

関数・論理型プログラミング実験 第11回

江口 慎悟

押川 広樹

塚田 武志

講義のサポートページ

<http://www2.kb.is.s.u-tokyo.ac.jp/~tsukada/cgi-bin/m/>

- 講義資料等が用意される
- レポートの提出先
- 利用にはアカウントが必要
- 名前/学籍番号/希望アカウント名をメールを
tsukada@kb.is.s.u-tokyo.ac.jp
までメールしてください。
 - 件名は「FL/LP実験アカウント申請」
 - アカウント名/パスワードを返信
 - PCからのメールを受け取れるように

論理型プログラミング（全3回）

第10回 Prolog の使い方

- Prolog を使ってみよう

第11回 手続き的側面

- 評価メカニズム

第12回 論理的側面

- 完全性・健全性
- 否定と閉世界仮説

今日の内容

○ Prolog の動作原理

- SLD 導出
- Prolog の戦略
- カットと否定

Prolog の動作原理

SLD導出

Prolog の戦略

カットと否定

SLD導出

○ Prolog の動作原理

- S: Selective
- L: Linear
 - P_1, \dots, P_n を確認するのに、サブゴール P_i を「ひとつ」「選ぶ」
- D: Definite clauses
 - $Q \leftarrow R_1, \dots, R_m$ の形の節（要は rule と fact のこと）

例

```
male(kobo).  
male(koji).  
male(iwao).  
female(sanae).  
female(mine).  
parent(kobo, koji).  
parent(kobo, sanae).  
parent(sanae, iwao).  
parent(sanae, mine).  
  
father(X,Y) :- parent(X,Y), male(Y).  
mother(X,Y) :- parent(X,Y), female(Y).
```

動作例

```
?- father(kobo, koji).
```

Goal

成否を確認したい式

動作例

```
?- father(kobo, koji).
```

Goal

成否を確認したい式

```
father(X,Y) :- parent(X,Y), male(Y).
```

動作例

```
?- father(kobo, koji).
```

Goal

成否を確認したい式



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, koji), male(koji).
```

動作例

```
?- father(kobo, koji).
```

Goal

成否を確認したい式



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, koji), male(koji).
```

```
parent(kobo, koji).
```

動作例

?- father(kobo, koji).

Goal

成否を確認したい式



father(X,Y) :- parent(X,Y), male(Y).

?- parent(kobo, koji), male(koji).



parent(kobo, koji).

?- male(koji).

動作例

`?- father(kobo, koji).`

Goal

成否を確認したい式



`father(X,Y) :- parent(X,Y), male(Y).`

`?- parent(kobo, koji), male(koji).`



`parent(kobo, koji).`

`?- male(koji).`

`male(koji).`

動作例

?- father(kobo, koji).

Goal

成否を確認したい式



father(X,Y) :- parent(X,Y), male(Y).

?- parent(kobo, koji), male(koji).



parent(kobo, koji).

?- male(koji).



male(koji).

? -

動作例

?- father(kobo, koji).

Goal

成否を確認したい式



father(X,Y) :- parent(X,Y), male(Y).

?- parent(kobo, koji), male(koji).



parent(kobo, koji).

?- male(koji).



male(koji).

? -

成功

全てのゴールの成立を確認

動作例

```
?- father(kobo, Z).
```


動作例

```
?- father(kobo, Z).
```

```
father(X,Y) :- parent(X,Y), male(Y).
```

動作例

```
?- father(kobo, Z).
```



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, Z), male(Z).
```

動作例

```
?- father(kobo, Z).
```



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, Z), male(Z).
```

```
parent(kobo, koji).
```

動作例

```
?- father(kobo, Z).
```



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, Z), male(Z).
```



```
parent(kobo, koji).
```

```
?- male(koji). [Z ↦ koji]
```

動作例

```
?- father(kobo, Z).
```



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, Z), male(Z).
```



```
parent(kobo, koji).
```

```
?- male(koji). [Z ↦ koji]
```

```
male(koji).
```

