論理型のプログラミング言語: Prolog

前回課題の解説

論理型のプログラミング言語: Prolog

- **Pro**gramming in **Log**ic
- 論理型プログラミング言語
- 1970年代にフランスで生まれた
- 人間が導出反駁木により推論を行う過程をコンピュータ上で再現
- 昔の人工知能システムに広く使われていた

節

・P1,P2,…,Pn←Q1,Q2,…,Qmという形をしている述語論理

(Pi,Qjは原子述語：最も基本的な述語）

・「Q1かつQ2かつ…かつQmならば、P1またはP2または…またはPnである」という意味を表す

（ Q⇒P = ¬Q∨P のため）

＝（ ¬Q1∨¬Q2∨,…,∨¬Qm∨P1∨P2 ∨…∨Pn ）

・ P1,P2,…,Pnは結論部、Q1,Q2,…,Qmは条件部と呼ぶ

・論理結合記号も、限定記号（全称記号と存在記号）も使わない、非常に単純な形をしている

・どんな複雑な述語論理式も節を使って書くことができる

（証明は省く）

ホーン節による述語論理

・ホーン節：結論が高々一つの節

P←Q1,Q2,…,Qm	**規則節**：規則に関する知識 「Q1,Q2,…,Qmならば、Pである」 あるいは、「Pを証明するには、 Q1,Q2,…,Qmを証明すればよい 」
←Q1,Q2,…,Qm	質問節（条件部しかない） 「Q1,Q2,…,Qmですか？」
P←	**事実節**（結論部しかない） 「Pである」
←	空節：「false」、矛盾

