2007年度 計算知能論A
知識表現と Prolog の基本

2007年4月16日

上田 和紀
早稲田大学理工学部CS学科

知識のいろいろ (6.1節)

- 記号的でない知識（暗黙知）
  - 例：自転車の乗り方，友達の顔
- 記号的な知識（形式知）
  - 構成的な知識
    - 手続き型 (procedural) ＝ “how” の表現
      - 例：逆行列を求めるプログラム
  - 構成的でない知識
    - 宣言型 (declarative) ＝ “what” の表現
      - 例：逆行列の数学的定義

コンピュータに知的能力をもたせる

- どちらが難しい？ $x^n - 1 = (x-1)(x^{n-1} + x^{n-2} + \ldots + 1)$

  Q1. 数式の展開 vs. 因数分解
  Q2. 式の微分 vs. 積分

  $\frac{df(x)}{dx}$ vs. $\int f(x)dx$

- しかしこれらはできるようになってきた
  - 積分をするプログラムは AI 草創期の主要成果
  - 将棋もプロ一歩手前（奨励会３段）
- 作るのがはるかに難しいのは
  - 子供の知的能力（例：コトバの学習）
  - 動物の知的能力（相手の認識，運動神経）

形式知
⇕
暗黙知

知的コンピュータの第一歩は記号処理

- コンピュータは数値計算の需要から生まれた
- しかしコンピュータが登場すると，その能力の人間との類似性に注目して新たな使い方を考える人々がすぐに現れた
  - ➔ 「記号」（シンボル）を扱う能力，具体的には
    - 記号や記号の連鎖をコンピュータの中に記憶したり，読み出したり，変形したりする能力
    - 二つの記号が等しいかどうかの判断能力

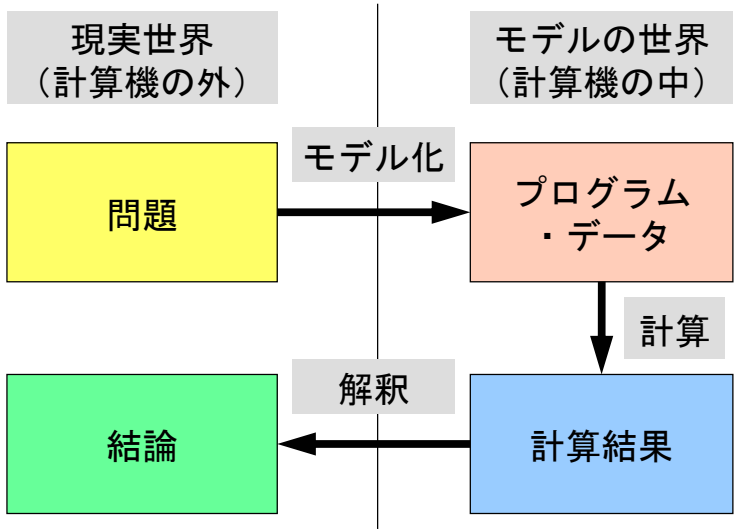
概念と記号 (6.2節)

- 宣言的な知識は，記号化することによって計算機に載せることができる．
- 記号 ＝ 概念につけた名前（cf. 単語）
  - 名前のつけ方には任意性がある
    - 例：12 vs. XII vs. twelve ...
  - 記号には３つのレベルがある
    1. 記号が表わす概念
    2. 記号自身
       - 動詞は名詞でないが「動詞」は名詞！
    3. 記号のつづり

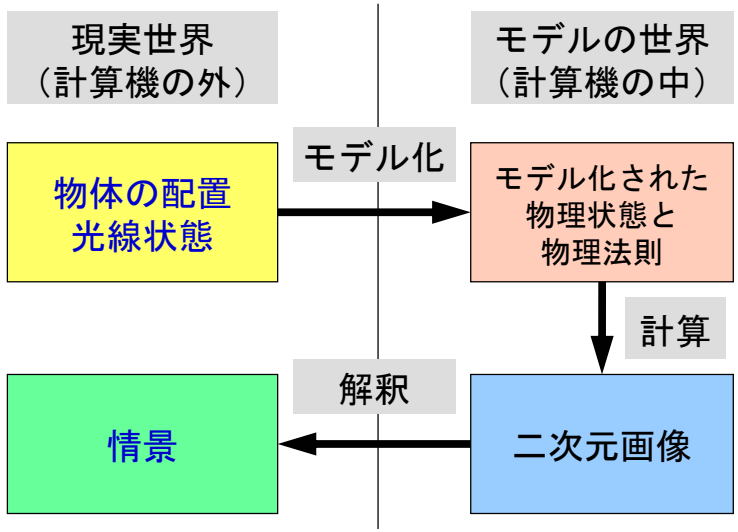
記号 ≒ 単語

- 記号とは，概念につけた名前のこと（cf. 単語）
  - 特徴 1: 名前のつけ方には任意性がある
    - 例：12 vs. XII vs. twelve ...
  - 特徴 2: 名前を分解すると意味をなさなくなる
    - 「りんご」 vs. “り” “ん” “ご”
- 記号を組み合わせることによって，より複雑な概念を表すことができる
  - 熟語 < 句 < 節 < 文 < 段落 < ...
- 単語どうしは相互に関係している
  - 辞書は単語のネットワーク

