証明と推論の機械化

久木田水生

計算・推論の機械化

- 計算の分野では古くから，ある問題に対して解を求めるための機械的な手続き（アルゴリズム）が発見され，使われてきた.
- 演繹的推論に関しては，形式的論理学の発展により，20世紀初頭には，例えば命題論理や一階述語論理の恒真式の判定や，形式的体系での証明を構成するためのアルゴリズムが発見された.
- 一方，帰納的な推論，仮説生成的推論に関しても，ベーコンやニュートン，ミルなどによって，その方法論が提案されてきた.
- 1980年代以降，コンピュータの発展とともに，帰納推論を行う機械学習システムが著しく発展した.
- 本発表では導出原理に基く，一階述語論理の定理判定アルゴリズムと，そのコンピュータによる実装を紹介する.
- 最後に導出原理を応用して作られた論理プログラミング，そしてさらにその帰納推論，仮説推論への応用について簡単に触れる.

例） ユークリッドの互除法

- 以下は，二つの自然数の最大公約数を求める手続きで，「ユークリッドの互除法（Euclidean algorithm）」として知られている.
- 1. m, nに与えられた自然数を入力して2に進む.
- 2. q := mod(n, m)と置く. 3に進む.
- 3. q = 0ならば手続きを終了する，そうでなければ4に進む.
- 4. n := m, m := qと置く. 2に戻る.

  ※ mod(n, m)はnをmで割った余りを表す.
- 手続きを終了した時点のmの値が，入力された二つの自然数の最大公約数である.

（→EuclideanAlgorithm.java）

例） 割り算のための「書き換えアルゴリズム」 (Terese [2003])

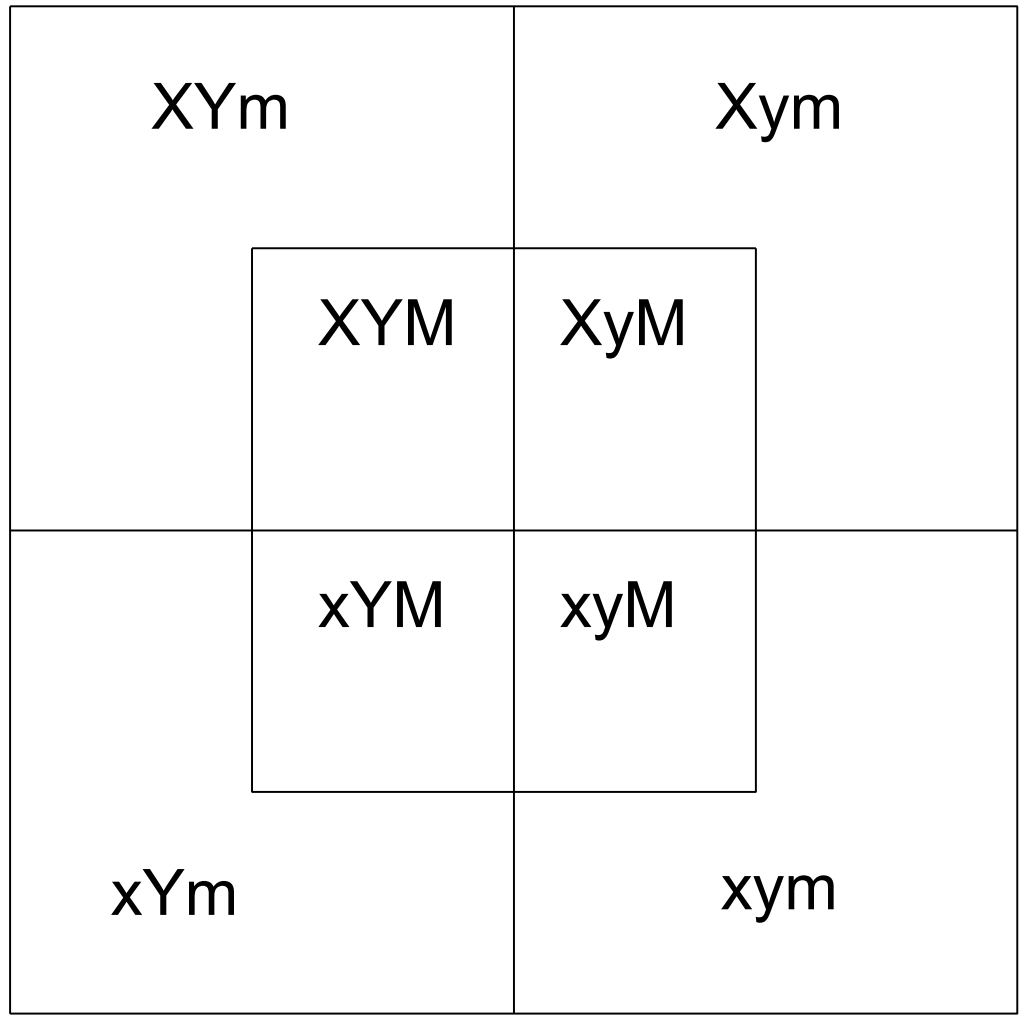
- 「n」は「S(S(...S(0)...))」（Sがn個現れる）の略記とする.
- 初期状態： quot(n, m) （ただしn >= 0, m > 0）
- 終了状態： n （ただしn >= 0）
- 規則1： minus(n, 0) → n
- 規則2： minus(S(n), S(m)) → minus(n, m)
- 規則3： quot(0, S(n)) → 0
- 規則4： quot(S(n), S(m)) → S(quot(minus(n, m),S(m)))

割り算のための「書き換えアルゴリズム」

- quot(4, 2)
  - S(quot(minus(3, 1), 2))
  - S(quot(minus(2, 0), 2))
  - S(quot(2, 2))
  - S(S(quot(minus(1, 1), 2)))
  - S(S(quot(minus(0, 0), 2)))
  - S(S(quot(0, 2)))
  - S(S(0)) = 2

例） ルイス･キャロルの論理ゲーム（Carrol [2005])

ルイス･キャロルは三段論法的推論を行う次のようなゲームを考えた.
下図のXとx，Yとy，Mとmはそれぞれある性質の存在，不在を表す.

