Le contrôle PROLOG

- Contrôle du backtrack
  - Soit une fonction **f** dont une définition Prolog peut être :

    f(**X**, 0) :- **X** < 3.
    f(**X**, 2) :- 3 =< **X**, **X** < 6.
    f(**X**, 4) :- 6 =< **X**.

  - Que se passe-t-il si on pose la question ?
    ?- **f**(1,**Y**), 2 < **Y**.

?- f(1,Y), 2 < Y.

f(**X**, 0) :- **X** < 3.
f(**X**, 2) :- 3 =< **X**, **X** < 6.
f(**X**, 4) :- 6 =< **X**.

Démonstration de f(1,Y)

Première règle : Y = 0
Démonstration de 2 < Y :- **échec**

Deuxième règle : **échec**

Troisième règle : **échec**

Prolog : extensions à la logique pure
Cut rouge ou vert, négation

Prolog prévoit UN moyen d'agir sur le parcours de l'arbre de preuve

Il s'agit d'un prédicat, appelé Cut/0, et noté '!'

r(...):-r1(...), ..., ri(...),**!**,rj(...),.,rn(...)

Interprétation

- Cut est vrai (le but '!' s'efface immédiatement)
- Passé le Cut, ce qui le précède ne sera plus remis en cause : aucune autre alternative ne sera explorée pour r(...), ni pour r1(...) ... ri(...)
- Autres façons de voir la chose :- redo(!) nous renvoie directement à redo(X), où X est le but qui a fait apparaître le Cut
- revenir sur ! par bactracking nous remonte au père du noeud où est apparu le Cut

Cut … suite !

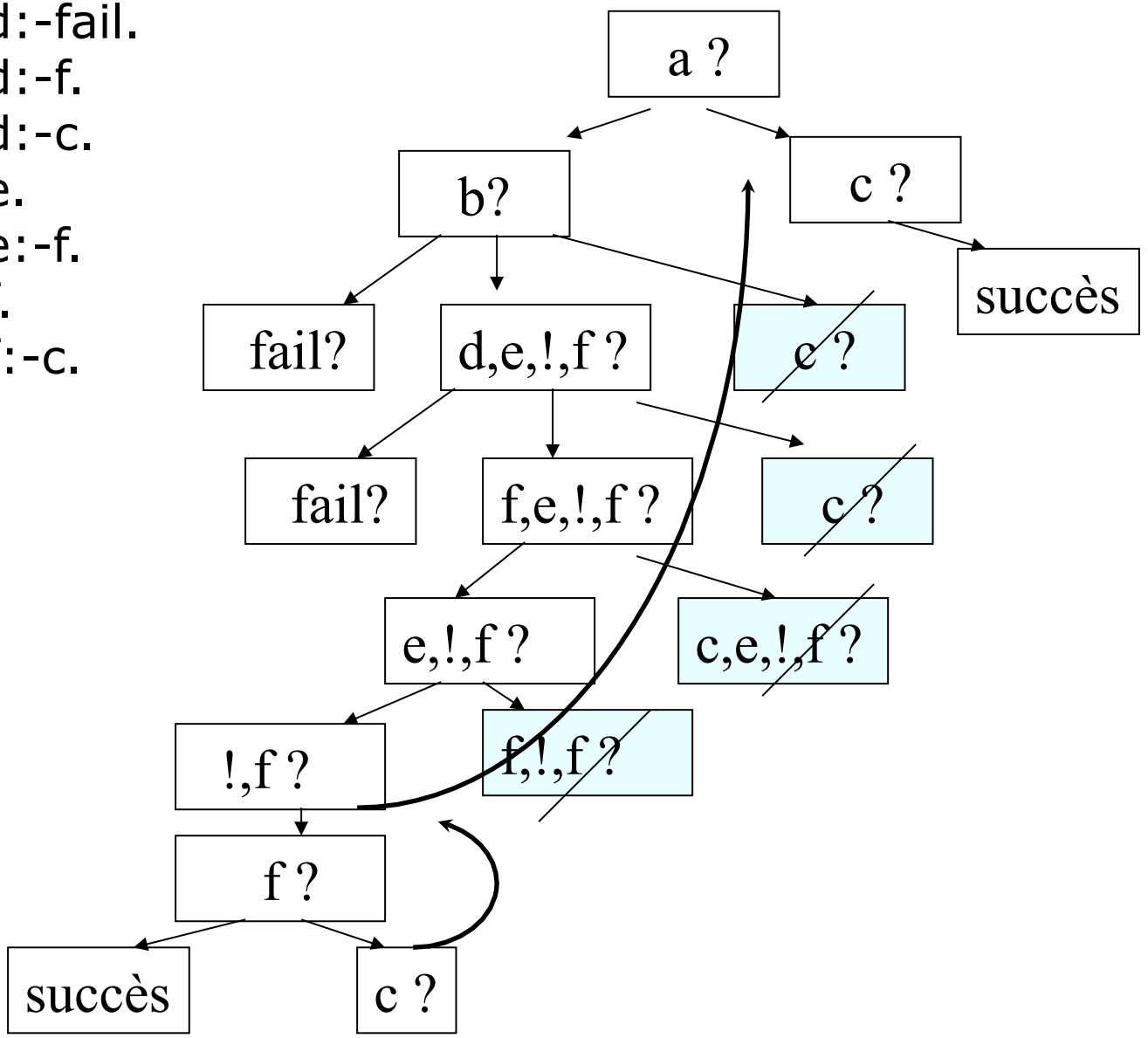
- **Deux utilisations possibles**
- On peut utiliser le Cut pour gagner en efficacité : si plusieurs règles sont mutuellement exclusives, le ! évitera au moteur Prolog d'essayer les autres règles (qui ne donnent de toutes façons rien)