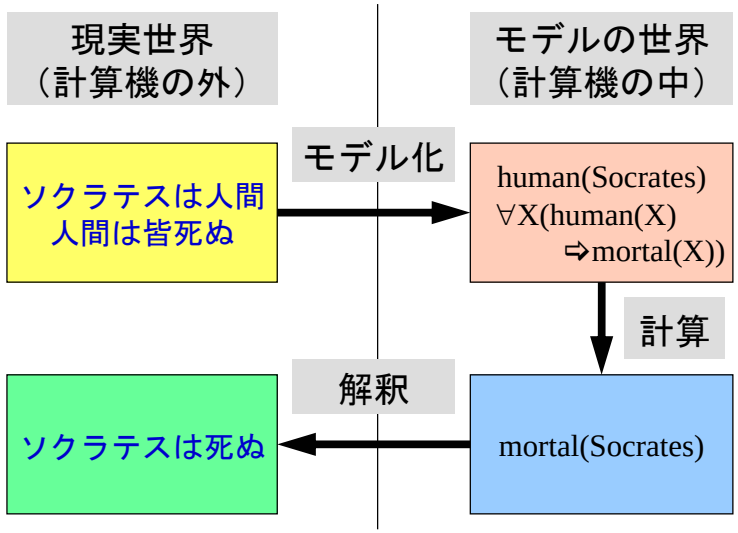
モデル化と解釈



モデル化と解釈の例：３次元ＣＧ



モデル化と解釈の例：三段論法



記号処理 (6.2 ～6.3節)

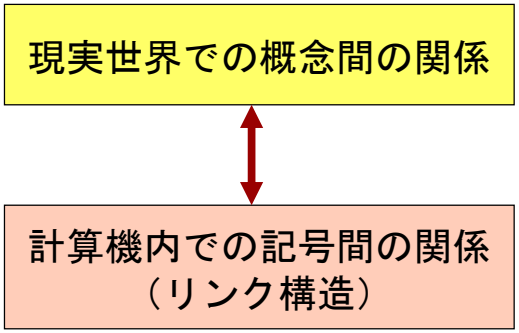
- 計算機で記号を扱うための基本機能は
  - 同一性の判定
  - 記号構造の形成と操作

  注：記号のつづりは入出力用，内部の処理では使わない
- 計算機内部に記号を格納するには？
  - 同一性の判定が高速にできればよい

  ➔ 外部から読み込んだ同一記号は常に同じ識別番号に変換されるようにする

記号による知識表現 (6.4節)

- 概念どうしを関係づけることが知識表現の基本 (cf. 辞書)



物理記号系 (Newell and Simon) (6.5節)

- A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure).
- Thus, a symbol structure is composed of a number of instances (or tokens) of symbols related in some physical way (such as one token being next to another).

物理記号系（つづき）

- At any instant of time the system will contain a collection of these symbol structures.
- Besides these structures, the system also contains a collections of processes that operate on expressions to produce other expressions: proesses of creation, modification, reproduction and destruction.

物理記号系（つづき）

- A physical symbol system is a machine that produces through time an evolving collection of symbol structures.
- Such a system exists in a world of objects wider than just these symbolic expressions themselves.

― Allen Newell and Hubert Simon: Computer Science as Empirical Inquiry: Symbols and Search (1976)

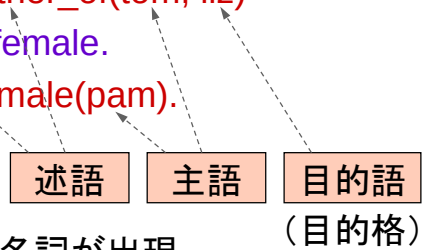
記号的知識は平叙文で書ける

- 個別知識（辞書にはあまり出ない）
  - Tom is the father of Liz.
  - Pam is female.
- 一般知識（辞書に出る）
  - A mother is a parent who is female.
  - A grandmother is a mother of a parent.
  - An ancestor is either (i) a parent or (ii) a parent of an ancestor.