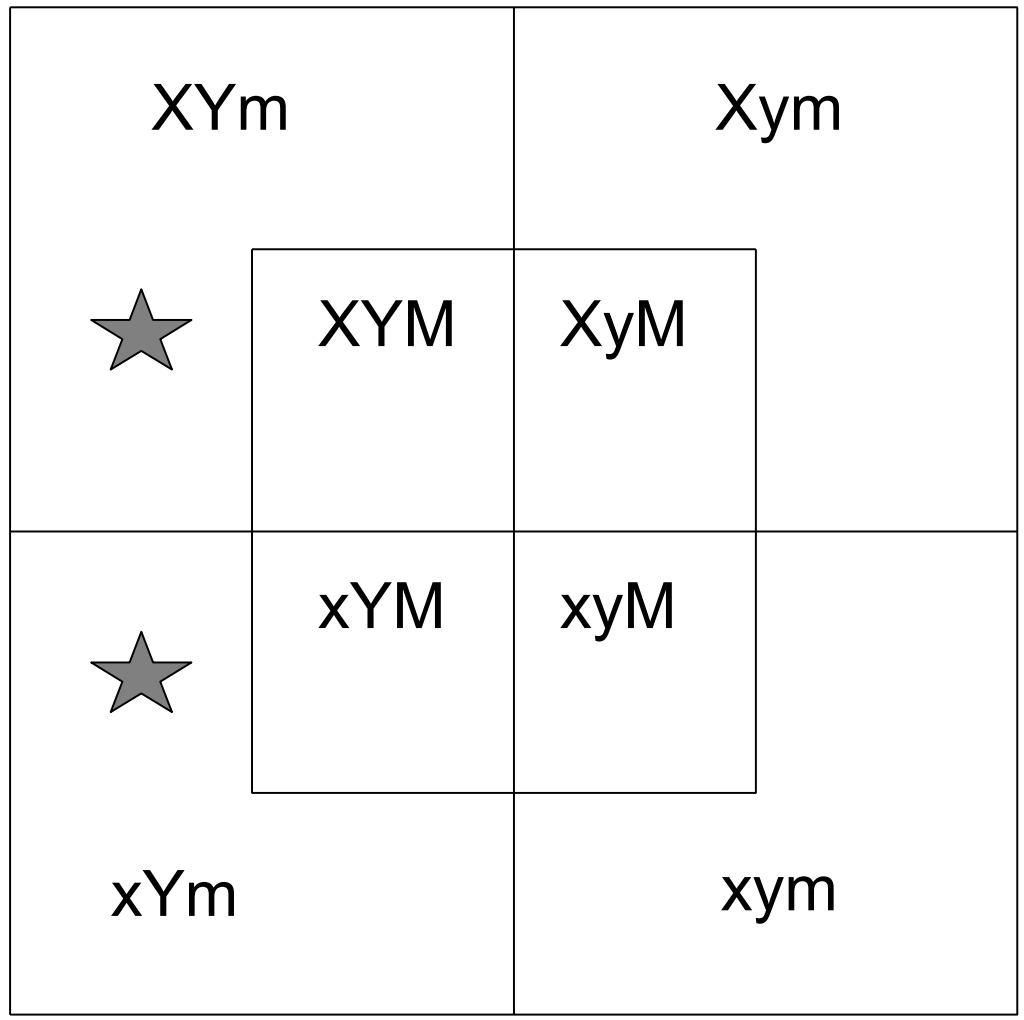


例えば左上外側の区画はXとYという性質を持ち，Mという性質を持たない対象の領域を表す.

その区画に対象が存在しないとき，そこには灰色のコマを置く.

その区画に対象が存在するとき赤いコマを置く.

ルイス･キャロルの論理ゲーム



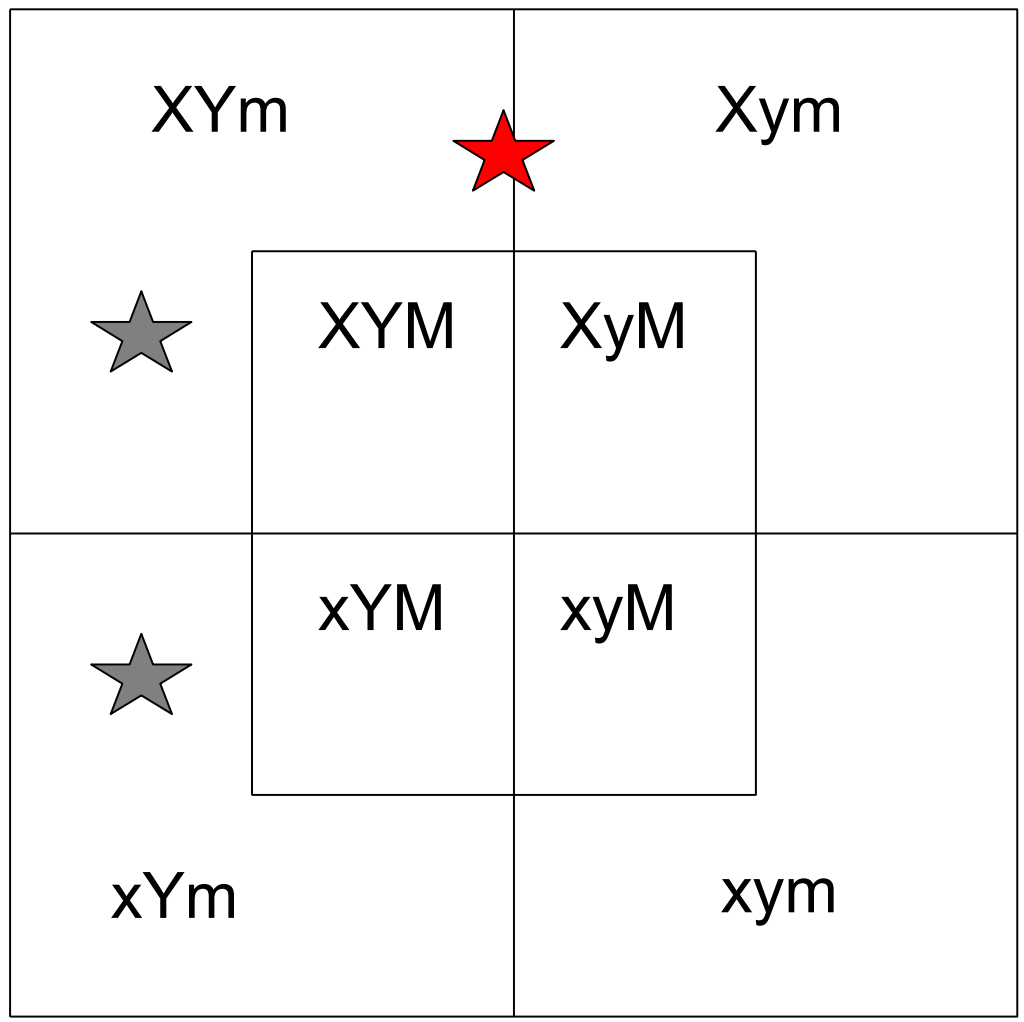
次の二つの前提が与えられたとしよう.

1　あるXはmである.

2　すべてのYはmではない.

2から，Yとmを含む区画の両方に，そこに対象が存在しないことを表す灰色のコマを置く.