動作例

```
?- father(kobo, Z).
```



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, Z), male(Z).
```



```
parent(kobo, koji).
```

```
?- male(koji). [Z ↦ koji]
```



```
male(koji).
```

```
? -
```

動作例

```
?- father(kobo, Z).
```



```
father(X,Y) :- parent(X,Y), male(Y).
```

```
?- parent(kobo, Z), male(Z).
```



```
parent(kobo, koji).      parent(kobo, sanae).
```

```
?- male(koji). [Z ↦ koji]
```



```
male(koji).
```

```
? -
```

動作例

?- father(kobo, Z).



father(X,Y) :- parent(X,Y), male(Y).

?- parent(kobo, Z), male(Z).



parent(kobo, koji). parent(kobo, sanae).

?- male(koji).

[Z \mapsto koji]

[Z \mapsto sanae]
?- male(sanae).



male(koji).

? -

動作例

?- father(kobo, Z).



father(X,Y) :- parent(X,Y), male(Y).

?- parent(kobo, Z), male(Z).



parent(kobo, koji).

parent(kobo, sanae).

?- male(koji).

[Z ↦ koji]

[Z ↦ sanae]

?- male(sanae).



male(koji).

? -



FAIL

動作例

```
? - mother(kobo, Z).
```

動作例

```
? - mother(kobo, Z).
```

```
mother(X,Y) :- parent(X,Y), female(Y).
```

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```

```
parent(kobo, koji).
```

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```



```
parent(kobo, koji).
```

$[Z \mapsto \text{koji}]$

```
?- female(koji).
```

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```



```
parent(kobo, koji).
```

$[Z \mapsto \text{koji}]$

```
?- female(koji).
```



FAIL

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```



```
parent(kobo, koji).      parent(kobo, sanae).
```

$[Z \mapsto \text{koji}]$

```
?- female(koji).
```



FAIL

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```



```
parent(kobo, koji).      parent(kobo, sanae).
```

$[Z \mapsto \text{koji}]$

$[Z \mapsto \text{sanae}]$