➔ これらはすべて論理学のコトバでも表現できる

記号的知識をコンピュータに格納する

- 個別知識
  - Tom is the father of Liz.
    ➔ is_father_of(tom, liz)
  - Pam is female.
    ➔ is_female(pam).
  - 特徴
    - 固有名詞が出現
    - 単文



記号的知識をコンピュータに格納する

- 一般知識
  - A mother is a parent who is female.
    - X is the mother of Y **if** X is a parent of Y **and** X is female.

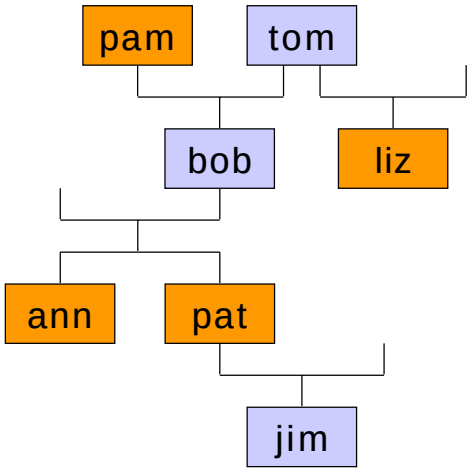
    ➔ is_mother_of(X,Y) ⇐
    is_parent_of(X,Y) ∧ is_female(X)
  - A grandmother is a mother of a parent.

    ➔ is_grandmother_of(X,Y) ⇐
    is_mother_of(X,Z) ∧ is_parent_of(Z,Y)

Prolog 入門 (7節)

- Prolog = PROgramming in LOGic
- 条件文の形をした論理式（$h \Leftarrow b_1 \wedge \ldots \wedge b_n$）（≒ 平叙文）を手続き（プログラム） として解釈
  - cf. C 言語や Java などは命令文を使う
- 30余年の歴史，多くのすぐれた処理系
  - 無料の処理系（Unix, Windows）は SWI-Prolog, B-Prolog, XSB, GNU Prolog など多数
  - 理工UNIXシステムでは SICStus Prolog

家族関係 (1) — 系図の登録 (family.pl)




```
is_parent_of(pam, bob).
is_parent_of(tom, bob).
is_parent_of(tom, liz).
is_parent_of(bob, ann).
is_parent_of(bob, pat).
is_parent_of(pat, jim).

female(pam).
female(liz).
female(pat).
female(ann).
male(tom).
male(bob).
male(jim).
```

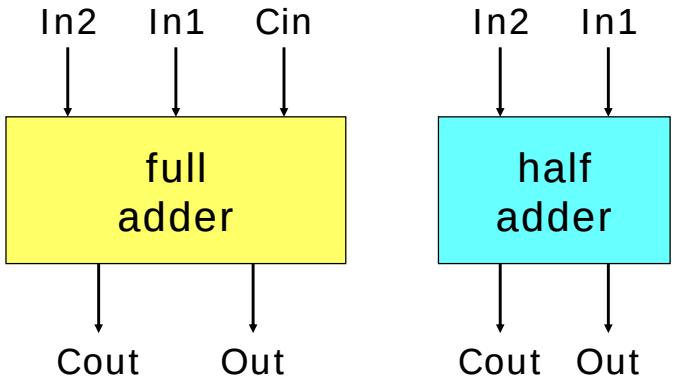

家族関係 (2) — 用語の意味の定義

```
is_mother_of(X,Y) :-
      is_parent_of(X,Y), female(X).
is_sister_of(X,Y) :-
      is_parent_of(Z,X),
      is_parent_of(Z,Y),
      female(X), X ≠ Y.

has_a_child(X) :- is_parent_of(X,_).

is_ancestor_of(X,Y) :- is_parent_of(X,Y).
is_ancestor_of(X,Y) :-
      is_parent_of(X,Z),
      is_ancestor_of(Z,Y).
```

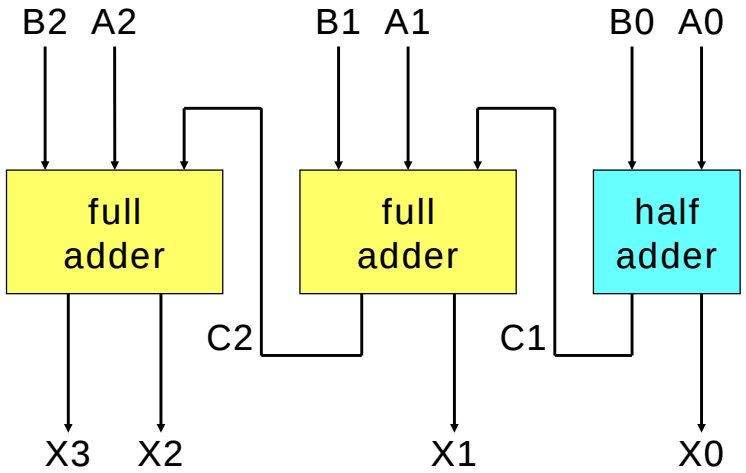
【補足】トレース機能 (8節)