ルイス・キャロルの論理ゲーム

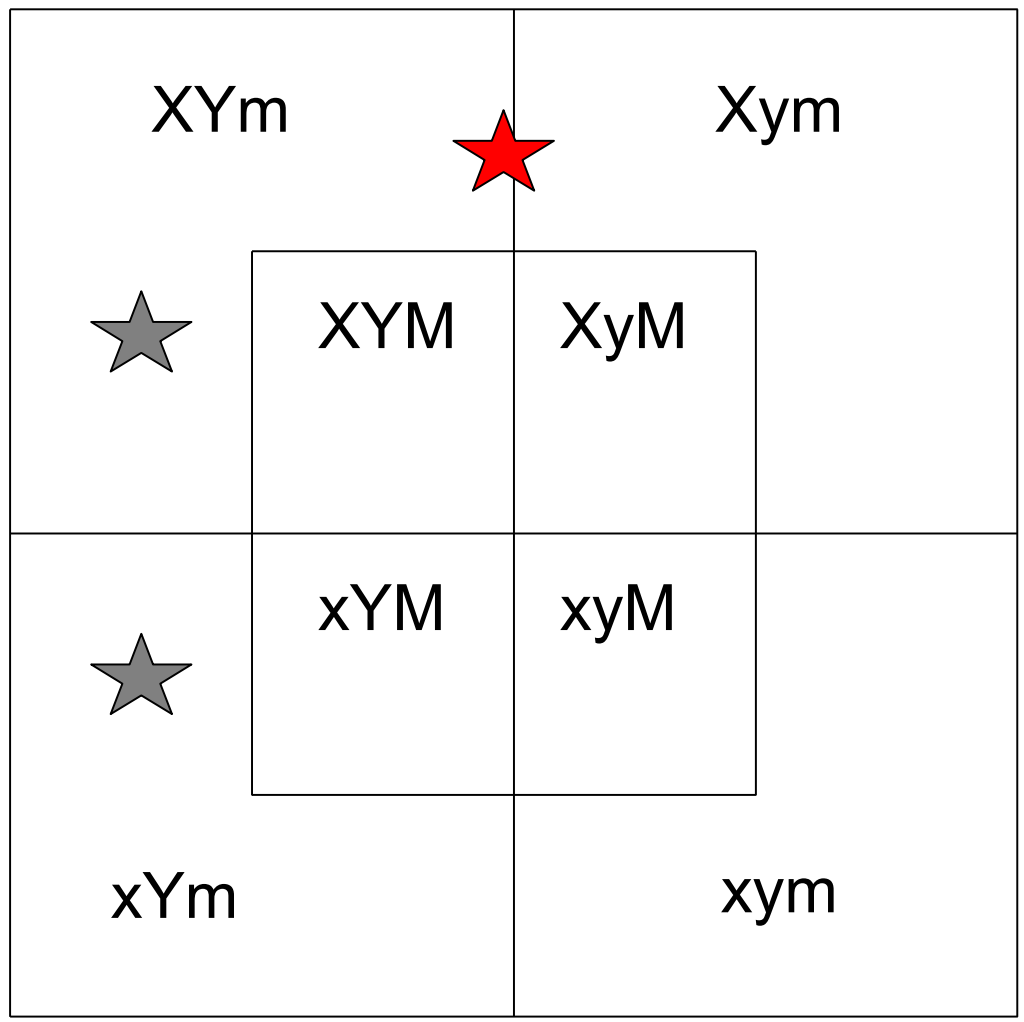


1　あるXはmである.

2　すべてのYはmではない.

次に1の前提を考慮する. Xとmを含む区画に，肯定を表す赤いカウンターを置く. このときXYmとXymの区画の境界をまたぐように置く.

ルイス･キャロルの論理ゲーム

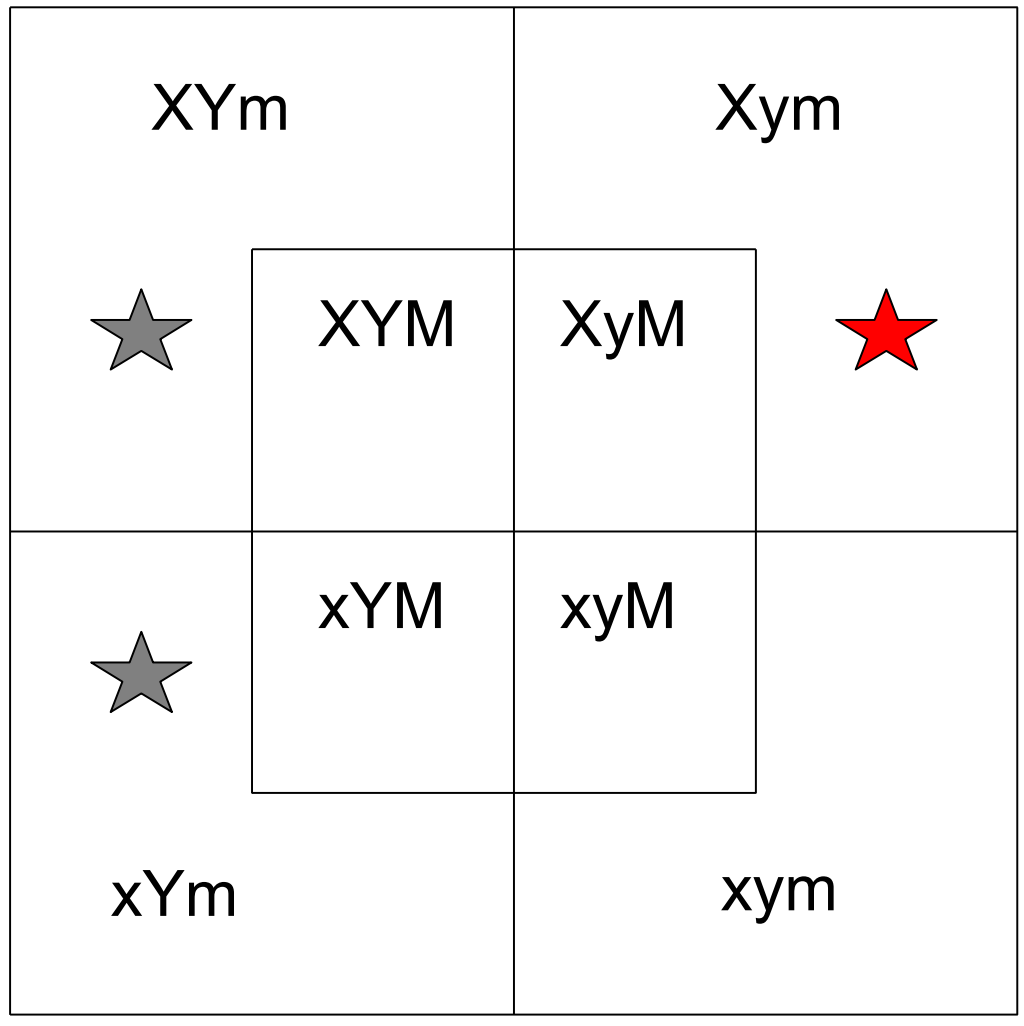


1　あるXはmである.

2　すべてのYはmではない.

しかしXYmの区画は既に灰色のコマが置かれているので，赤いコマをXymの側に移動させる.

ルイス･キャロルの論理ゲーム



1　あるXはmである.

2　すべてのYはmではない.

最後にMとmの境界線を消し，Mとmの文字を消す.

またM,mの境界の片方にしか灰色のコマがない場合はそのコマを消去をする.

ルイス･キャロルの論理ゲーム

XY	Xy
	★
xY	xy

１　あるXはmである.

２　すべてのYはmではない.

