述語論理

講義計画

1	ガイダンス（授業のテーマや到達目標及び授業の方法について説明する）	
2	論理型知識の表現1（意味ネットワーク・フレーム・ルール）	論理的知識
3	**論理型知識の表現2（述語論理）**	論理的知識
4	論理型のプログラミング言語Prolog入門	論理的知識
5	論理型のプログラミング言語Prolog演習1（基礎編）	論理的知識
6	論理型のプログラミング言語Prolog演習2（実践編）	論理的知識
7	Prolog総合演習	論理的知識
8	機械学習1（線形回帰）	機械学習
9	機械学習2（ロジスティック回帰）	機械学習
10	機械学習3（単純ベイズ分類器）	機械学習
11	機械学習4（決定木）	機械学習
12	機械学習5（サポートベクタマシン）	機械学習
13	機械学習6（深層学習）	機械学習
14	第1回目から第13回目までの講義内容について質疑応答を行う	機械学習
15	理解度テストと解説	

述語論理

- 命題

  命題とは、真偽が判定できる言明（宣言）のことである。

  【命題の例】 仮に、「太郎」という男性がいるとする。

  太郎は男性である（真）

  太郎は女性である（偽）

  【命題でない言明の例】 太郎は金持ちである（真偽が定まらない）

- 述語論理

  述語論理とは、変数を含み、その値によって真偽が異なるような命題である。

  【例】Xは男性である（一階述語論理:個体に対する量化が可能）

※二階述語論理（高階述語論理）：個体の集合や関数に対する量化を可能にする。

述語論理の記号表現

【例１】Xは人間である： human(X), 人間(X)
【例２】Xは金持ちである： rich(X), 金持ち(X)
【例３】Xには親がいる： parents(X), 親(X)
【例４】Xは死ぬ： die(X), 死ぬ(X)

【例５】XはYを好きである： like(X,Y), 好き(X,Y)
【例６】XはYを持っている： have(X,Y),持つ(X,Y)
【例７】XはYをほしがる： want(X,Y),ほしがる(X,Y)

述語論理式

- 述語論理式とは

  述語論理を論理記号によって結合したもの
- 論理記号

  ∧(かつ)、∨(または)、¬(否定:「not」と読む)、

  ⇒(ならば:if-then型の規則)
- 述語論理式の例

【例１】人間なら、いつか死ぬhuman(X)⇒die(X)

【例２】太郎はお金持ちで、花子を好きである

rich(Taro) ∧ like(Taro,Hanako)

【例３】太郎には親がいるか、人間でない

parents(Taro) ∨ (¬human(Taro))

述語論理式における等式と限定記号

- 等式
  - ¬(P∧Q) = ¬P∨¬Q
  - ¬(P∨Q) = ¬P∧¬Q
  - **P⇒Q = ¬P∨Q**（後ほど使用）
  - P⇒Q = ¬Q⇒¬P
- 限定記号
  - 全称記号

    ∀：すべての、任意の

    使用法：**∀x**：～ （すべてのxについて～）
  - 存在記号

    ∃：存在する

    使用法：**∃x**：～ （～であるxが存在する）

限定記号の使用例

次郎は太郎の持っているものを何でもほしがる

$\forall x[(have(太郎,x) \Rightarrow want(次郎,x)]$

太郎が持っており、次郎がほしがっているものがある

$\exists x[(have(太郎,x) \wedge want(次郎,x)]$

太郎も次郎も持っていないが、どちらかがほしがっているものがある

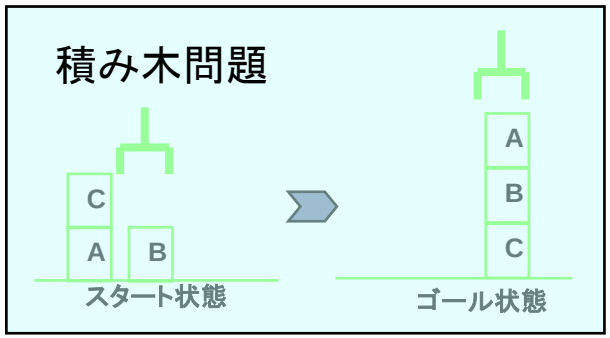
$\exists x[\neg has(太郎,x) \wedge \neg has(次郎,x) \wedge (want(太郎,x) \vee want(次郎,x))]$

述語論理（宣言型の知識）による問題解決の例

積み木問題（オペレータの適用順の問題）

積み木を持ち上げ、移動することのできるマジックハンドを1つ持つロボットがある。このロボットに積み木の作業をさせたい。与えられたスタート状態から作業を始めてゴール状態を達成するにはどのような作業手順に従って作業を行ったらよいか？（答えがたくさんあるが、最短の作業手順を求めることに注意すること）