- ?- trace.　　トレース実行開始
- ?- notrace.　　トレース実行終了
- Call, Exit, Redo, Fail
  - Call: 最初に呼ばれた
  - Exit: 答が求まった
  - Redo: 答の見直しを求められた
  - Fail: もう答がない

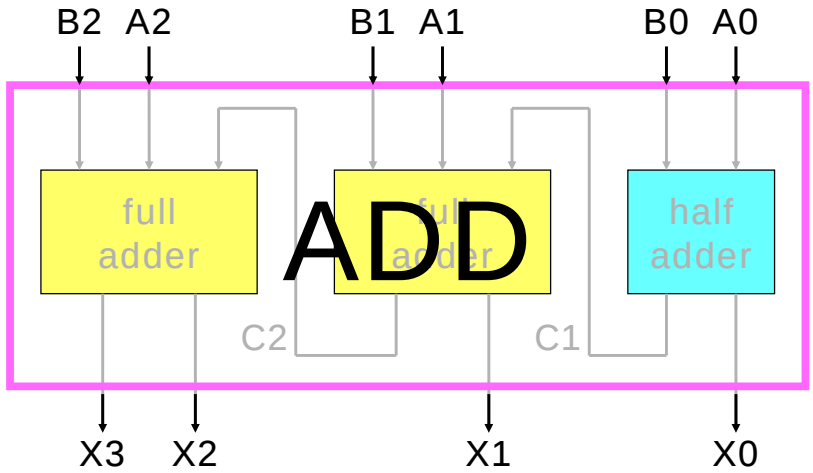
半加算器と全加算器 (adder.pl) (9節)



3ビット加算器 (adder.pl) (9節)



3ビット加算器 (adder.pl) (9節)



Prologと一階述語論理で扱う記号 (10節)

(1) 述語（述語記号）
- 「もの」のカテゴリへの所属　student(bob)
- 「もの」の性質　tall(bob)
- 「もの」の動作　runs(bob)
- 「もの」と「もの」との関係　loves(bob,liz)

◆ 「もの」を引数として与えると原子論理式 (atomic formula) となる.

◆ 原子論理式は自然言語の単文に相当し，真偽を論ずる対象となる

◆ 述語は他の言語の「手続き」に対応する

Prologと一階述語論理で扱う記号 (10節)

(2) 定数（定数記号）　bob, 0, june, nil
- きまった「もの」を表わす

(3) 変数（変数記号）　X, Y, Who
- いろいろな「もの」を表わす

(4) 関数（関数記号）　s(X), date(dec,31)
- 「もの」から「もの」を作る

◆ (2)〜(4) を用いて「もの」や「データ」を表わす項 (term)を構成する

◆ “一階”とは，変数には「もの」を表す変数（一階の変数）しかないことを表す.

記号 vs. 記号の意味 (10節)

◆ 記号は何らかの意味を担っている

◆ 記号の意味とは
- （計算機の中の）記号と
- （計算機の外の）「もの」「概念」

との対応関係のこと

◆ 知識を計算機に格納するときは，記号の意味はプログラマが決める（cf. 自然言語による表現）

◆ 論理式が言及している（外の世界の）「もの」の集合のことを，対象領域 (domain, universe) と呼ぶ.

記号による知識表現の例 (10節)

◆ 生死を扱うとき（対象領域：人間の集合）
- 定数：　Socrates
- 述語：　human, mortal

◆ 自然数を扱うとき（対象領域：**N**）
- 定数：　0, 1, 2, . . .
- 関数：　+, !, . . .
- 述語：　=, >, . . .

関数記号の役割 (10節)

- 単純な記号を組み合わせてより複雑な「もの」を表現
  - date(27,may,1994)
  - c(1,2)
  - s(s(s(s(s(0)))))
  - 1+2*3　または　+(1,*(2,3))
  - rectangle(point(X0,Y0),point(X1,Y1))
- “関数”と考えるよりも“データ構造”と考えた方が自然な場合もある
- 関数も述語も，引数の個数 (arity) は固定
- 定数は，0 引数の関数と考えても良い

手続き的解釈と宣言的解釈 (11節)

