解集合プログラミングの宣言的問題解決に関する計算機実験

文理学部情報科学科
5419045 高林 秀

2022 年 1 月 10 日

概要

本稿は、今年度論理と計算 2 の課題研究として、具体的な問題に対して ILAPS システムを用いて解答するものである。本稿前半部では、解答に必要な計算理論の説明を行う。後半部では、実際に ILAPS システムを使用して、与えられた問題に回答していく計算機実験を行う。

目次

1 目的

本稿は今年度論理と計算 2 の第 3 回目の課題研究として、ILASP システムを用いた宣言的問題解決を通じ，解集合プログラムに基づく帰納推論に関する学修内容を振り返ることを目的とする。必要な計算理論の説明を通して学習内容の復習を図るとともに、本稿後半部に記載する問題の計算機実験を通して、内容の定着を図るものとする。

2　計算理論説明

この章では、今回の計算機実験に使用した各計算理論の解説を行う。前提となる論理記号等の定義は、下記レポートを参照されたい。

- 命題論理に関するレポート:https://drive.google.com/file/d/1p_jezNTWJEisTFzSJxPcckGNfjLirWUJ/view?usp=sharing
- 述語論理に関するレポート:https://drive.google.com/file/d/1-WFu3JJW4gAdWUn3gqoyokSYy8N1S1hr/view?usp=sharing

2.1　推論の概要

推論とは、デジタル大辞泉には以下のように記されている。

> ある事実をもとにして、未知の事柄をおしはかり論じること。「実験の結果から推論する」

すなわち、現在知っている事実。知識を元に新たな事実を導くことを示す。

推論にはいくつかその手法により種類が存在する。

- 演繹推論
- 帰納推論
- 発想推論
- 類推推論

■演繹推論　演繹推論は単に演繹法とも呼ばれ、後述する帰納法とは反対の推論手法となる。一般的、すなわち普遍的な事実やルールを前提（条件）とし、特定の場合、ケースに適用して結論を得る推論手法である。

具体例を以下に示す。既知の普遍的な事実として以下 2 つのルールが与えられているとする。

1. パソコンは電気を使う。
2. 電気を使うのは機械である。

この 2 つのルールから、次の新たなルール、事実が導き出せる。

パソコンは機械である。

いま、前提 1,2 から上記の新たな事実を導いた。このように、前提となる事実・ルールから新たな事実を論ずるのが演繹推論である。

ただし演繹推論では、前提に偏った観点や、論理が混在した場合、その論理は成立しなくなることに注意が必要である。前提の論理が正しく確立していれば、強力な論理として成立させることが可能な推論方法である。

■帰納推論　帰納推論は単に帰納法とも呼ばれ、既知の事実や事例から読み取れる傾向を総合し結論を論ずる推論方法である。特定のケース、条件と結論のセットからあるルールを導出する。

具体例を示す。既知の事実として以下の情報が与えられているとする。

1. 朝のニュース番組で原油価格についての報道があった。加えて、近所のガソリンスタンドの 1 リッターあたりの単価が以前より高くなっていた。
2. 友人からもガゾリン代が高くなったので車での外出は控えているという話を聞いた。

以上 2 つの既知の事実から、全国的にガソリン価格が高騰している、という結論が導き出せる。このように、既知の事実から同一の傾向を抽出し結論を導く、これが帰納推論である。つまり、ケースと結論の対関係からルールを導出するということである。

詳細については後述する。

■発想推論　発想推論とは、普遍的なルールと結論から、あるケース、条件を導出する推論手法である。これまでの推論では、あくまで前提から結論を導出していたのに対して、この発想推論では前提部のケース、条件を結論から推論する。詳細は後述する。

