計算の理論

後半第5回

「前回のつづき(証明支援システム)と論理プログラミング」

小林

本日の内容

- Curry-Howard同型（補足と復習）
  - 証明 = プログラム
- 証明支援システム
- 論理プログラミング

単純型つきラムダ計算の拡張: 直積型

- 構文

  τ (型) ::= b | $\tau_1 \to \tau_2$ | $\tau_1 \times \tau_2$

  b (基本型) ::= int | bool | ...

  M（項） ::= ... | (M_1, M_2) | fst(M) | snd(M)

- 簡約

  ...

  $fst(M_1, M_2) \to M_1$

  $snd(M_1, M_2) \to M_2$

- 型付け

$$\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash (M, N) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash fst(M) : \tau_1}$$

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash snd(M) : \tau_2}$$

直和型を加えた拡張

- 構文

  τ (型)　　::= ... | $\tau_1+\tau_2$

  M（項）　　::= ... | inl(M) | inr(M)

  | case M_0 of inl(x)=>M_1 | inr(y)=>M_2

- 簡約

  case inl(M) of inl(x)=>M_1 | inr(y)=>M_2 $\rightarrow$ [M/x]M_1

  case inr(M) of inl(x)=>M_1 | inr(y)=>M_2 $\rightarrow$ [M/y]M_2

- 型付け

$$\frac{\Gamma \vdash M: \tau_1}{\Gamma \vdash \mathrm{inl}(M): \tau_1+\tau_2}$$

$$\frac{\Gamma \vdash M: \tau_2}{\Gamma \vdash \mathrm{inr}(M): \tau_1+\tau_2}$$

$$\frac{\Gamma \vdash L: \tau_1+\tau_2 \quad \Gamma, x: \tau_1 \vdash M: \tau \quad \Gamma, y: \tau_2 \vdash N: \tau}{\Gamma \vdash \mathrm{case}\ L\ \mathrm{of}\ \mathrm{inl}(x)=>M \mid \mathrm{inr}(y)=>N: \tau}$$

計算と論理の不思議な対応関係 (Curry-Howard同型)

計算の世界	論理の世界
型	命題
プログラム	証明
計算	証明の簡単化
型検査	証明の整合性の検査

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	
$A \vee B$	
$A \rightarrow B$	

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	
$A \rightarrow B$	

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \rightarrow B$	

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \to B$	Aの証明をもらってBの証明を返す関数

例. $A \to (B \to (A \wedge B))$の証明:

$= \lambda x{:}A.$「 $B \to (A \wedge B)$の証明」

$= \lambda x{:}A.\, \lambda y{:}B.$「 $A \wedge B$の証明」

$= \lambda x{:}A.\, \lambda y{:}B.$ (「 Aの証明」,「 Bの証明」)

$= \lambda x{:}A.\, \lambda y{:}B.\, (x, y)$

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \rightarrow B$	Aの証明をもらってBの証明を返す関数

例.「$(A \vee B) \rightarrow (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$の証明」

$= \lambda x\colon (A \vee B).$「$(A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$の証明」

$= \lambda x\colon (A \vee B).\ \lambda y\colon (A \rightarrow C).$「$(B \rightarrow C) \rightarrow C$の証明」

$= \lambda x\colon (A \vee B).\ \lambda y\colon (A \rightarrow C).\ \lambda z\colon (B \rightarrow C).$「$C$の証明」

$= \lambda x\colon (A \vee B).\ \lambda y\colon (A \rightarrow C).\ \lambda z\colon (B \rightarrow C).$

case x of inl(u) =>「Cの証明」 /* Aが成り立つ場合*/

| inr(v) =>「Cの証明」 /* Bが成り立つ場合*/

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \to B$	Aの証明をもらってBの証明を返す関数

例.「$(A \vee B) \to (A \to C) \to (B \to C) \to C$の証明」

$= \lambda x{:}\ (A \vee B).$「$(A \to C) \to (B \to C) \to C$の証明」

$= \lambda x{:}\ (A \vee B).\ \lambda y{:}\ (A \to C).$「$(B \to C) \to C$の証明」

$= \lambda x{:}\ (A \vee B).\ \lambda y{:}\ (A \to C).\ \lambda z{:}\ (B \to C).$「$C$の証明」

$= \lambda x{:}\ (A \vee B).\ \lambda y{:}\ (A \to C).\ \lambda z{:}\ (B \to C).$

```
  case x of inl(u) => y(u)
          | inr(v) => z(v)
```

λ項の存在と証明可能性

定理:

型Aの閉じた単純型付きλ項が存在する

⇔

Aが直観主義命題論理で証明可能

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \rightarrow B$	Aの証明をもらってBの証明を返す関数
$\forall x{:}S.P(x)$	
$\exists x{:}S.P(x)$	

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \rightarrow B$	Aの証明をもらってBの証明を返す関数
$\forall x{:}S.P(x)$	Sの要素xを引数としてP(x)の証明を返す関数
$\exists x{:}S.P(x)$	

Curry-Howardの対応による証明の表現

論理式	証明
$A \wedge B$	(Aの証明, Bの証明)
$A \vee B$	inl(Aの証明)またはinr(Bの証明)
$A \to B$	Aの証明をもらってBの証明を返す関数
$\forall x{:}S.P(x)$	Sの要素xを引数としてP(x)の証明を返す関数
$\exists x{:}S.P(x)$	Sの要素aとP(a)の証明の組

例. $(\forall x{:}nat.x<x+1) \to \forall y{:}nat.\ \exists z{:}nat.(y<z)$の証明:

= $\lambda p{:}\ (\forall x{:}nat.x<x+1)$.「$\forall y{:}nat.\ \exists z{:}nat.(y<z)$の証明」

= $\lambda p{:}\ (\forall x{:}nat.x<x+1)$. $\lambda y{:}nat.$「$\exists z{:}nat.(y<z)$の証明」

= $\lambda p{:}\ (\forall x{:}nat.x<x+1)$. $\lambda y{:}nat.$ (y+1, 「y<y+1の証明」)

= $\lambda p{:}\ (\forall x{:}nat.x<x+1)$. $\lambda y{:}nat.$ (y+1, p y)

証明とプログラム抽出

$\forall x{:}S.\ \exists y{:}T.P(x,y)$ の証明：

$\lambda x{:}S.\ (e,$ “$P(x,e)$ の証明”$)$

$P(x,e)$の証明部分を除いて得られる

$\lambda x{:}S.\ e$

は、型Sの値aを受け取ってP(a,b)を満たすbを返す関数

例：$\forall x{:}$ nat list.$\exists y{:}$ nat list.

(sorted(y) $\wedge$ permutation(x,y))

を証明すれば、その証明からソーティング関数を抽出できる!

本日の内容

- Curry-Howard同型（補足と復習）
  - 証明 = プログラム
- 証明支援システム
- 論理プログラミング

定理証明支援器
(Coq, Isabelle/HOL, Agda, ...)

- 対象(数、関数、データ、プログラムなど)や
  その性質を表す命題およびその証明を形式的に記述するための言語を提供
- 証明の正しさを機械的に検査
- 証明を行う手助け(半自動証明)
- 証明からプログラムを抽出

定理証明支援器
(Coq, Isabelle/HOL, Agda, ...)

- 対象(数、関数、データ、プログラムなど)や
  その性質を表す命題およびその証明を形式的に記述するための言語を提供

型付きλ計算！

- 証明の正しさを機械的に検査
- 証明を行う手助け(半自動証明)
- 証明からプログラムを抽出

定理証明支援器
(Coq, Isabelle/HOL, Agda, ...)

- 対象(数、関数、データ、プログラムなど)や
  その性質を表す命題およびその証明を形式的に記述するための言語を提供
- 証明の正しさを機械的に検査
- 証明を行う手助け(半自動証明)
- 証明からプログラムを抽出

型付きλ計算！

型検査

定理証明支援器
(Coq, Isabelle/HOL, Agda, ...)

- 対象(数、関数、データ、プログラムなど)や
  その性質を表す命題およびその証明を形式的に記述するための言語を提供
- 証明の正しさを機械的に検査
- 証明を行う手助け(半自動証明)
- 証明からプログラムを抽出

型付きλ計算！

型検査

証明=プログラムだから
当たり前！

証明支援器の必要性

- 証明は
  - 面倒 (特に大量の場合分けを要する場合など)
  - 間違いが混入しやすい(*)

  (*)数学の専門家でも。

⇒ 計算機による証明の検査や半自動証明

証明支援器の応用事例

- 数学
  - 4色問題
  - Fermatの最終問題(n=3,4のケース)
  - 代数学の基本定理
  - Odd Order Theorem （群論に関する定理）
  - ケプラー予想
- コンピュータサイエンス
  - Gödelの不完全性定理
  - 種々のアルゴリズムの証明
    (Presburger算術、暗号アルゴリズムなど)
  - 種々のソフトウェアの検証
    (コンパイラ、オペレーティングシステム)

Coq(*)のデモ

(*) http://coq.inria.fr

Coqのコマンド

- トップレベルで用いるコマンド(Vernacular command)
  - 定義のためのコマンド(Definition, Inductive, Fixpoint, Variableなど)
  - 定理のためのコマンド(Theorem, Lemma)
  - 定義等の確認のためのコマンド(Check, Print)
  - ライブラリを読み込むためのコマンド(Require)
- 証明中に使うコマンド(tactics)
  - 自動証明のためのコマンド(auto, tauto)
  - 細かい証明の指示のためのコマンド(intro, induction, apply, exact, rewrite, simpl)
  - 証明のおわり(Qed)

トップレベルで用いる
Coqのコマンド

定義のためのコマンド
Definition

フォーム:

Definition <名前> := <項>.

例:

Definition X := 1.

(** X を1という項を表す名前として定義 **)

定理を述べるためのコマンド
Theorem

フォーム:

Theorem <定理名>: <命題>.

Proof. <証明> Qed.

例:

```
Theorem id: forall A:Prop, A -> A.
Proof.
  exact (fun A:Prop =>fun x:A => x).
Qed.
```

注: Lemma も全く同じ。公理はTheorem のかわりに Axiomと書き、証明はつけない。

命題を記述するための構文

True:　　真

False:　　偽

A ／＼ B :　　Aかつ B

A ＼／ B :　　AまたはB

A -> B:　　AならばB

~A :　　Aの否定（A->Falseの略）

forall x:A, B　　任意のAの要素xについてB

exists x:A, B　　あるAの要素xについてB

帰納的定義のためのコマンドInductive

フォーム:

```
Inductive  <名前>: <型> :=
    <コンストラクタ名> : <型>
  | ...
  | <コンストラクタ名>: <型>
```

例:

```
Inductive  mynat: Set :=
    Z : mynat
  | S : mynat -> mynat.
(*** mynat を、ZとSのみから構成される項の型として
 定義 ***)
```

再帰的関数の定義
Fixpoint

フォーム:

