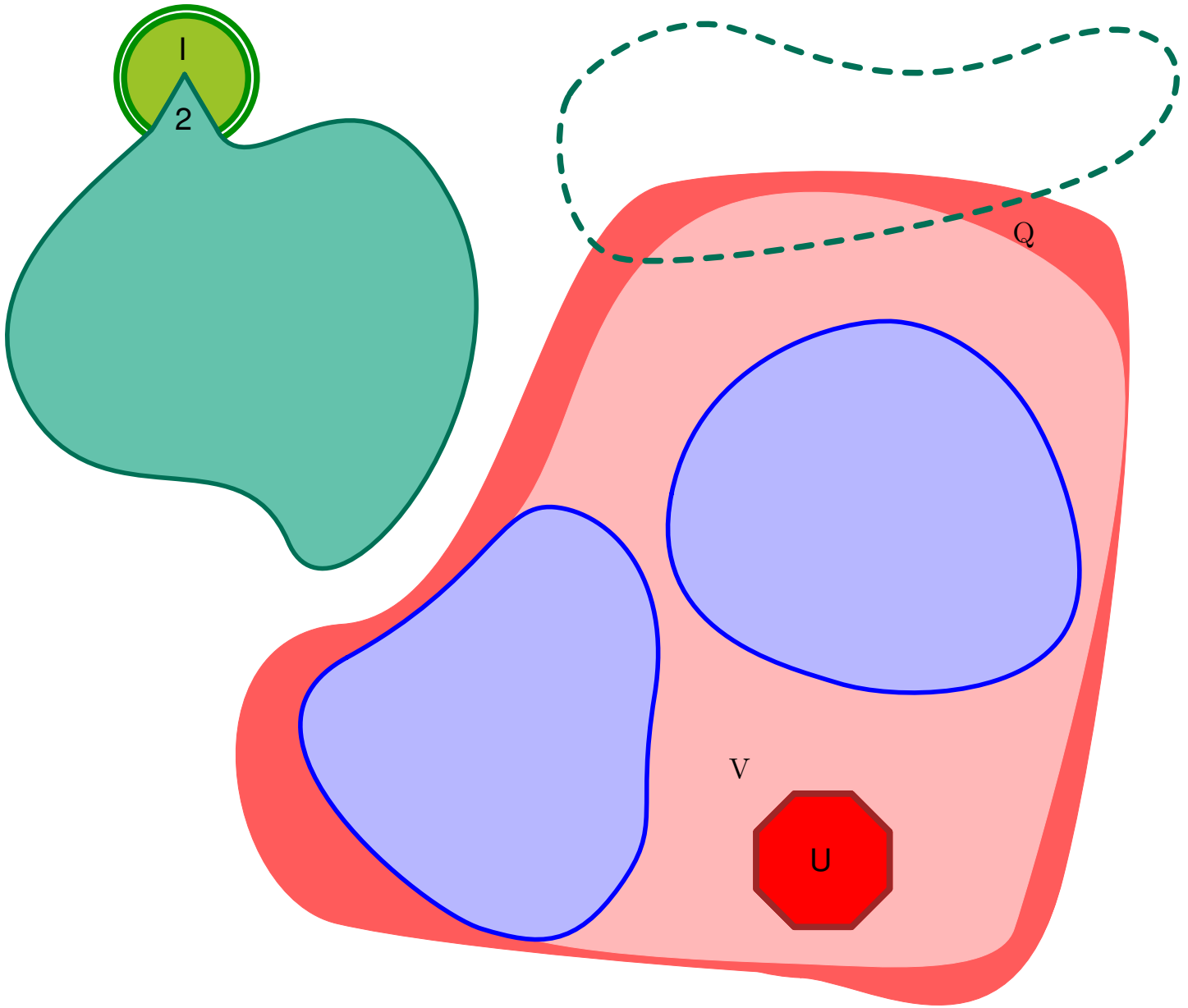


BRAB : intuition

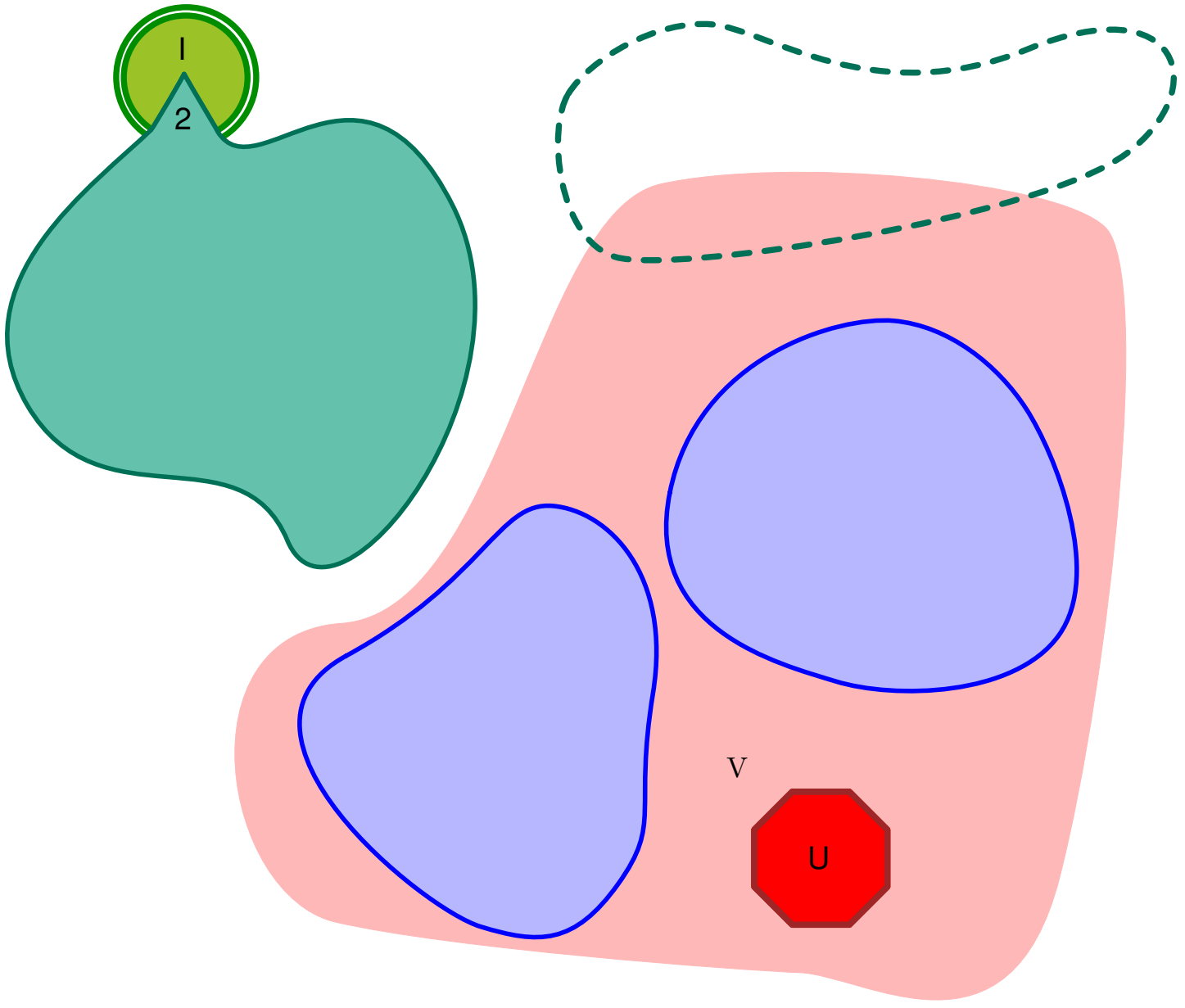




BRAB : intuition



BRAB : intuition



Framework

- Framework symbolique pour les systèmes paramétrés
- États : formules dans un fragment décidable de FOL
- Pre-image effectivement calculables
- Post-image effectivement calculables pour les instances finies

Framework

- Framework symbolique pour les systèmes paramétrés
- États : formules dans un fragment décidable de FOL
- Pre-image effectivement calculables
- Post-image effectivement calculables pour les instances finies

Dans Cubicle $\rightarrow$ systèmes de transitions à tableaux

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

Client i :

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

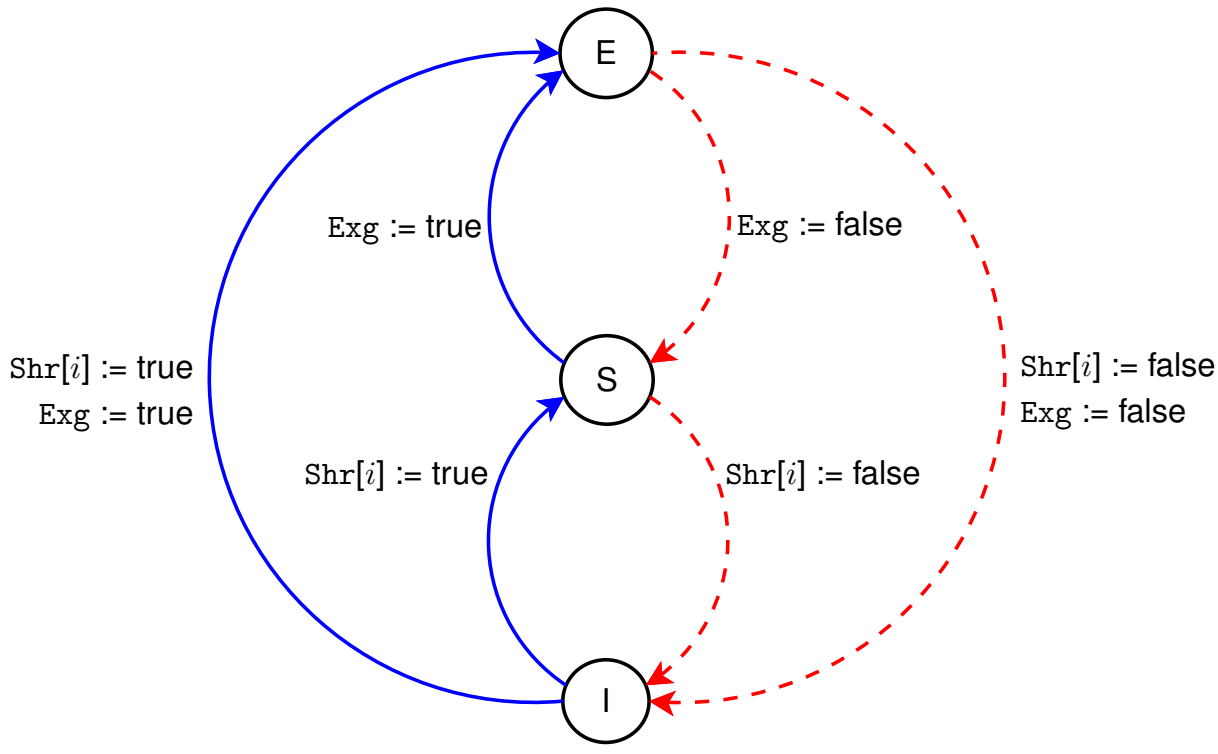
Directory :

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

$\texttt{Ptr} \in proc$

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

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



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 : protocole de cohérence de cache German-*esque*

Client i :