■類推推論　デジタル大辞泉に、類推の意味が次のように書かれている。

1. 類似の点をもとにして、他を推しはかること。「過去の事例から類推する」
2. 論理学で、二つの事物の間に本質的な類似点があることを根拠にして、一方の事物がある性質をもつ場合に他方の事物もそれと同じ性質をもつであろうと推理すること。結論は蓋然的。類比推理。類比。比論。アナロジー。
3. ある語形または文法形式との関連から、本来の語形または文法形式とは別の新しい語形または文法形式を作ろうとする心理的な作用。この種の働きによって、多くの不規則な語形が規則化されていくことがある。

これまでの推論手法は、基となる事実があり、そこから結論またはルールを導き出す。これに対し、類推推論では、似たような事実から結論を導き出すということを行う。すなわち、推論をする際に用いる事実や知識が他の推論手法と異なっているという点で違いがある。

2.2　発想推論 (abduction) の説明

発想推論とは、一言で述べると仮説の形成である。a を前提（仮定）,b を結論としたとき、b にある規則「a ならば b」を当てはめて a を推論する。帰納推論が仮定と結論の傾向から、ルールや規則を導出したり、演繹推論が、前提となる事実・ルールから新たな事実を論ずるのとは異なり、発想推論は結論と規則から「仮定 a」を求めるという点で大きな違いがある。

■形式的定義　発想推論の形式的定義として、論理式 P と観測事実 G に関して $P \nvDash G$ であるとき、$(P \cup \Delta \models G) \wedge (p \cup \Delta)$ は矛盾しないことを満足させる、論理式の集合 Δ を求める、というように示すことができる。P に Δ を保管することにより、G を説明、証明することができる。

Δ を「発想的説明」と呼ぶことがある。加えて、$P \cup \Delta$ を「発想的拡張」と呼ぶことがある。Δ に入れることができる文の区分（クラス）を「候補仮説集合」と呼ぶ。このとき、Δ の条件として以下の点を満たさなけ

ればならない。

- 候補仮説集合の要素はすべて原子文であること
- 与えられた一貫性制約を満たすこと
- 説明が基本的であること
- 説明が極小 (minmal) であること

■発想推論の注意すべき点　発想推論において注意すべき点としては、推論して導出した仮定 a が真理値的に真 (true) であることが保証されないという点である。これに関しては次の具体例で説明する。

以下のようにそれぞれ記号の意味を定めるとする。

- na:アリバイなし
- c:犯人である
- ルール:$c \Rightarrow na$(犯人であるならばアリバイなし。)

このとき、発想推論を行い前提（仮定）部分の c を導出してみる。節集合として $\{c, c \Rightarrow na\}$ と $\{na, c \Rightarrow na\}$ があり以下のような伴意関係が導ける。

$$\{c, c \Rightarrow na\} \models na \cdots 1$$
$$\{na, c \Rightarrow na\} \models c \cdots 2$$

このとき、式 1 に関しては、節集合のどれか一つが true であるとき、例えば、$c = true$ すなわち犯人であることが真であるときは、必然的に犯人であるので「na アリバイなし」は成立する。したがって、式 1 に関しては成立する。しかし、式 2 に関して、$na = true$ すなわち「アリバイがない」という条件だけでは犯人であるか否かを決定することができない、すなわち c が $true$ か $false$ であるかを決定することはできないので、式 2 に関しては成立できない。したがって、仮に発想推論の結論として仮定 a を求めたとしてもその結論が真であるか否かは保証されない。よって求めた仮定が本当に正しいかどうかに関しては別で議論する必要がある。

2.3　帰納推論の説明

先に示したように、帰納推論は既知の事実や事例から読み取れる傾向を総合し結論を論ずる推論方法である。特定のケース、条件と結論のセットからあるルールを導出する。

フランシス・ベーコンによって提案された概念で、これをより人間学に近いよう噛み砕いたものは特に「ジョン・ロックの経験論」と呼ばれる。

帰納推論は、その手法から概念学習と称されることがある。個別事実の細部を無視し、共通して見られる部分や傾向を抽出することで一般的な概念を学習する。これは帰納学習とも呼ばれる。

