

完全・不完全ルールの使用頻度を考慮した知識洗練化

桂田 浩一 大原 剛三 馬場口 登 北橋 忠宏

大阪大学 産業科学研究所

概要

本稿では、例外を許容する不完全知識の存在下における知識の洗練化手法を提案する。知識の洗練化とは、推論の効率化のために知識集合を再構成することをいう。ルール形式で表現した知識ベースでは、不完全知識を表現する不完全ルールに対してルールを満たさない不整合事例が増加した場合、不整合事例に関するルールの増加によって推論効率が低下する。このような場合に本手法では、推論効率を向上するために不完全ルールの結論部を書換え、不整合事例に関するルールを削除し、推論結果の維持のために必要なルールを追加する。こうした洗練化を実現するために、推論の際のルールの使用頻度に基づいた推論効率の評価方法を提案し、ルール集合を表現するグラフにおいて洗練化をモデル化する。

Knowledge Refinement of Complete/Incomplete Rules Based on Frequency of Rule Using

Kouichi KATSURADA Kouzou OHARA
Noboru BABAGUCHI Tadahiro KITAHASHI

I.S.I.R., Osaka University

abstract

In this paper, we propose a method of knowledge refinement under the condition that the incomplete knowledge including exceptions exists. Knowledge refinement means a reconstruction of a knowledge-base for the purpose of the efficient inference. In the knowledge-base represented by rules, if examples inconsistent with an incomplete rule representing the incomplete knowledge increase, rules related with these examples would increase accordingly, and the increase would make the performance of the inference worse. In order to improve the performance, we rewrite the head of the incomplete rule to delete rules increased according to the inconsistent examples, and add some rules to prevent the change of the conclusion derived from the knowledge-base.

1 はじめに

新たな知識の追加が頻繁に起こる知識ベースでは、現有知識の変化に伴い、不要となる知識や誤りの多い知識が増加する場合がある。このような知識は推論効率の低下など、知識ベースの性能悪化の一因となり得る。こうした背景から、推論効率の向上、および、より正確な知識の獲得のために知識を再構成する知識洗練化 (knowledge refinement)[1] が検討されてきた。

しかしながら、従来の知識洗練化手法では常に成り立つ完全知識のみから知識ベースが構成されることを前提としており、常に成り立つとは限らない不完全知識を取り扱うことができなかった。そこで、本稿では完全・不完全知識の存在下における、推論効率向上のための知識洗練化手法 [2][3] を提案する。

提案手法では、完全・不完全知識をそれぞれ完全・不完全ルールで表している。不完全ルールを満たさない不整合事例が増加した場合、不整合事例に関するルールが増加するために、推論効率が低下することがある。こうした場合に、知識コンバージョン (Knowledge Conversion: 以後 KC と略記する)[4] を用いることで不完全ルールの整合事例と不整合事例を逆転する。KC とは完全・不完全知識を質的に変換させるメカニズムをいい、筆者らはこれまでに一つの完全ルールを不完全ルールに変換する最も基本的な KC[4] を提案している。本手法ではこの KC を拡張して複数の完全・不完全ルールを変換することで、図 1 のような知識洗練化を実現する。

こうした洗練化を、推論効率が向上する場合に限って実行するために、推論の際のルールの使用頻度に基づく推論効率の評価方法を検討する。また、ルール集合を表現するグラフにおいて、リンクの追加・削除の操作を用いることで洗練化をモデル化する。