- Prolog のプログラムは，宣言型の知識としても手続き型の知識としても解釈できる
- 手続き的解釈 ～ プログラムを “how” の表現として解釈すること
  - では Prolog 処理系はどのように質問に答えているのか？
- 宣言的解釈 ～ プログラムを “what” の表現として解釈すること
  - では Prolog 処理系が求めた答はどのような意味で正しいのか？

Prologプログラムの宣言的解釈 (11.1節)

- プログラムは規則の（ANDでつながった）並び
- 規則 (rule)　ー 専門用語では「節」(clause)
  - 条件文に対応
  - 条件部のないものは事実 (fact)（肯定平叙文）
- 質問 (query) も「節」
  - 否定疑問文に対応
- Prologにないもの：
  - 否定平叙文　(cf. ‘dif ’)
  - OR でつながった複文

Prologプログラムの宣言的解釈 (11.1節)

- “P implies Q” は “(not P) or Q” と同じ
- “(P and Q and R) implies S” は “(not P) or (not Q) or (not R) or S” と同じ
- つまり，節とは，原子論理式やその否定を or でつなげたもの．ここで
  - 条件部の原子論理式には not をつける
  - 結論には not をつけない
  - 規則は not がつかない原子論理式を１個もつ
  - 質問はすべての原子論理式に not がつく

?- is_ancestor_of(X, jim) の解釈 (11.1節)

- 解釈１：否定（疑問）文

  false :- is_ancestor_of(X, jim)

  = For all X, X is not an ancestor of jim.
  = Is there no ancestor of jim?
  - Yes! Pat is an ancestor!
  - Yes! Ann is an ancestor!
- 否定されると，背理法による証明手続きが起動

?- is_ancestor_of(X, jim) の解釈 (11.1節)

- 解釈２：条件文の略記

  yes(X) :- is_ancestor_of(X, jim)

  = If X is an ancestor of jim, say “yes” and answer X.
  - Yes, Pat is an answer.
  - Yes, Ann is an answer.
- yes と答えようとして証明手続きが起動
- 「P ならば yes」と「not P」は非常に近い

変数の役割 (11.1節)

- 規則 C の中の変数は「すべての ～ について C が成立」と解釈
- しかし，
  - 質問の中の変数は，直感的には「～ が存在するか？」
  - 規則の条件部だけに出てくる変数は，直感的には「～ が存在すれば」
    - g_father(X,Y) :- father(X,Z), father(Z,Y).
- 解釈の反転は，ド・モルガンの法則のしわざ

手続き的解釈 ＝ 三段論法による演繹 (11.2節)

- 前向き (bottom-up)

$$\frac{\Rightarrow P \quad P \Rightarrow Q}{\Rightarrow Q}$$

- 後向き (top-down)

$$\frac{Q \Rightarrow \quad P \Rightarrow Q}{P \Rightarrow}$$

(Prologの基本メカ)

- どちらも下記のルールの特殊な場合にすぎない

$$\frac{A \Rightarrow B \quad B \Rightarrow C}{A \Rightarrow C}$$

Prologの実行メカニズムの基本 (11.2節)

- 知識

  P :- Q,R. (= P if Q and R)

  P :- S,T. (= P if S and T)

  は

  P if ((Q and R) or (S and T))

  と等価．つまり P が成り立つことを示すには
  - Q と R が両方成り立つ
  - S と T が両方成り立つ

  のいずれかを示せばよい

Prologの実行メカニズムの基本 (11.2節)

- したがって，知識

  P :- Q, R.

  P :- S, T.

  の下で P が成り立つことを示すには
  - 目標 P を（副）目標 Q, R に展開し，これらが成り立つか（左から）調べる
  - 途中で行き詰まったら目標 S, T に展開しなおし，それらが成り立つか（左から）調べる

  とすればよい（バックトラック：次頁）．

バックトラック (11.2節)

- 通常の（決定的）アルゴリズムは，次にすべきことが一通りにきまっている
- 非決定的 (nondeterministic) アルゴリズムは，次にすべきことが複数通りある
  - 可能性が枝分かれしてゆく
  - 現実の（決定的な）計算機で非決定的アルゴリズムをシミュレートするために，バックトラック (backtracking) を使う（試行錯誤）
  - cf. 非決定性オートマトン

バックトラック (11.2節)

- 宣言的解釈では，規則の順序や，一つの規則の中の条件の順序は問題にならない．
- しかし手続き的解釈ではこれらが問題になる．
  - 上に書いた規則から順に試す
  - 左に書いた条件から順に確かめる
- 一般には，
  - 単純な規則を先に書く
  - 一つの規則の中では，解の少ない（または単純な）条件を先に書く

  のがよい．

展開と単一化 (12節)