```
?- female(koji).
```

```
?- female(sanae).
```



FAIL

動作例

```
?- mother(kobo, Z).
```



```
mother(X,Y) :- parent(X,Y), female(Y).
```

```
?- parent(kobo, Z), female(Z).
```



```
parent(kobo, koji).
```

$[Z \mapsto \text{koji}]$

```
?- female(koji).
```



FAIL

```
parent(kobo, sanae).
```

$[Z \mapsto \text{sanae}]$

```
?- female(sanae).
```

```
female(sanae).
```

動作例

?- mother(kobo, Z).



mother(X,Y) :- parent(X,Y), female(Y).

?- parent(kobo, Z), female(Z).



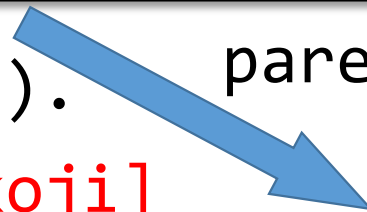
parent(kobo, koji).

[Z \mapsto koji]

?- female(koji).



FAIL



parent(kobo, sanae).

[Z \mapsto sanae]

?- female(sanae).



female(sanae).

?-

やっていること

- Goal の一部を rule や fact を用いて置き換え、Goal が空になったか確認
 - 置き換え時には $[Z \mapsto koji]$ などの代入が生じる
- 利用できるrule や fact が複数ある場合には、ひとつずつ試す
 - 例： `parent(kobo, koji).` と `parent(kobo, sanae).`

復習：単一化

(第六回を参照)

- 自由変数を含む項 s と t に対して
 $s \theta = t \theta$ となる代入 θ を求めること
 - $s \theta$: s 中の変数 x を $\theta(x)$ で置き換えたもの
