فصل چهارم
کنترل عقبگرد

سید ناصر رضوی

razavi@Comp.iust.ac.ir

۱۳۸۳

مقدمه

- جلوگیری از عقبگرد (backtracking).
- چند مثال برای استفاده از cut.
- نفی (negation).
- مشکلات cut و negation.

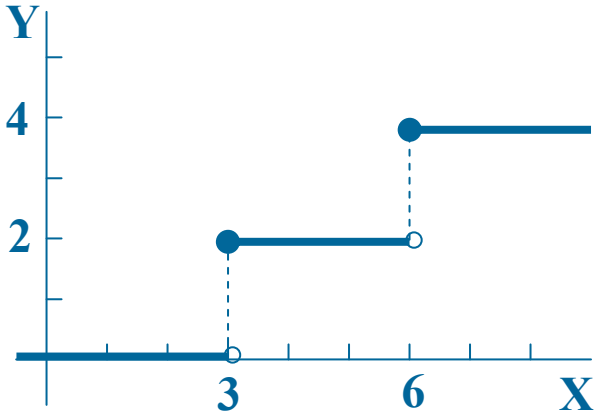
جلوگیری از عقبگرد

- نحوه کنترل اجرای برنامه توسط برنامه نویس:
  - جا به جا کردن ترتیب فراکردها و اهداف.
  - استفاده از عملگر cut.
- اگر مکانیزم عقبگرد خودکار در پرولوگ کنترل نشود، ممکن است باعث ناکارآمدی برنامه شود.
- بنابراین، برخی اوقات ما می خواهیم از عقبگرد جلوگیری کنیم و آنرا کنترل کنیم.(با استفاده از cut).

جلوگیری از عقبگرد

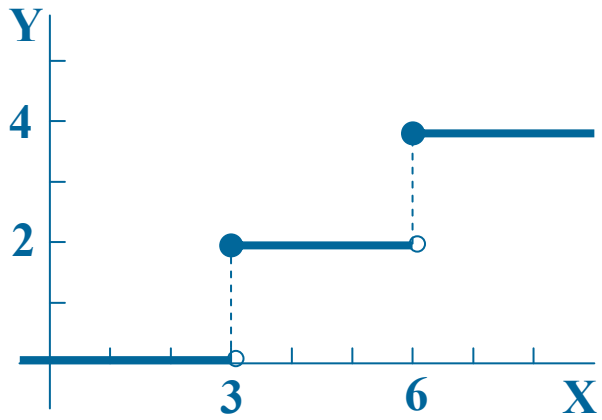
- ابتدا رفتار یک برنامه ساده را که شامل برخی موارد غیرضروری از عقبگرد می باشد، بررسی می کنیم و نقاطی را که در آنها عقبگرد بی فایده می باشد را شناسایی می کنیم.
- **تابع دو پله ای:**

- *Rule1*: if $X < 3$ then $Y = 0$
- *Rule2*: if $3 \leq X < 6$ then $Y = 2$
- *Rule3*: if $6 \leq X$ then $Y = 4$



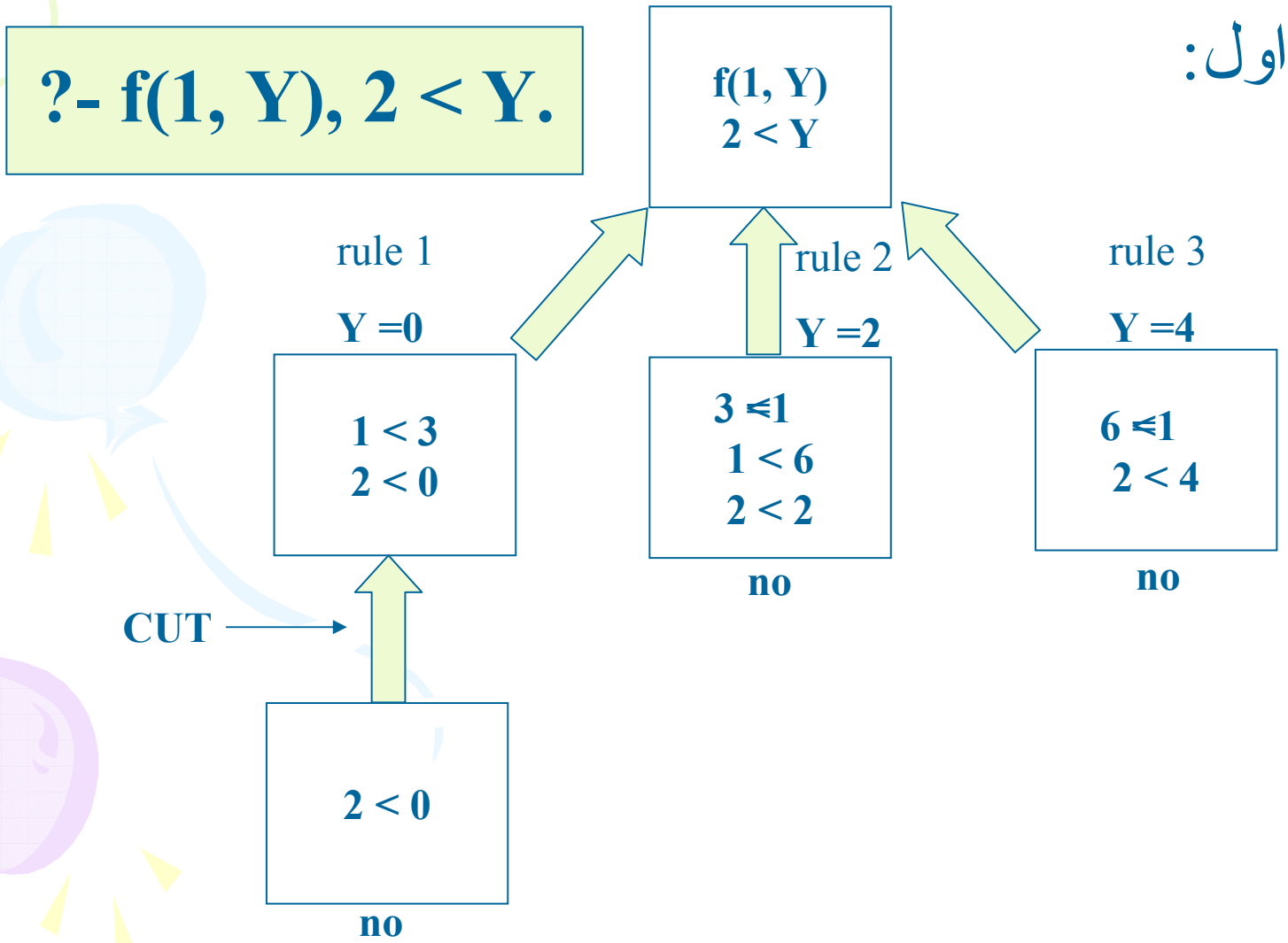
جلوگیری از عقبگرد

```
f( X, 0) :- X < 3.                % Rule1
f( X, 2) :- 3 =< X, X > 6.        % Rule2
f( X, 4) :- 6 =< X                % Rule3
```