- これまでの説明は述語の引数には言及していなかった
- 変数を含む節を展開するときは、キャンセルするものどうしをまず「同じ形」にそろえる
- 例：

$$\frac{\text{human(socrates)} \quad \forall X(\text{human}(X) \Rightarrow \text{mortal}(X))}{\text{mortal(socrates)}}$$

では、一般的な知識

∀X(human(X)⇒mortal(X)) をまず具体化する

単一化 (12節)

- 形を揃える操作 ＝ 単一化 (unification)

例：human(socrates) と
human(X) ⇒ mortal(X) の左辺は、
X ← socrates と化ければ同じ形になる

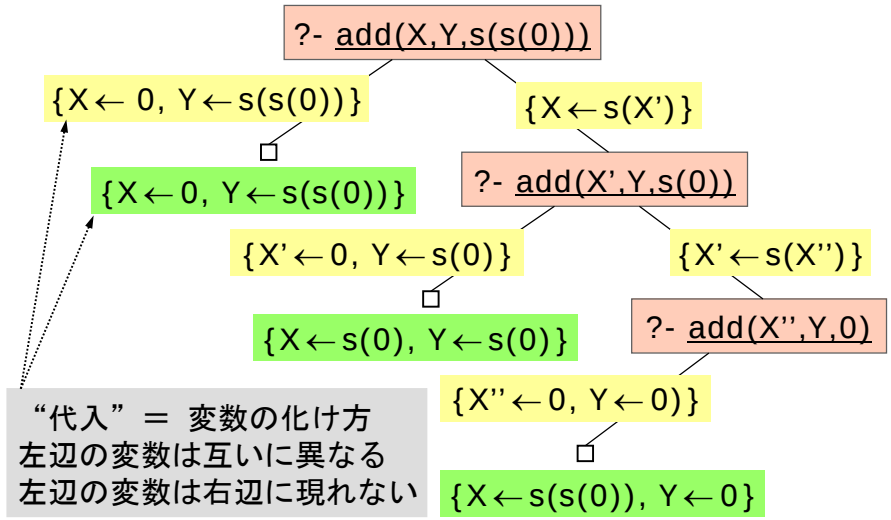
- Prolog は、プログラミング言語に必要な
  - 代入機構
  - 同一性の比較機構
  - 引数の受渡し機構

をすべて単一化によって実現している

展開の詳細 (12節)

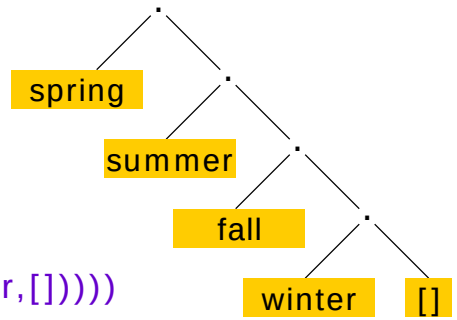
- 後向き三段論法 (top-down) の１ステップ
  - 他の言語における手続き呼出しに相当
- 呼出し add(X,Y,s(s(0))) を
  規則 add(s(X),Y,s(Z)) :- add(X,Y,Z) で展開する：
  1. 規則の中の変数をつけかえる．結果を
     add(s(X'),Y',s(Z')) :- add(X',Y',Z') とする
  2. 規則の頭部と呼出しとを単一化する．結果は
     {X ← s(X'), Y' ← Y, Z' ← s(0)} と書ける
  3. 呼出しを，代入を施した後の規則の本体で置き換える．結果は add(X',Y,s(0)) となる

SLD木 ＝ 実行の木 (12節)



リスト (13節)

- 0 個以上の要素を “,” で区切って並べたもの
  - [spring, summer, fall, winter]
  - [name([r,a,kowalski]),age(65)]
  - [singleton]
  - []
- 実は二分木の特別な記法にすぎない
  - .(spring, .(summer, .(fall, .(winter,[]))))



リスト (13節)

- ドット対 .(X,Y) を [X | Y] とも書く.
- 以下の三者は同一のもの
  - .(spring, .(summer, .(fall, .(winter,[]))))
  - [spring | [summer | [fall | [winter | []]]]]
  - [spring, summer, fall, winter]
- 略記のルール： “| [” と対応する “]” を省略して, “| [” のかわりに “,” を書く. ただし “,” の右に要素がなければ何も書かない.
- “|” と “,” の混在も可.
  - 例： [2,3,5,7,11|X]

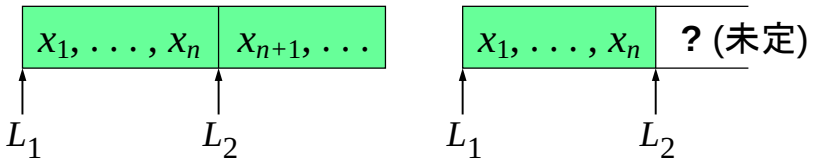
リスト (13節)