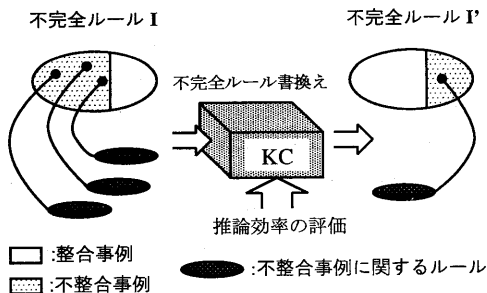


図 1: 知識の洗練化

2 知識表現

2.1 ADAG 表現

本稿では、完全知識、不完全知識および事実を完全ルール、不完全ルールおよびファクトとして次のように表す。

完全ルール: $A \leftarrow B_1, \dots, B_n$

「 $B_1, \dots, B_n$ ならば必ず A である。」

不完全ルール: $A \leftarrow B_1, \dots, B_n$

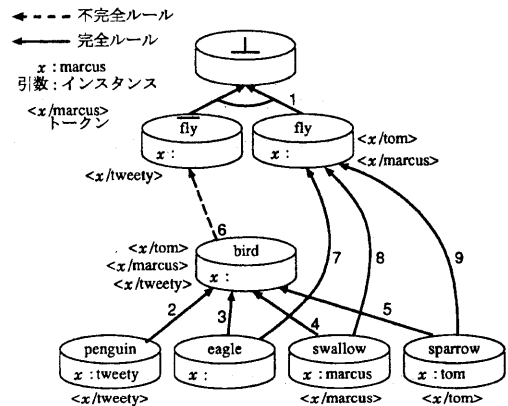
「 $B_1, \dots, B_n$ ならば通常 A である。」

ファクト: $p(c_1, \dots, c_m)$

「 $c_1, \dots, c_m$ は p である。」

ここで、 $A, B_i (1 \leq i \leq n)$ は原子式、 p は述語記号、 $c_i (1 \leq i \leq n)$ は定数を表す。完全ルールのうち結論部が矛盾を表す記号 $\perp$ であるものは制約と呼び、条件部がすべて成り立つ場合に矛盾することを表す。

以上のルール、ファクトの集合を図 2 のような属性付き有向アサイクリックグラフ (Attributes Added Directed Acyclic Graph, 以後 ADAG と略記する)[4] を用いて表す。ADAG は一種の AND-OR グラフであり、完全ルールおよび不完全ルールはそれぞ



知識

1. 「飛ぶと飛ばないは同時に成り立たない」
2. 「ペンギンは鳥である」
3. 「ワシは鳥である」
4. 「ツバメは鳥である」
5. 「スズメは鳥である」
6. 「鳥は通常飛ばない」
7. 「ワシは飛ぶ」
8. 「ツバメは飛ぶ」
9. 「スズメは飛ぶ」

事実

「tweety はペンギンである」「marcus はツバメである」「tom はスズメである」

図 2: ADAG 表現

れ実線および破線のリンクで、ファクトはノード中のインスタンスとして記述する。図中の番号は各ルールの ID 番号である。

ADAG ではトークンを伝播することによって推論を行う。トークンは変数 x_i とその代入例である定数 c_i の組から成り立っており、 $\langle x_1/c_1, \dots, x_n/c_n \rangle$ と表記する。推論の際には、まず、インスタンス $(x_1 : c_1, \dots, x_n : c_n)$ に対応するトークン $\langle x_1/c_1, \dots, x_n/c_n \rangle$ を各ノードに設置し、以下のトークン伝播規則に従ってトークンを伝播する。

定義 1 (完全ルールのトークン伝播規則) 完全ルール「 $A \leftarrow B_1, \dots, B_n$ 」において、 $B_1, \dots, B_n$ 中の変数を $x_1, \dots, x_m$ 、 A 中の変数を $x_1, \dots, x_l (l \leq m)$ とする。このとき、条件部中のトークンから代入 $\{x_1/c_1, \dots, x_m/c_m\}$ を求めることができるならば、 A に $\langle x_1/c_1, \dots, x_l/c_l \rangle$ を伝播する。□

定義 2 (不完全ルールのトークン伝播規則) 不完全ルール「 $A \leftarrow B_1, \dots, B_n$ 」において、 $B_1, \dots, B_n$ 中の変数を $x_1, \dots, x_m$ 、 A 中の変数を $x_1, \dots, x_l (l \leq m)$ とする。このとき、条件部中のトークンから代入 $\{x_1/c_1, \dots, x_m/c_m\}$ を求めることができ、かつ、トークンの伝播によって矛盾が生じないならば、 A に $\langle x_1/c_1, \dots, x_l/c_l \rangle$ を伝播する。□