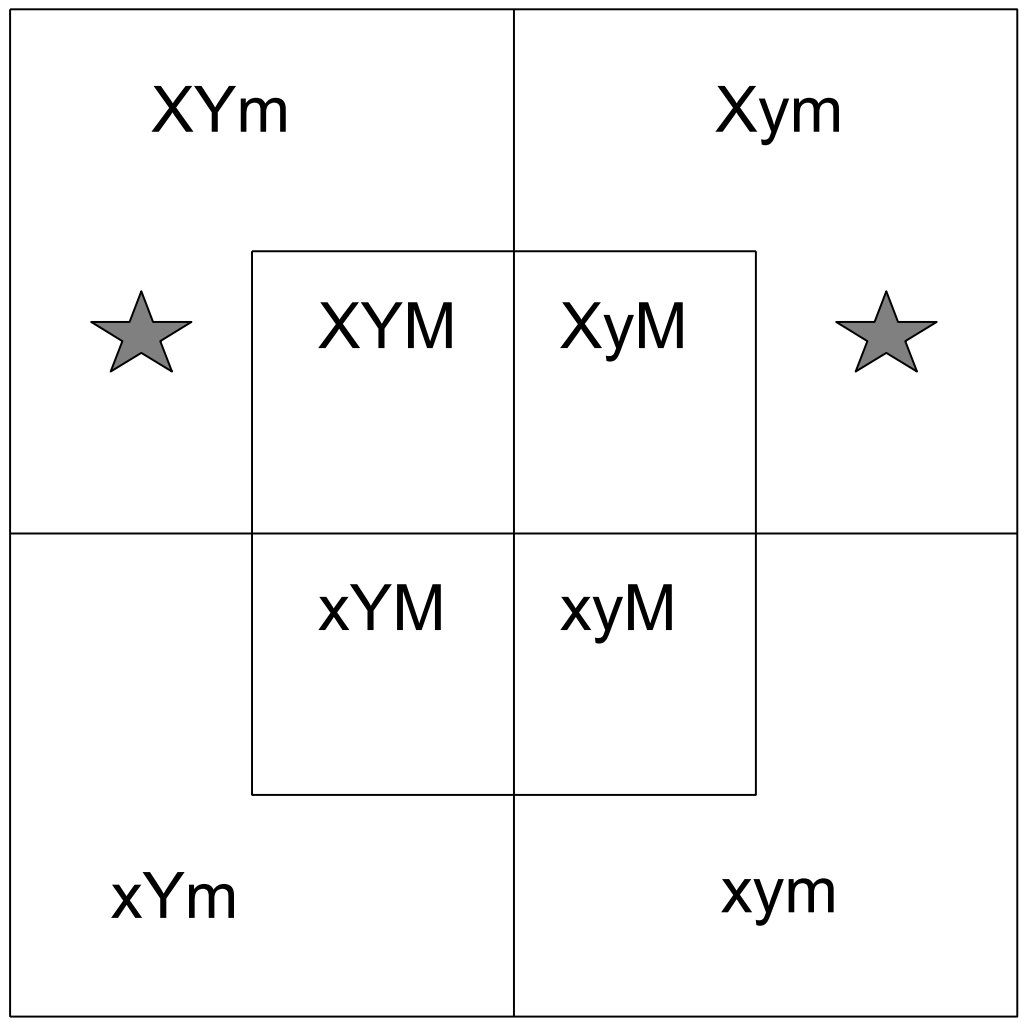
図から読み取れる結論は

あるXはyである

ということになる.

ルイス･キャロルの論理ゲーム

また次のような前提が与えられたとする.

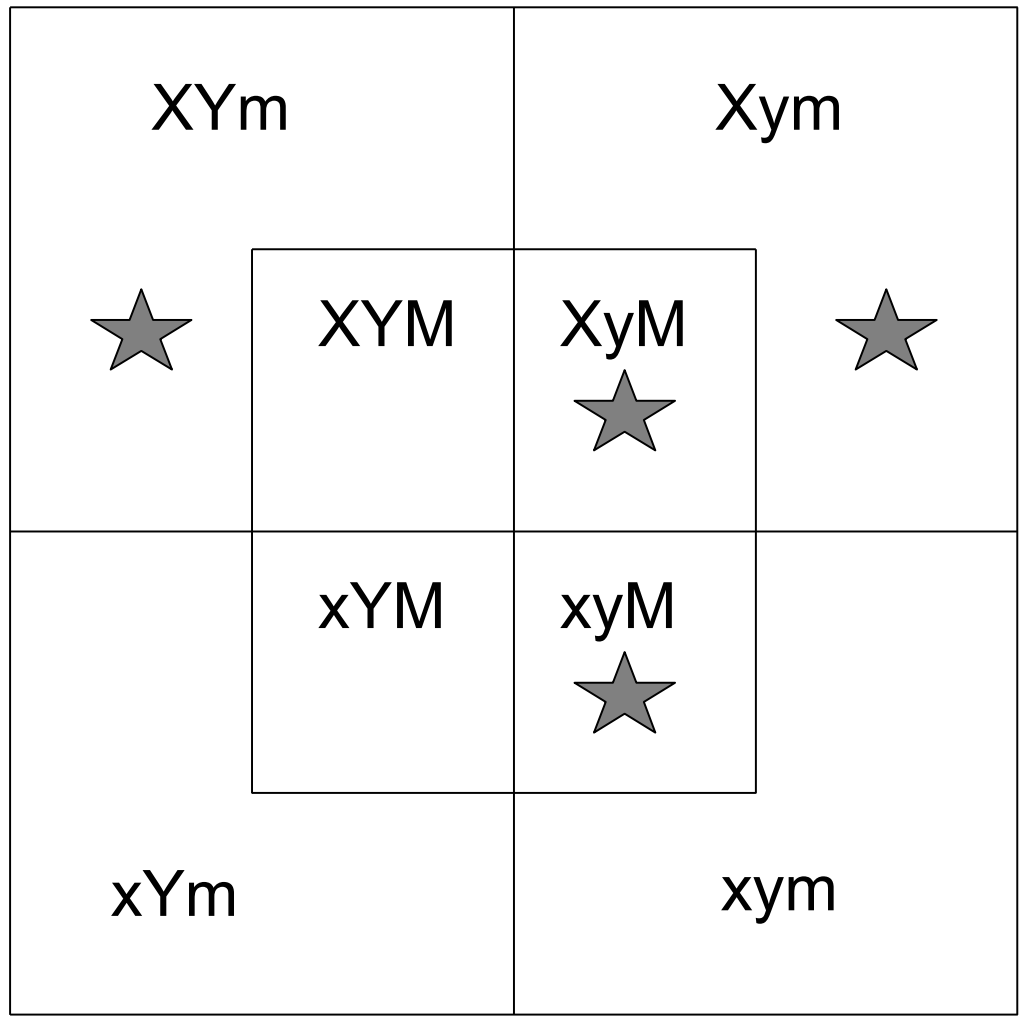


1　すべてのXはMである.

2　すべてのMはYである.

1から，Xとmを含む区画の両方に，そこに対象が存在しないことを表す灰色のコマを置く.