 - 代入 θ は単一化子、単一化代入と呼ばれる
- 例：
 - X と koji について $[X \mapsto \text{koji}]$
 - $s(X)$ と Y について $[Y \mapsto s(X)]$
 - $s(X)$ と Y について $[X \mapsto s(z), Y \mapsto s(s(z))]$
 - $s(X)$ と z は単一化不能

復習：最汎単一化(MGU) (第六回を参照)

- 単一化子の中でもっとも一般的なもの
 - θ_1 が θ_2 より一般的
 \Leftrightarrow 代入 η が存在して $\forall X. \theta_2(X) = \eta(\theta_1(X))$
 - この関係は代入の上の前順序を成す
- 例： $s(X)$ と Y の最汎単一化子は $[Y \mapsto s(X)]$
- 最汎単一化子は存在すれば一意
 - ただし変数の名前代えるは同一視

SLD導出 in Prolog

○ 入力

- ゴール P_1, \dots, P_n .
- $Q :- Q_1, \dots, Q_m$ の形の rule のリスト
 - $m=0$ が fact に相当

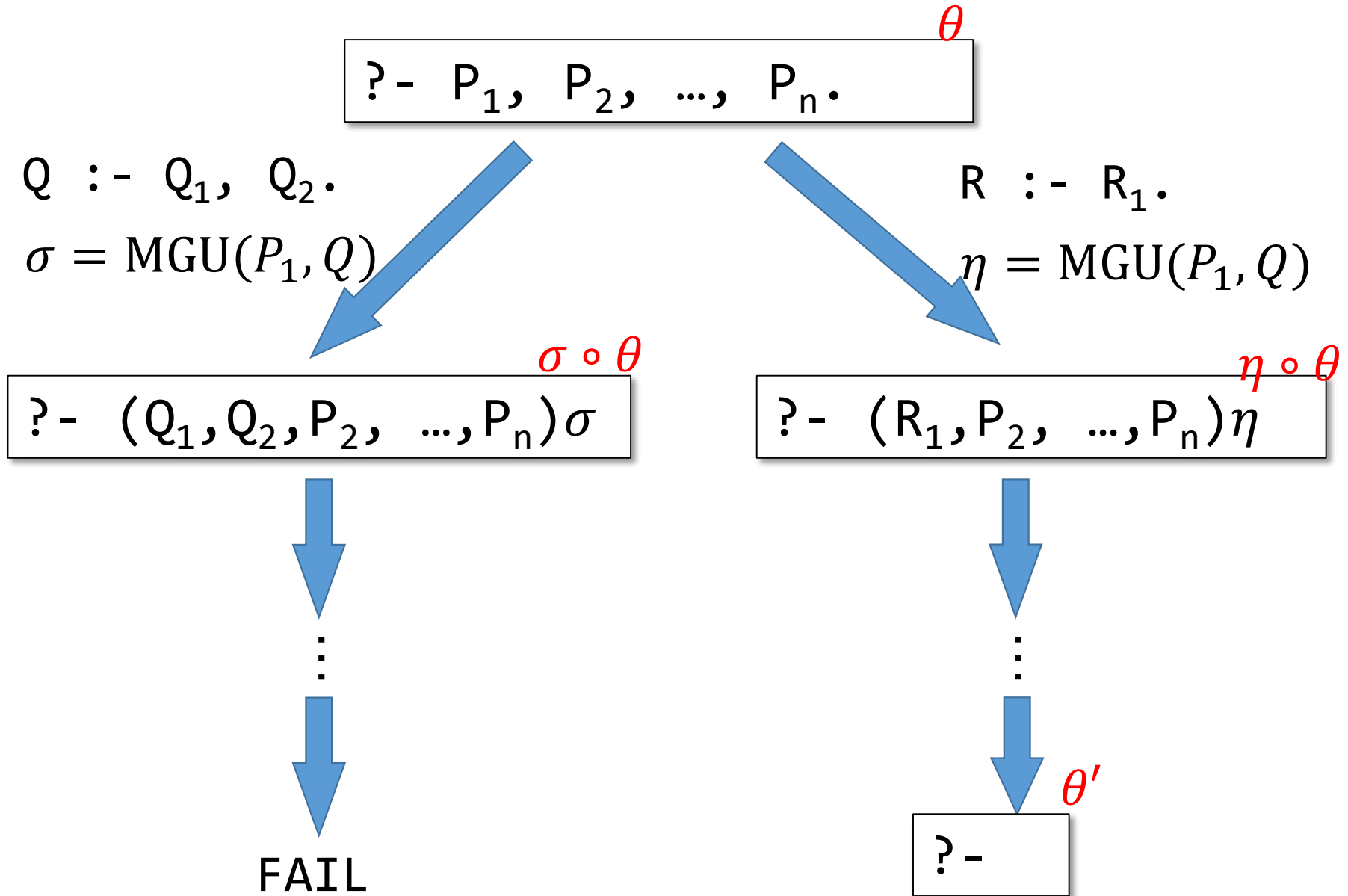
○ 出力

- 成否 + 成功する場合は変数への代入

SLD導出 in Prolog

- Goal P_1, \dots, P_n と代入 θ について
 - Goal が空なら成功。代入 θ を返す
 - そうでなければ、各ruleについて以下を行う
 - $Q :- Q_1, \dots, Q_m$ の結論 Q が P_1 と単一化可能か調べる。単一化不能ならば次のruleを試す。
 - 最汎単一化子を σ をとして、
新しいゴール $(Q_1, \dots, Q_m, P_2, \dots, P_n)\sigma$ と
新しい代入 $\sigma \circ \theta$
に再帰的にこの手続きを適用する
 - すべてのルールで失敗したら失敗

イメージ



例

?- father(kobo, Z).



father(X,Y) :- parent(X,Y), male(Y).