各ルールの条件部を満たす代入例には、重複がないとする。上の伝播規則に従ってトークンを伝播するとき、ADAG におけるトークンの伝播可能性を次のように定義する。

定義 3 (伝播可能性) $N, N_1, \dots, N_m$ をノードとする。各 $N_i (1 \leq i \leq m)$ にトークンがあるとき、トークン伝播規則に従って、 N にトークンを伝播することができるならば、 $N_1, \dots, N_m$ から N に伝播可能であると呼ぶ。伝播することができないならば、伝播不可能であると呼ぶ。伝播可能、伝播不可能などの性質を伝播可能性と呼ぶ。□

伝播可能性の定義より、ノード $N_1, \dots, N_m$ から N に伝播可能である場合には、 $N_1, \dots, N_m$ と N の間にトークンを伝播するためのルール集合が存在する。このルール集合を $N_1, \dots, N_m$ から N への推論パスという。

ADAG では、伝播可能なすべてのトークンを伝播し終えた時の、各ノードのトークンの状態を推論

結果とする。このときの各ノードのトークンの状態を伝播完了状態と呼ぶ。

例えば、図 2 のトークンは伝播完了状態である。この場合、ノード penguin からノード fly へは伝播可能であり、ルール集合 $\{2, 6\}$ はノード penguin からノード fly への推論パスである。

定義 2 の伝播規則により、不完全ルールについてはトークンを伝播できる場合とできない場合がある。そこで、不完全ルールの整合事例、および、不整合事例を次のように定義する。

定義 4 (不完全ルールの整合事例・不整合事例) I を不完全ルール「 $A \leftarrow B_1, \dots, B_n$ 」、 $N_1, \dots, N_m$ をノードとし、 $N_1, \dots, N_m$ から $B_1, \dots, B_n$ の全てに伝播可能であるとする。このとき、 $N_1, \dots, N_m$ から A に伝播可能であるなら、 $N_1, \dots, N_m$ を I に対する整合事例と呼び、矛盾するために伝播不可能であるなら、 $N_1, \dots, N_m$ を I に対する不整合事例と呼ぶ。□

ADAG では制約の条件部を満たす代入例があるかどうかで矛盾検査をするが、本手法では、不完全ルールのトークン伝播における矛盾検査の際に、その不完全ルールの結論部を条件部に持つ制約のみを検査すれば良いように、知識が構成されているとする。

不完全ルールの結論部を A 、制約を「 $\perp \leftarrow A, A_1, \dots, A_m$ 」とするとき、上の定義 4 より、整合事例から $A \rightarrow$ は伝播可能であり、不整合事例から $A_1, \dots, A_m \rightarrow$ は伝播可能である。本研究では、制約の条件部は整合事例、および、不整合事例の相反する性質を表すと捉え、 A を整合事例の性質、 $A_1, \dots, A_m$ を不整合事例の性質と呼ぶ。

不整合事例を条件部、不整合事例の性質を結論部とするルールは、不整合事例がその性質を持つことを表すルールとみなすことができる。以下では、このようなルールを不整合事例の性質を表すルールと呼ぶ。

図 2 の場合、penguin は不完全ルール 6 に対する整合事例、eagle, swallow, sparrow は不整合事例である。また、ノード fly はルール 6 の不整合事例の性質であり、ルール 7, 8, 9 は不整合事例の性質を表すルールである。

2.2 ADAG の階層化

ADAG におけるトークンの伝播順序が一意に定まっていない場合、つまり、任意の二つのルール r_1

と r_2 について、どちらの条件部のトークンを先に伝播するかが決まっていなない場合には、不完全ルール I におけるトークンの伝播後に、 I に対する不整合事例の性質に新たなトークンが伝播するときがある。このとき、改めて矛盾検査をやり直して矛盾が検出された場合には、先に伝播したトークンのいくつかを削除しなければならない。

こうした冗長な処理を解消するために、ADAG のノードの階層化を考える。階層に従って各ノードのトークンを順に求めることによって、不完全ルールにおけるトークン伝播の際の矛盾検査を1度に限ることができる。