ルイス･キャロルの論理ゲーム

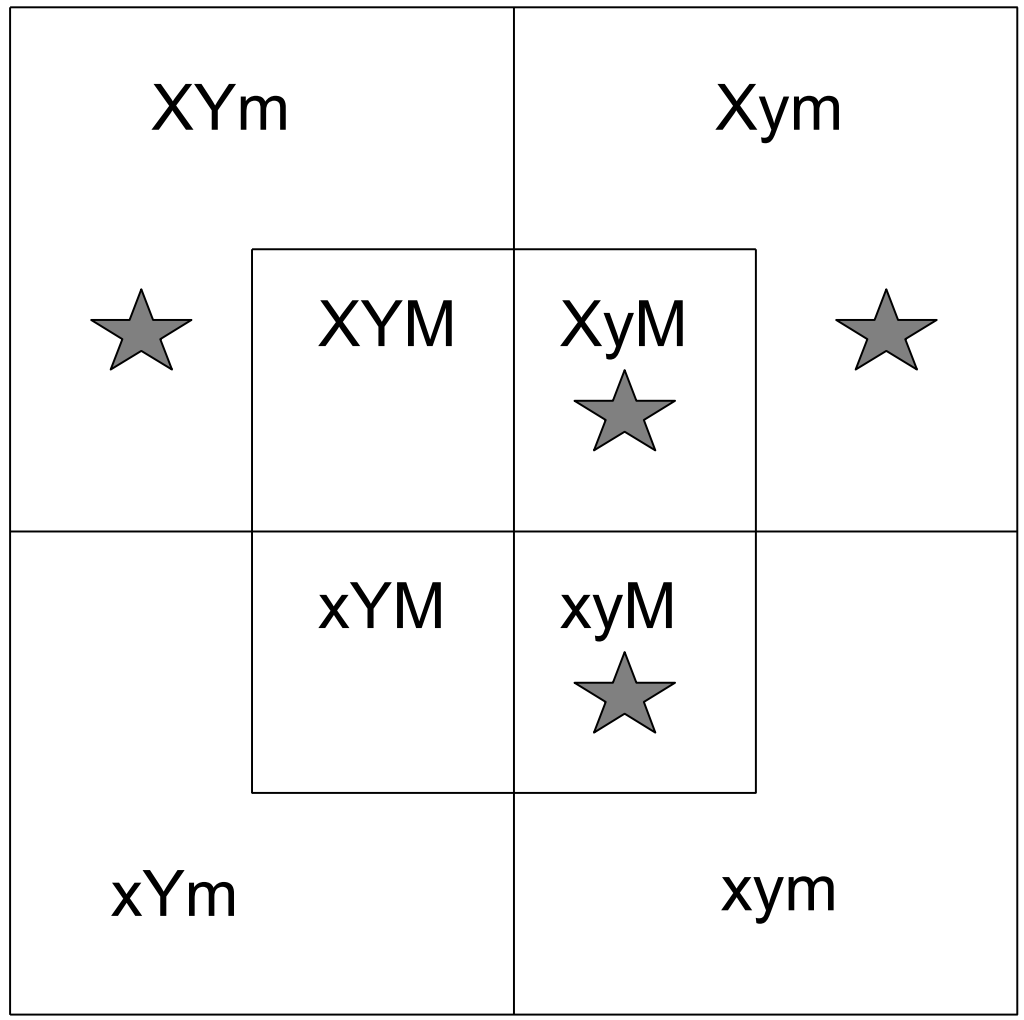


1　すべてのXはMである.

2　すべてのMはYである.

2から，Mとyを含む区画の両方に，そこに対象が存在しないことを表す灰色のコマを置く.

ルイス･キャロルの論理ゲーム



1　すべてのXはMである.

2　すべてのMはYである.

最後にMとmの境界を取り除き，
境界の両方に灰色のコマがある
区画（ここでは右上）にだけコマを
残す.

ルイス･キャロルの論理ゲーム

XY	Xy
	★
xY	xy

1　すべてのXはMである.

2　すべてのMはYである.

最後にMとmの境界を取り除き，
境界の両方に灰色のコマがある
区画（ここでは右上）にだけコマを
残す．Mとmの文字を消す．

ルイス･キャロルの論理ゲーム

XY	Xy ★
xY	xy

1　すべてのXはMである.

2　すべてのMはYである.

図から読み取れる結論は,

すべてのXはYである

ということになる.

ルイスキャロルの論理ゲーム

- ルイス·キャロルの論理ゲームは，論理的な推論を，一定のルールに従って行うゲームとして表現している.
- このゲームの図やコマの配置，そしてそれを操作するルールは私たちの知識や推論をモデル化したものであると言うことができる.
- 最初の図とコマの配置が前提を表し，最後の図とコマの配置が結論を表現している.

例）トートロジーをチェックするアルゴリズム

- 以下はエルブランによって発見された，論理式の書き換えアルゴリズムである(Terese [2003]) .

- 規則1: $p \Rightarrow q \rightarrow p \cdot q + p + 1$
- 規則2: $p \vee q \rightarrow p \cdot q + p + q$
- 規則3: $\neg p \rightarrow p + 1$
- 規則4: $p + 0 \rightarrow p$
- 規則5: $p + p \rightarrow 0$
- 規則6: $p \cdot 0 \rightarrow 0$

トートロジーをチェックするアルゴリズム

- 規則7: p･1 → p
- 規則8: p･p → p
- 規則9: p･(q + r) → p･q + p･r
- 規則10: ･と+に関する結合律，および交換律

- この規則に従って論理式を書き換え，1で終了するとき，そしてそのときに限り，最初の式はトートロジーである.

トートロジーをチェックするアルゴリズム

- $p \vee \neg p$

  - $p \cdot \neg p + p + \neg p$
  - $p \cdot \neg p + p + (p + 1)$
  - $p \cdot \neg p + (p + p) + 1$
  - $p \cdot \neg p + 0 + 1$
  - $p \cdot \neg p + 1$
  - $p \cdot (p + 1) + 1$
  - $p \cdot p + p + 1$
  - $(p + p) + 1$
  - $0 + 1$
  - 1

AIによる自動証明の試み

- ある推論を遂行するためのアルゴリズムを確立することによって，その推論を機械的な手続きにすることができる.
- そこに十分な工学的技術が加わることで，推論の自動化が可能になる.
- このような試みの一環は，論理学の定理の自動証明という形で，1950年代頃から行われるようになった.
- AIに関する研究集会，ダートマス会議（1956）では，Whitehead & Russell, *Principia Mathematica*の定理を証明するLogic TheoristというプログラムがNewell, Simon, Shawによって発表された.
- このプログラムは*PM*の第2章に挙げられた52個の定理のうち，32個の証明を見つけることができた.

節形論理

- Robinson [1965]は，一階述語論理の式が充足不可能であることを判定するための手続きを与えた.
- ここでは節形論理（clausal form logic）という特殊な論理体系が採用される.
- この手続きは，実際にその式が充足不可能であるときは有限のステップで判定結果を出す.
- 式が充足可能である場合には手続きが終了しないこともある.

節形論理：構文論と解釈（命題論理）

- 原子式，およびその否定をリテラルという.
- リテラルの有限集合（空でもよい）を節という.
- 節集合{ C1，...，Cn }は
  (∨C1)∧...∧（∨Cn)
  という連言標準形の論理式と同一視される.
- 任意の論理式はそれと論理的に同値な連言標準形を持つことに注意.

※節{L1, ..., Ln}に対して∨{L1, ..., Ln}はL1∨...∨L3という論理式を表す.

※∨{ }は矛盾⊥を表す．空節を□によって表す.

節形論理：導出原理（命題論理）

- リテラルLとL*に対して，一方が他方の否定であるとき，LとL*は相補的であるという．
- 節Cと節Dがそれぞれ，相補的なリテラルL，L*を含むとき，(C－{L})∪(D－{L*})を，CとDからのリゾルベントと呼ぶ．
- 導出原理：C1，C2は節，CはC1とC2からのリゾルベントであるとする．このとき，C1とC2から節Cを導出してよい．
- 節集合Xに，Xに含まれる任意の二つの節から導出可能なすべての節を加えた節集合をR(X)と表す．

節形論理:導出原理(命題論理)

- $R^0(X) = X$,　$R^{n+1}(X) = R(R^n(X))$　と表す.
- 任意の節集合X = {C1, C2, ... , Cn}(n >= 0)に対してc(X) は論理式$(\vee C1)\wedge(\vee C2)\wedge\ldots\wedge(\vee Cn)$を表すものとする.
- 定理(導出原理の完全性)

  あるn>=0に対して　$\square \in R^n(X)$であるのはc(X)が充足不可能であるとき, そしてそのときに限る.