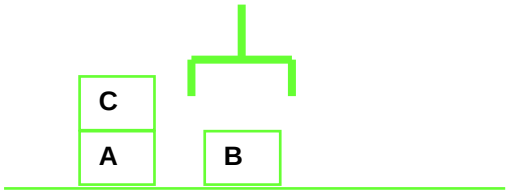
手順：
①状態を記述するための述語を定義
②①の述語により、スタート状態とゴール状態を記述
③①の述語によりオペレータ（動作）を表現
④述語論理式によりオペレータを記述



①状態を記述する述語

- ON(x, y)
  物体xが物体yの上にある
- CLEAR(x)
  物体xの上には何もない
- HOLDING(x)
  マジックハンドは物体xを持っている
- EMPTY
  マジックハンドは何も持っていない

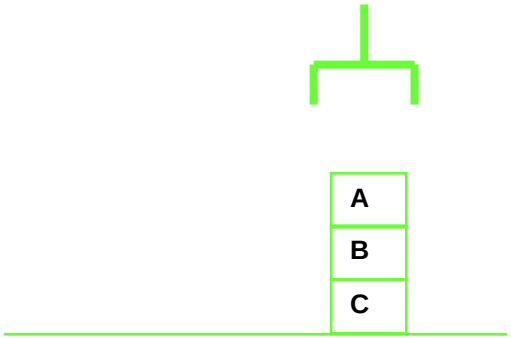
②スタート状態とゴール状態を記述する述語



スタート状態

ON(A, TABLE)
ON(B, TABLE)
ON(C, A)
CLEAR(B)
CLEAR(C)
EMPTY

ON(A, B)
ON(B, C)
ON(C, TABLE)
CLEAR(A)
EMPTY



ゴール状態

③オペレータの表現

オペレータ：オペレータ適用前の状態⇒オペレータ適用後の状態

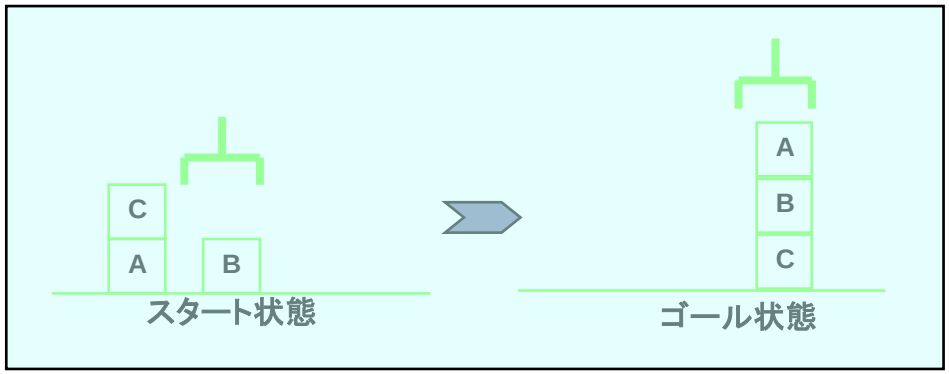
オペレータ	適用前の状態	削除リスト	追加リスト
PICKUP(x) (xを机の上から持ち上げる)	ONTABLE(x) CLEAR(x) EMPTY	ONTABLE(x) CLEAR(x) EMPTY	HOLDING(x)
UNSTACK(x,y) (xをyの上から持ち上げる)	ON(x,y) CLEAR(x) EMPTY	ON(x,y) CLEAR(x) EMPTY	HOLDING(x) CLEAR(y)
PUTDOWN(x) (xを机の上に置く)	HOLDING(x)	HOLDING(x)	ONTABLE(x) CLEAR(x) EMPTY
STACK(x,y) (xをyの上に置く)	HOLDING(x) CLEAR(y)	HOLDING(x) CLEAR(y)	ON(x,y) CLEAR(x) EMPTY

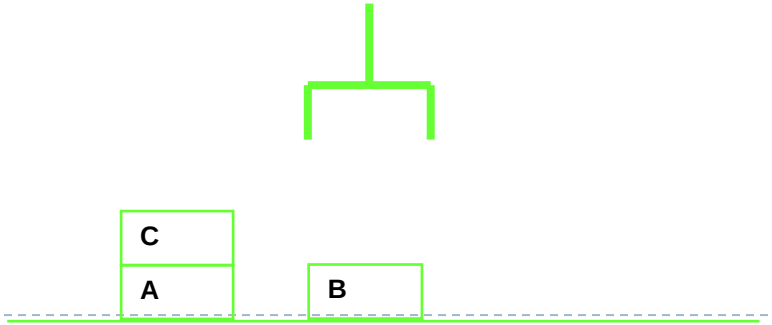
④述語論理式によるオペレータの記述

オペレータ	述語論理式
PICKUP(x)	∀x[ONTABLE(x)∧CLEAR(x)∧EMPTY⇒HOLDING(x)]
UNSTACK(x,y)	∀x∀y[ON(x,y)∧CLEAR(x)∧EMPTY⇒HOLDING(x)∧CLEAR(y)]
PUTDOWN(x)	∀x[HOLDING(x)⇒ONTABLE(x)∧CLEAR(x)∧EMPTY]
STACK(x,y)	∀x∀y [HOLDING(x)∧CLEAR(y)⇒ON(x,y)∧CLEAR(x)∧EMPTY]