Prologの基本的動作

・ホーン節の条件部を左から右へと、各原子述語の実行が成功するかを調べていく

・実行が成功するとは、その原子述語が

①直接評価され、その結果が真であった場合

②事実の結論部と単一化できた場合、

③規則の結論部と単一化でき、かつその条件部がすべて成功した場合

のいずれかである

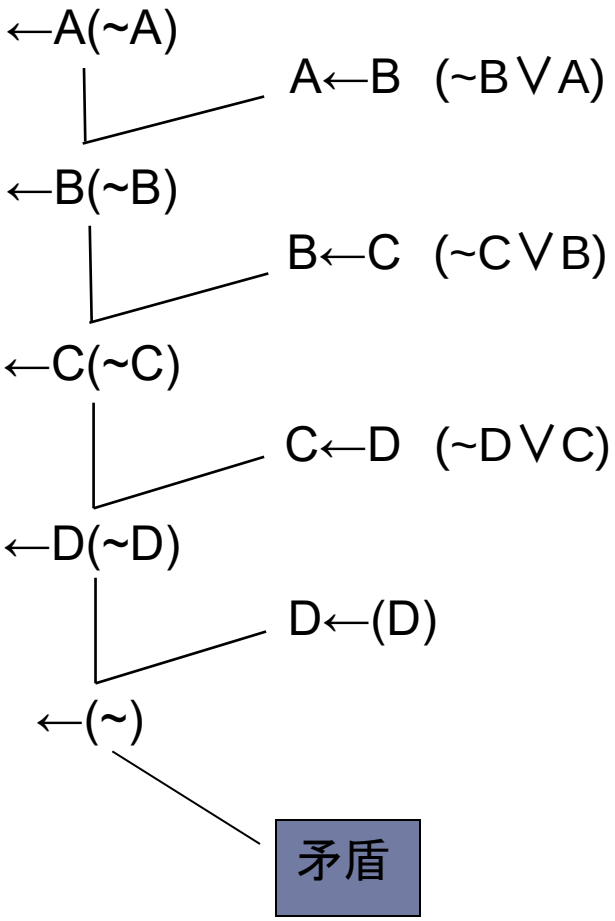
動作例

事実：D←, E←

規則：A←B, B←C,
C←D

質問：←A

矛盾が導出されれば、
答えが”YES”となる
（Aである）



バックトラック

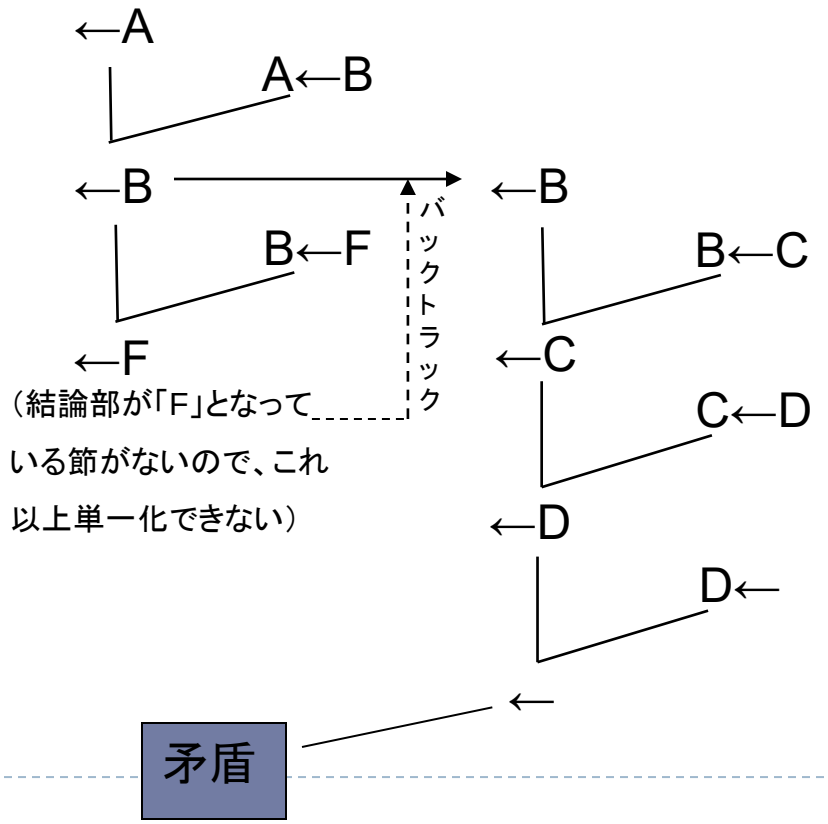
原子述語の実行が失敗した場合には、直前に行われた単一化までさかのぼり、別のホーン節を選択する動作

事実：D←, E←

規則：A←B, B←F,
B←C, C←D

質問：←A

答えが”YES”となる



SWI-Prolog

- 「SWI-Prolog」は「Prolog」の処理系である。
- オランダにあるアムステルダム大学で開発された
- オープンソースであるため、無料で利用できる

  http://www.swi-prolog.org/download/stable

  にある

  <SWI-Prolog 8.4.0-1 for Microsoft Windows (64 bit)>

  をインストール

  （32ビットWindowsやMacの場合はOSに応じて選択すること）

SWI-Prologプログラムの書き方

- 規則節： P←Q1,Q2 ➡ P:-Q1,Q2.
- 変数： 英大文字か“_”で始まる文字列
  （例： X, NAME, Name, N1, _name,…）
- 定数： 英大文字や“_”以外で始まる文字列
  （例： tom, 名前, 世田谷, …）

必ず“.”で節を終わらせる必要がある

※ 「%」で始まるものはコメントと見なす

プログラムの作成・実行手順

- 規則節と事実節をテキストファイルに書き込んで、ファイル名は任意で、拡張子として“.pl”で保存(例：sample.pl)
- SWI-Prologを起動（次々頁）
- Prologのプログラムを読み込む(['sample.pl'].)
- 質問節（ゴール節）を実行時に入力
  - 処理系が質問に対して推論を行い
  - 推論で得られた回答を表示
- 回答が複数あるとき“;”で別解を表示させる

SWI-Prologのインストール

- ダウンロードしたインストーラをダブルクリック
- いくつかの確認画面が出てくるが、デフォルトのままで問題ないので、「次へ」などのボタンを押してインストールを遂行

SWI-Prologの起動方法（Windowsの場合）

- 起動方法１:

  インストール時に選択した拡張子(.pl)を持つファイルをダブルクリックする(この場合には、同時にファイルの読み込みが行われる).