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

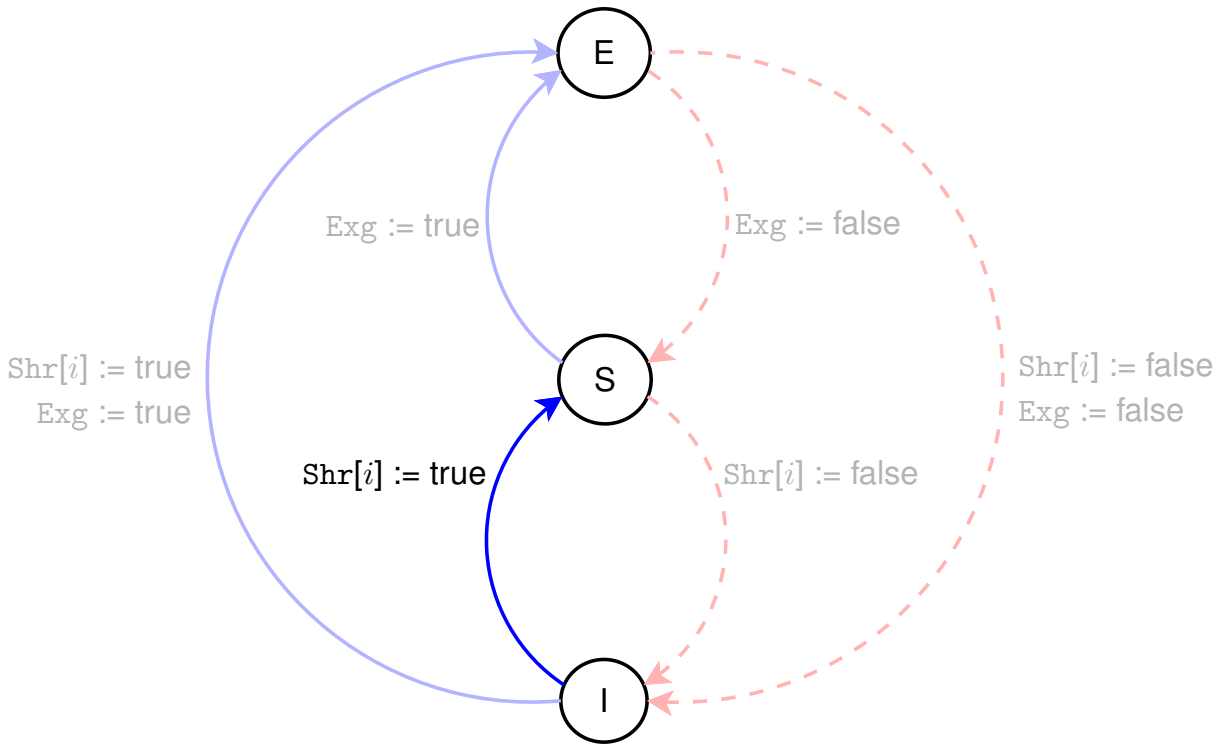
Directory :

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

$\texttt{Ptr} \in proc$

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

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



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

L'algorithme BRAB

I : états initiaux U : états dangereux (cubes) $\mathcal{T}$: transitions

---

BRAB () :

B := $\emptyset$; Kind(U) := Orig ; From(U) := U ;

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

while BWDA() = unsafe **do**

  if Kind(F) = Orig **then return** unsafe

  B := B $\cup$ { From(F) } ;

return safe

L'algorithme BRAB

I : états initiaux U : états dangereux (cubes) $\mathcal{T}$: transitions

---

BWD () :

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

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

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

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

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

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

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

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

return safe

L'algorithme BRAB

I : états initiaux U : états dangereux (cubes) $\mathcal{T}$: 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 return** unsafe

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

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

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

  return safe

L'algorithme BRAB

I : états initiaux U : états dangereux (cubes) $\mathcal{T}$: transitions

---

Approx$_{\mathcal{T}}$ (φ) :

  foreach ψ in `candidates`(φ) **do**

    if $\psi \notin$ B $\wedge$ $\mathcal{M} \not\models \psi$ **then**

      `Kind`(ψ) := Appr ;

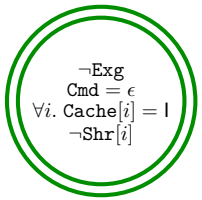
      ...

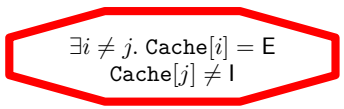
      return ψ

    ...

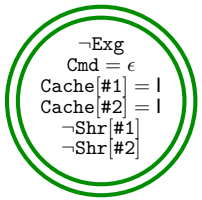
  return `pre`$_{\mathcal{T}}$(φ)

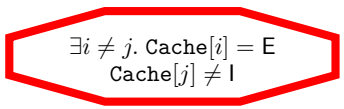
Exemple : BRAB sur German-*esque*



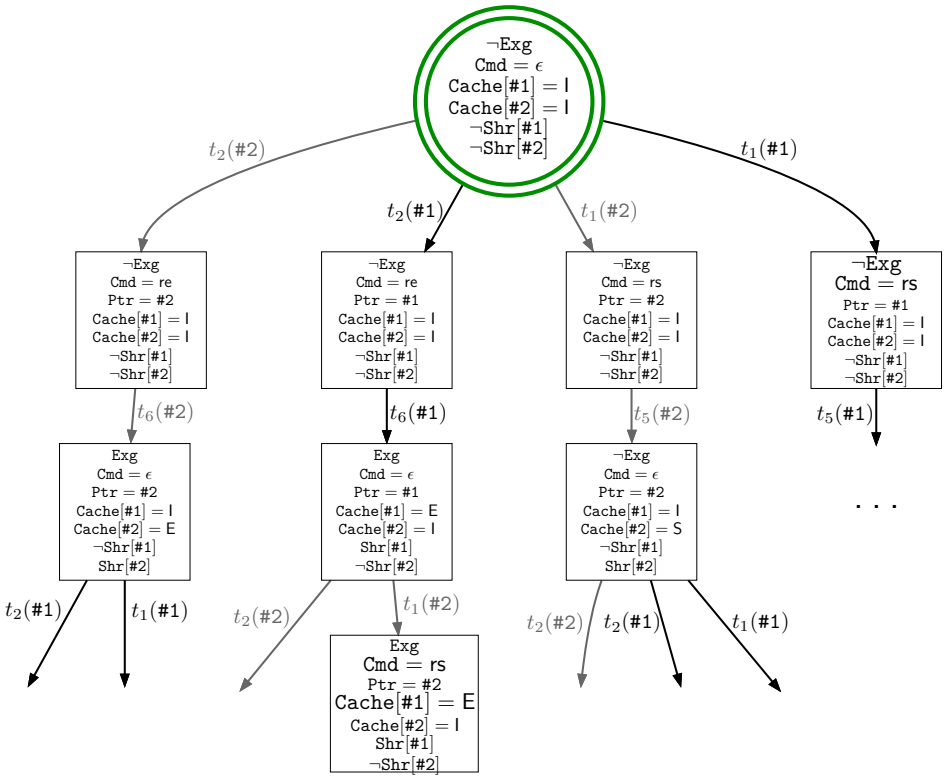


Exemple : BRAB sur German-*esque*



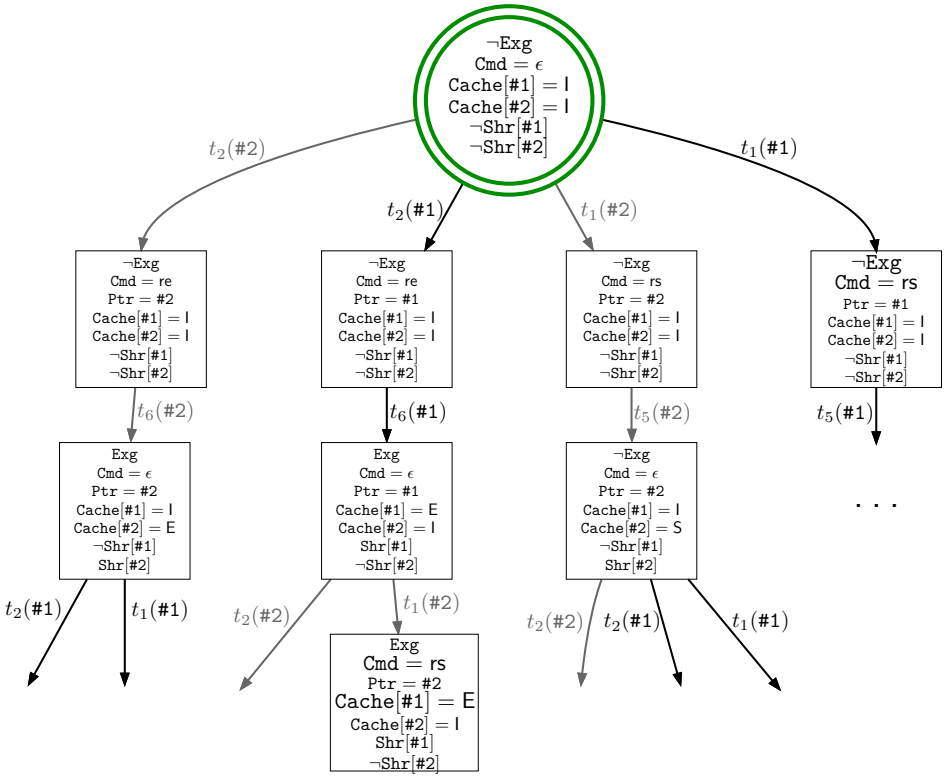


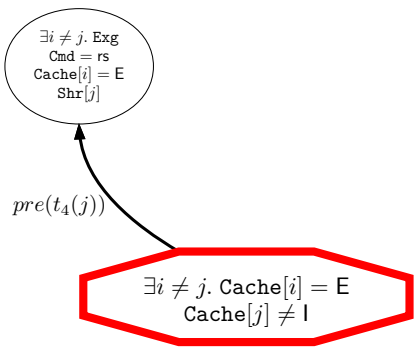
Exemple : BRAB sur German-*esque*



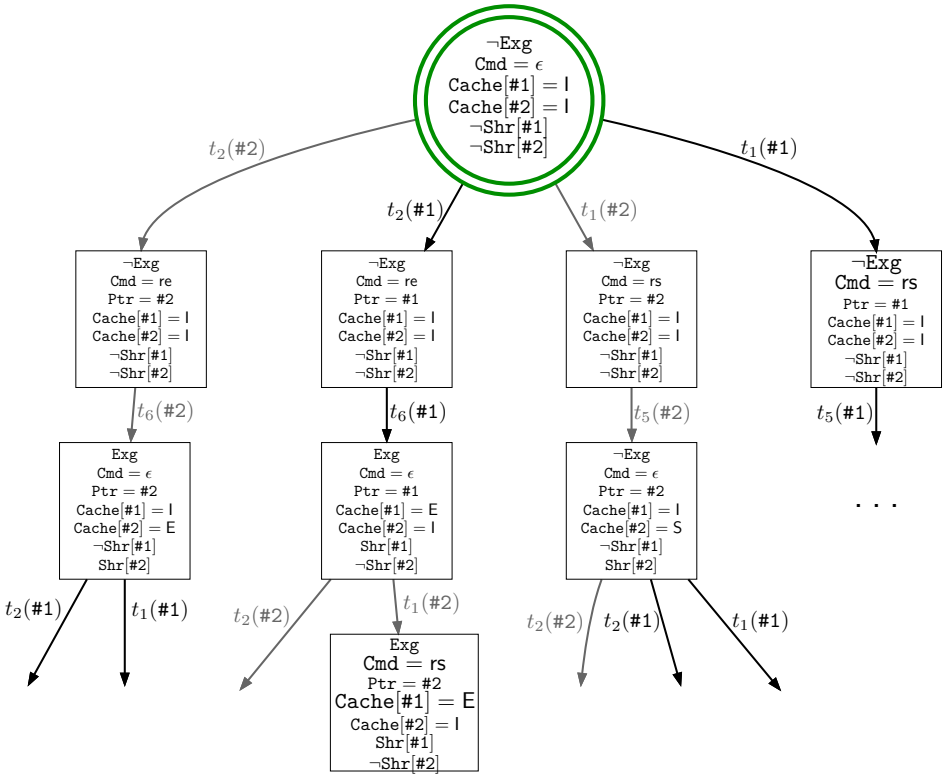
$\exists i \neq j.\ \mathtt{Cache}[i] = \mathsf{E}$
$\mathtt{Cache}[j] \neq \mathsf{I}$