جلوگیری از عقبگرد

- آزمایش اول:



جلوگیری از عقبگرد

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

عملگر ***CUT***

اگر همان پرسش قبل انجام شود فقط شاخه چپ شکل قبل بررسی می شود.

این برنامه از برنامه اولیه کارآتر می باشد و هر وقت پاسخ منفی باشد، این برنامه زودتر به آن می رسد.

در این مثال عملگر ! فقط بر معنای رویه ای (چگونگی اجرا) تاثیر دارد نه بر نتیجه برنامه.

جلوگیری از عقبگرد

- آزمایش دوم

?- f(7, Y)
Y = 4

آزمایش قانون ۱: 3 < 7 مردود می شود، عقبگرد و آزمایش قانون ۲ (اجرا به عملگر cut نمی رسد).

آزمایش قانون ۳: **7 =< 3** موفقیت آمیز می باشد، اما 6 < 7 مردود می شود، عقبگرد و آزمایش قانون ۳ (اجرا به عملگر cut نمی رسد).

آزمایش قانون ۳: **7 =< 6** موفقیت آمیز می باشد و Y = 4.

نکات:

3 < 7 درست نیست، پس دیگر نباید 7 =< 3 بررسی شود(نقیض)

6 < 7 درست نیست، پس نیازی به بررسی 7 =< 6 در قانون سوم نمی باشد، زیرا حتما این شرط درست می باشد.

جلوگیری از عقبگرد

- در اینجا مسأله به شکل بهتری بیان شده است:

if X < 3 then Y = 0,
otherwise if X < 6 then Y = 2,
otherwise Y= 4.

نسخه سوم برنامه:

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

این برنامه از هردو نسخه قبلی سریعتر اجرا می شود، اما اگر در آن عملگرهای cut حذف شوند برنامه چندین پاسخ تولید می کند که برخی از آنها نادرست می باشند(؟).

نسخه سوم در مقایسه با نسخه دوم، علاوه بر تأثیر روی معنای رویه ای بر پاسخ های تولید شده نیز تأثیر می گذارد.

جلوگیری از عقبگرد

- بررسی دقیق عملکرد cut:

فراکرد زیر را در نظر بگیرید:

H :- B1, B2, …, Bm, !, …, Bn.

فرض می کنیم که این فراکرد توسط هدف G که با H تطابق دارد احضار شود. به G**هدف پدر** می گوییم. لحظه ای که اجرا به cut می رسد، سیستم پاسخهایی را برای اهداف B1، B2، ... و Bm یافته است. پس از اجرای cut این پاسخها **منجمد** شده و پاسخهای دیگر برای اهداف قبل از cut دور ریخته می شوند. همچنین هدف G به این فراکرد **ملزم** شده و از هر تلاشی برای تطابق G با بخش سرآیند فراکردهای دیگر ممانعت می شود.