?- parent(kobo, Z), male(Z).



parent(kobo, koji).

[Z \mapsto koji]

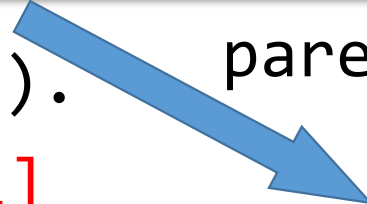
?- male(koji).



male(koji).

? -

[Z \mapsto koji]



parent(kobo, sanae).

[Z \mapsto sanae]

?- male(sanae).



FAIL

別の例

$\text{add}(z, Y, Y).$

$\text{add}(s(X), Y, s(Z)) \text{ :- } \text{add}(X, Y, Z).$

$?- \text{add}(s(s(z)), s(z), A).$



$\text{add}(s(X), Y, s(Z)) \text{ :- } \text{add}(X, Y, Z).$

$[A \mapsto s(B)]$

$?- \text{add}(s(z), s(z), B).$



$\text{add}(s(X), Y, s(Z)) \text{ :- } \text{add}(X, Y, Z).$

$[B \mapsto s(C), A \mapsto s(s(C))]$

$?- \text{add}(z, s(z), C).$



$\text{add}(z, Y, Y).$

$[C \mapsto s(z), B \mapsto s(s(z)), A \mapsto s(s(s(z)))]$

$?- [A \mapsto s(s(s(z)))]$

Prolog の動作原理

SLD導出

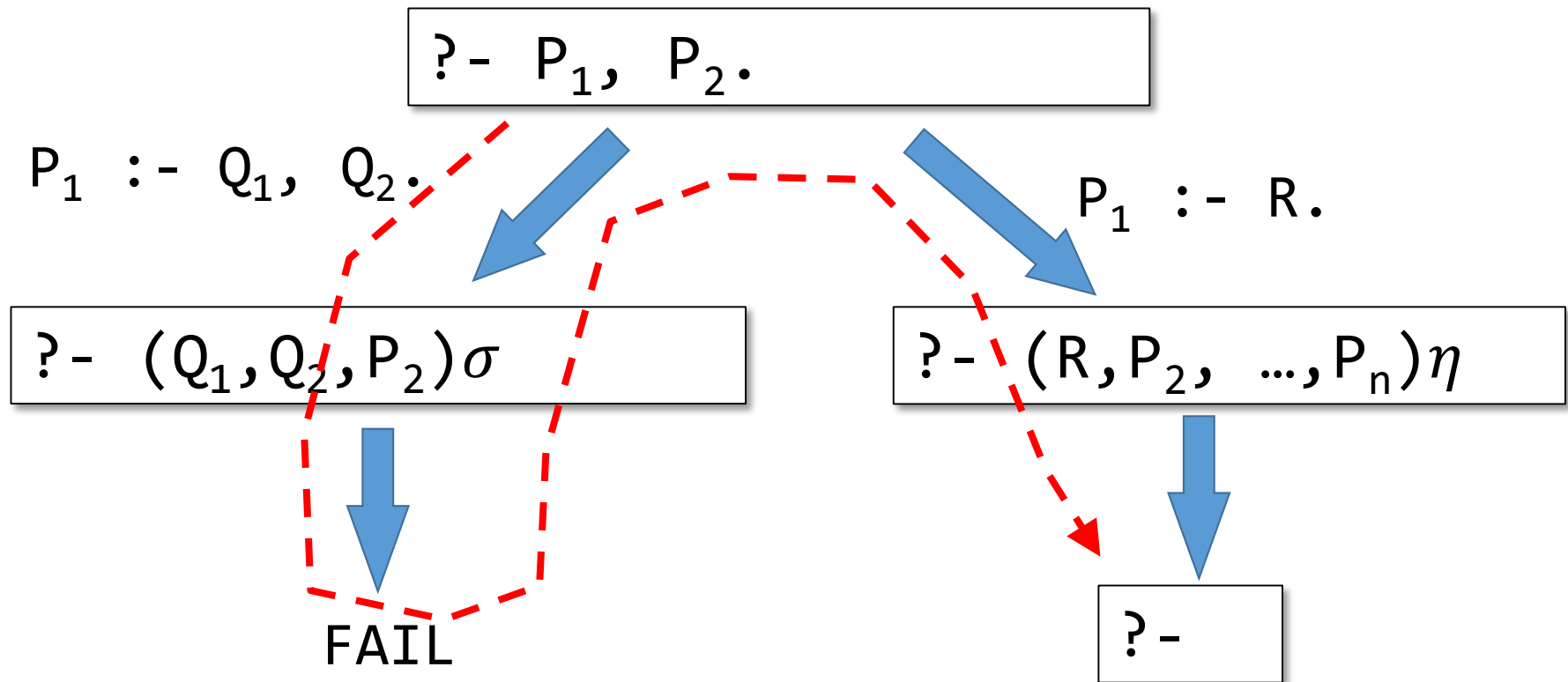
Prolog の戦略

カットと否定

Prolog の探索順

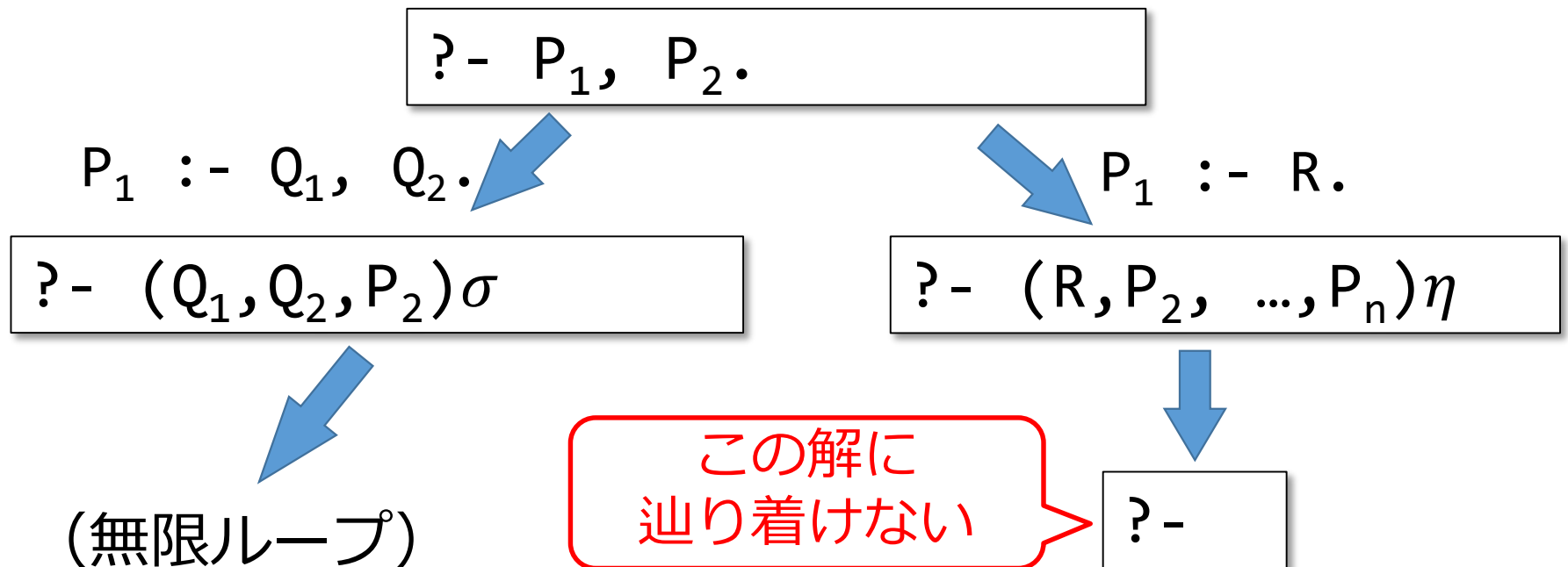
○ 深さ優先探索 (DFS)

- 上から下の順でルールを当てはめようとする
- ゴールは左から右の順でチェックする



深さ優先探索の特徴

- 利点：メモリ消費量が少ない
 - cf. 幅優先探索
- 欠点：解はあるのに見つけれられないことも



例