- ここで典型的なリスト処理プログラムのデモ
  - プログラムの間の「形の類似性」に注目！
    - 自然数の加算プログラム vs. リストの連結プログラム
    - リストの長さを求めるプログラム vs. リスト反転プログラム（naive 版）

差分リスト (difference lists) (13.2節)

- 論理変数を利用した強力なプログラミング技法
- リスト L_1 から要素 $x_1, \ldots, x_n$ $(n \geq 0)$ を除去するとリスト L_2 が得られるとき, L_1 と L_2 の対を差分リストという

  cf. time *vs.* duration, position *vs.* displacement



差分リスト (13.2節)

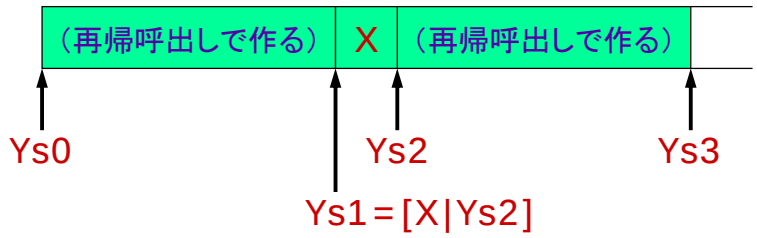
- 差分リスト L_1-L_2 と L_3-L_4 は，L_2 が不定ならば定数時間で連結できる
- 例：
  L_1 = [the,course | L_2] ,
  L_3 = [on,computational,intelligence | L_4] のとき，
  L_2 と L_3 の単一化（L_2 = L_3 の実行）によって
  L_1 = [the,course,on,computational,intelligence | L_4]
  となる
  - cf. append によるリスト連結

Quicksort (13.2節)

```
qsort(Xs,Ys) :- qsort(Xs,Ys,[]).
qsort([],      Ys,Ys).
qsort([X|Xs],Ys0,Ys3) :-
    part(X,Xs,S,L), qsort(S,Ys0,Ys1),
    Ys1=[X|Ys2], qsort(L,Ys2,Ys3).

part(_,[],      [],  []).
part(A,[X|Xs],[X|S],L) :- A>=X, part(A,Xs,S,L).
part(A,[X|Xs],S,[X|L]) :- A <  X, part(A,Xs,S,L).
```

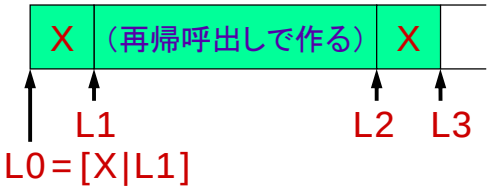

Quicksort (13.2節)

```
qsort(Xs,Ys) :- qsort(Xs,Ys,[]).
qsort([],      Ys,Ys).
qsort([X|Xs],Ys0,Ys3) :-
    part(X,Xs,S,L), qsort(S,Ys0,Ys1),
    Ys1=[X|Ys2], qsort(L,Ys2,Ys3).
```



Palindrome (13.2節)

```
element(a).  element(b).  element(c).
palindrome(L) :- palin(L,[]).
palin(L,      L).
palin([X|L],L) :- element(X).
palin(L0,L3) :-
      L0=[X|L1], palin(L1,L2), L2=[X|L3],
      element(X).
```



後に決めた部分（両端）が先に見直しの対象になる

差分リスト (13.2節)

- 差分リスト技法を使うと，リストの各部分を任意の順序で作成できる
  - あとで他のリストと連結する予定のあるリストは，差分リストとして作っておくのがよい
  - 応用例：自然言語処理
- 差分リストをリストに変換する手間は O(1)．その逆は append と同じ手間がかかる

経路探索 (14.1節)

- 末尾再帰 ＝ 深さ優先探索（縦型探索）

```
path(D,Visited,D,[D|Visited]).
path(S0,Visited,D,PathR) :-
   reach(S0,S), not_in(S,Visited),
   path(S,[S0|Visited],D,PathR).
```

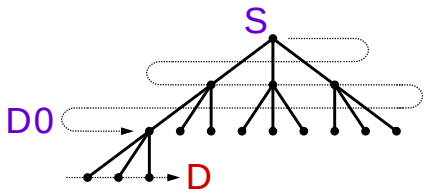
- 左再帰 ＝ 反復深化探索

```
path(D,D,[D]).
path(S,D,[D|PathR]) :-
   path(S,D0,PathR), reach(D0,D).
```