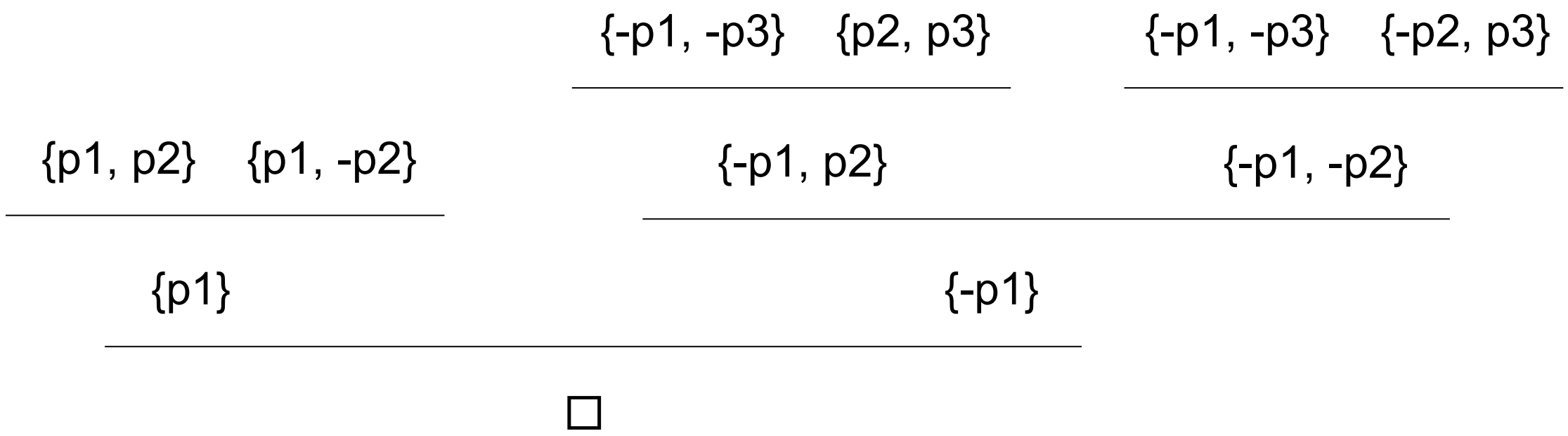
節形論理:導出原理(命題論理)

- 例） X = { {-p1, p2}, {-p2, p3}, {p1}, {-p3} }とする.
  このとき以下の導出が可能.

$$\dfrac{\dfrac{\{-p1, p2\} \quad \{p1\}}{\{p2\}} \qquad \dfrac{\{-p2, p3\} \quad \{-p3\}}{\{-p2\}}}{\square}$$

節形論理:導出原理(命題論理)

- 例） X = { {p1, p2}, {p1, -p2}, {-p1, -p3}, {p2, p3}, {-p2, p3} }とする. このとき以下の導出が可能.



(→RefutationTest1.java)

節形論理：構文論と解釈（述語論理）

- 項：
  - 変数，定数は項である．
  - n項関数に対して，引数としてn個の項の配列を与えたものは項である．
- 原子式：n項述語に対して，引数としてn個の項の配列を与えたものは原子式である．
- リテラル：原子式，およびその否定はリテラルである．
- 節：リテラルの有限集合を節という．
- 相補的なリテラル：命題論理の場合と同様．

節形論理：構文論と解釈（述語論理）

- 節集合{C1,... , Cn}に現れる変数をv1,... , vmとするとき，Xは全称冠頭連言標準形の論理式

$$\forall v1 \ldots \forall vm((\vee C1) \wedge \ldots \wedge (\vee Cn))$$

と同一視される.

- 任意の論理式Pに対して，ある全称冠頭連言標準形の論理式Qが存在して，

Pが充足不可能　⇔　Qが充足不可能

が成り立つことに注意.

節形論理:不一致集合

- 式P1とP2の不一致集合とは，それぞれの式を左から見て，最初に異なる記号が現れる箇所から始まる項を要素として持つ集合である.
- 例） 式P(v2, f1(v1, c1))とP(v2, f1(f2(v3), c1))との不一致集合は，{v1, f2(v3)}.
- 式の集合から作られる不一致集合も同様に定義される.

（→DisagreementSetTest1.java）

節形論理:置換

- 変数vと項tに対して[t / v]を置換または代入という.
- 任意の表現Eに対してE[t / v]はEに現れる変数vをすべてtに置き換えて得られる表現を表す.
- 表現の集合{E1, ..., En}と置換[t / v]に対して,

  {E1, ..., En} [t / v] = {E1[t / v], ..., En[t / v]}

  とする.
- 表現Eと置換の集合θ = { [t1 / v1], ..., [tn / vn] }に対してEθはEに現れるv1, ..., vnをそれぞれt1, ..., tnによって置き換えて得られる表現を表す.
- このようなθを同時置換(または単に置換)と呼ぶ.

(→SubstituteAtOnceTest1.java, AvoidVariableClashTest1.java)

節形論理: 置換

- 表現の集合{E1, ..., En}と同時置換θに対して,

  $$\{E1, ..., En\}\theta = \{E1\theta, ..., En\theta\}$$

  とする.
- 同時置換θ = { [t1 / v1], ..., [tn / vn] }とμ = { [s1 / u1], ..., [sm / um] }に対して, θ･μ は集合 { [t1μ / v1], ..., [tnμ / vn], [s1/ u1], ..., [sm / um] }から, ある[ti / vi]に対してvi = ujであるような[sj / uj]をすべて取り除いた集合を表すものとする.
- 任意の同時置換θ, μ, νに対して, 結合律

  $$\theta\cdot(\mu\cdot\nu) = (\theta\cdot\mu)\cdot\nu$$

  が成り立つ.

節形論理:単一化

- 表現の集合Sと置換θに対して，Sθが単元集合であるとき，θをSの単一化子と呼ぶ．また単一化子が存在するとき，Sは単一化可能であるという．
- μはSの単一化子であり，かつ任意の単一化子θに対して，ある置換vが存在してθ= μ･vが成り立つとする．このときμをSの最汎単一化子（most general unifier; mgu）と呼ぶ．
- mguは一般に複数存在するが，μとvがともにSのmguだとすると，あるθとδが存在してμ･θ = vかつv･δ = μが成り立つ．

節形論理: 単一化アルゴリズム

1. 表現の集合Sを入力する. μ:= { }, DはSの不一致集合とする.
2. Dに変数が含まれていないとき, 手続きを終了する.
3. Dに含まれる他の項の部分項になっていないような変数が存在しないとき, 手続きを終了する.
4. Dに含まれ, Dの他の要素の部分項になっていない変数vと, vとは異なるDの要素tを選び出し, μ:= (μ･{ [t / v] })と置く.
5. S:= Sμとし, Dを新たにSの不一致集合とする.
6. D = { }のとき, 手続きを終了する. そうでないときは2に戻る.

節形論理:単一化定理

- 単一化アルゴリズムが，2，3のステップで終了するとき，もとのSは単一化不可能である.
- 6のステップで終了するとき，もとのSは単一化可能であり，最後に得られたμはその単一化子である.
- さらには，このμはSの最汎単一化子である.
- 定理（単一化定理）

  表現の集合Sが単一化可能であるならば，Sは最汎単一化子を持つ.

（→UnificationAlgorithmTest1.java）

節形論理:導出原理(述語論理)

- 節C, Dに対して, θとδは変数を変数に置き換える置換であり, CθとDδに共通の変数は現れないものとする. あるL∈Cθ, M∈Dδに対して, M*はMの相補的なリテラルであり, L, M*が単一化可能であるとする. μはLとM*のmguであるとする. このとき, 節((Cθ−{L})∪(Dδ−{M}))μをCとDからのリゾルベントと呼ぶ.
- 導出原理:C1, C2は節集合, CはC1とC2からのリゾルベントであるとする. このとき, C1, C2から節Cを導出してよい.
- Xに, Xに含まれる任意の二つの節から導出可能なすべての節を加えた節集合をR(X)と表す.