Fixpoint <関数名> (<引数>*: <型>)
{struct <引数>}: <返り値の型> :=
<関数本体>

例: 足し算を行う関数plus の定義

```
Fixpoint plus (m n: mynat) {struct m} : mynat :=
  match m with
     Z =>  n
   | S m' => S(plus m' n)
  end.
```

注: 関数本体中の再帰呼び出しの引数は、structで指定された引数に関して小さくなっていなければならない。

再帰関数に関する注意

- Fixpointによる関数定義では、停止すること（全関数であること）がわかっている関数しか記述できない
- もし停止しない関数が書けたとすると以下のように矛盾が導ける

Fixpoint f (x:nat) = 1+f(x)

関数の定義より f(0)=1+f(0)

∀x,y,z:nat.(x=y -> x-z = y-z) より

f(0)-f(0) = 1+f(0)-f(0)

∀x:nat.(x-x=0) と +に関する結合法則より

0 = 1

証明のためのコマンド

証明モード

CoqIdeを用いると、右上のウィンドウに現在証明すべきゴール（複数ある場合はその一つ目のみ）が表示

ゴールの形式：

```
  H1: A1
  ...
 Hn: An
-------------------------
  G
```

「仮定H1:A1, ..., Hn:An を用いてGを導け」という意味

証明の開始・終わり

Proof.　証明の開始（省略可）

Qed.　証明のおわり
（ゴールのウィンドウに
”No more subgoals”
と表示されたときのみ有効）

自動証明のためのコマンド

- auto.
- tauto.