一般的に、帰納とは人間のような知的判断力を有する生物が行動を学習する原理を定式化したものである。以下に他の推論との比較表を挙げる。（引用元 : Wikipedia「帰納」`https://ja.wikipedia.org/wiki/%E5%B8%B0%E7%B4%8D`）

	演繹推論	狭義的な帰納推論	発想推論	類推推論
例	前提 l ：a ならば b である 前提 2 ：a である 結論：b である	前提 l ：a1 は P である 前提 2 ：a2 も P である 結論：(たぶん) すべての a は P である	前提 l ：a である 前提 2 ：H と仮定すると a が説明できる 結論：(たぶん) H である	前提 l ：a は P である 前提 2 ：b は a と似ている 結論：(たぶん) b は P である
情報量増加	×	○	○	○
真理保存性	○	×	×	×

表 1　推論手法の比較

ここには書かれていないが、表内の補足事項として以下を挙げる。

演繹推論における情報量増加が起きないことに関しては、結論の内容はすべて前提の内容に含まれていることが理由である。また、真理値保存性に関しては、妥当な演繹推論は前提が正しければ必ず結論は正しいからである。

他の推論方式に関して、まず情報量の増加が起こることに関しては、結論が、前提に含まれていた内容を超える内容を持つのが理由である。また、真理値保存性に関しては、先に示した発想推論のように前提が正しくても、結論の正しさは保証されないのが理由である。

また、帰納推論の欠点として、推論の際に複数の事実や事例をすべて考慮しなければ証明したことにならないという欠点がある。

■帰納学習の例　これまでの講義の中から帰納学習に該当する機械学習がある。それは決定木学習である。決定木は、各属性の分岐条件に対する true,false による分類で構成される木構造を用いたルール表現であり、得られたルール（木構造）から入力が何に分類されるか推論することができる。すなわち、決定木は単に機械学習の一手法というだけでなく、命題論理に置き換えれば、帰納学習を使用した学習器、学習モデルと言える。

また、余談だが帰納学習を用いた具体例として、Google が開発している「alphaGo」が挙げられる。盤面における適切な手や勝利条件をもとに結論を導いている点において帰納学習が大きく関わっている。

2.3.1　帰納論理プログラミング（Inductive logic programming）

帰納推論を使用した論理プログラミングを帰納論理プログラミングと呼ぶ。コンピュータが理解できる知識表現の範囲で帰納推論を行うための手法の一つとしてこの帰納論理プログラミングが挙げられる。帰納論理プログラミングでは、様々な分類問題を扱うことができるのに加え、背景知識を利用した事例、事実の一般化を行える。また、ある学習結果を別の学習に利用することができる転移学習も行える。加えて、帰納推論の特徴である、事例間の共通した傾向、パターンの抽出を行える。

帰納論理プログラミングを用いる例として、家族関係の学習や、得られたルールを利用して論理プログラムを自動的に合成したりできる。

■問題設定　論理プログラミングにおける問題設定とは、入力と出力に関する論理的関係を示したもので、得られる解が満たすべき論理的性質を定めたものを意味する。その設定の内容により、以下のように区分されている。

- 伴意からの学習（Learning from Entailment）
- 解釈からの学習（Learning from Interpretation）
- 充足可能性からの学習（Learning from Satisfiability）
- 解集合からの学習（Learning from Answer sets）

問題設定を行う上で必要な事項を挙げると、まず初めに与えられた問題の形式的な記述表現が求められる。すなわち、入力値としてどう表現するか設定する必要がある。次に、共通パターンの形式的記述すなわち、出力で得られる仮説をどのように表現するか設定する。次に、各事例に特定のパターンが出現するか確認し、仮説空間と設定を行う。最後に得られる仮説に関して、評価関数を使用して仮説の順位付けをする。

なお今回の実験で使用するのは解集合からの学習（Learning from Answer sets）であるので、ここでは問題設定の一例として伴意からの学習のみ説明する。問題設定は下記。