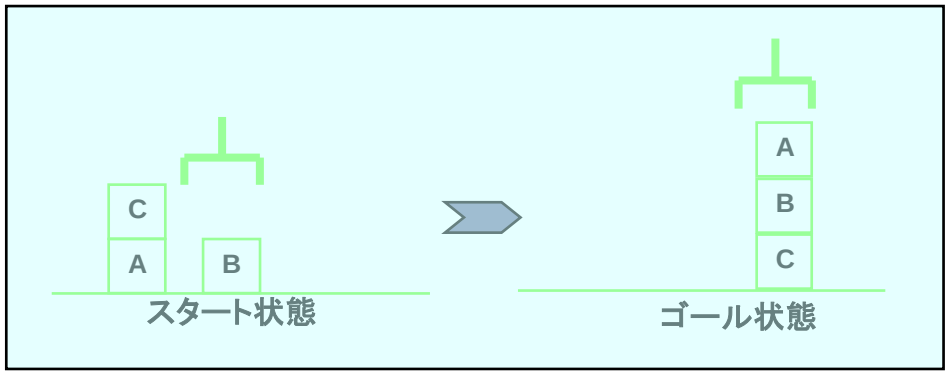
オペレータの適用順の問題を述語論理の問題に変わる

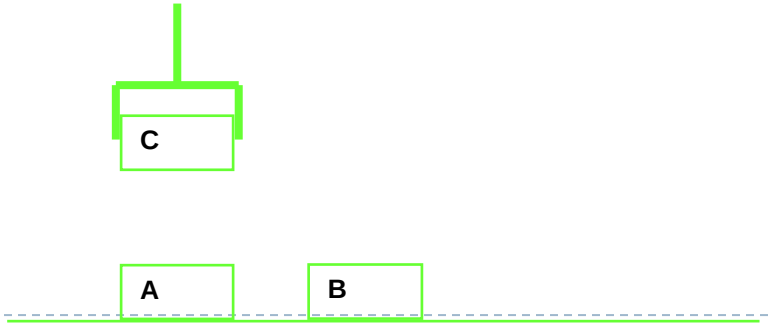
オペレータの適用過程



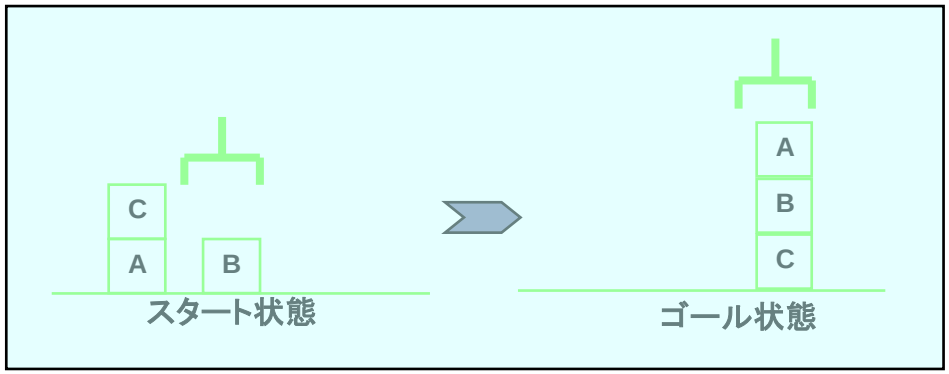


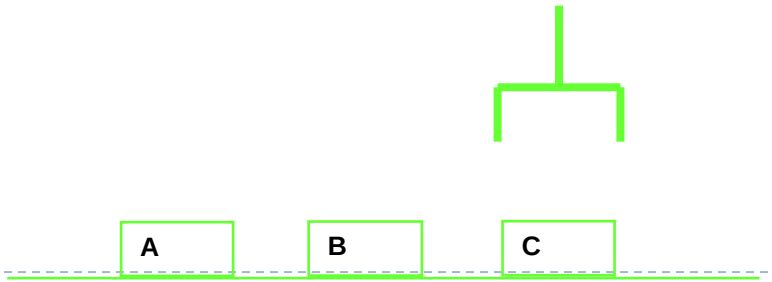
オペレータの適用過程



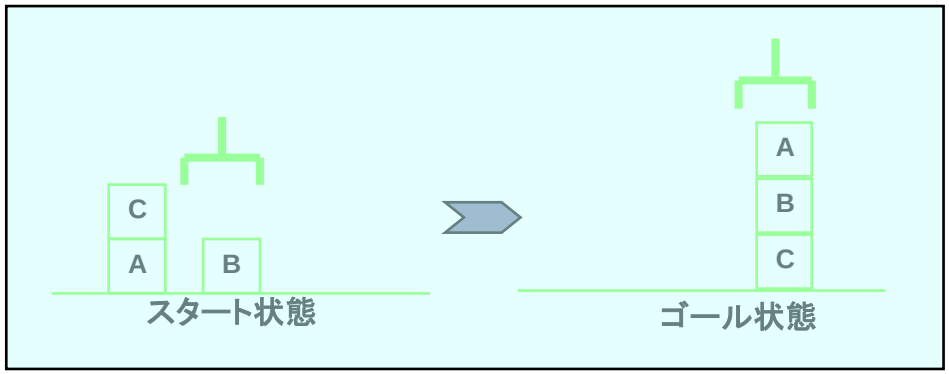


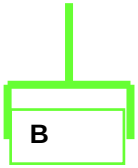
オペレータの適用過程



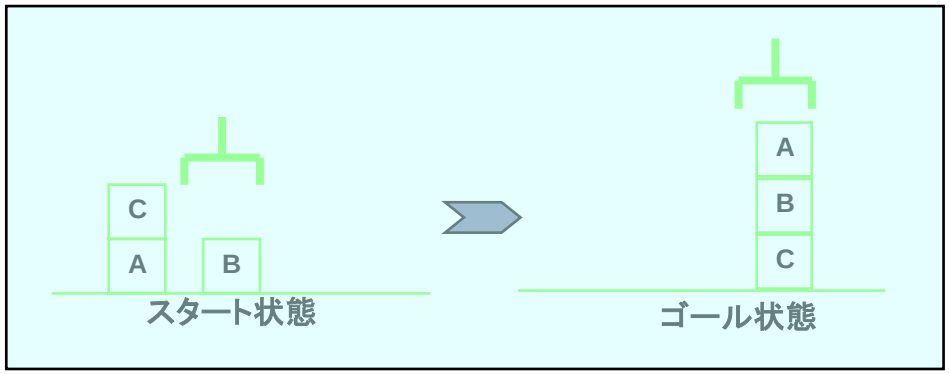


オペレータの適用過程

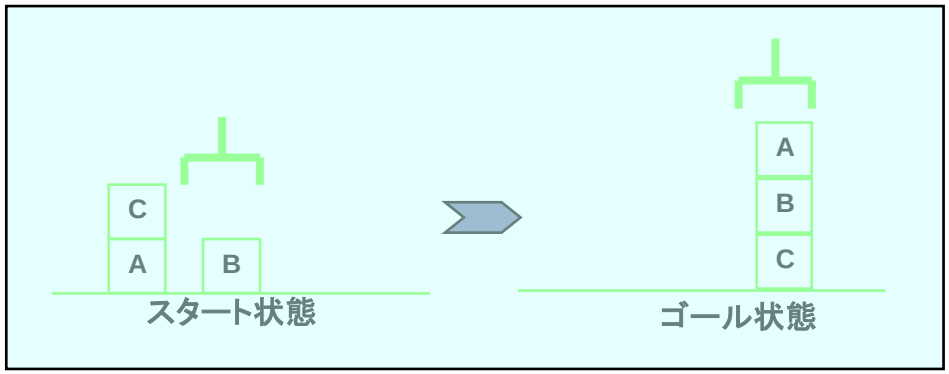


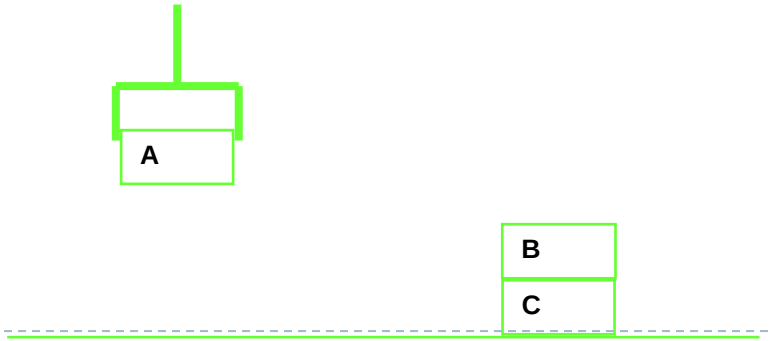


オペレータの適用過程

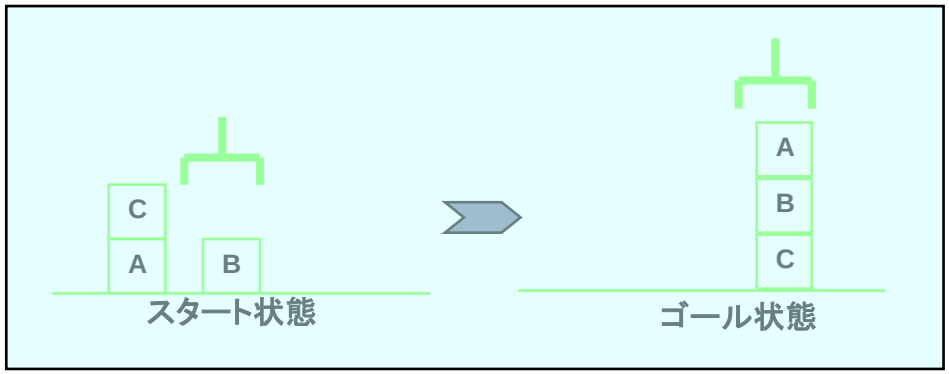


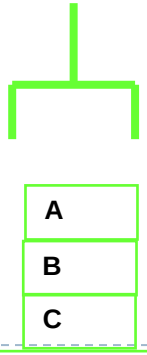
オペレータの適用過程



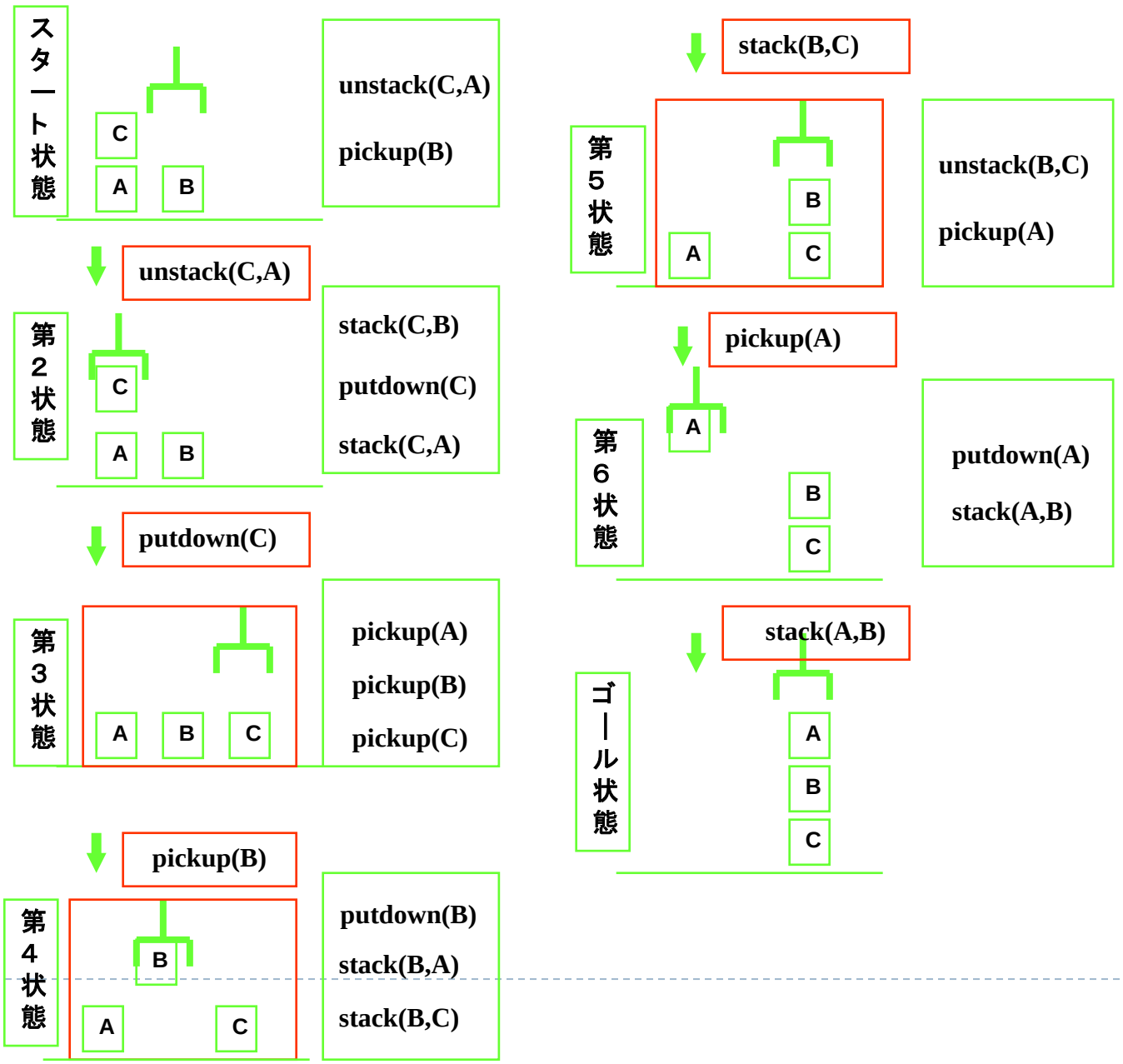