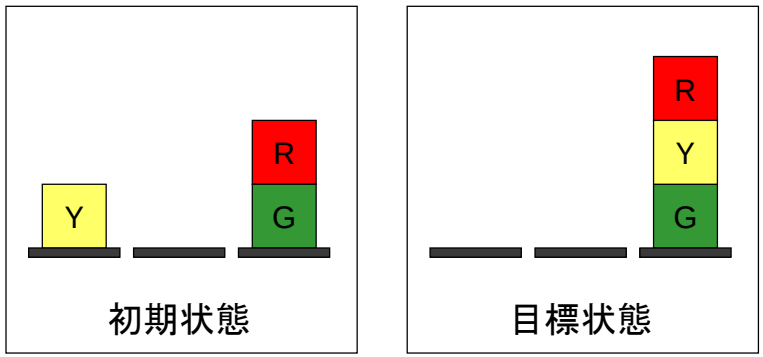

経路探索 (14.1節)

- 左再帰 ＝ 反復深化探索

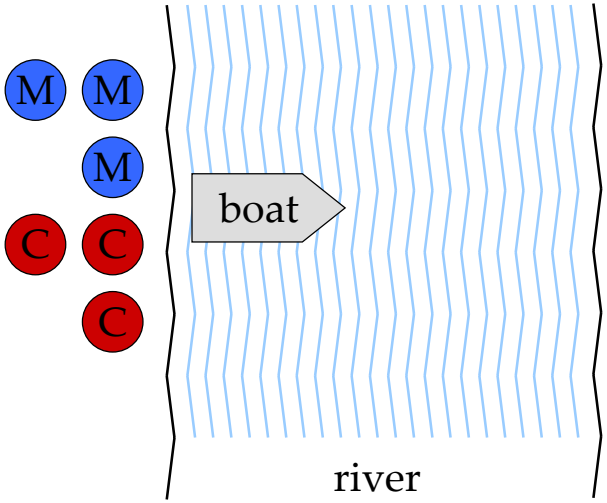
```
path(D,D,[D]).
path(S,D,[S|PathR]) :-
    path(S,D0,Path), reach(D0,D).
```



計画を立てる



計画を立てる



英文の構造を解析する

"The student passed the practical course with a computer."

- 文法：
  - 文 ::= 名詞句，動詞句.
  - 名詞句 ::= 限定詞，修飾語，名詞，前置詞句.
  - 動詞句 ::= 動詞，［名詞句］，前置詞句.
  - 前置詞句 ::= ［前置詞，名詞句］.
  - 修飾語 ::= ［形容詞，修飾語］.
  - 限定詞 ::= ［ a | the ］
  - . . .

レポート課題2

- 0 と s を使った自然数の処理に関連する探索問題（バックトラックによってさまざまな解を探し求める問題）を自分で設定して，プログラムを作る．問題設定，考え方，作成したプログラム，実行結果，考察・感想を記す．
- 期限：6月2日（土）24:00
- **report**@ueda.info.waseda.ac.jp 宛てにメールで提出（受領確認メールが返ります）
- Prolog 関係の自由課題は随時送ってください．問21～問32 を解くなど．

課題2：ウォーミングアップ

1. 自然数 0, s(0), s(s(0)), ... を順に列挙するプログラム nat を書いてみる
2. プログラム add による自然数の分割を手続き的に理解する
3. add の変種として，与えられた正整数を二つの正整数に分割するプログラムを書いてみる
4. 問16（与えられた自然数を三つの数の和に分割）をやってみる
5. 問20（二数の足し算のすべての場合を順に列挙）をやってみる

課題2：問題例

1. 三つの自然数の足し算のすべての場合を列挙する
   - つまり $x + y + z = u$ を満たすような x, y, z, u の組を $u = 0, 1, 2, \ldots$ について列挙する
     - cf. 問16
2. 与えられた正整数を1個以上の正整数の和に分解するすべての方法を列挙する
3. 二つの正整数の掛け算のすべての場合を列挙する

課題2：提出時の注意

- plain text で送ってください.
- メール本文の最初に学籍番号と氏名を記入してください.
- レポートを添付するのでなく，メール本文そのものをレポートとしてください.
- 質問は **ueda**@ueda.info.waseda.ac.jp に送って下さい.

課題2：補足

- Prolog 処理系は，大きなデータ構造の一部を “. . .” で省略して表示することがある.
- SWI-Prolog の結果出力で省略が起きないようにするには，結果が出力されたところ（Enter やセミコロンを入力するところ）で **w** (write) を入力すると完全表示モードに移る.
  - 画面上では **[write]** と表示される
- 省略表示モードに戻すには **p** (print) を入力する.