- 入力：正例 E^+、負例 E^-、関連知識 BK
  ※すべて述語論理で表現。また、$BK \nvDash E^+$ （背景知識のみでは正例を説明できない）。
- 出力：仮説 H
  ※すべて述語論理で表現。H は以下の条件を満たす。

  (1).$BK \cup H \models E^+$
  (2).$BK \cup H \nvDash E^-$

■補足　数学的帰納法や、構造的帰納法などの手法は、名称に帰納とついているが、手法としては演繹に当たる。

2.4　解集合からの学習

標準論理プログラム P は、複数個の安定モデルないしは解集合をもつ。P の解集合全体の集合を $AS(P)$ と表記する。標準論理プログラム P の帰納推論に関する問題設定には以下のものが存在する。

- Cautious Induction
- Brave Induction
- Induction of Stable Models
- Learning from Answer sets(解集合からの学習)

■Cautious Induction　問題設定は下記。

- 入力：背景知識 B、仮説空間 S_M、正例の集合 E^+、負例の集合 E^-
- 出力：仮説 H
  ※ H は以下の条件を満たす。

  (1).$H \subseteq S_M \cdot H$ は仮説空間
  (2).$AS(B \cup H) \neq \phi \cdot B$ と併せた際に解集合をもつ
  (3).$\forall A \in AP(B \cup H)[E^+ \subseteq A, E^- \cap A = \phi]$　　·すべての解集合は正例のみを含み負例を含まない

■Brave Induction　問題設定は下記

- 入力 (Cautious Induction と同一)：背景知識 B、仮説空間 S_M、正例の集合 E^+、負例の集合 E^-

- 出力：仮説 H
  ※ H は以下の条件を満たす。

  (1).$H \subseteq S_M \cdot H$ は仮説空間
  (3).$\exists A \in AP(B \cup H)[E^+ \subseteq A, E^- \cap A = \phi]$　　·正例のみを含み負例を含まない解集合が存在する

以上の 2 つの問題設定における違いは、出力される仮説の条件 (3) にある。Cautious Induction では、A は全称限量され解集合すべてが条件を満たさなければならない。Brave Induction では存在限量され、一部の解集合にて条件を満たせば良い。

■Induction of stable models　複数の解釈からの学習を行うには、Induction of stable models と呼ばれる問題設定を使用する。基礎アトム（基礎原子式）を e とすると、部分解釈は e^{inc}, e^{exc} で示され、それぞれ以下を示す。

- e^{inc}:解釈に含まれるべきアトムの集合
- e^{exc}:解釈に含まれるべきではないアトムの集合

このとき、ある解釈 I が条件「$e^{inc} \subseteq I, e^{exc} \cap I = \phi$」を満たすとき、$I$ は e を拡大する、と言う。問題設定は下記。

- 入力：背景知識 B、仮説空間 S_M、部分解釈集合 E
- 出力：仮説 H
  ※ H は以下の条件を満たす。

  (1).$H \subseteq S_M \cdot H$ は仮説空間
  (2).$\forall < e^{inc}, e^{exc} > \in E, \exists A \in AP(B \cup H)[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$

  上記式は、事例となる各部分解釈に対し、それぞれ拡大する任意の解集合が存在することを示す。

なお、Brave Induction との違いとして、Brave Induction はすべての事例を説明するのが 1 つの解集合であるのに対し、Induction of stable models は事例ごとにそれを説明する解集合は異なっている。

■**解集合からの学習（Learning from Answer Sets）**　最後に後述する ILASP システムの基本的な問題設定とされる解集合からの学習（Learning from Answer Sets）について述べる。問題設定は下記。

- 入力：背景知識 B、仮説空間 S_M、部分解釈集合 E^+, E^- ※ Induction of Stable Models に負例の部分解釈集合を追加する。
- 出力：仮説 H
  ※ H は以下の条件を満たす。

  (1).$H \subseteq S_M \cdot H$ は仮説空間
  (2).$\forall < e^{inc}, e^{exc} > \in E^+, \exists A \in AP(B \cup H)[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$
  (3).$\forall < e^{inc}, e^{exc} > \in E^-, \nexists A \in AP(B \cup H)[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$