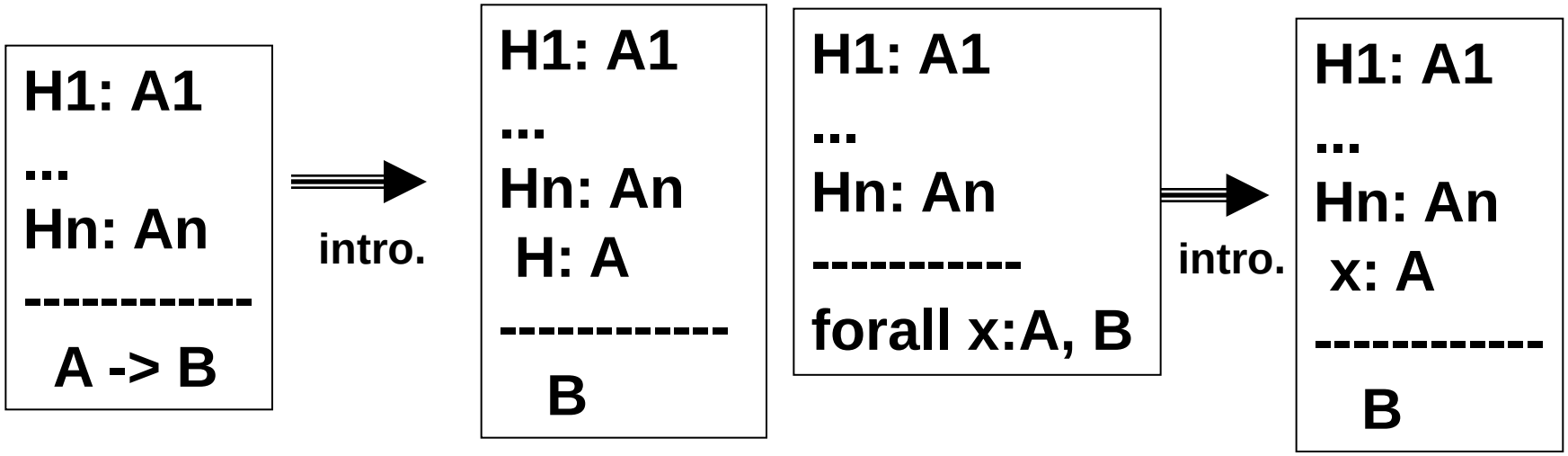
など。

コマンドによって証明戦略が異なる。

ヒントをうまく使うと証明が簡単に。

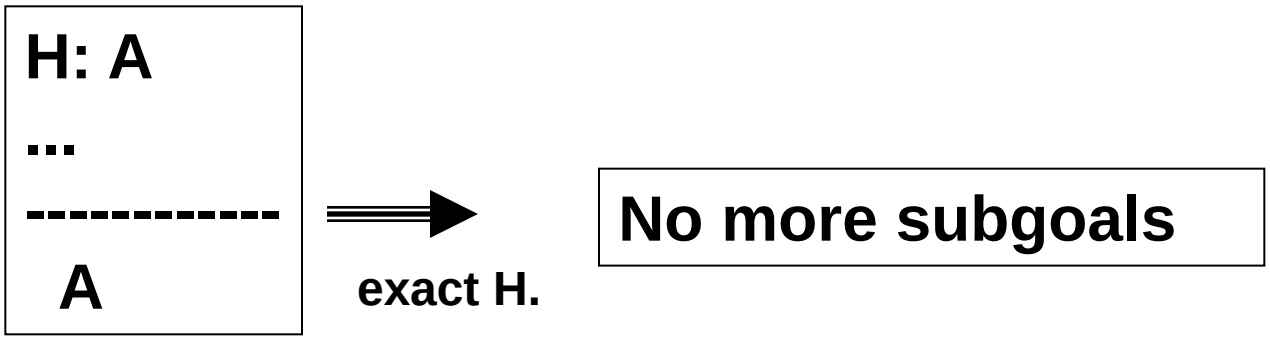
intro

ゴールがA-> Bやforall x:A, Bのときに適用可

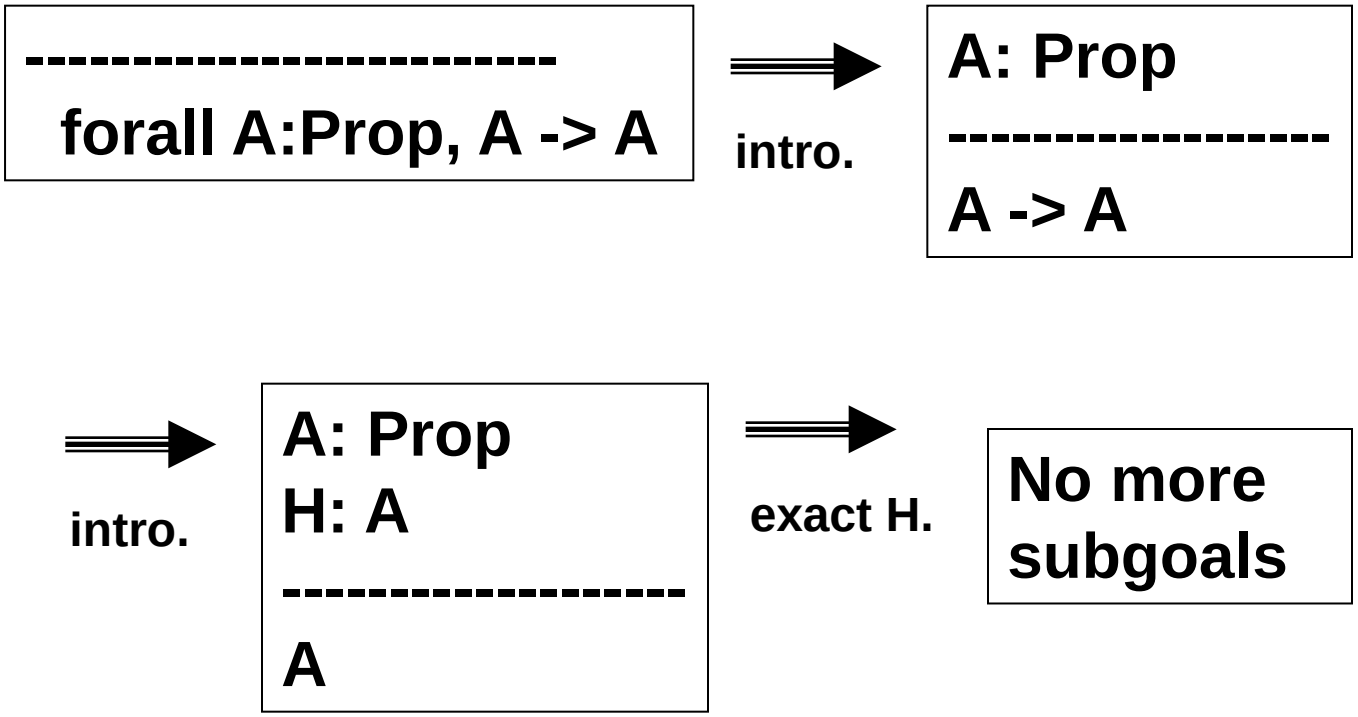


exact