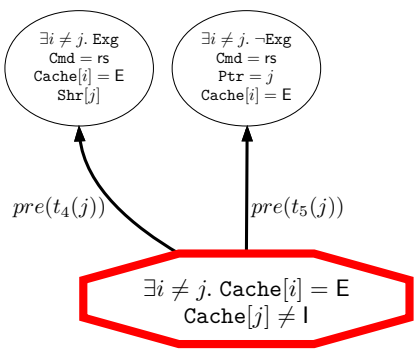
Exemple : BRAB sur German-*esque*



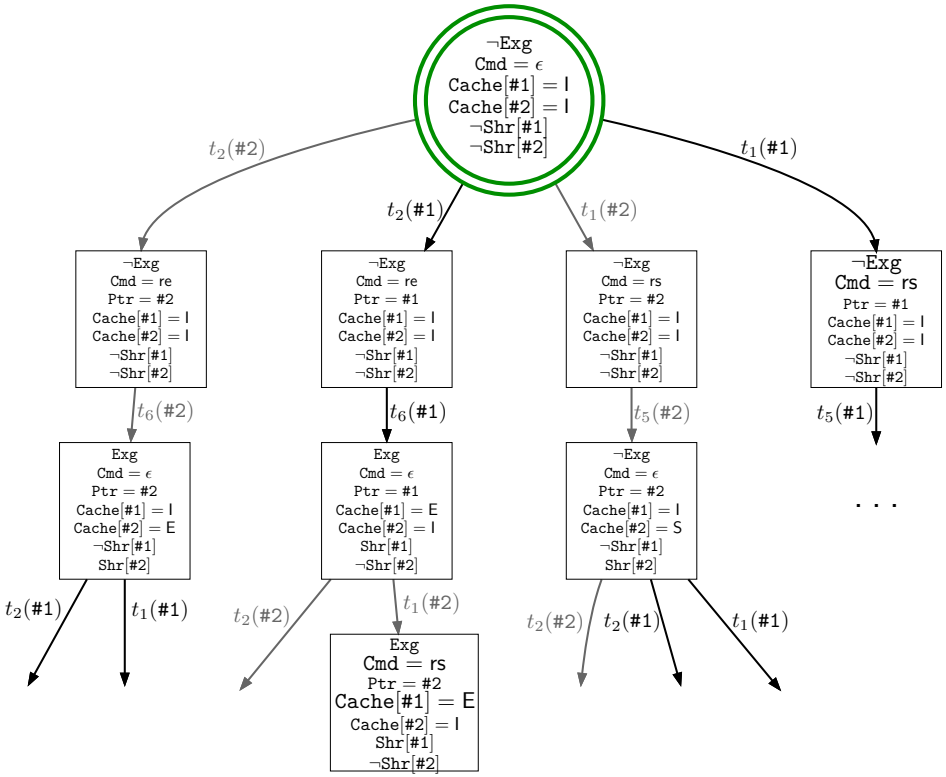


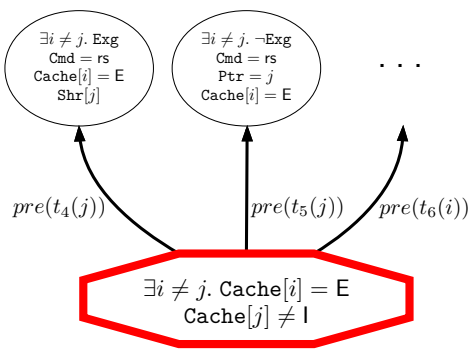
Exemple : BRAB sur German-*esque*



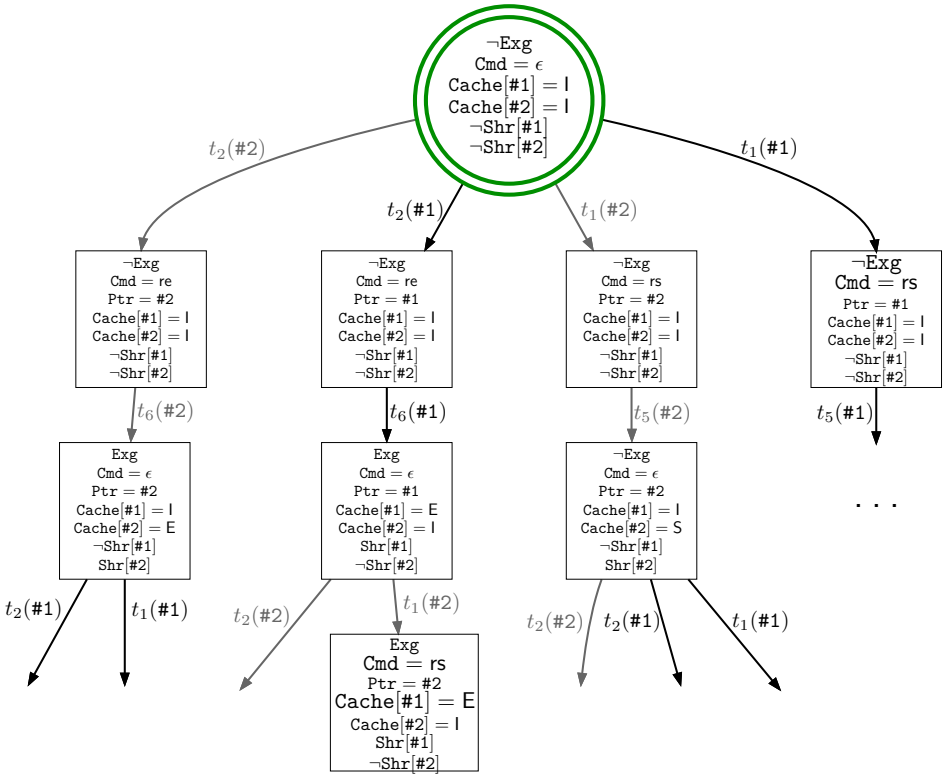


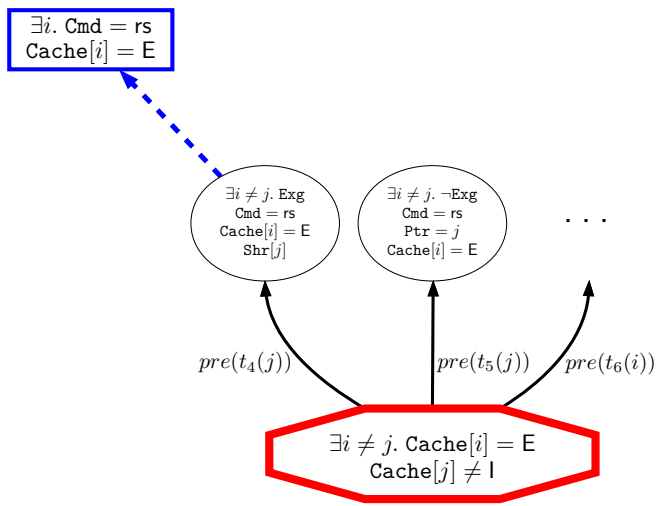
Exemple : BRAB sur German-*esque*