オペレータの適用過程





積み木の状態遷移


スタート状態
C
A
B
unstack(C,A)
pickup(B)
unstack(C,A)
第２状態
C
A
B
stack(C,B)
putdown(C)
stack(C,A)
putdown(C)
第３状態
A
B
C
pickup(A)
pickup(B)
pickup(C)
pickup(B)
第４状態
B
A
C
putdown(B)
stack(B,A)
stack(B,C)
stack(B,C)
第５状態
B
A
C
unstack(B,C)
pickup(A)
pickup(A)
第６状態
A
B
C
putdown(A)
stack(A,B)
stack(A,B)
ゴール状態
A
B
C

述語論理によるオペレータの適用順の問題解決

背理法

証明したい結論を否定し、他の既知条件とあわせて考えると矛盾することを示すことによって、結論が成り立つことを間接的に証明する。

「クモは昆虫ではない」
ことを証明したい

クモは昆虫であると仮定する

↓

昆虫の体は頭部・胸部・腹部の３つの部分からなり、足の数は６本である

↓

クモの体は頭部・腹部の２つの部分からなり、足の数は８本である

↓

クモが昆虫だとすると、クモの体の構造と足の数は昆虫と矛盾する。

↓

この矛盾はクモが昆虫であるとしたから生じたものである