節形論理:導出原理(述語論理)

- $R^0(X) = X$, $R^{n+1}(X) = R(R^n(X))$ と表す.
- 任意の節集合X = {C1, C2, ... , Cn}(n >= 0)に対して, c(X) は論理式
  $\forall v1 ... \forall vn((\vee C1) \wedge (\vee C2) \wedge ... \wedge (\vee Cn))$を表すものとする. ただしv1,..., vnはXに現れる変数のすべてであるとする.
- 定理(導出原理の完全性)

  あるn>=0に対して □ $\in R^n(X)$であるのはc(X)が充足不可能であるとき, そしてそのときに限る.

（→RefutationTest3.java）

節形論理:決定可能性，半決定可能性

- 命題論理の場合は，任意の節集合Xに対してあるn >= 0が存在して，$R^n(X) = R^{n+1}(X)$となる．つまりある段階において，それ以上導出原理によって新しいリゾルベントを導出することができなくなるということである．従ってこの段階で□ ∈ $R^n(X)$でなければc(X)が充足可能であるということが分かる．
- 従って，導出原理は命題論理に対する完全な決定手続きになっている．
- 述語論理では，c(X)が充足可能である場合，どのn >= 0に対しても$R^n(X)$から新しいリゾルベントが導出できる可能性がある．
- 一方c(X)が充足不可能であるときには必ずあるn >= 0に対して□∈$R^n(X)$が成り立つ．
- 従って，導出原理は述語論理に対しては，半決定的な手続きということになる．

（→RefutationTest4.java, RefutationTest2.java）

論理プログラミング

- 節形論理と導出原理は，ある背景知識のもとで，ある問題に対する解を求めるアルゴリズムに応用されている．それがPrologに代表される論理プログラミングである．
- 正のリテラルを一つだけ含む節をホーン節と呼ぶ．正のリテラルをそのホーン節の頭という．それ以外の節を本体と呼ぶ．
- 正のリテラルを含まない節をゴール節という．
- ホーン節のみからなる有限節集合を論理プログラムと呼ぶ．

論理プログラミング

- 以下は論理プログラムである．X，Y，Zは変数，それ以外の項は定数であるとする．

{

{ grandparent(X, Y), - parent(X, Z), - parent(Z, Y) },

{ parent(john, bob) },

{ parent(john, beth) },

{ parent(jane, peter) },

{ parent(beth, tom) },

{ parent(bob, paul) }

}

論理プログラミング

- ユーザは質問を与えることによってこのプログラムを利用することができる．例えば先ほどのプログラムに，

  ?- grandparent(john, paul).

  という質問を打ち込むと，インタプリタはゴール節

  { - grandparent(john, paul) }

  をプログラムに与え，導出原理を適用する．
- □が導出されたらインタプリタはyesを，そうでないときはnoを出力する．
- 従ってこの場合はyesが出力される．

論理プログラミング

- またユーザは例えば

  ?- grandparent(john, W).

  という質問を打ち込むこともできる.
- このときインタプリタはゴール節

  { - grandparent(john, W) }

  をプログラムに与え，導出原理を適用する.
- □が導出されたらインタプリタは，このゴール節に適用されたすべての同時置換に含まれる置換[t / W]のtを出力する.
- 従ってこの場合は，tomとpaulが出力される.

論理プログラミング

- ここで，インタプリタは

  grandparent(X, Y) ← parent(X, Z), parent(Z, Y)

  parent(john, beth)

  parent(beth, tom)

  - grandparent(john, W)

  から，一つには，同時置換{ [john / X], [tom / Y], [beth / Z], [tom / W] }を施すことによって導出原理により口を導出した.
- 従って答えの一つはtomになっていた.
- もし最初の質問が ?-grandparent(tom, W) であったらどのような代入によっても口は導出されないので，答えはnoになる.

論理プログラミング

- このプログラムを与えられたインタプリタは

  ?- grandparent(john, peter)

  という質問に対してはnoを出力する．ゴール節

  { - grandparent(john, peter) }

  をこのプログラムに加えても，□が導出されないからである．
- これを，質問ではなく，説明されるべき事実として与えるというアイディアがある．つまり現在のプログラムに，さらにどのようなホーン節を加えたら，与えられたゴール節から□が導出されるかをコンピュータに推論させるのである．
- これはコンピュータによる帰納的推論（induction），もしくは仮説的推論（abduction）であるといえる．

帰納論理プログラミング，仮説論理プログラミング

- この例では，ゴール節

  { - grandparent(john, peter) }

  に対しては，

  parent(john, jane)

  が一つの可能な仮説（の一つ）である．
- 論理プログラミングの枠組みを利用して，導出原理の逆演算として，帰納的，仮説的推論を行わせるのが，帰納論理プログラミング（inductive logic programming）や，仮説論理プログラミング（abductive logic programming）と呼ばれ，近年発展しているAIの分野である．
- ALP，ILPはエキスパート・システムや機械学習といったAIの分野で大きな成功を収めている．

帰納推論の機械化，帰納論理の実現？

- ベーコンやミルなどは，帰納推論や仮説推論の方法の定式化を試みた.
- しかし，一般的には，このような推論は，決まったルールに基くような論理の問題ではなく，科学的に重要な法則の発見は科学者の直観によるしかないという見解が強い.
- 一方Gillies (1996)はILPに代表される機械学習システムは，帰納的推論を機械化した成功例であり，歴史上初めての帰納論理の体系の実現であると評価している.

ILPのさらなる応用

- ILPは現在も目覚しい発展を続けている．例えば，次のような研究が現在行われている．
  - 述語論理と確率論の統合への応用．
  - 遺伝的アルゴリズムの導入．
  - 倫理的判断を行うエキスパート・システムへの応用．
- こういったことが，論理や計算の哲学に対するどのようなインパクトを持つのかについての判断は，今後の発展を待ちたい．

文献

- Carrol, Lewis (2005).『不思議の国の論理学』, 柳瀬尚紀編訳, ちくま学芸文庫.
- Kowalski, R. (1979). *Logic for Problem Solving*, Elsevier North Holland, Inc.（1992, 浦昭二監修, 山田眞市・菊池光昭・桑野龍夫訳,『論理による問題の解法』, 培風館.）
- 古川康一・尾崎知伸・植野研 (2002).『帰納論理プログラミング』, 共立出版.
- Gillies, D. (1996). *Artificial Intelligence and Scientific Method*, Oxford University Press.
- Muggleton, S. (1992). `Inductive Logic Programming', in Muggleton (Ed.), in *Inductive Logic Programming*, Academic Press (pp. 3-27).
- Muggleton, S. & Buntine, W. (1992). `Machine Invention of First-order Predicates by Inverting Resolution', in Muggleton (Ed.), *Inductive Logic Programming*, Academic Press (pp. 261-280).
- 小野寛晰 (2005).『情報科学における論理』, 日本評論社.
- Robinson, J. A. (1965). `A Machine-oriented Logic Based on the Resolution Principle', in *Journal of the Association for Computing Machinery, 12, no. 1*, (pp. 23-41).
- Terese (2003). *Term Rewriting Systems*, Cambridge Tracts in Theoretical Computer Science 55.
- 内井惣七 (1995).『科学哲学入門』, 世界思想社.
- 結城浩 (2005).『Java言語プログラミングレッスン』（上）（下）, ソフトバンク　クリエイティヴ.