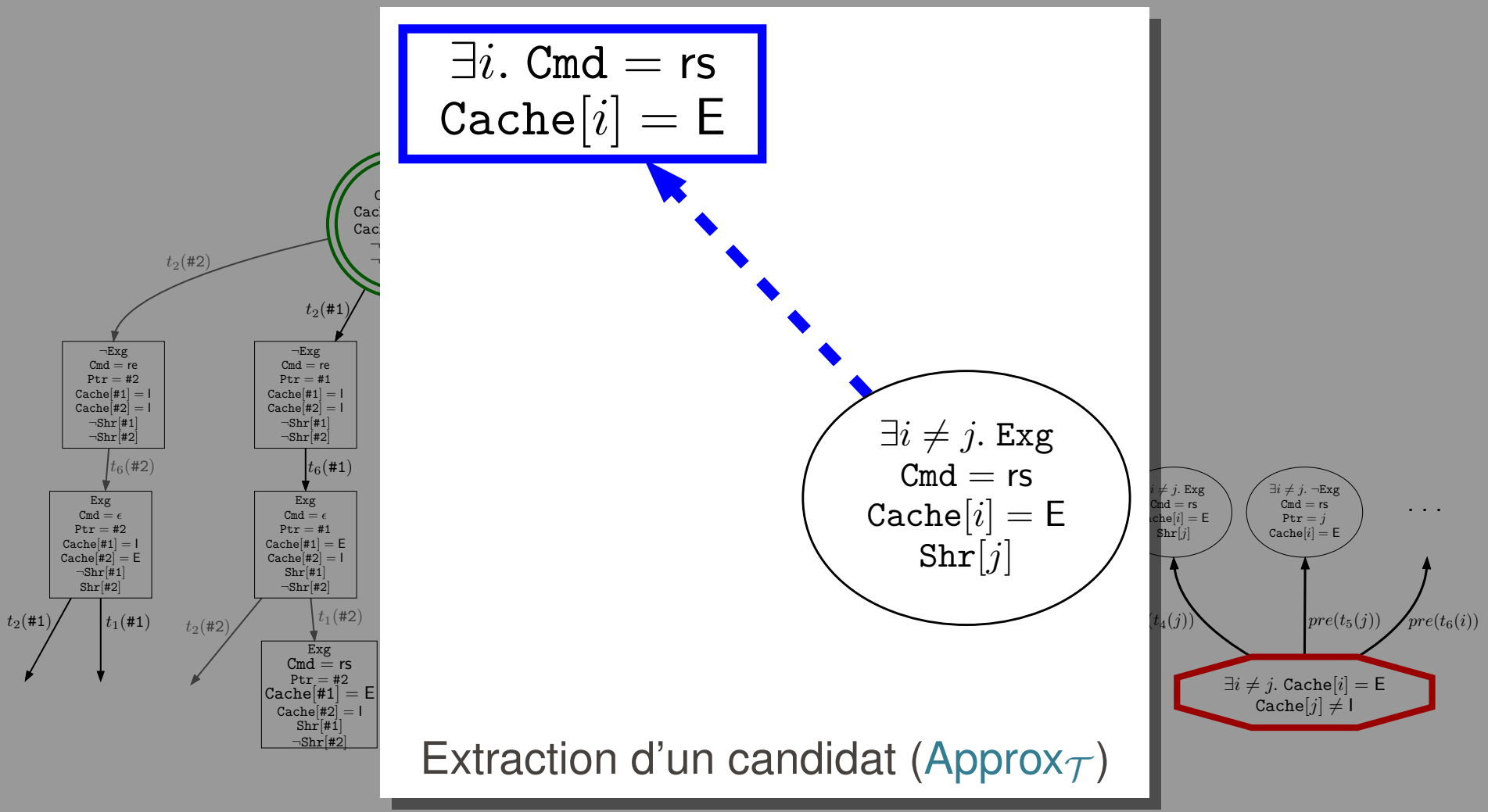


Exemple : BRAB sur German-*esque*



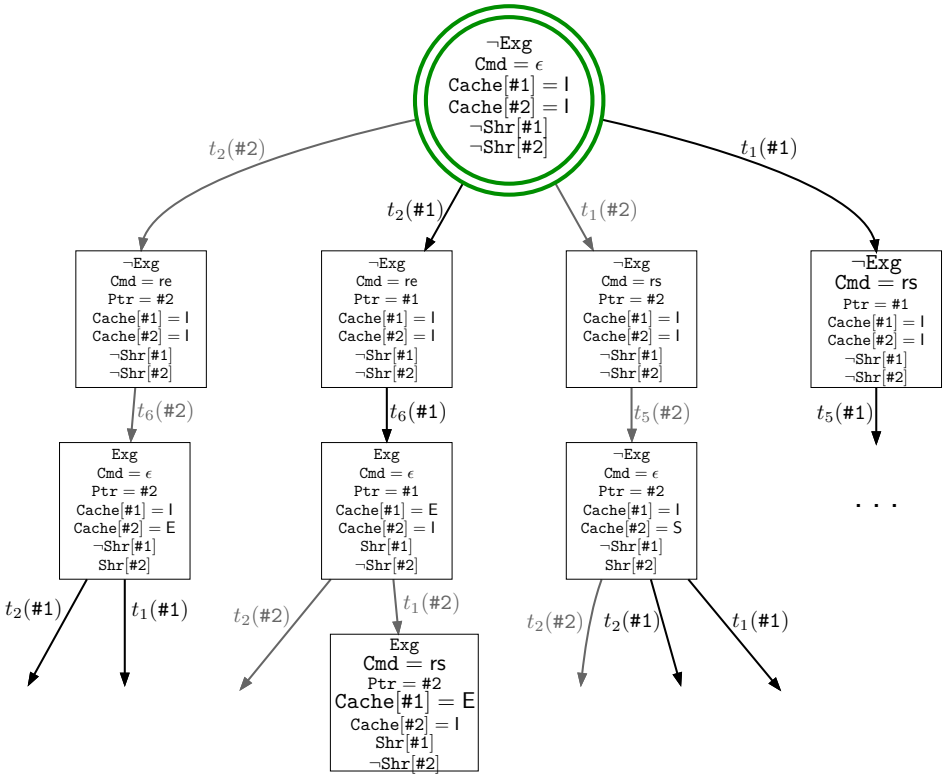


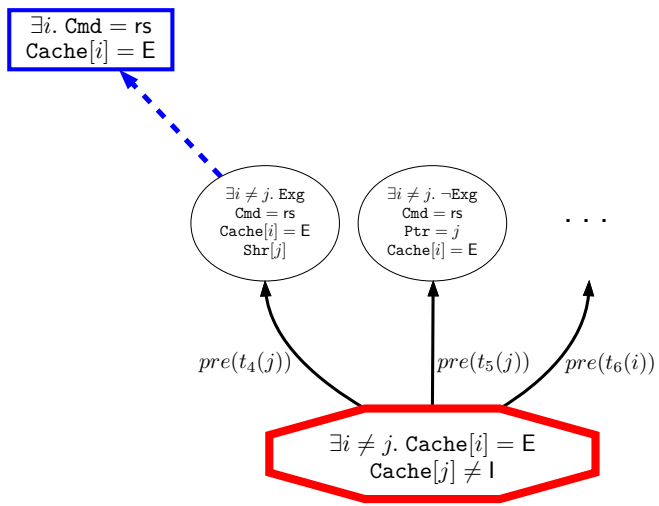
Exemple : BRAB sur German-*esque*



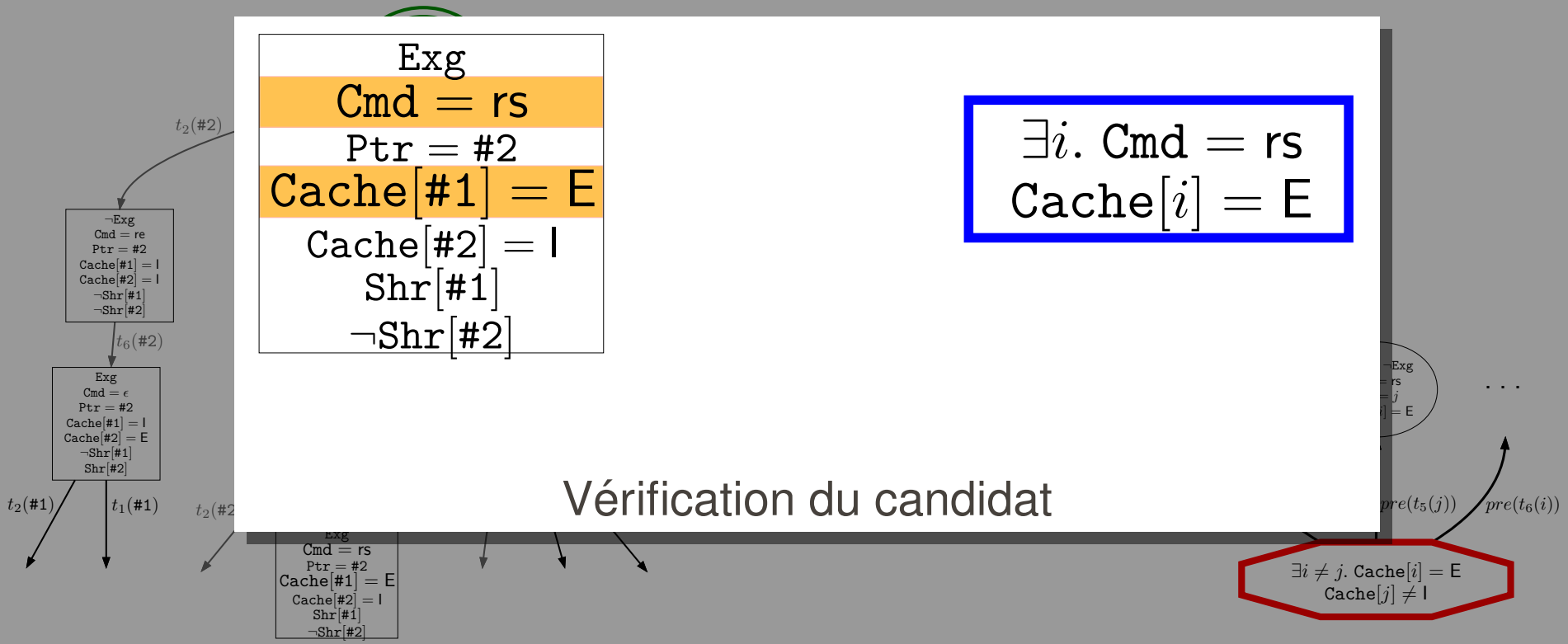
Extraction d'un candidat (Approx$_{\mathcal{T}}$)