ADAG では条件部から結論部へトークンを伝播するので、条件部の階層を結論部の階層に比べて低くすることによって、下位階層から順にトークンを求めることができる。ただし、不完全ルールに関してはトークン伝播の前に矛盾検査を行うので、不完全ルールに対する不整合事例の性質であるノードの階層は、不完全ルールの結論部に比べて低くしなければならない。そこで ADAG 中の完全ルールの集合を R_C 、不完全ルールの集合を R_I と表すとき、ノード N の階層 $St(N)$ を次のように定義する。

- I. if ($A \leftarrow B_1, \dots, B_n \notin R_C$,
 $A \leftarrow D_1, \dots, D_n \notin R_I$)
then $St(A) := 0$
 - II. (1) if ($A \leftarrow B_1, \dots, B_n \in R_C$,
 $St(B_1), \dots, St(B_n)$ が定義済)
then $St'(A) > \max(St(B_1), \dots, St(B_n))$
 - (2) if ($A \leftarrow D_1, \dots, D_n \in R_I$,
 $\perp \leftarrow A, E_1, \dots, E_m \in R_C$,
 $St(D_1), \dots, St(D_n)$,
 $St(E_1), \dots, St(E_m)$ が定義済)
then $St'(A) > \max(St(D_1), \dots, St(D_n),$
 $St(E_1), \dots, St(E_m))$
- $St(A) := (1), (2)$ の $St'(A)$ の取り得る最小の整数

ADAG では、推論結果が複数となる場合、すなわち、非単調論理における多重拡張 (multiple extensions)[5] が起こる場合がある。多重拡張とは、二つの不完全ルールの適用順序によって異なる推論結果が得られることをいう。このような場合には、上の定義では階層化が不可能となる。

こうした問題を解消するために、一方のルールの条件部が他方のルールの条件部に比べ特殊である場合、つまり、一方のルールの条件部から他方のルー

ルの条件部への推論パスがある場合に、特殊な条件部を持つルールを優先して用いる。この優先関係を以下のように定義する。

定義 5 (不完全ルールの優先関係) ルール r の条件部ノードを $Body(r)$ と表す。不完全ルール I_1, I_2 について、 $Body(I_1)$ から $Body(I_2)$ への推論パスが存在するとき、 I_1 の方が I_2 より優先するといい、 $I_1 \ll I_2$ と書く。 $\square$

この優先関係によって、妥当な推論結果の選択が可能となるが、上記の優先関係が必ずしも存在するとは限らないため、多重拡張が解消しない場合がある。本手法では、定義 5 の優先関係がすべての不完全ルールについて定義されているものと仮定する。この優先関係を反映した次の規則を階層の定義 II. に追加し、階層 $St(A)$ の定義を変更する。

- (3) if ($r : A \leftarrow D_1, \dots, D_n \in R_I$,
 $r_1 : E_1 \leftarrow F_1, \dots, F_{l_1} \in R_I$,
 $\vdots$
 $r_l : E_l \leftarrow F_{l_1}, \dots, F_{l_l} \in R_I$,
 $\perp \leftarrow A, E_1, \dots, E_l, E_{l+1}, \dots, E_m \in R_C$,
 $r \ll r_1, \dots, r_l$,
 $St(D_1), \dots, St(D_n)$,
 $St(E_{l+1}), \dots, St(E_m)$,
 $St(F_{1_1}), \dots, St(F_{l_l})$ が定義済)
then $St'(A) > \max(St(D_1), \dots, St(D_n),$
 $St(E_2), \dots, St(E_m),$
 $St(F_1), \dots, St(F_l))$

$St(A) := (1) \sim (3)$ の $St'(A)$ が取り得る最小の整数

以上のような階層化の代表例として、従来手法では層状データベース [6] における述語の階層化がある。層状データベースでの階層化は、ルールの条件部と結論部が同一の階層に属する場合があるのに対して、本手法の階層化はルールの条件部の階層は結論部に比べて必ず低くなるという点でこれらは異なる。階層の定義に従った階層化の例を図 3 に示す。

3 知識の再構成法

不完全ルールに対する不整合事例の性質を表すルールが増加した場合、不完全ルールの結論部をその不整合事例の性質に書換える事によって、その不整合事例の性質をあらわすルールを削除できる。一方で、書換え前の不完全ルールに対する整合事例に