```
nat(s(X)) :- nat(X).  
nat(z).
```

?- nat(A).

nat(s(X)) :- nat(X).

$[A \mapsto s(B)]$

?- nat(B).

nat(s(X)) :- nat(X).

$[A \mapsto s(s(C))]$

?- nat(C).

(無限ループ)

nat(z).

?-

$[A \mapsto z]$

nat(z).

?-

$[A \mapsto s(z)]$

これらには
辿り着けない

Prolog の動作原理

SLD導出

Prolog の戦略

カットと否定

カット

○!

- 探索のバックトラックの制御に使う。
導出は必ず成功し、以降の導出に失敗した場合、
同じ結論の別の導出は試みない。

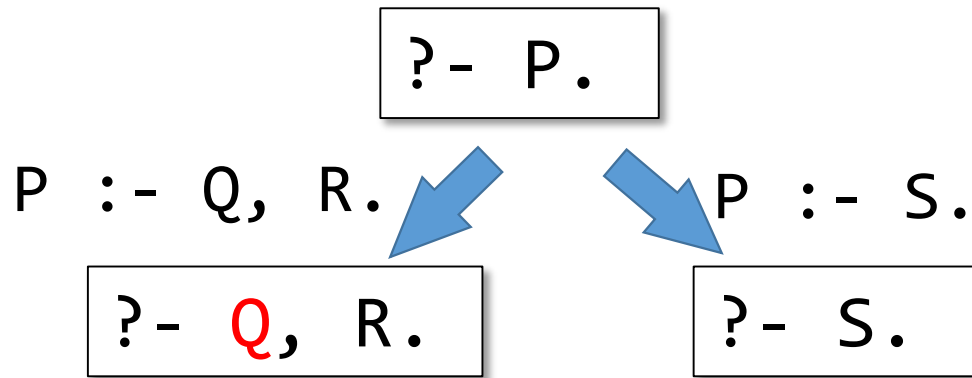
○ 例

$X > Y$ に成功したら例えば $Z \neq X$ でも下のルールは使わない

$\text{max}(X, Y, Z) :- X > Y, !, Z \text{ is } X.$

$\text{max}(X, Y, Z) :- Z \text{ is } Y.$

このルールが使われるということは $X > Y$ で失敗
 $X > Y$ ではないので、最大値は Y