(2) は、各部分解釈に対し、それぞれその部分解釈を拡大する解集合が存在することを示す式である。
(3) は、負例となっている各部分解釈に対し、その部分解釈を拡大する解集合が存在しないことを示す。

2.4.1 仮説の評価関数

仮説の評価関数として「記述長最小原理」など複数の評価関数が提案されている。

■記述長最小原理 記述長最小原理は、正例を「仮説＋例外」の形式置き換えて、記述量の差を計算し仮説を評価する。この値が最大の仮説が良い仮説とされ、順位付けされる。近似式は下記。

$$\begin{aligned} CompressionGain(\text{評価値}) &= \text{説明される正例数} - \text{仮説のリテラル長} - \text{説明される負例の数} \\ &= |\text{概念の外延表記}| - |\text{概念の内包表記}| - |\text{例外表現}| \end{aligned}$$

解集合からの学習における仮説の長さは、仮説を構成するリテラル数で決定される。すなわち各ルールを構成するリテラル数の総和となる。例として、{q :- notr., r :- not q.}という仮説が与えられたとする。この仮説に含まれるリテラル数の総和は、2+2 = 4 となる。

では、仮説をどのように導出するのかについて述べる。

まず、以下のように記号を定める。

- Positive Solution:背景知識とともに各正例を説明する仮説、ルール集合。
  条件 : $\forall < e^{inc}, e^{exc} >\in E^+, \exists A \in AP(B \cup H)[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$ を満たすこと。
- Violating Solution:負例も説明する仮説、ルール集合。
  条件 : $\forall < e^{inc}, e^{exc} >\in E^+, \exists A \in AP(B \cup H)[e^{inc} \subseteq A, e^{exc} \cap A = \phi] \wedge \exists < e^{inc}, e^{exc} >\in E^-, \nexists A \in AP(B \cup H)[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$

このとき、Positive Solution の集合を $pos_sol(T)$Violating Solution の集合を $vio_sol(T)$ と表記する。T は入力の集合 $T =< B, S_M, E^+, E- >$ を示す。

そして、求める仮説 H は、$pos_sol(T)$、$vio_sol(T)$ のうちで、前述した記述長が最小のものである必要がある。したがって、仮説空間 S_M の部分集合を 1 つずつ調べていき、記述長が最小の仮説を取り出せば良いことになる。しかし、部分集合の大きさは全部で、$2^{|S_M|}$ とルール集合の大きさ次第で指数関数的に増加するので、計算量として現実的とは言えない。

この問題を解決するため、後述する ILASP システムでは、設定された帰納推論の問題を解集合プログラミングへ変換して計算している。より具体的には、前回のレポートの実験で使用した SAT ソルバーである clingo を使用して、問題を解いている。

2.5 ILASP システム（Inductive Learning of Answer Set Programs）

論理ベース、すなわち論理プログラミングに基づいた機械学習システムの構築を目的としたシステムが、ILASP システムである。公式サイト `https://ilasp.com/?no_animation` には以下のように記されている。

＜原文＞

INTERPRETABILITY:The learned knowledge can be translated into English, meaning that it can be explained to users and verified or corrected where necessary.

USE OF EXISTING KNOWLEDGE:Logic-based systems do not need to learn everything from

scratch, and can instead start from an existing knowledge base, containing anything known before the learning starts or even previously learned rules.
GENERALISATION:Logic-based systems can generalise from very few examples, making it possible to learn complex knowledge without needing large datasets.
＜訳文＞
インタープリタビリティ：学習した知識を英語に翻訳し、ユーザーに説明し、必要に応じて検証・修正することができる。
既存の知識の利用：論理ベースのシステムは、すべてを一から学習する必要がなく、学習開始前に知っていたことや、以前に学習したルールなどを含む既存の知識ベースから学習を開始することができる。
一般化：ロジックベースのシステムは、非常に少ない例から一般化することができるため、大規模なデータセットを必要とせずに複雑な知識を学習することができる。