付録） 形式的体系Measures

- 項:
  1. 一はMeasuresの項である.
  2. XがMeasuresの項であるとき，X○はMeasuresの項である.
- 従って，一，一〇，一〇〇，一〇〇〇等がMeasuresの項である.
- 式： X, YがMeasuresの項であるとき,

  X | Y

  はMesuresの式である.

形式的体系Measures

- 公理: ー｜ー
- 導出規則1: ー〇^{n}|ー から ー〇$^{n+1}$|ー を導出して良い.
- 導出規則2: ー〇^{n}|ー〇^{m} から ー〇^{n}|ー〇$^{m+n}$ を導出して良い.

ただし任意のn>0に対して「 〇^{n}」は〇をn個並べた記号列を表す.

形式的体系Measures

- 以下はMeasuresにおける証明である
  1. ー | ー
  2. ーO | ー
  3. ーOO | ー
  4. ーOOO | ー
  5. ーOOO | ーOOO
  6. ーOOO | ーOOOOOO
  7. ーOOO | ーOOOOOOOOO

形式的体系Measures

- Measuresに関して次の性質が成り立つ.
  1. ー〇^{n}|ー〇^{m}がMeasuresの定理ならば, nはmの約数.
  2. nがmの約数ならば, ー〇^{n}|ー〇^{m}はMeasuresの定理.
- 1は, Measuresが自然数(0を含む)上の約数関係に対して健全であることを, 2は完全であることを述べている.

形式的体系Measures

- Measuresの健全性と完全性が示されたならば，この体系が決定可能であることは容易に示される.
- 実際，任意の式に対して｜の左に現れる〇の数と右に現れる〇の数を数え，前者が後者の約数になっていれば，その式は定理である．そうでなければ，その式は定理ではない.
- また任意のMeasuresの定理に対して，その証明を構成する手続きも容易に考えられる.
- Measuresに対しては，その定理を自動的に証明するプログラムが書ける.

形式的体系Measures

- ⊢○^{n} | ⊢○^{kn} を証明する手続き

1. まず導出規則1をn回適用する. すると
   ⊢○^{n} | ⊢ が得られる.
2. 次に導出規則2をk回適用する. すると
   ⊢ ○^{n} | ⊢○^{kn} が得られる.

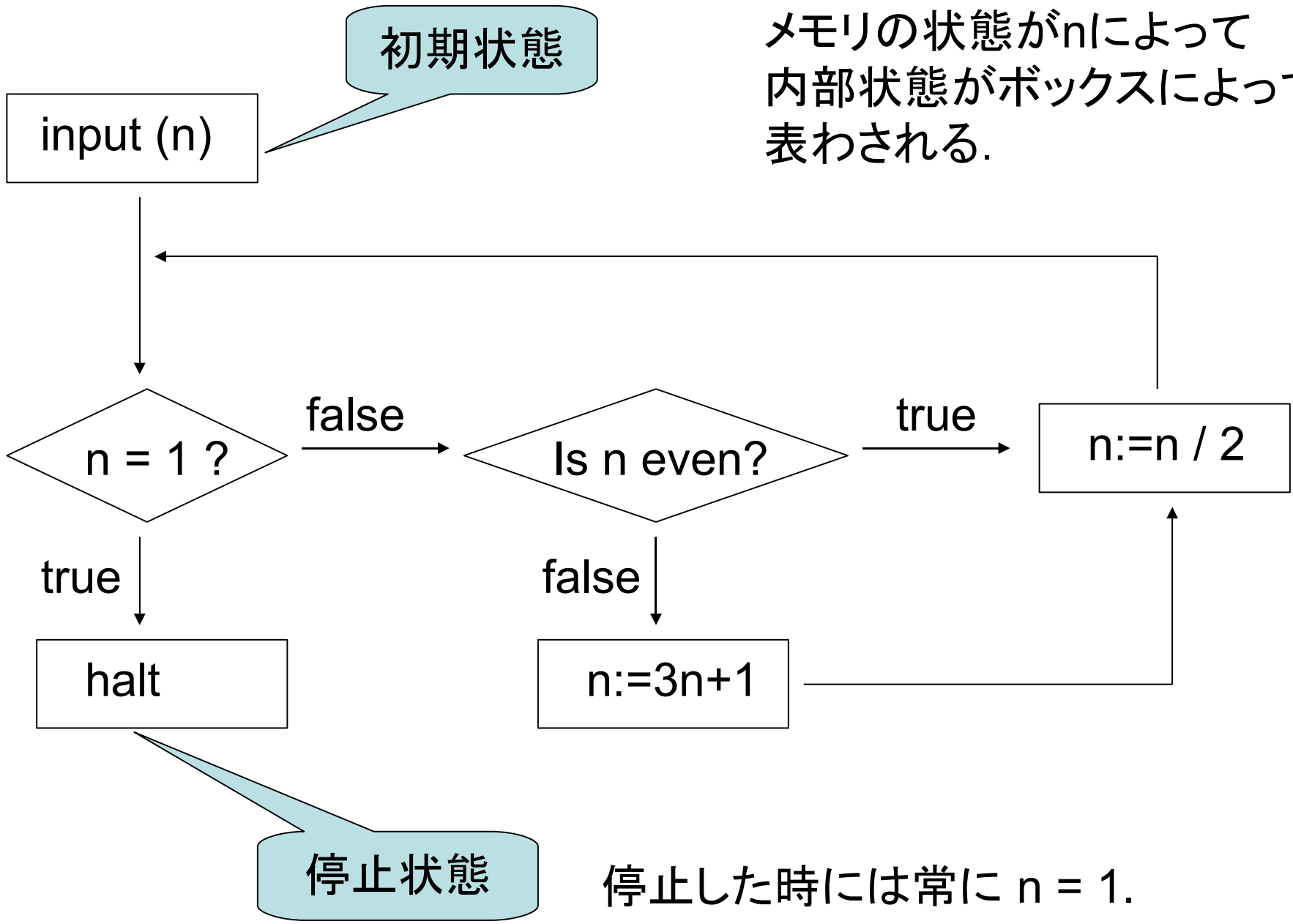
（→Measures.java）

付録）「シラキュース問題」

- 次の手続きを「シラキュース・アルゴリズム」と呼ぶ.
  1. 正整数nを受け取る.
  2. n = 1ならば停止する. そうでなければ3行目に行く.
  3. nが偶数ならば4行目, そうでなければ5行目に行く.
  4. n:= n / 2として2行目に行く.
  5. n:= 3*n + 1として4行目に行く.
- この計算では, nの値(メモリ)と, いま実行している命令が何行目であるかということ(内部状態という)がシステムの状態を決定する.

(→SyracuseProblem.java)

「シラキュース問題」



メモリの状態がnによって
内部状態がボックスによって
表わされる.

停止した時には常に n = 1.

「シラキュース問題」

input (n:= 7)

- n:= 3*11 + 1 = 34
- n:= 34/2 = 17
- n:= 3*17 + 1 = 52
- n:= 52/2 = 26
- n:= 26/2 = 13
- n:= 3*13 + 1 = 40
- n:= 40/2 = 20
- n:= 20/2 = 10
- n:= 10/2 = 5
- n:= 3*5 + 1 = 16
- n:= 16/2 = 8
- n:= 8/2 = 4
- n:= 4/2 = 2
- n:= 2/2 = 1
- halt.

この計算では正規形は数1のみである. 上の例では数1にたどりついて計算が終了している. しかしすべての正整数についてこの計算が1にたどりつくかどうかは知られていない. これがシラキュース問題である.