```
P :- Q, R.  
P :- S.  
Q :- T, !, U.  
Q :- V.  
T.  
T :- W.
```

```
P :- Q, R.  
P :- S.  
Q :- T, !, U.  
Q :- V.  
T.  
T :- W.
```

?- P.

P :- Q, R.

P :- S.

?- Q, R.

?- S.

Q :- T, !, U.

?- T, !, U, R.

Q :- V.

?- V, R.

```
P :- Q, R.  
P :- S.  
Q :- T, !, U.  
Q :- V.  
T.  
T :- W.
```

?- P.

P :- Q, R.

P :- S.

?- Q, R.

?- S.

Q :- T, !, U.

?- T, !, U, R.

Q :- V.

?- V, R.

T.

?- !, U, R.

T :- W.

?- W, !, U, R.

```

P :- Q, R.
P :- S.
Q :- T, !, U.
Q :- V.
T.
T :- W.

```

?- P.

P :- Q, R.

P :- S.

?- Q, R.

?- S.

Q :- T, !, U.

Q :- V.

?- T, !, U, R.

?- V, R.

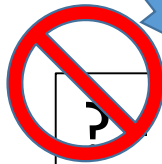
T.

T :- W.

?- !, U, R.

?- W, !, U, R.

?- U, R.



```
P :- Q, R.  
P :- S.  
Q :- T, !, U.  
Q :- V.  
T.  
T :- W.
```

?- P.

P :- Q, R.

P :- S.

?- Q, R.

?- S.

Q :- T, !, U.

?- T, !, U, R.

Q :- V.

?- V, R.

T.

?- !, U, R.

T :- W.

?- U, R.

?- W, !, U, R.

FAIL

```
P :- Q, R.  
P :- S.  
Q :- T, !, U.  
Q :- V.  
T.  
T :- W.
```

?- P.

P :- Q, R.

P :- S.

?- Q, R.

?- S.

Q :- T, !, U.

Q :- V.

?- T, !, U, R.

?- V, R.

T.

?- !, U, R.

T :- W.

?- U, R.

?- W, !, U, R.

FAIL

カット

○ 例

$X > Y$ に成功したら $Z \neq X$ でも下のルールは使わない

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

$X > Y$ ではないので、最大値は Y

```
?- max(10, 3, X).
```

```
X = 10.
```

```
?- max(3, 2, 2).
```

```
false.
```


カット

○ 例：カットがないと、、、

```
max(X,Y,Z) :- X>Y, Z is X.  
max(X,Y,Z) :- Z is Y.
```

```
?- max(10, 3, X).  
X = 10 ;  
X = 3.  
?- max(3, 2, 2).  
true.
```

否定

○ Prolog の否定は「失敗による否定」

- ゴール $\neg P$ を導出するには、まず P の導出を試みる
 - P の導出に成功した場合、 $\neg P$ の導出は失敗する
 - P の導出に失敗した場合、 $\neg P$ の導出は成功する

(この他に P の導出が無限ループになる場合がある)

否定

- Prolog プログラム中では、否定は \+ と書く

```
?- \+ female(koji).  
true.
```

```
?- \+ male(koji).  
false.
```

- カットを使うと否定は次のように定義できる

```
notMale(X) :- male(X), !, false.  
notMale(X).
```

注意

- 否定やカットは思わぬ振舞いをする事とも
⇒ 使用には細心の注意を
- 例：次の max の定義は正しくない。なぜか

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

例題

理解の確認をするための課題です

課題提出システム上での提出の必要はありません

例題を解きTAに見せることで出席とします

分からないことがあったら、積極的に質問しましょう

例題

- 以下のプログラムを書き変えて
?- nat(N). という問い合わせに、
すべての自然数を返すようにせよ