ILASP システムはその特性から、一般的なゲームや強化学習などに応用されている。また問題設定に関して、各事例に対する背景知識やノイズの許容、弱い制約の学習ができるといった拡張ができる。各事例に対する背景知識とは具体的に述べると、これまでの部分解釈集合 $< e^{inc}, e^{exc} >$ を $< e^{inc}, e^{exc}, e^{ctx} >$ に拡張できることを指す。e^{ctx} は、弱い制約を含まない解集合プログラミングのことである。これまでの背景知識 B に加え、この e^{ctx} と併せたときの解集合を考えている。

- $\forall < e^{inc}, e^{exc}, e^{ctx} > \in E^{+}, \exists A \in AP(B \cup H \cup e^{ctx})[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$
- $\forall < e^{inc}, e^{exc}, e^{ctx} > \in E^{-}, \nexists A \in AP(B \cup H \cup e^{ctx})[e^{inc} \subseteq A, e^{exc} \cap A = \phi]$

また、ノイズの許容とは、各事例に対してペナルティとなる値（コスト）を設定することを指す。このコストは、条件を満たさない事例に付与されるペナルティの総和の値となる。

■解集合プログラミングへの変換　先述したように、ILASP システムでは、設定された帰納推論の問題を解集合プログラミングへ変換して計算している。背景知識、仮説、事例をそれぞれ解集合プログラミングの形式へ変換することで、SAT ソルバーである clingo を利用できるようにしている。

具体的には、入力である「B, S_M, E^{+}, E^{-}」を変形する。

- B = {q :- r}
- E+ = {<{p,q}, {r,s}>, <{q}, {}>}
- E- = {<{p}, {q, t}>}
- SM = {p., q :- r, not s.}

このとき得られた仮説、ルール集合を T^{n}_{meta} と示す。

まず背景知識 B を変形する。各ルールに対し、ルール中のアトム（原子式）A を $e(A, X)$ へ置換し、ルール本体部に $ex(X)$ を追加した新たなルールを生成する。すると変形後は「q(q, X):- e(r,X), ex(X)」となる。

次に、仮説空間の変形を行う。固有の識別子”id”を持つルールに対し、ルール中の各アトム A を $e(A, X)$ へ置換し、ルール本体部に $active(id), ex(X)$ を追加した新たなルールを生成する。また、id を持つルール R に対してその長さ $|R|$ を示す length(id, |R|) を作成する。すると変形後は「e(p,X):- active(h1), ex(X)., length(h1,1)., e(q,X):- e(s,X), active(h2), ex(X)., lenght(h2,3)」となる。

次に、仮説長を n に限定するための補助ルール「n # sum { active(R) = X : length(R, X) } n.」を生成する。

次に、正例の変形を行う。識別子が id である正例 $< e^{inc}, e^{exc} >$ に対し、次の 3 つのルールを生成する。

1. ex(id)
2. :- not converted(id).
3. ルール頭部は converted(id), 本体部は e^{inc} 内の各アトム A に対する e(A, id) と e^{exc} 内の各アトム B に対する not e(B, id) の連言

変形後は、「{ex(pos1).,:- not covered(pos1).,covered(pos1):- e(p, pos1), e(q, pos1), not e(r, pos1), not e(s, pos1)., ex(pos2).,:- not covered(pos2).,covered(pos2):- e(q, pos2). }」となる。最後に負例の変形だが、各負例 $< e^{inc}, e^{exc} >$ に対し、次のルールを 1 つ生成する。

- 頭部は voilating, 本体部は $e^{i}nc$ 内の各アトムに対する e(A, neg) と e^{exc} 内の各アトムに対する not e(B,neg) の連言。

したがって、変形後は「{voilating :- e(p, neg), not e(q, neg), not e(t, neg).}」となる。
以上が解集合プログラミングへの変形の概要である。