Exemple : BRAB sur German-*esque*

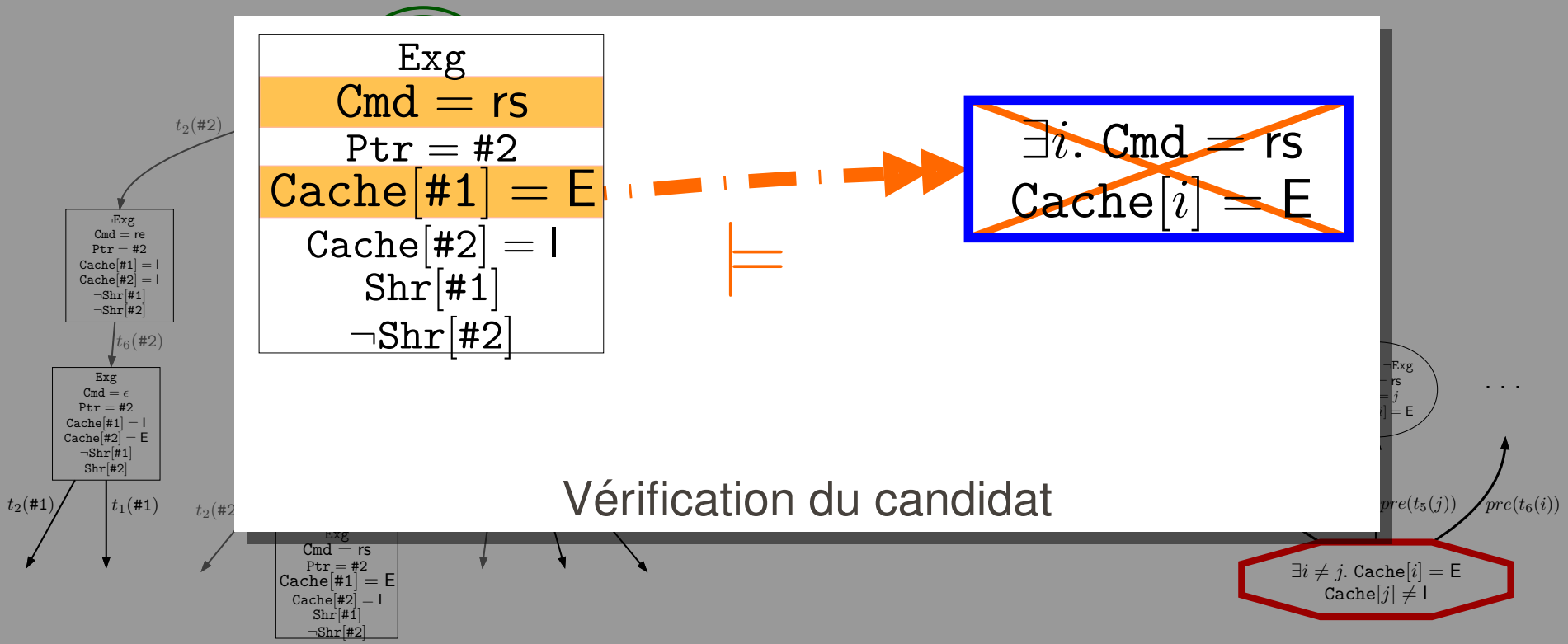




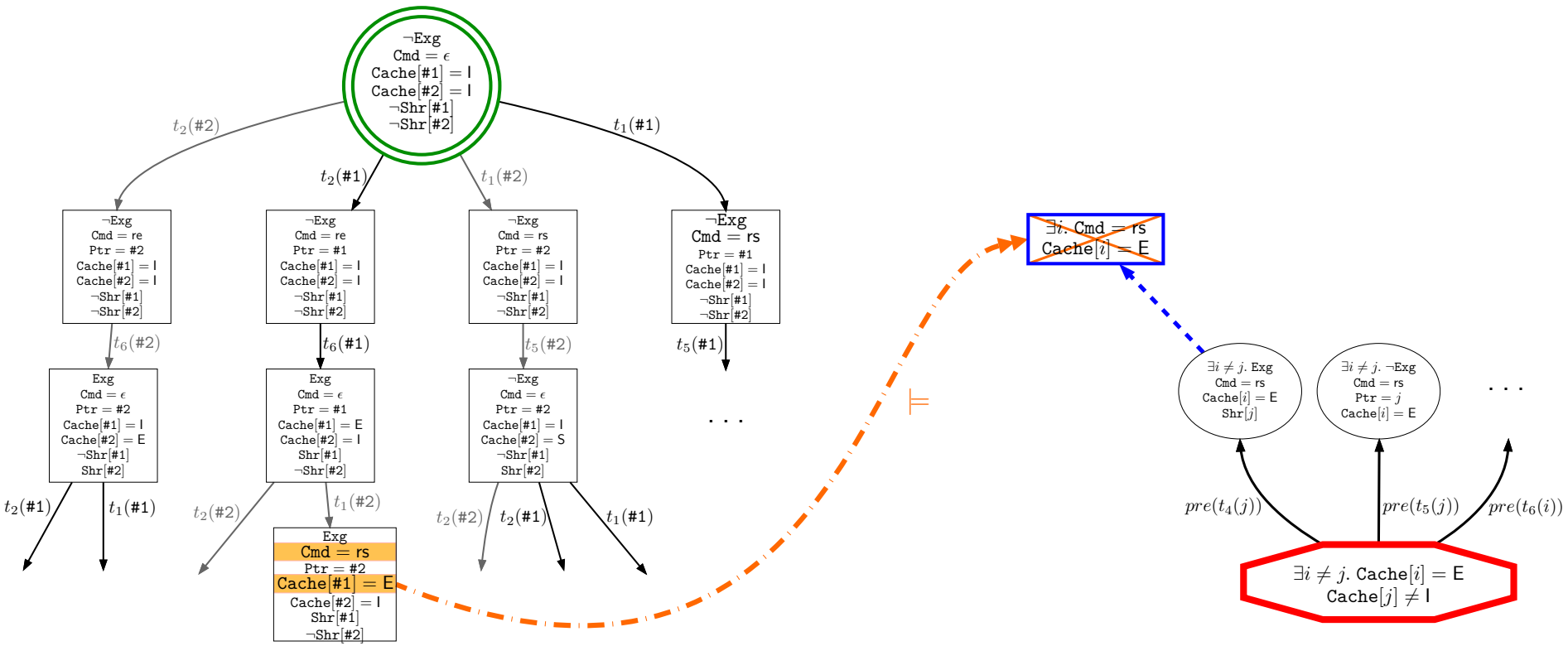
Exemple : BRAB sur German-*esque*



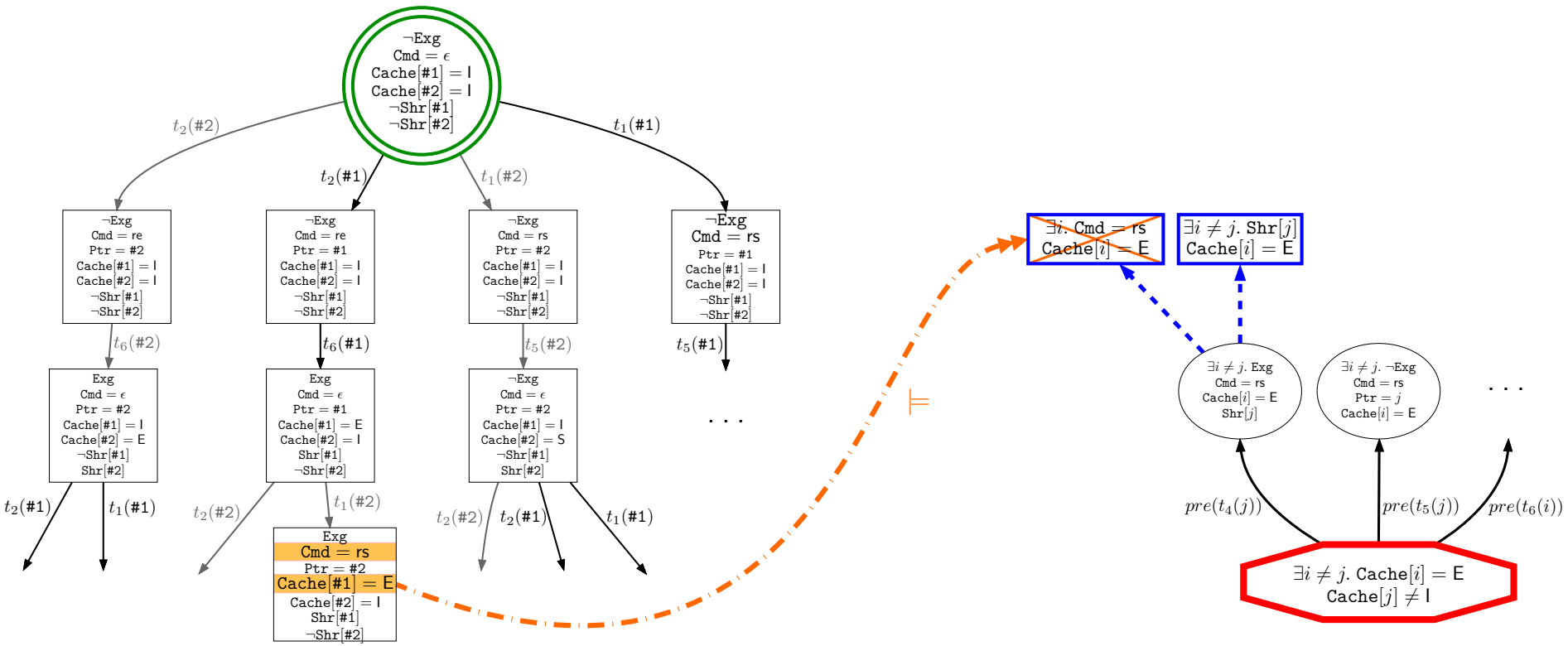
Exemple : BRAB sur German-*esque*



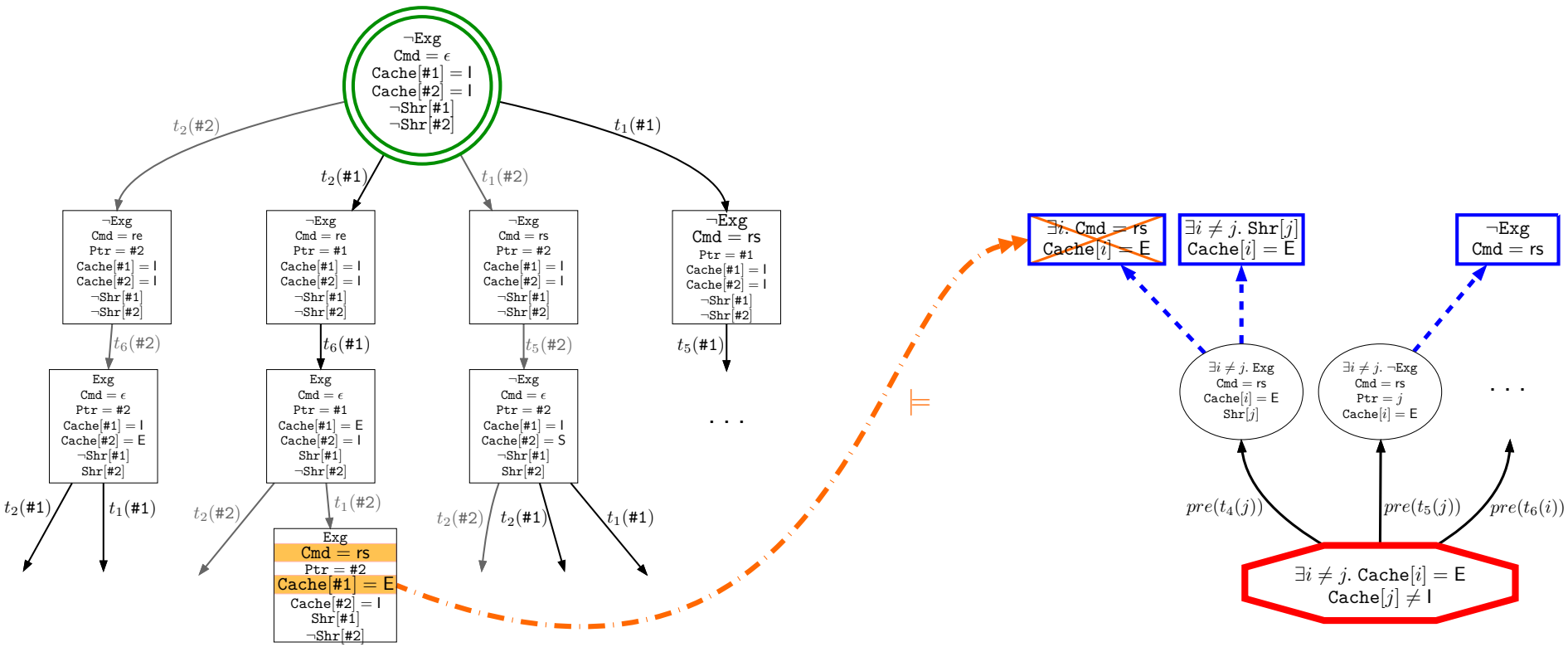
Exemple : BRAB sur German-*esque*



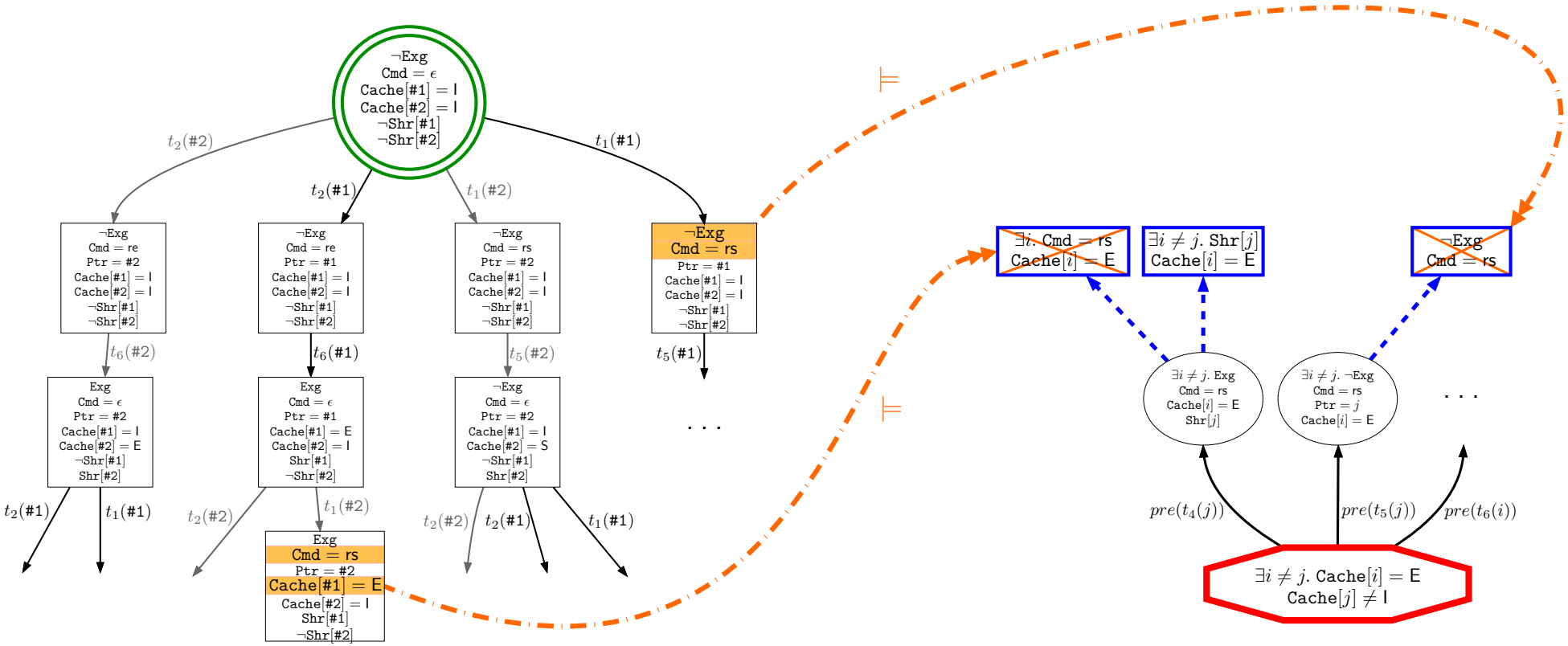
Exemple : BRAB sur German-*esque*



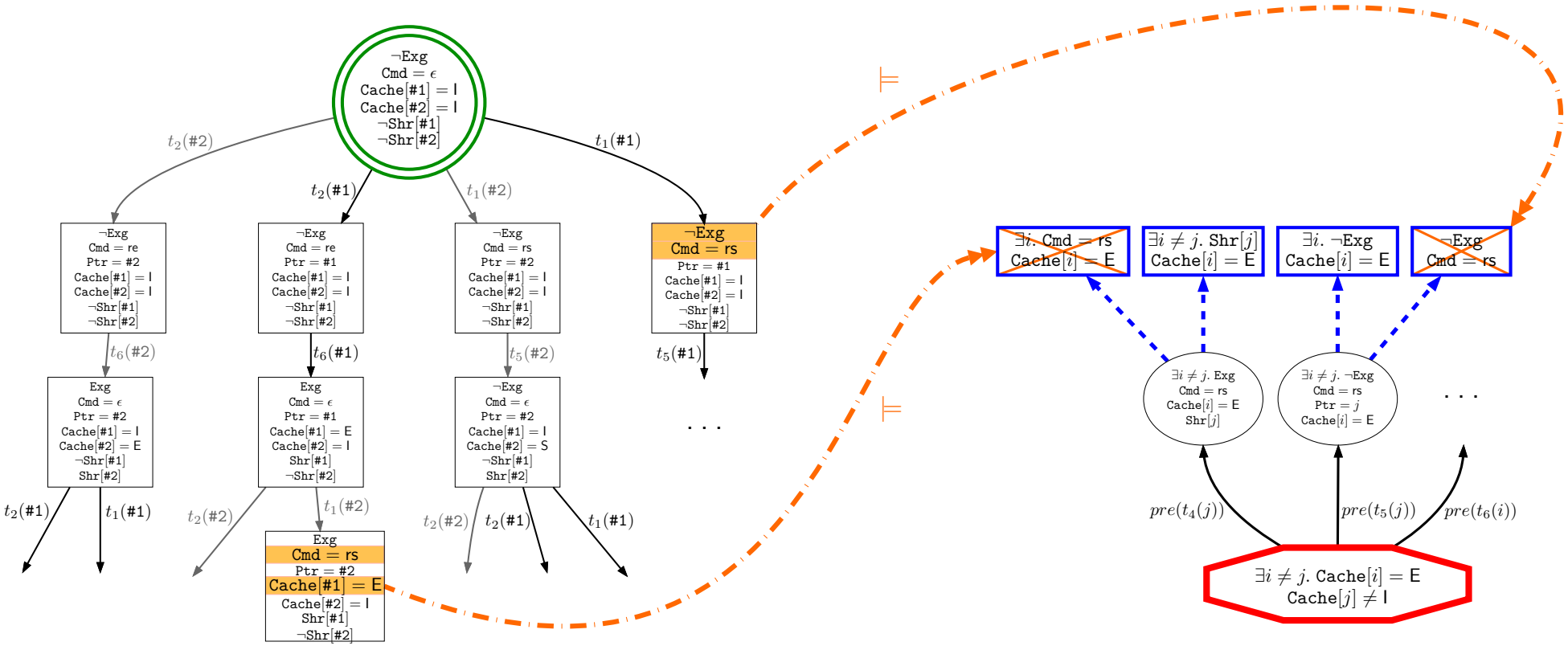
Exemple : BRAB sur German-*esque*



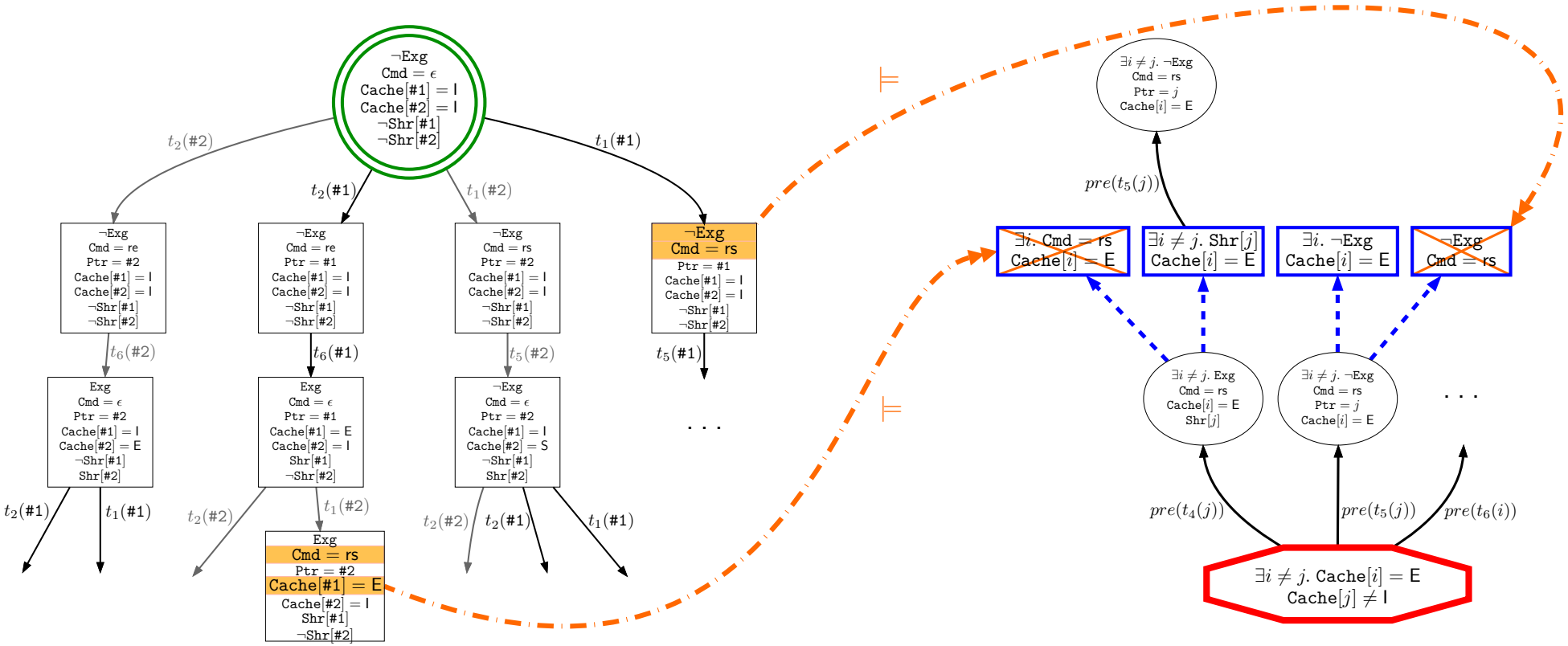
Exemple : BRAB sur German-*esque*



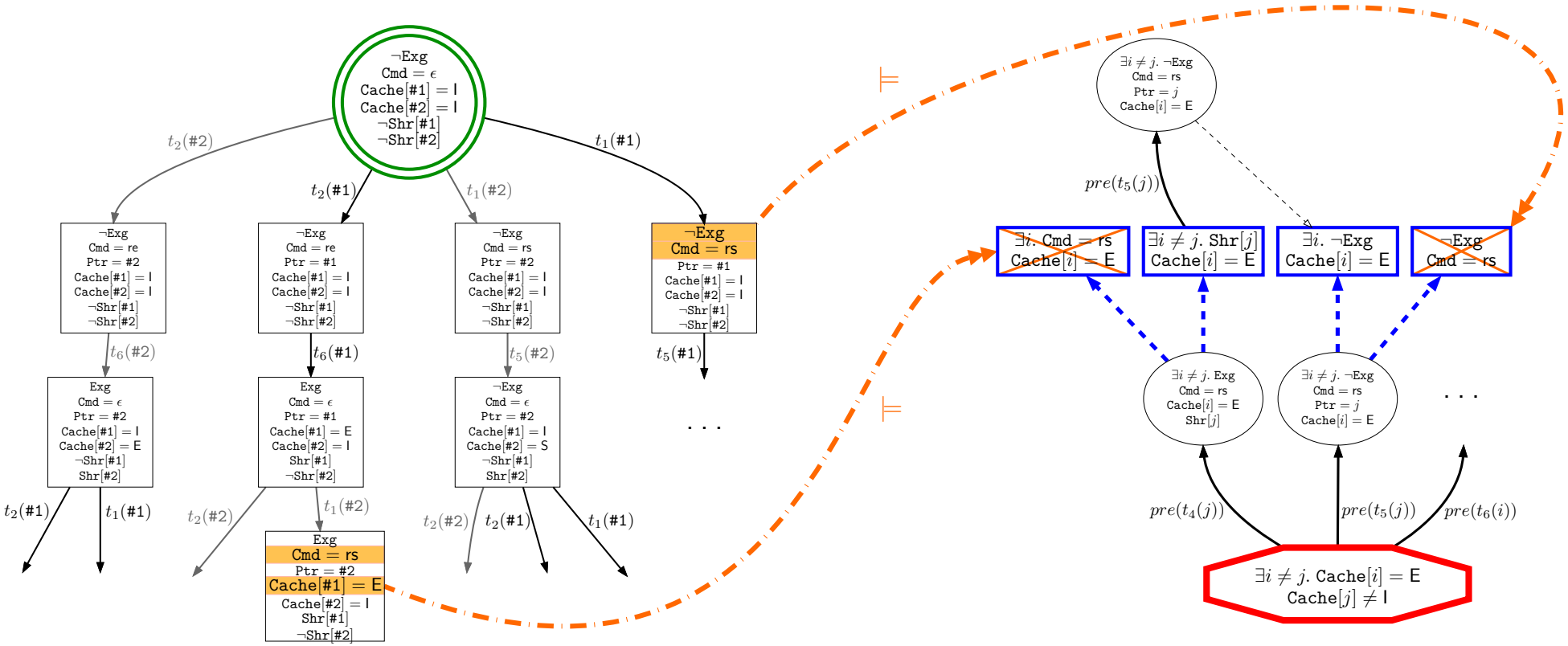
Exemple : BRAB sur German-*esque*



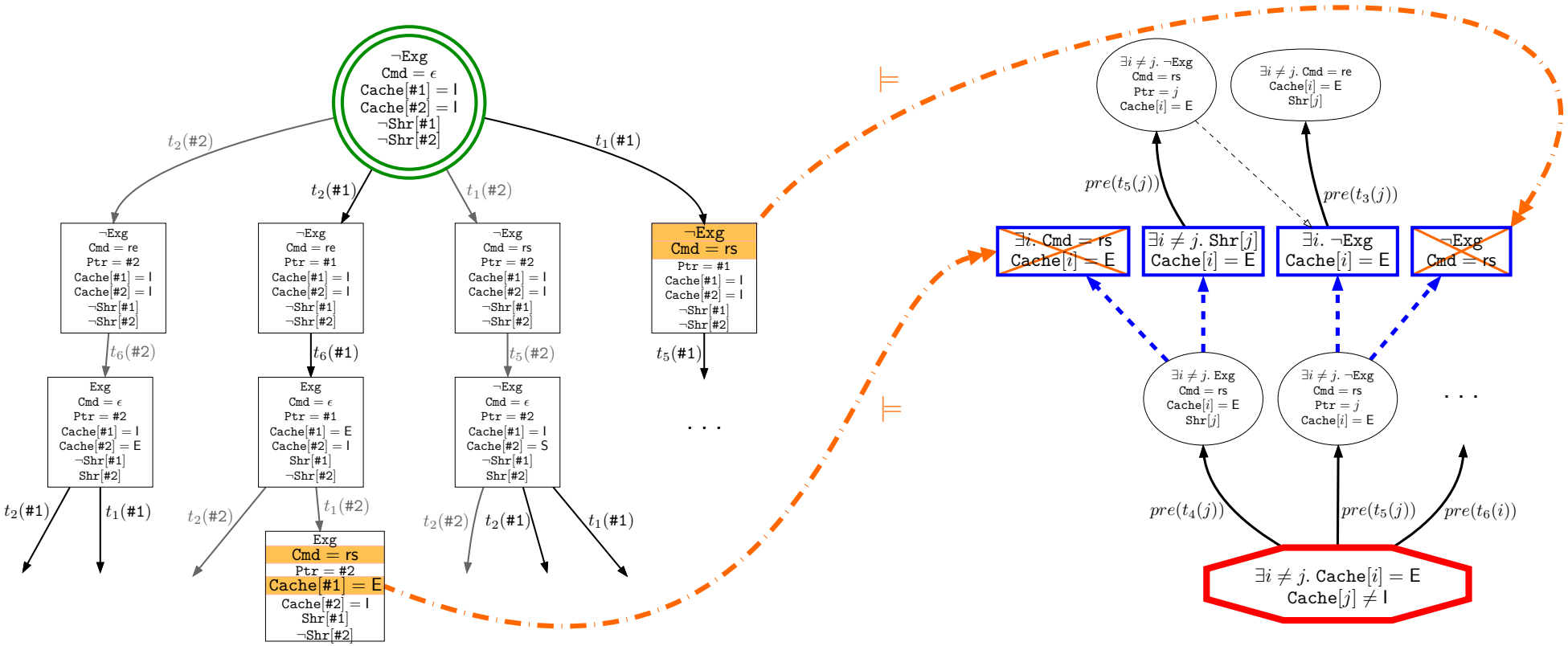
Exemple : BRAB sur German-*esque*