- MAIS : le Cut peut aussi changer radicalement la logique du programme !

Illustration du Cut dans l'arbre de preuve

```
a:-b.
a:-c.
b:-fail.
b:-d,e,!,f.
b:-c.
c.
```

```
d:-fail.
d:-f.
d:-c.
e.
e:-f.
f.
f:-c.
```



La coupure : !

- On appelle **but père** le but ayant permis d'unifier la clause contenant la coupure (**!** cut)
- L'effet du **cut** est de **couper** tous les points de choix restant depuis le but père. Les autres alternatives restent en place

  f(***X***, *0*) :- ***X*** < *3*, ***!***.
  f(***X***, *2*) :- *3* =< ***X***, ***X*** < *6*, ***!***.
  f(***X***, *4*) :- *6* =< ***X***.

- *?-* ***f***(*7*,***Y***).

Si … alors … sinon

- Le **cut** peut servir à exprimer des conditions mutuellement exclusives et ainsi **simplifier** l'écriture
- La clause suivant un **cut** peut être considérée comme un **sinon**

  f*(**X**, 0) :-* ***X*** *< 3,* ***!****.*
  f*(**X**, 2) :-* ***X*** *< 6,* ***!****.*
  f*(**X**, 4).*

- *?-* ***f****(1,2).*

Un usage délicat

- **Green cut** : la sémantique déclarative du programme **n'est pas** modifiée
  - on peut enlever le **cut** le programme fonctionnera toujours

- **Red cut** : la sémantique déclarative du programme **est** modifiée
  - Le retrait du **cut** conduit à un programme au fonctionnement erroné
  - Généralement, la version avec **cut** peut être **prise en défaut**

Autres prédicats de contrôle

- **true** est un but qui **réussit** toujours
  - p(a,b). ≡ p(a,b) :- **true**.
- **fail** est un but qui **échoue** toujours
- call(**X**) est un **méta but**. Il considère **X** comme un but et essaie de le résoudre.
  - ?- **Y**=b, **X**=member(**Y**, [a,b,c]), call(**X**).
    Yes

Application : négation

- Expression de la négation en Prolog
  - different(**X**, **Y**) :-
    X = **Y**, **!**, **fail**.
    different(**X**, **Y**).

- Un prédicat plus général :
  - not(**P**) :-
    P, **!**, **fail**.
    not(_).

Problèmes avec le not !

***r**(a).*
***q**(b).*
***p**(**X**) :- **not**(**r**(**X**)).*

- *?- **q**(**X**), **p**(**X**).*
- *?- **p**(**X**), **q**(**X**).*

Théorie du monde clos

- **not**(**X**)
  - ne veut pas dire
    - ***X*** *est toujours faux*
  - veut simplement dire
    - *Je n'ai pas assez d'information pour prouver* ***X***

- **Prolog** considère ce qui n'est pas vrai comme faux et vice-versa
  - c'est la théorie du **monde clos**

- A quoi peut servir : **not**(**not**(**P**)) ?