証明をλ式で直接指定

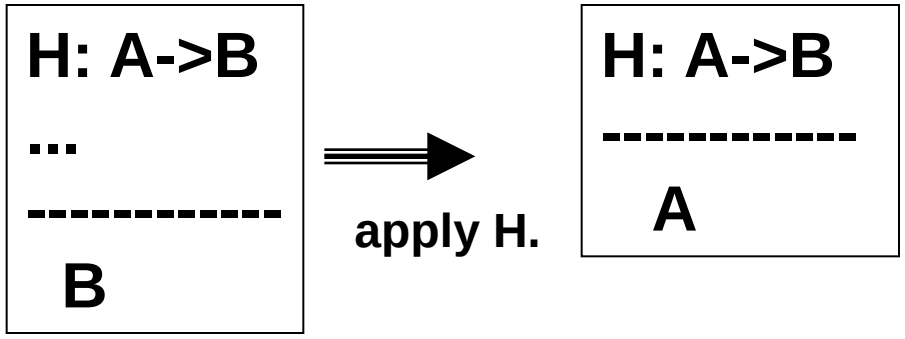


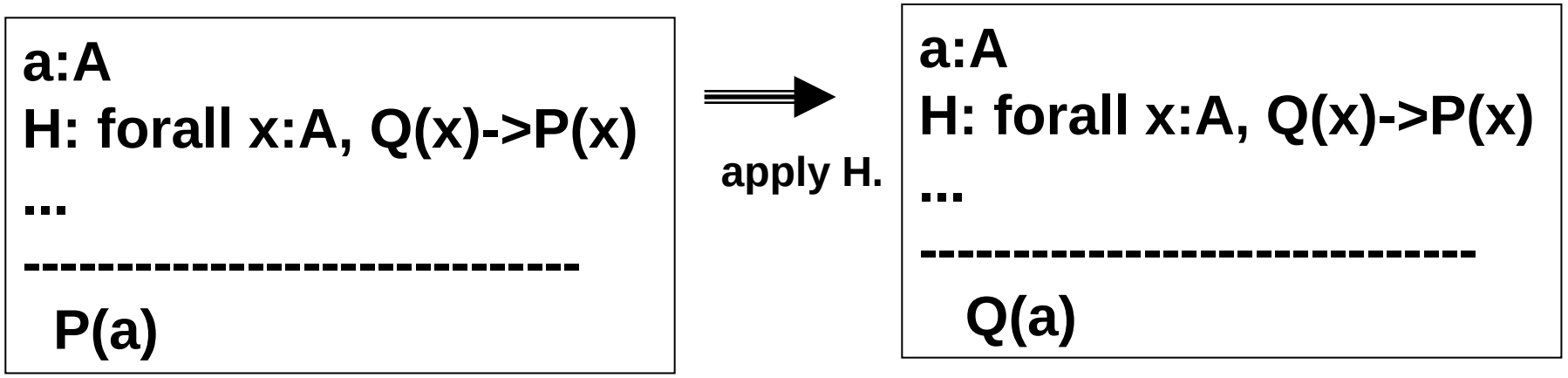
introとexactの組み合わせ例



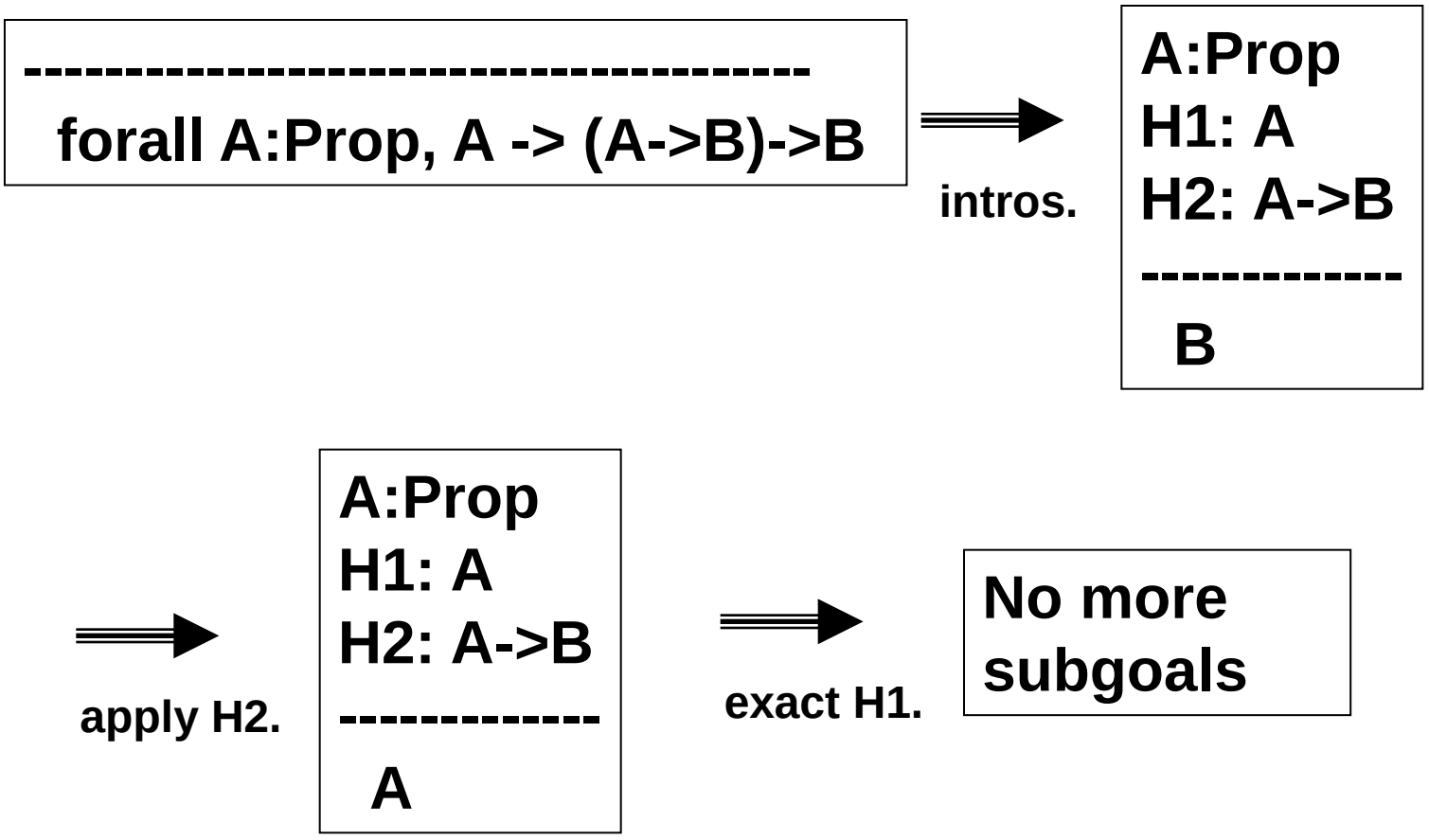
apply

- ゴールがBでA->Bという形の仮定（またはすでに証明された定理）がある場合に適用可能