については推論結果の変化を防ぐために、新たなルールを追加する必要が生じる。

本章では、推論効率が向上する場合にのみこうした操作を実行するための推論効率の評価方法、洗練化に必要なルールの追加・削除の操作、および、洗練化アルゴリズムについて述べる。

3.1 推論効率の評価

3.1.1 ルールの重み

ADAG ではトークンの伝播によって推論を行うことから、推論効率の向上はトークンを伝播するコストの削減を意味する。ルールにおいてトークンを伝播するときの主な処理は、条件部を満たす代入例を検出することである。そこで、条件部を満たす代入例を検出する関数 **detect** を定義し、その呼び出し回数をトークン伝播の際のコストと考える。関数 **detect** の引数は、 $\langle$ ノードの集合, 検出の情報 $\rangle$ とし、出力は $\langle$ ノードの集合を満たす代入例, 検出の情報 $\rangle$ の組とする。検出の情報とは、これまでにどのトークンを調べたかといった、検出作業の途中経過の情報であり、探索済のトークンを重複して調べることを防ぐために用いる。ノードを満たす代入例が検出されない場合、**detect** は $\langle$ nil, end $\rangle$ を返すものとする。

ADAG における推論の際には伝播完了状態を求めることから、そのための **detect** の呼び出し回数が減少する洗練化を実現する。本手法では、各ルールにおいて **detect** を呼び出す回数を各ルールの重みとし、重みの総和が減少する場合に上述の知識の再構成を実行する。ただし、関数 **detect** の代入例検出のコストは、条件部の変数の数、重複の度合などに依存するが、本研究では関数 **detect** 自体のコス

トは考慮しないため、ここでの推論効率の評価はあくまで大まかな指標であることを断っておく。

完全ルールにおけるトークン伝播の際に **detect** を呼び出す回数は、(条件部を満たす代入例の数+検出終了を知るための1回)である。一方、不完全ルールにおいてはトークン伝播の際に矛盾検査をする必要があるので、**detect** の呼び出し回数は(条件部を満たす代入例の数+1)と(制約の条件部を満たす代入例の数+1)との和となる。例えば次の例では、不完全ルール「 $\overline{\text{fly}}(x) \leftarrow \text{bird}(x)$ 」におけるトークン伝播の際に **detect** を呼び出す回数は、鳥であるものの検出に3回、制約の条件部を満たす代入の検出に2回の合計5回となる。

例： $\perp \leftarrow \text{fly}(x), \overline{\text{fly}}(x), \quad \overline{\text{fly}}(x) \leftarrow \text{bird}(x)$
 $\text{bird}(\text{tweety}), \text{bird}(\text{bill}), \text{fly}(\text{bill})$

以上から、ルールの重みを次のように定める。

完全ルールの重み

(条件部を満たす代入例の数+1)

不完全ルールの重み

(条件部を満たす代入例の数+1)+

(不完全ルールの結論部を条件部に持つ制約の条件部を満たす代入例の数+1)

上の定義中の条件部を満たす代入例は、ルールの条件部ノード中のトークンから求まる。しかしながら、各ノードのトークンを実際に求めることは効率が悪いので、ここでは期待値として各ノードのトークン数を求め、代入例の数を計算する。ノードのトークン数の期待値は、ノード中のインスタンス、および、そのノードを結論部とするルールに従って伝播するトークン数の期待値から求まる。以下では、これらの期待値を求める方法を検討する。

3.1.2 条件部を満たす代入例の数

例えば、「 $\text{bigger}(x, y) \leftarrow \text{ostrich}(x), \text{swallow}(y)$ 」のように、ルールの条件部に変数の重複がない場合、条件部を満たす代入例の数は条件部の各ノードのトークンの全組合せ数となる。すなわち、条件部が $B_1, \dots, B_n$ であるルールにおける各 $B_i (1 \leq i \leq n)$ 中のトークン数が T_{B_i} である場合、 $\prod_i T_{B_i}$ となる。