2.6 初期の ILAPS における仮説導出アルゴリズム

以下に ILASP のアルゴリズムを擬似コード (JavaScript 風) で示す。各文字は、先述したものに対応する。

```
0  function ILASP(T) {
1    let solutions = [];
2    for(let n = 0; solutions.length === 0; n++) {
3      let vs = AS(T^n_meta ∪ {← not violating; ex(neg)});
4      let ps = AS(T^n_meta ∪ {constraint(meta^-1(V)): V ∈ vs});
5      solutions = {meta^-1(A) : A ∈ ps}
6    }
7    return solutions
8  }
```

流れを説明する。初めに、3、4 行目の処理で仮説長 n の vio_sol(T) を計算している。具体的には、$T^n_{meta} \cap \{:-notviolating., ex(neg)\}$ の解集合を計算している。このとき、制約として $pos\_sol(T) \quad \setminus vio\_sol(T)$ を計算している。また、「meta^1」は変形前の状態に戻すことを示す。「constraint(meta^1(V))」では、violating を導出する、各解集合 V を制約に変形している。
以上の処理を解が見つかるまで、仮説長 n をインクリメントしながら行う。

3 計算機実験

3.1 実験準備

3.1.1 実験環境

今回の実験は仮想マシン上で clasp のバイナリをダウンロードして行った。下記に実験時の環境を示す。

- ホスト OS：Window10 Home 20H2
- 仮想 OS：Ubuntu 20.04.2 LTS
- CPU：Intel(R)Core(TM)i7-9700K @ 3.6GHz
- GPU：Nvidia Geforce RTX2070 OC @ 8GB
- ホスト RAM：16GB
- 仮想 RAM：4GB
- ILASP version： 4.1.2

なお、ILASP 環境は下記 URL より圧縮ファイルをダウンロードし、任意のフォルダに展開して作成した。

- GitHub:`https://github.com/ilaspltd/ILASP-releases/releases`

実行時は、ダウンロードしたものを展開したディレクトリ（ILASP があるディレクトリ）に移動し「./ILASP ....」コマンドを使用して実行する。

3.2 問題説明

各設問ごとの問題を以下に示す。

3.2.1 ハミルトン経路

1. 「ハミルトン閉路」を表す解集合プログラムを求める問題に対する表現とその詳細な説明（配布資料：hamilton{X,Y}.las）
2. 配布資料：hamilton{X,Y}.las から得られる結果（プログラム）の詳細な説明．※併せて，具体的な実行コマンドと，実際に得られる結果，計算時間等も示すこと．

3.3 結果と考察

hamilton{X,Y}.las の内容の説明をする。次に、hamilton{X,Y}.las の仮説空間の大きさを示す。

```
..$ wc hamiltonX.las
104  672 3898 hamiltonX.las
..$  wc hamiltonY.las
200  1496 19656 hamiltonY.las
```

hamilton{X,Y}.las に対して、ILASP を実行するにはシェル上で「./ILASP –version=3 ./hamiltonX.las」打ち込む。以下は実行時の結果。

```
..$ ./ILASP --version=4 ./hamiltonX.las
%% score = 0
Pre-processing  : 0.006s
Solve time      : 0.004s
Total           : 0.009s
```

「score = 0」とあるが、これは得られた仮説がないことを示す。このときのデバック情報は下記

```
%% Iteration 1
%%
%% Found Hypothesis:
%% |H| = 0
%% expected penalty = 0
%% unexpected penalty = 0
%% score = 0
%% UncoveredEGS = {  }
```

4　巻末資料

本稿で使用した画像、プログラムコード等はすべて以下のリンク先に掲載している。必要に応じてご覧頂きたい。

- GoogleDrive:https://drive.google.com/drive/folders/1YZg84--BFR0XRyXArL1TTY5m5xGxITxG?usp=sharing
- GitHub:https://github.com/tsyu12345/logical_and_calculating_LectureCode/tree/master/No14