↓

よって、クモは昆虫ではない

http://optica.cocolog-nifty.com/blog/cat54802764/index.html より

述語論理によるオペレータの適用順の問題解決

単一化

述語の変数に適当な値を代入することによって、同じ述語にすることである

【例】like(X,hanako)　と　like(taro,Y)　があるとする。

もし、X=taro,Y=hanako　ならば、

like(X,hanako)とlike(taro,Y)は同じ述語like(taro,hanako)となる（単一化）。

述語論理によるオペレータの適用順の問題解決

導出原理

導出とは二つの節より新しい節を導き出す操作で、一方の節に含まれるリテラル l と、他方の節に含まれる否定リテラル $\sim l$ を除去し、その他のリテラルの論理和をとることで、新しい節を得ることができる。

$C_1 = \sim p \vee q$　　$C_2 = \sim q \vee r$

$C_3 = \sim p \vee r$

【証明】 $P \Rightarrow Q = \neg P \vee Q$

$C_1 = \sim p \vee q = p \Rightarrow q$

$C_2 = \sim q \vee r = q \Rightarrow r$

$C_3 = \sim p \vee r = p \Rightarrow r$

C_1: $p \Rightarrow q$
C_2: $q \Rightarrow r$

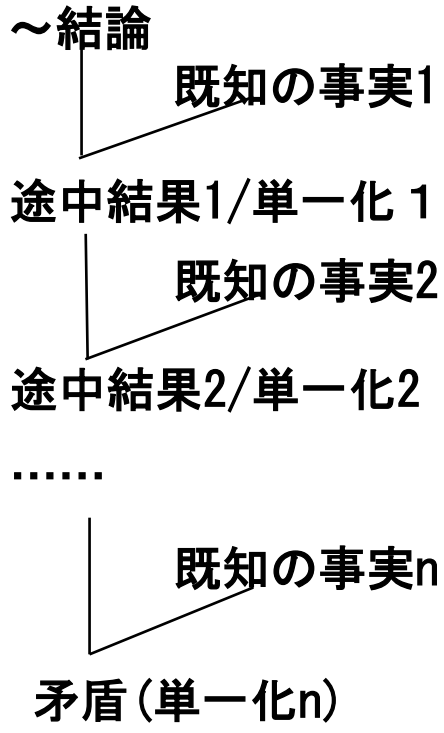
C_3: $p \Rightarrow r$

三段論法

述語論理によるオペレータの適用順の問題解決

導出反駁木

結論を否定した述語論理式から、単一化を行うことによって矛盾を導く過程を表す木。



述語論理による問題解決例ー計画問題

- 計画問題
  - 初期状態と目標状態とオペレータ集合が与えられ、
    オペレータをどのような順序で適用したら、目標状態を達成できるかを
    答える問題。
- 計画問題の形式化
  - 初期状態を論理式P、
    オペレータの集合を論理式の集合{R1,R2,...,Rn}、
    目標状態を論理式Qで表わせば、
    背理法により次の論理式が矛盾を含むことを証明することになる。
    (P∧(R1∧R2∧...∧Rn)∧～Q)
  - 問題の解、すなわち**オペレータの適用順序は、この証明プロセスから抽出**される。
- 計画問題の問題点
  - 計画問題では、『順番（時間）』という要素が含まれている。
  - 時間を『状態の遷移』として捉えた論理表現を用いる。