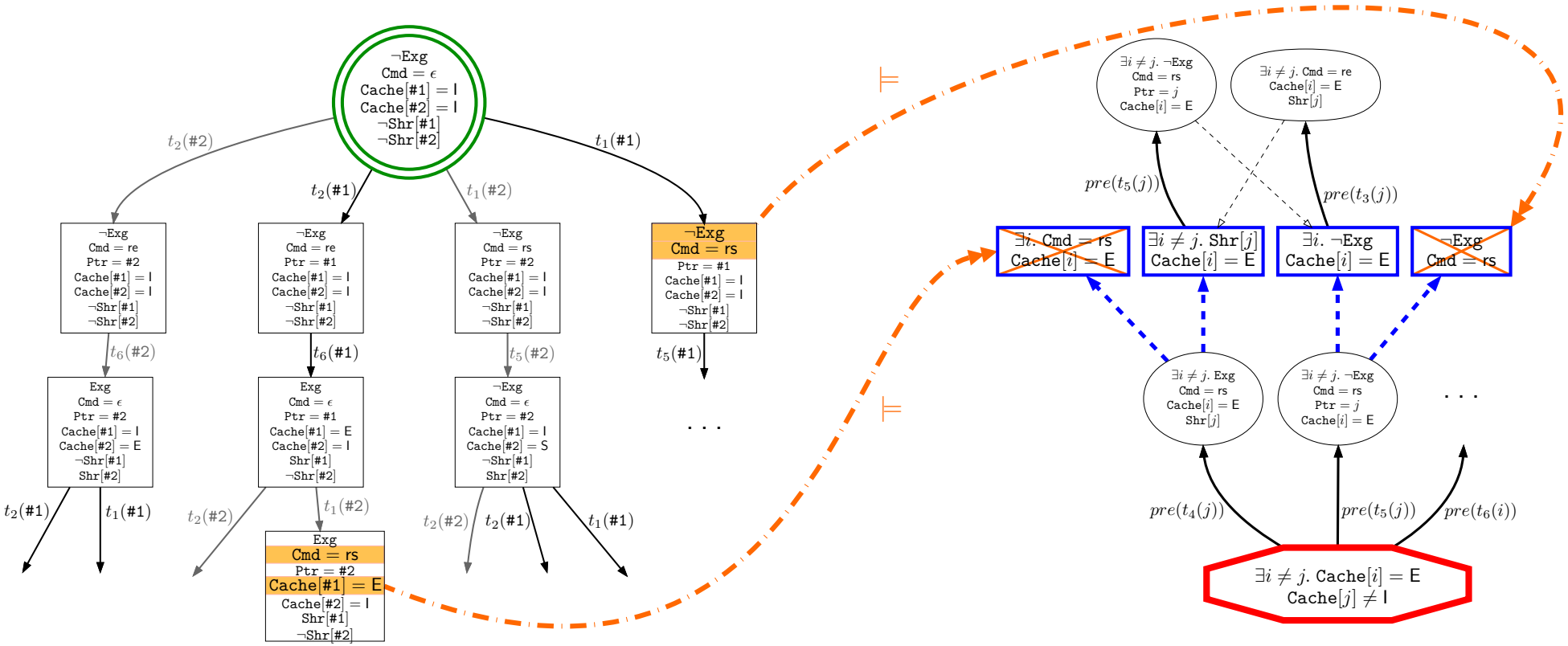
Exemple : BRAB sur German-*esque*



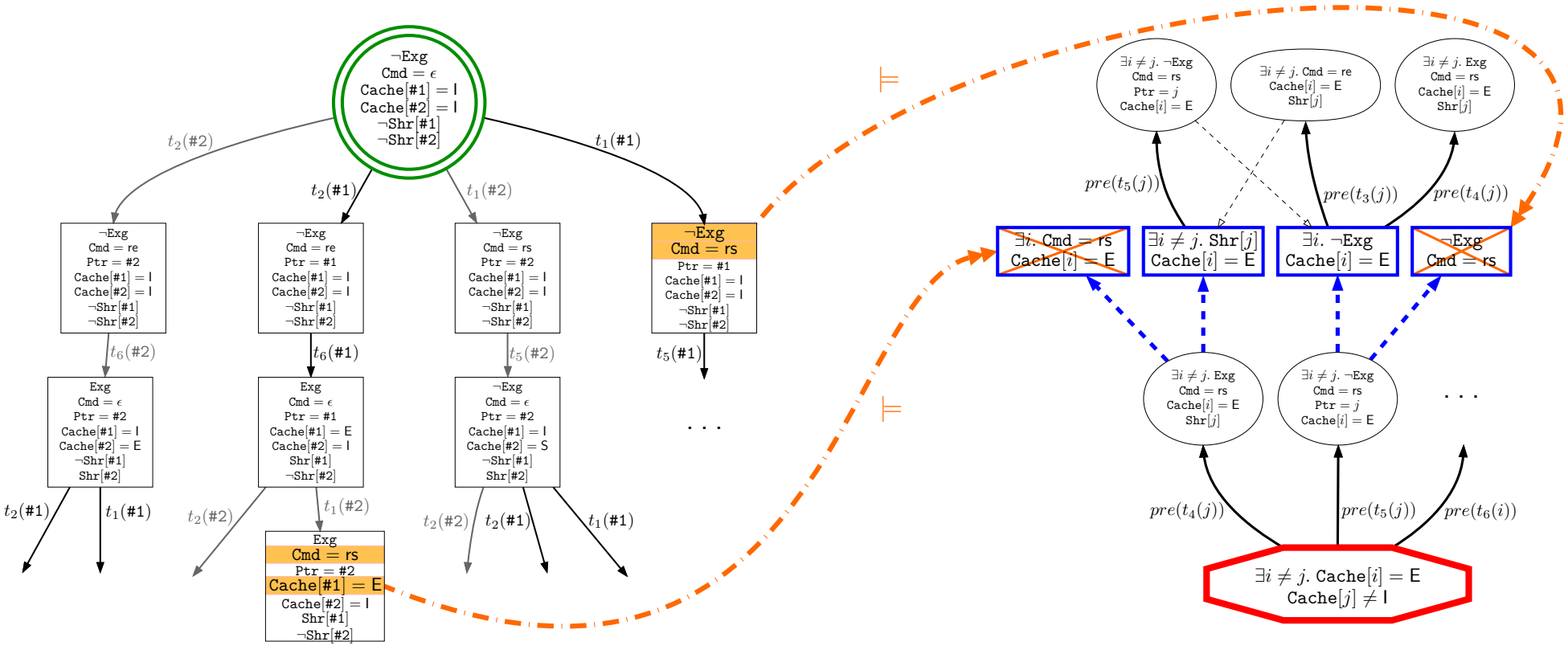
Exemple : BRAB sur German-*esque*



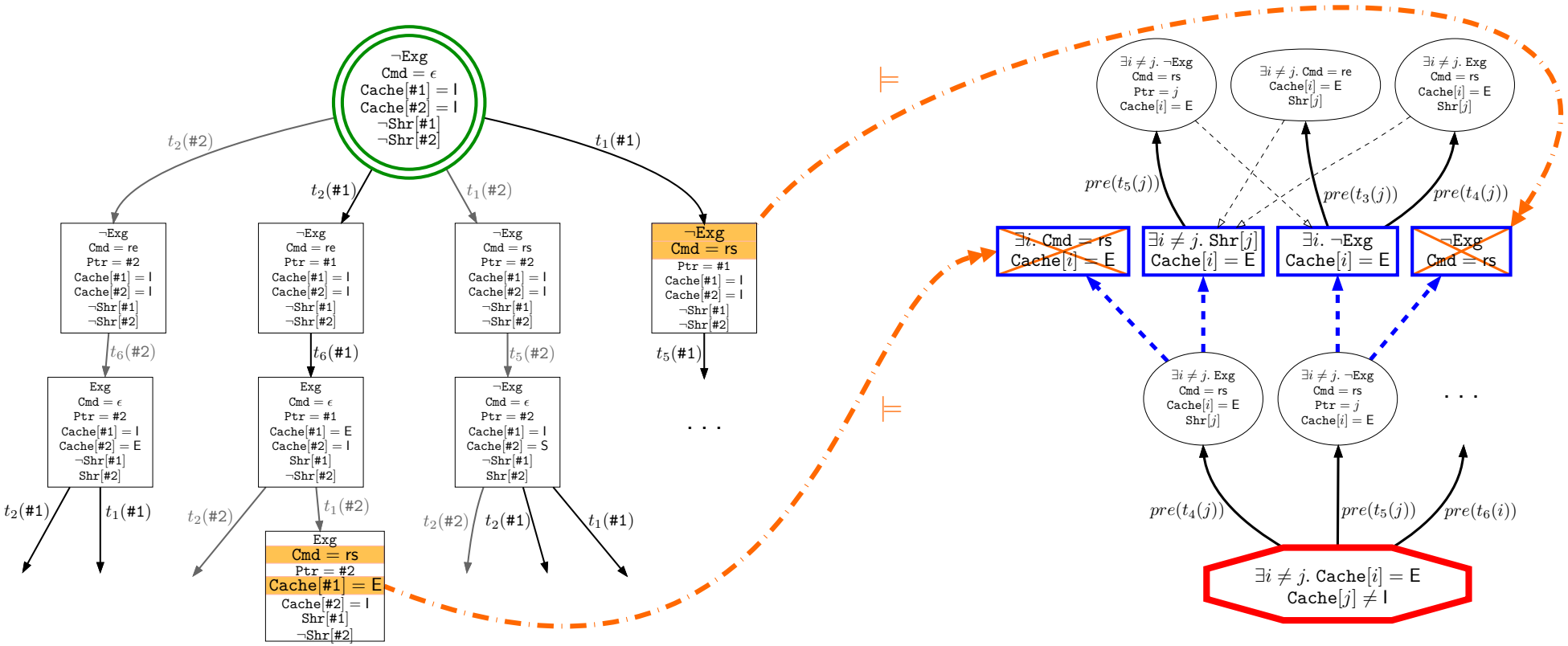
Exemple : BRAB sur German-*esque*



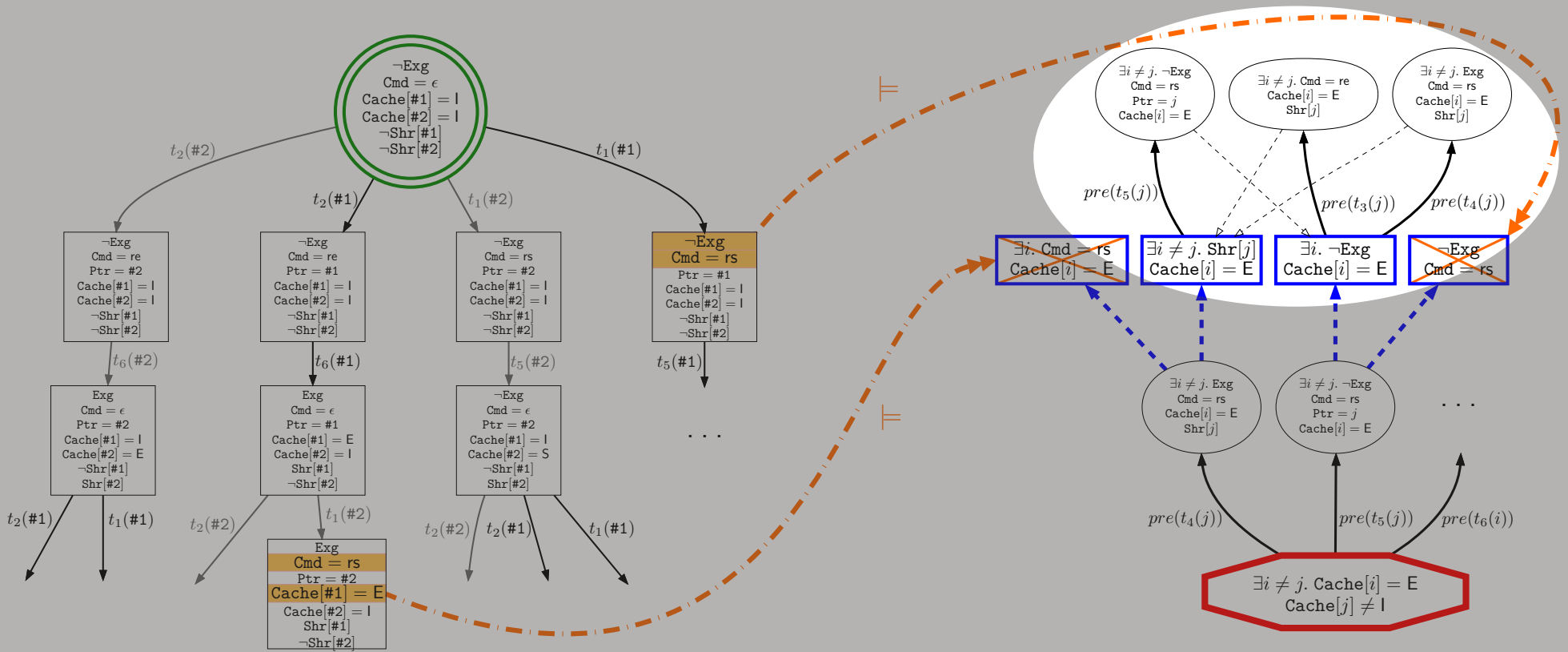
Exemple : BRAB sur German-*esque*



Exemple : BRAB sur German-*esque*



Exemple : BRAB sur German-*esque*



Quelques benchmarks

	BRAB	Cubicle	CMurphi		
Szymanski_at	**0.14s**	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

Remarks

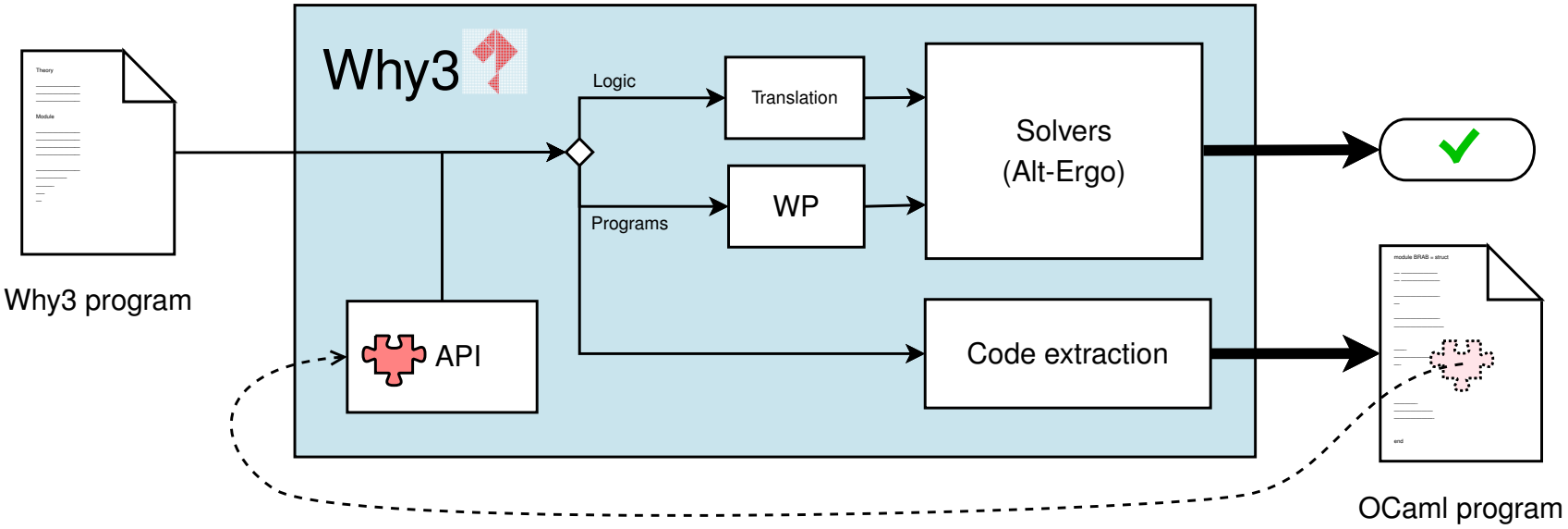
- BRAB est complet seulement si le framework admet une analyse d'atteignabilité arrière complète
- Cubicle va au-delà du fragment décidable des systèmes à tableaux