しかしながら、「 $\text{fly}(x) \leftarrow \text{bird}(x), \text{injured}(x)$ 」のように、条件部のノードの変数が重複するために、代入される定数が一致しなければならない場合がある。例えばこのルールにおいて、 $\text{bird}(c_1)$ と $\text{injured}(c_2)$ が共に真である場合、 x に代入可能な定数が Con_x

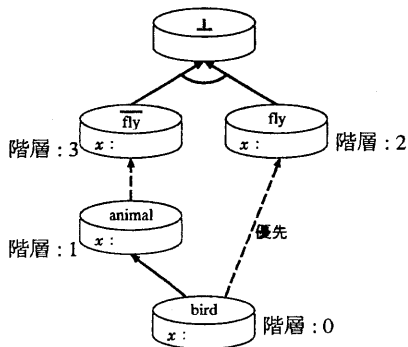


図 3: 不完全ルールの優先関係とノードの階層化

種類あると仮定すると、 c_1 と c_2 が一致する確率は $1/Con_x$ となる。

一般に、ルール r の条件部ノード中の変数を $x_1, \dots, x_m$ 、変数 $x_j (1 \leq j \leq m)$ が現れるノード数を $|x_j|$ 、変数 x_j に代入可能な定数の総数を Con_{x_j} とする場合、 x_j の現れるすべてのノードにおいて x_j に代入される定数が一致する確率は、 $1/Con_{x_j}^{|x_j|-1}$ となる。したがって、ルールの条件部を満たす代入例の期待値 T は次のように表すことができる。

$$T = \frac{\prod_i T_{h_i}}{\prod_j Con_{x_j}^{|x_j|-1}} \quad \left(\begin{array}{l} 1 \leq i \leq n \\ 1 \leq j \leq m \end{array} \right)$$

3.1.3 ルールを伝播するトークン数

一般的にホーン節では、条件部に現れる変数が必ずしも結論部に現れるとは限らない。このために、ルールの条件部を満たす代入例から、伝播するトークンを生成する際に重複が生じる場合がある。例えば「fly(y) ← penguin(x), parent(x, y)」というルールがあった場合、二つの代入 $\{x/sweetie, y/tweetie\}$ と $\{x/veety, y/tweetie\}$ は x が異なっているが、 y が同じであるために同一のトークン $\langle y/tweetie \rangle$ が伝播される。

ここでこのルールにおいて、一つの代入 $\{x/c_1, y/c_2\}$ が与えられた場合に、この代入から求めるトークンが別のトークンと重複する確率を考える。条件部を満たす代入例には重複がないので、 $\{x/c_1, y/c_2\}$ を除く代入例の全組合せは、変数 x, y に代入可能な定数の種類をそれぞれ Con_x, Con_y 種類とすると、 $Con_x \times Con_y - 1$ となる。また、伝播するトークンが一致する、つまり、 y が一致する組合せの総数は $\{x/c_1, y/c_2\}$ を除くと $Con_x - 1$ となる。すなわち、この例では伝播するトークンが一致する確率は $(Con_x - 1)/(Con_x \times Con_y - 1)$ となる。

一般に、完全ルール C を伝播するトークンの期待値 T_C は、条件部に現れる変数を $x_j (1 \leq j \leq m)$ 、条件部に現れて結論部に現れない変数を $x_k (1 \leq k \leq l, l \leq m)$ 、条件部を満たす代入例を T とするとき次式で表される。

$$T_C = T \times \left(1 - \frac{\prod_k Con_{x_k} - 1}{\prod_j Con_{x_j} - 1}\right) \quad \left(\begin{array}{l} 1 \leq j \leq m \\ 1 \leq k \leq l \end{array} \right)$$

不完全ルールでは、このトークン数から、矛盾するために伝播できないトークン数を差し引かなければならない。つまり、完全ルールとみなして伝播したトークン数から、この場合の制約の条件部を満た