جلوگیری از عقبگرد

- بررسی دقیق عملکرد cut:

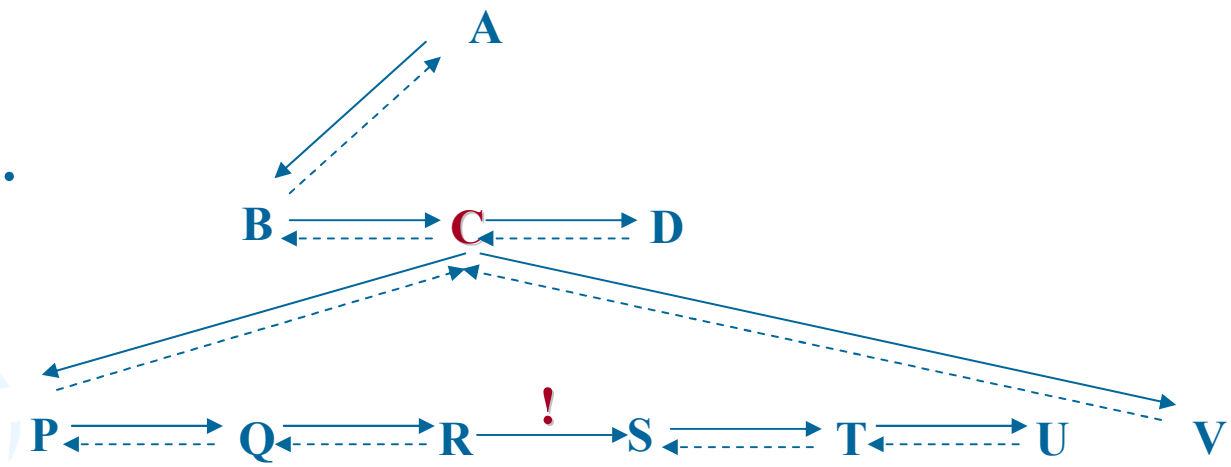
مثال زیر را در نظر بگیرید:

C :- P, Q, R, !, S, T, U.

C :- V.

A :- B, C, D.

?- A.



مثالهایی با استفاده از cut

- محاسبه ماکزیمم

```
max( X, Y, X) :- X >=Y, !.
max( X, Y, Y).
```

بررسی عضویت تک جوابی

```
member( X, [X| _] ) :- !.
member( X, [Y| T] :- member( X, T).

?- member( X, [a, b, c] ).
   X = a;
   no
```

مثالهایی با استفاده از cut

- اضافه کردن طک عنصر به لیست بدون تکرار

```
add( X, L, L) :- member( X, L), !.
add( X, L, [X| L] ).

?- add( a, [b, c], L ).
    L = [a, b, c]
?- add( X, [b, c], L).
    L= [b, c]
    X = b
?- add( a, [b, c, X], L).
    L = [b, c, a]
    X = a
```

منفی سازی با *fail*

- “ مریم تمام حیوانات را به جز مار دوست دارد”. چگونه می توان این جمله را در پرولوگ نمایش داد؟

```
likes( maryam, X) :-
    snake( X), !, fail.
likes( maryam, X) :-
    animal( X).
```

شکل فشرده تر:

```
likes( maryam, X) :-
    snake( X), !, fail
    ;
    animal( X).
```

منفی سازی با *fail*

- تعریف different(X, Y) :
  - اگر X و Y قابل تطابق نباشند رابطه بالا درست است.

```
different( X, X) :- !, fail.
different( X, Y).
```

شکل فشرده تر:

```
different( X, Y) :-
   X = Y, !, fail
   ;
   true
```

منفی سازی با *fail*

- تعریف not:

```
not( P) :-
    P, !, fail
    ;
    true.
```

توجه: not در برخی از پیاده سازیهای پرولوگ تعریف شده است.

```
likes( maryam, X) :-
    animal( X),
    not snake( X).
different( X, Y) :-
    not (X = Y).
```

مزایا و معایب cut

- مزایا
  - افزایش کارآیی برنامه
  - بیان قوانین انحصار متقابل (mutually exclusive) به شکل زیر:

if condition P *then* conclusion Q,
otherwise conclusion R

مزایا و معایب cut

- معایب:
  - تأثیر cut بر معنای توصیفی برنامه (نتایج)
- مثال:

```
P :- a, b.
P :- c.
```

معنای توصیفی: P درست است اگر a و b هر دو درست باشند و یا c درست باشد.
در اینجا ما می توانیم ترتیب فراکردها را تغییر دهیم بدون اینکه معنای توصیفی تغییر کند.

```
P :- a, !, b.
P:- c.
```

معنای توصیفی : P درست است اگر a و b هر دو درست باشند یا a درست نباشد و c درست باشد.

$(a \wedge b) \vee (\sim a \wedge c)$

اگر ترتیب فراکردها را عوض کنیم، معنای توصیفی نیز تغییر می کند:

```
P :- c.
P :- a, !, b.
```

معنای توصیفی:

$c \vee (a \wedge b)$

مزایا و معایب cut

- انواع cut:

green cut –

- بر معنای توصیفی تأثیر ندارد.
- خوانایی برنامه را کاهش نمی دهد.
- استفاده از آن نسبتا قابل قبول است.

red cut –

- بر معنای توصیفی تأثیر می گذارد.
- فهم برنامه را مشکل می کند.
- باید به دقت استفاده شود.