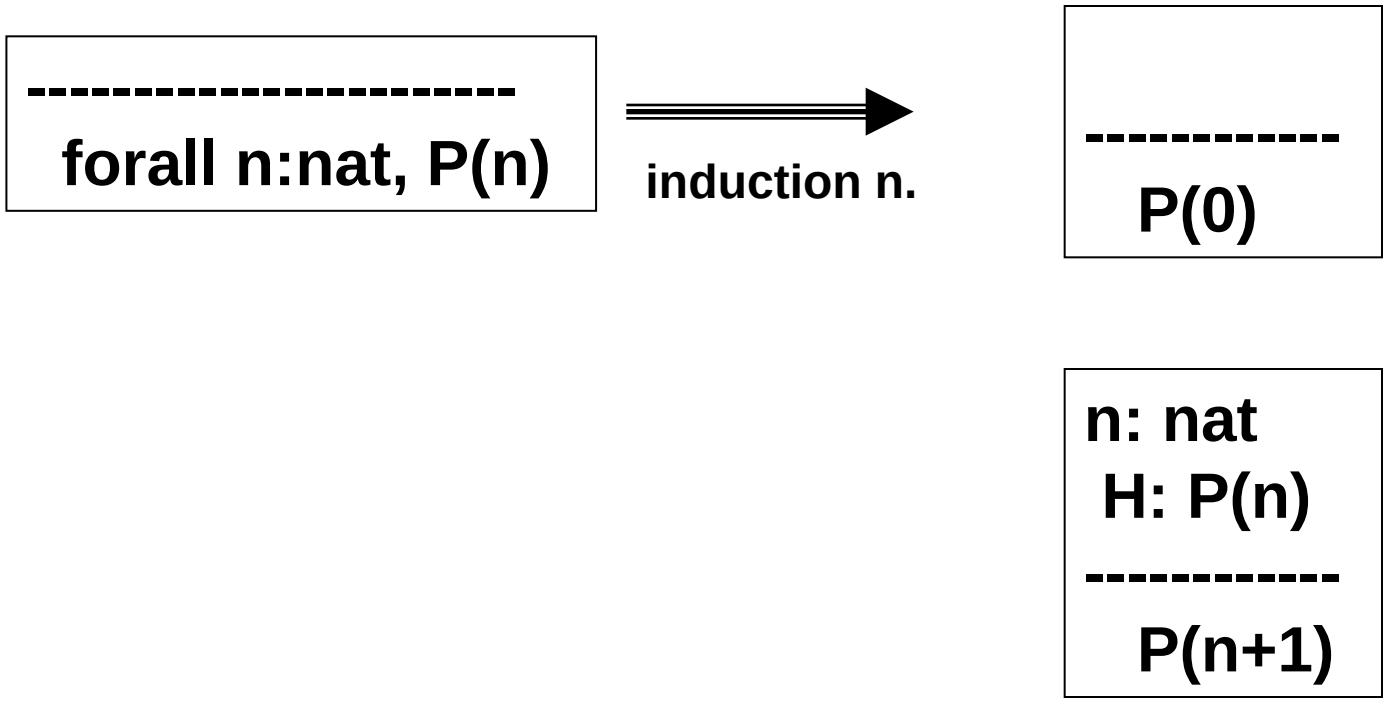




introとapply, exactの組み合わせ例

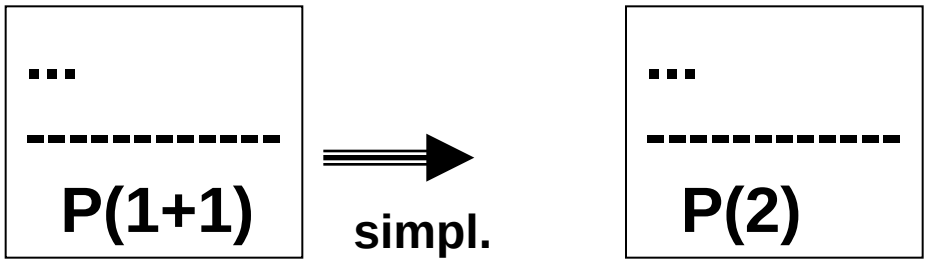


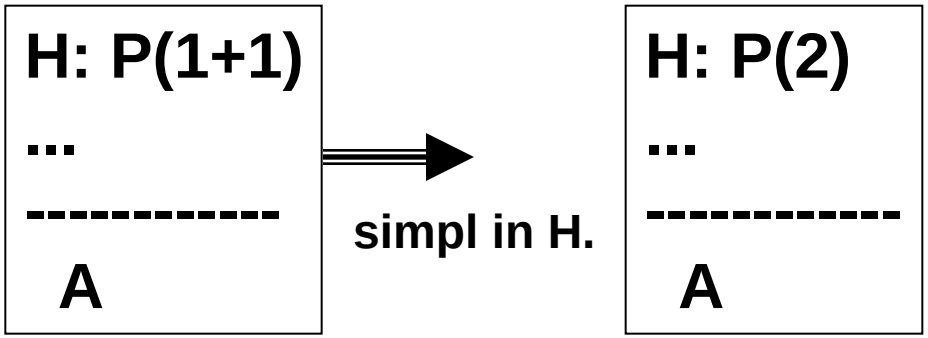
induction



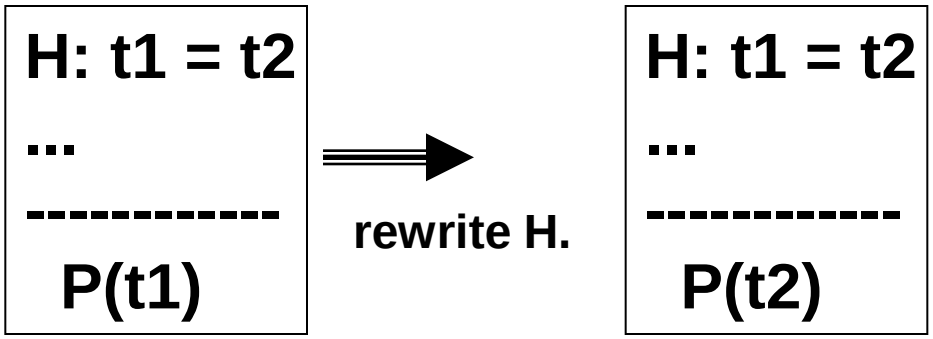
simpl