  ※ファイルのパスに日本語が含まれる場合，うまく起動できない場合がある。

- 起動方法２:

  Windowsの[スタート]→ [SWI-Prolog]→[Prolog]を選択する.

  その後は「file」メニューの「consult」でソースファイルを開く.

- 起動方法３:

  DOSプロンプトなどから，インストールパスにあるフォルダ「bin」内の「swipl-win.exe」を実行する.

  その後は「file」メニューの「consult」でソースファイルを開く.

SWI-Prologの基本コマンド

別解を求める	**;**
メモリ内のプログラム表示	**listing.**
改行	**nl.**
無限ループからの脱出	**Ctrl-c　//キーボード操作**
常に失敗する	**fail.**
常に成功する	**true.**
SWI-Prologを中断	halt.
デバッグモードに入る	trace.
GUIのデバッグモードに入る	gtrace.
デバッグモードから脱出	nodebug.

※**下線部**は常用コマンドを表す

SWI-Prologを動かしてみよう

【例1】規則節：人間なら、いつか死ぬ
事実節：太郎は人間である。
質問節：太郎は死ぬのか？

SWI-Prologを動かしてみよう

【例1】規則節：人間なら、いつか死ぬ
事実節：太郎は人間である。
質問節：太郎は死ぬのか？

```
%life.pl
die(X):-human(X).
human(taro).
```

ここの「?-」は処理系がユーザ入力を促す記号なので、プログラムの一部ではない

（質問節）

```
?- die(taro).
```

（処理系の出力）

```
true.
```

さらに、こんな質問もしてみよう
1．太郎は人間なのか？
2．次郎（jiro）は人間なのか？
3．次郎（jiro）は死ぬのか？

SWI-Prologを動かしてみよう

【例2】事実節1：太郎はサッカーが好きである。
事実節2：太郎は野球が好きである。
質問節：太郎は何が好きなのか？　（ヒント：「何」を変数で置き換える）

SWI-Prologを動かしてみよう

【例2】事実節1：太郎はサッカーが好きである。
事実節2：太郎は野球が好きである。
質問節：太郎は何が好きなのか？　（ヒント：「何」を変数で置き換える）

```
%hobby.pl
like(taro,soccer).
like(taro,baseball).
```

さらに、こんな質問もしてみよう
1．太郎は野球が好きなのか？
2．野球が好きな人は誰？
3．サッカーが好きな人は誰？
4．次郎（jiro）は何が好きなのか？
5．誰が何を好きなのか？（複数解）