➢ ＧＲＥＥＮによる定式化※

・状態に依存した命題を表現する１つの方法は、
述語に状態を表す項を導入することである。

Ｂの状態は変化していないので、説明を簡単にするため、Ｂの状態記述を省略

【例】右図に示すような積木の問題を考える。

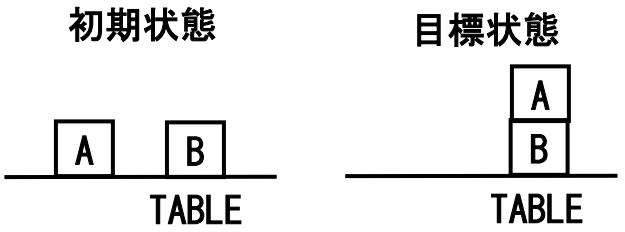
・状態を表わす述語の引数として状態sを追加する。

《初期状態》 ONTABLE(A, s_0)

《目標状態》 ヨs ON(A,B,s)

（ある状態sにおいて、AがBの上にある）

簡単な積み木の問題



状態変数sを最後の引数として追加する

オペレータ

■LIFT(x)： 積木xを持ち上げる
前提条件： ONTABLE(x)
削除リスト： ONTABLE(x)
追加リスト： HOLDING(x)

■PUT(x,y)： 積木xを積木yの上に置く
前提条件： HOLDING(x)
削除リスト： HOLDING(x)
追加リスト： ON(x,y)

※Green,C, "Application of Theorem Proving to Problem Solving", Proc. Int. Joint Conf. on AI,pp219-240,1969.

➢ 解決過程

・状態sにおいてオペレータ(例lift)を適用した結果、状態s'が生じたとする。この状態変化を関数afterを用いて、s'=after(lift(x),s)と表す。

《LIFT(x)》 ∀x ∀s [ONTABLE(x,s)⇒HOLDING(x,after(lift(x),s))]

《PUT(x,y)》∀x ∀y ∀s [HOLDING(x,s)⇒ON(x,y,after(put(x,y),s))]

・次の論理式が充足不能であることを証明すればよい。

《初期状態》∧《LIFT(x)》∧《PUT(x,y)》∧～《目標状態》

・下図のような導出反駁木が得られる。

・用いた単一化置換を変数sに関してトレースすると

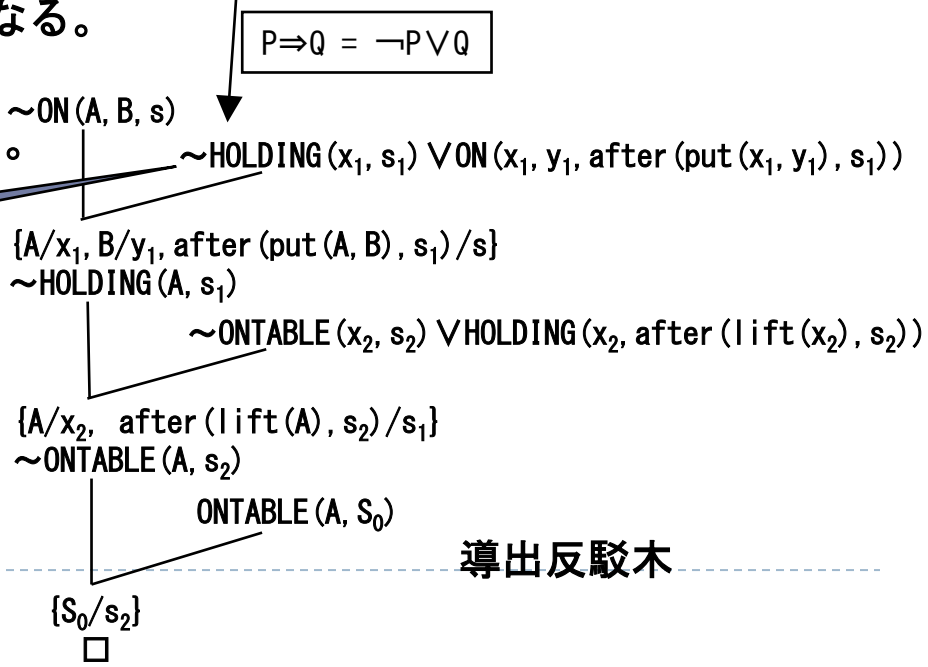
{after(put(A,B),s_2)/s}・{after(lift(A),s_1)/s_2}・{s_0/s_1}であるから