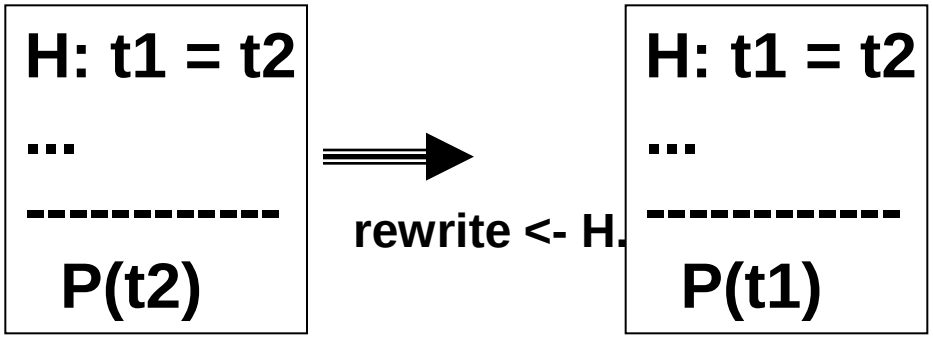
ゴールや仮定中のterm を簡約



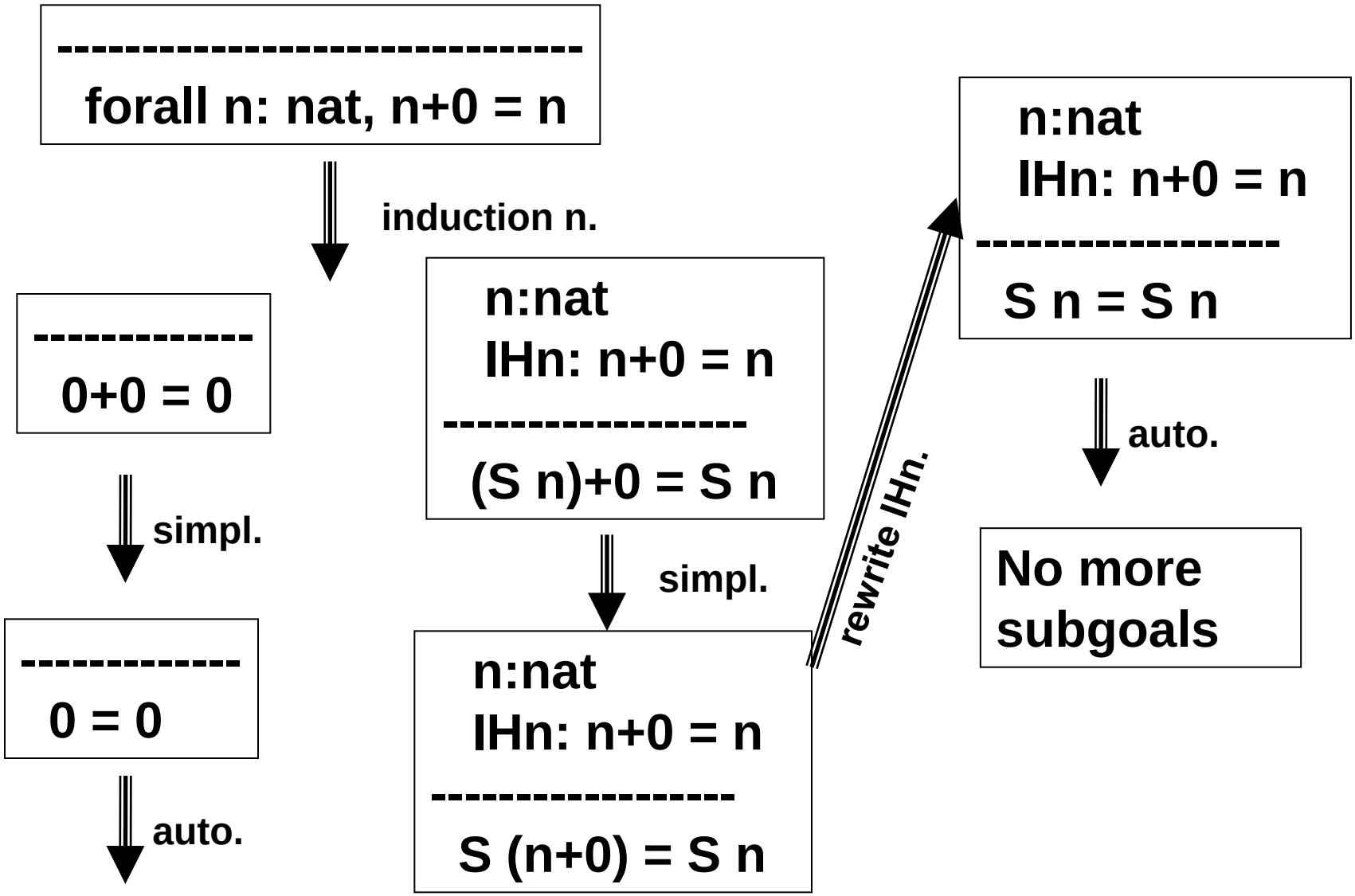


rewrite





induction, simpl, rewriteの組み合わせ例

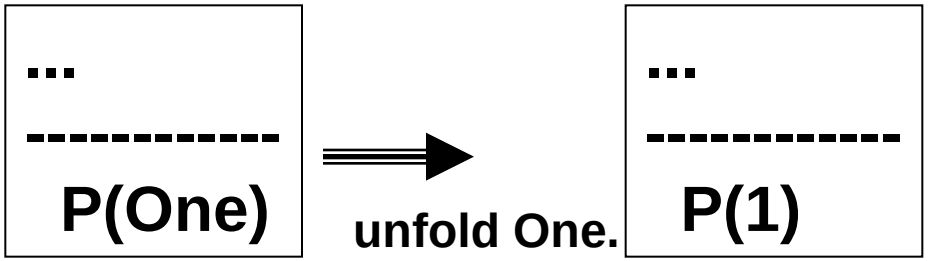


unfold

定義を展開

例 : Definition One := 1.