（質問節）

```
?- like(taro,X).
```

ここの「?-」は処理系がユーザ入力を促す記号なので、プログラムの一部ではない

（処理系の出力）

```
X = soccer ;
X = baseball.
```

ここの「;」は別解を求めるよう、ユーザが入力したもの

デバッグ

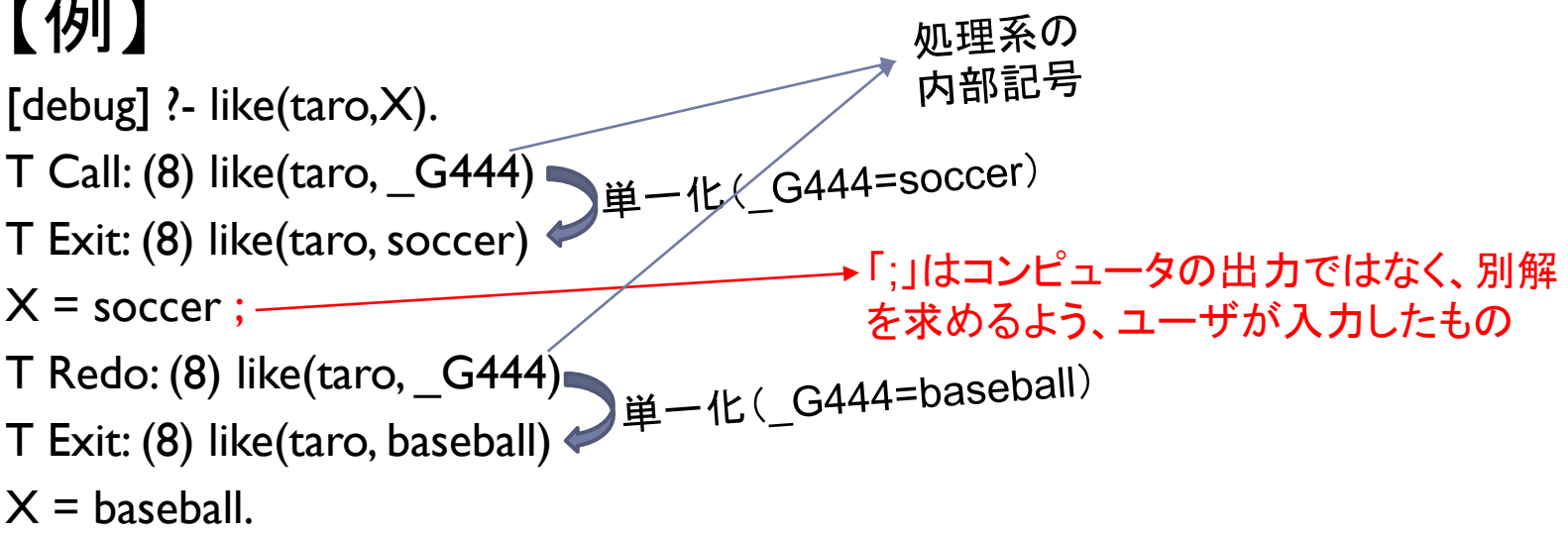
- デバッグモードに入る

  ?- trace.

  true.

  [trace] ?- 述語名(引数).

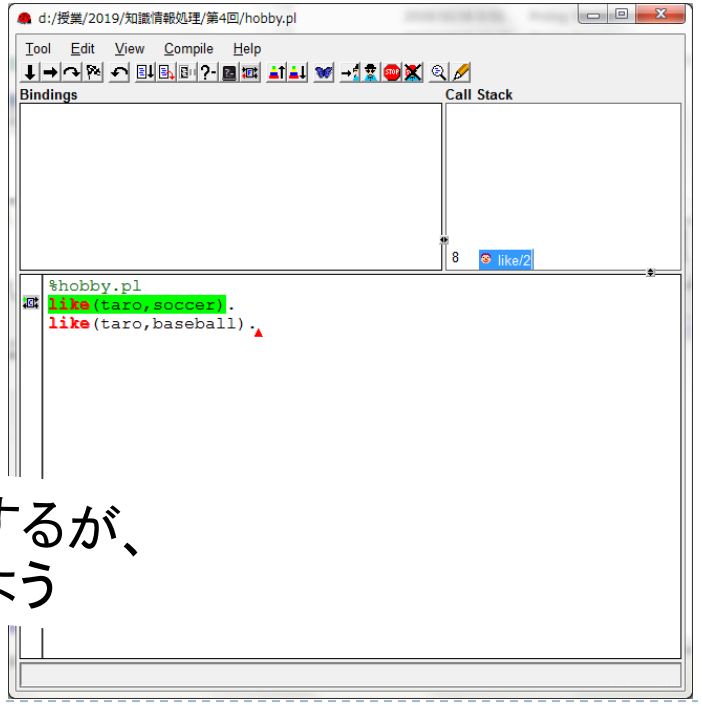
【例】



GUIのトレーサー

- GUIのデバッグモードに入る

```
?- gtrace.
% The graphical front-end will be used for
    subsequent tracing
true.
[trace]  ?- like(taro,X).
```



詳細については次回説明するが、
とりあえず触って遊んでみよう