```
nat(s(X)) :- nat(X).  
nat(z).
```

■ 動作例

```
?- nat(N).  
N = z ;  
N = s(z) ;  
N = s(s(z))
```

レポート課題11

締切：2018/7/10 13:00(JST)

問 1

- 以下の述語が上手く動作しない理由を述べよ

```
ancestor(X,Y) :- ancestor(Z,Y), parent(X,Z).  
ancestor(X,Y) :- parent(X,Y).
```

- 例えば `ancestor(kobo, iwao)` という問合せはどうなるか

問 2

○ 次のプログラムについて、

`nat_list([z, s(z), z])` や

`nat_list([z, X])` は期待通り動作するが

`nat_list(X)` はそうでない。何故か

- 例えば `nat_list(X)`, $X=[s(Y)]$ は解を返さず無限ループする

```
nat(z).  
nat(s(N)) :- nat(N).  
nat_list([]).  
nat_list([N|X]) :- nat(N), nat_list(X).
```

問 3

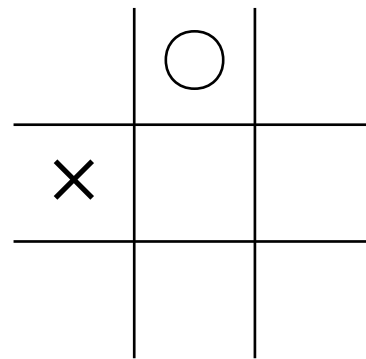
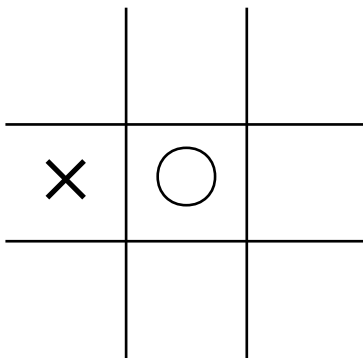
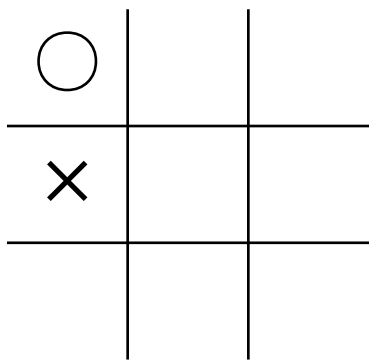
- tic-tac-toe (3×3 の $\bigcirc \times$ ゲーム) は
双方最善を尽くすと引き分けである。
このことを Prolog により確認せよ
- 例えば次の述語を定義する
 - win(P, B)
 - 盤面が B で P の手番なら、P の必勝
 - lose(P, B)
 - 盤面が B で P の手番なら、P の相手の必勝
- その上で、否定を使って、初期盤面が
どちらの必勝でもないことを問い合わせる

問 3 (つづき)

○ 同じプログラムを用いて

以下の局面が先手必勝であることを確認せよ

■ 先手：○、後手：×



ヒント

- $\text{win}(P, B)$ は、相手を Q とすると、
 - 自分のマークが既に三つ並んでいるか、
 - 相手のマークはまだ三つ並んでおらず、かつ、 B から一手進めたある局面 B' が存在して $\text{lose}(Q, B')$ が成立する
- $\text{lose}(P, B)$ は、相手を Q とすると、
 - 相手のマークが既に三つ並んでいるか、
 - 終局ではなくて、かつ、 B から一手進めた任意の局面 B' について $\text{win}(Q, B')$

下のルールを Prolog で書くには工夫が必要

発展 1

○ Prolog ライクな論理型言語を実装せよ

- 構文解析器を準備する必要はない
- カットや否定はなくてよい

せっかくなので swi-prolog とは違う挙動をさせよう！

- 幅優先探索をする
- 単一化で（ちゃんと）出現検査をする
- 否定を評価するのは変数がないときだけにする

※ 次回の講義も参考に