- FLASH est exprimé hors de ce fragment
- BRAB reste correct

Travaux futurs

Améliorations :

- Expérimenter avec des protocoles industriels de taille réelle
- Améliorer le backtracking
- Découverte de candidats pour des invariants numériques ardue

Travail en cours : Certification de Cubicle en Why3



- Prouver formellement la correction des algorithmes d'atteignabilité arrière
- Extraire le code avec Why3 vers OCaml
- Réaliser les fonctions axiomatisées avec l'API de Why3

But : un model checker efficace et certifié

Preuve de sûreté de BRAB

- sat : `formula`

$$sat(f : \texttt{formula}) = \exists m : \texttt{structure}.\ m \models f$$

- pre : `formula` $\rightarrow$ `formula`

$$\forall f : \texttt{formula}.\ pre^*(pre(f)) \vee f = pre^*(f)$$

- $reachable(I, f)$ est vrai si un état de f est atteignable à partir d'un état de I et défini comme suit :

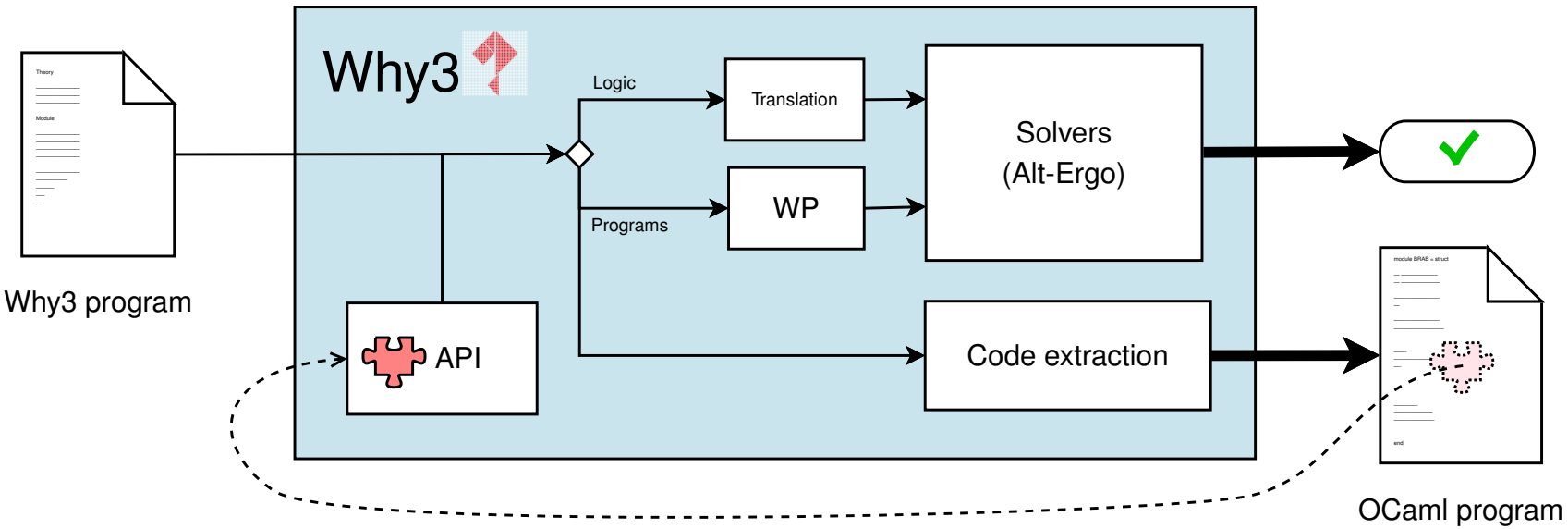
$$reachable(I, f) = sat(pre^*(f) \wedge I)$$

Preuve de sûreté de BRAB

$$\texttt{BRAB}(I, \Theta) = \text{Safe} \implies \neg reachable(I, \Theta)$$

$$\texttt{BRAB}(I, \Theta) = \text{Unsafe} \implies reachable(I, \Theta)$$

Défis restants



- Réalisation des fonctions axiomatisées avec l'API de Why3 :
  - Notre prédicat sat = appel à un solveur
  - Fonction pre = calcul de plus faible pré-condition (WP) de Why3
- Limiter la *base de confiance* :
  - Alt-Ergo + Why3
  - Intégrer l'API d'Alt-Ergo dans Why3

Merci

http://cubicle.lri.fr