s = after(put(A,B),after(lift(A),s_0)) となる。

・解は

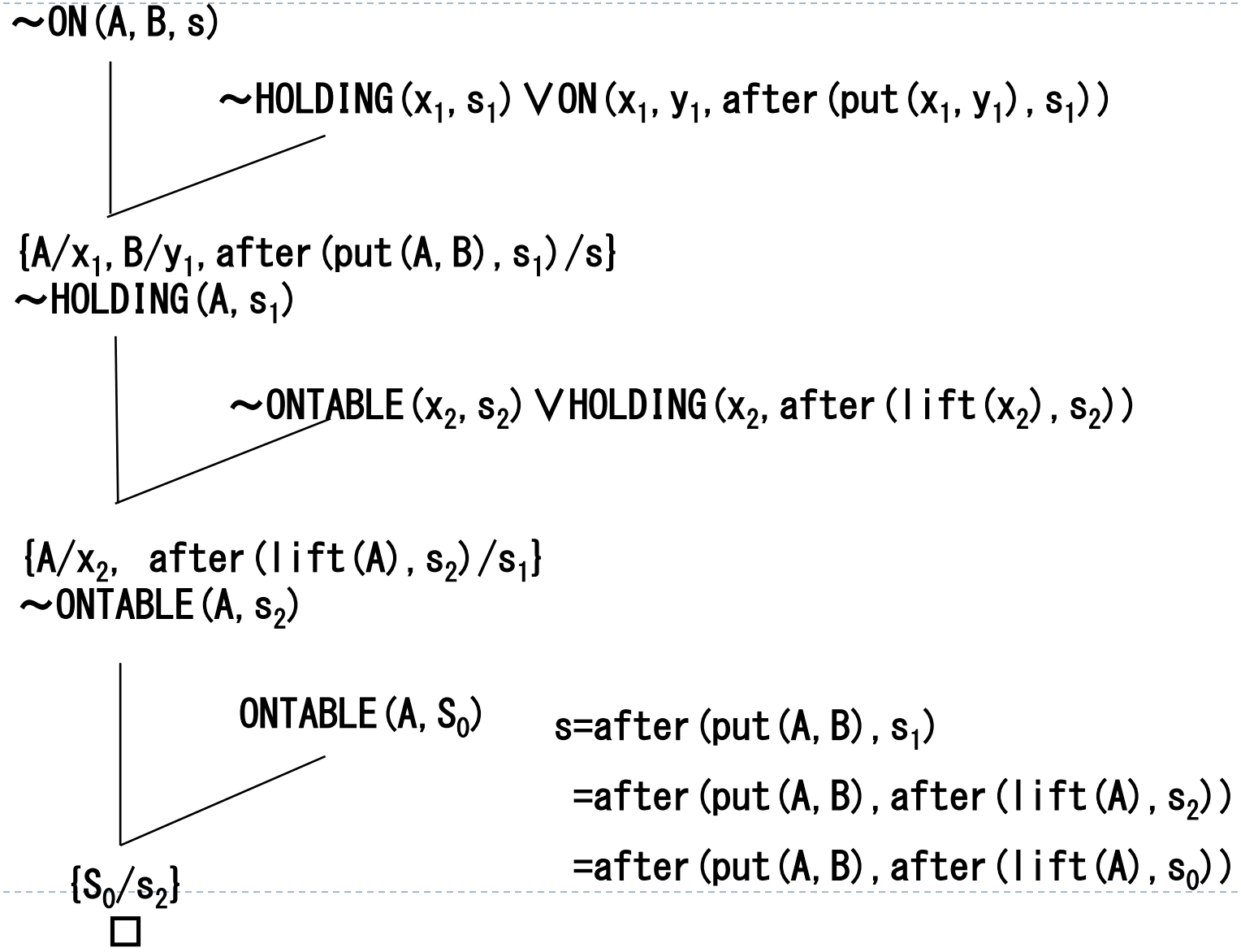
LIFT(A)→PUT(A,B) というオペレータ順序列。

LIFT(x,y)を先に適用しようとしても単一化はできなくなる



導出反駁木

導出反駁木(拡大図)


~ON(A, B, s)
~HOLDING(x1, s1) ∨ ON(x1, y1, after(put(x1, y1), s1))
{A/x1, B/y1, after(put(A, B), s1)/s}
~HOLDING(A, s1)
~ONTABLE(x2, s2) ∨ HOLDING(x2, after(lift(x2), s2))
{A/x2, after(lift(A), s2)/s1}
~ONTABLE(A, s2)
ONTABLE(A, S0)
{S0/s2}
□
s=after(put(A, B), s1)
=after(put(A, B), after(lift(A), s2))
=after(put(A, B), after(lift(A), s0))