と定義されているとすると...

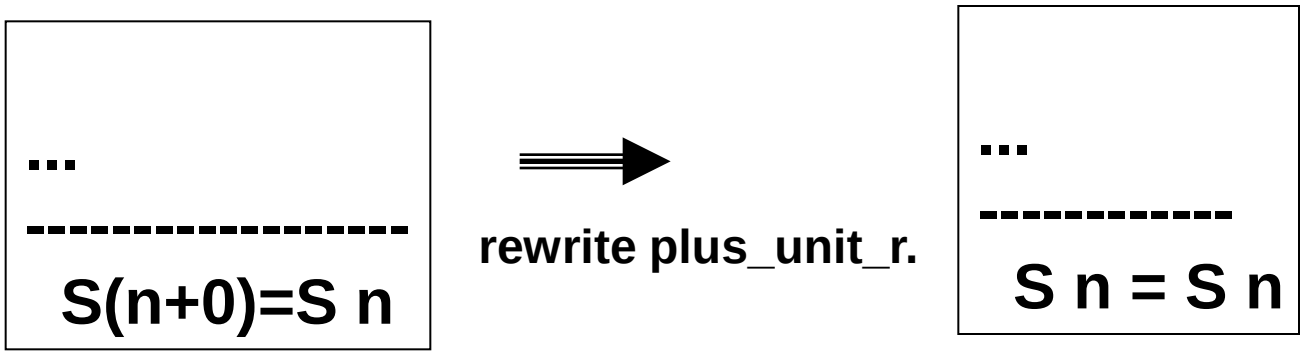


すでに証明された定理の利用

- apply, rewrite, exact などの引数として、ゴールの仮定に明示されているもの以外に、公理や証明ずみの定理を使用可

例:

Theorem plus_unit_r: forall n:nat, n+0 = n.
が証明されているとして



本日の内容

- Curry-Howard同型（補足と復習）
  - 証明 = プログラム
- 証明支援システム
- 論理プログラミング

論理型プログラミング

- プログラム = 論理式
- プログラムの実行 = 証明探索
  (cf. Curry-Howard同型:
  論理式＝型、プログラム=証明)
- 代表的な言語
  - Prolog
    - 1970年代に人工知能の研究用に開発
    - エキスパートシステム、自然言語処理などの知的処理を伴うプログラムの開発に応用
  - GHC, KL1
    - 論理型言語に並行プログラミングの要素を盛り込んだもの
    - 日本の「第5世代コンピュータ」プロジェクトで活発に研究

Prologのプログラム例

```
plus(0, X, X).                    /* 0+X = X */
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).
                  /* X+Y=Zならば(X+1)+Y = Z+1 */
```

自然数の表現:

0	0
1	s(0)
2	s(s(0))
3	s(s(s(0)))
…	

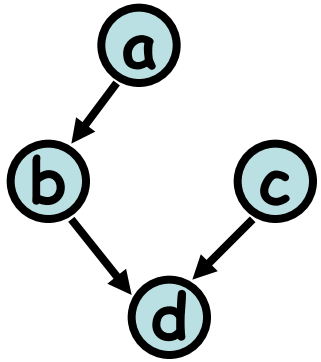
Prologのプログラム例

```
plus(0, X, X).                          /* 0+X = X */
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).
            /* X+Y=Zならば(X+1)+Y = Z+1 */


?- plus(s(0), s(s(0)), X).   /* 1+2は？ */
 X = s(s(s(0)))
?- plus(X, Y, s(s(0))).  /* X+Y=2を満たすX,Yは？*/
 X = 0, Y = s(s(0));
 X = s(0), Y = s(0);
 X = s(s(0)), Y = 0
```

Prologのプログラム例

```
path(X, X).
path(X,Y) :- arc(X,Z), path(Z, Y).
arc(a, b).
arc(c, d).
arc(b, d).
```




```
?- path(a, X).   /* aから到達可能な点は？ */
```

Prologのプログラム例(魔方陣)

```
mem(X, [X|L], L).
mem(X, [A|L1], [A|L2]) :- mem(X, L1, L2).

permutation([], []).
permutation(L, [X|L2]) :-  mem(X,L,L1), permutation(L1, L2).

magic([X11,X12,X13,X21,X22,X23,X31,X32,X33]) :-
  permutation([1,2,3,4,5,6,7,8,9],
              [X11,X12,X13,X21,X22,X23,X31,X32,X33]),
  X1 is X11+X12+X13, X1 = 15,
  X2 is X21+X22+X23, X2 = 15,
  X3 is X31+X32+X33, X3 = 15,
  Y1 is X11+X21+X31, Y1 = 15,
  Y2 is X12+X22+X32, Y2 = 15,
  Y3 is X13+X23+X33, Y3 = 15,
  Z is X11+X22+X33, Z = 15,
  W is X13+X22+X31, W = 15.
```

X11	X12	X13
X21	X22	X23
X31	X32	X33