す代入例を引いた数が不完全ルールのトークン数となる。ここで、不完全ルール I の結論部を条件部に持つ制約を $Inc_1, \dots, Inc_n$ とする。 I を完全ルールとみなしてトークンを伝播した場合の、 $Inc_h (1 \leq h \leq n)$ の条件部を満たす代入例の数を T_{Inc_h} とするとき、 I を用いて伝播するトークン数の期待値 T_I は、次の式で表される。

$$T_I = T_C - \sum_h T_{Inc_h} \quad (1 \leq h \leq n)$$

3.1.4 各ノードのトークン数

トークンはインスタンスから生成され、ルールに沿って伝播する。したがって、伝播完了状態におけるノード N 中のトークンは、 N 中のインスタンス、および、 N を結論部とする各ルール $r_1, \dots, r_n$ を伝播するトークンから求めることができる。これらの各トークンに重複がない場合、 N 中のトークン数は、 N 中のインスタンス数と $r_i (1 \leq i \leq n)$ を伝播するトークン数の和となるが、重複がある場合があるため、これを除いた結果が N 中のトークン数となる。

N 中の変数を $x_1, \dots, x_m$ 、 $x_j (1 \leq j \leq m)$ に代入可能な定数の総数を Con_{x_j} とするとき、二つのトークンが一致する確率は $1/\prod_j Con_{x_j}$ となる。すなわち、 N 中のインスタンス数を $Inst_N$ 、 r_i を伝播するトークン数を T_{r_i} とするとき、 N 中のトークンの期待値 T_N は次のように求めることができる。

$$1. T_N := Inst_N$$

$$2. \text{ for all } r_i$$

$$T_N := T_N + \left(1 - \frac{T_N}{\prod_j Con_{x_j}}\right) \times T_{r_i} \quad \left(\begin{array}{l} 1 \leq i \leq n \\ 1 \leq j \leq m \end{array} \right)$$

以上の重みおよびトークン数の計算方法を用いて図2についてそれぞれの値を計算した結果を図4に示す。

3.2 洗練化におけるルールの追加・削除

不完全ルールの結論部、及び、不整合事例の性質は同一の制約の条件部であることから、本手法では各制約について洗練化の操作を行う。ここで、制約「 $\perp \leftarrow A, A_1, \dots, A_m$ 」の A を結論部とする不完全ルール I : 「 $A \leftarrow B_1, \dots, B_n$ 」について洗練化の操作を行うとする。

定義4より、 I に対する不整合事例は全ての $A_i (1 \leq i \leq m)$ にトークンが伝播可能であり、一方、 I に対する整合事例は、ある A_i にトークンが伝播不可能である。ここで、不完全ルールに対する整合事例の

集合を C -set, 不整合事例の集合を I -set とする. ある $j(1 \leq j \leq m)$ に対して, $s_I \in I$ -set を条件部とし, A_j を結論部とする図 2 の ID=7, 8, 9 のようなルール r が存在する場合, I の結論部を A_j に書換えることによって不要となる r を削除することができる. この際に, 不完全ルールの書換えによる推論結果の変化を最小限するために, $s_C \in C$ -set を条件部とし, A を結論部とするルールを付け加える. 図 2 においては不完全ルールを「鳥は通常飛ぶ」と変更することによって, ID=7, 8, 9 の不整合事例の性質を表すルールが削除可能となり, 「ペンギンは飛ばない」という新たなルールを付け加える必要が生じる. 本手法では各 A_j について上の操作を行ったと仮定し, 最も重みが減少する A_j に対して実際に上の操作をする. なお, ルールの追加の際には完全ルールを追加することによって矛盾しないならば完全ルールを, そうでなければ不完全ルールを追加する.

なお, 特殊なノードは一般的なノードの性質を継承することから, 最も一般的な整合・不整合事例のみについて洗練化の操作をすることで, 特殊なノードを考慮する必要がなくなる. 洗練化におけるこの制限を次に示す.

- 不完全ルールの条件部 $B_1, \dots, B_n$ の一つのノード B_k を結論部, $D_1, \dots, D_l$ を条件部とするルールを検出し, $B_1, \dots, B_{k-1}, D_1, \dots, D_l, B_{k+1}, \dots, B_n$ を洗練化の整合・不整合事例として用いる.

例えば図 2 で「penguin(x) $\leftarrow$ emperor-penguin(x)」というルールが加わった場合, emperor-pen

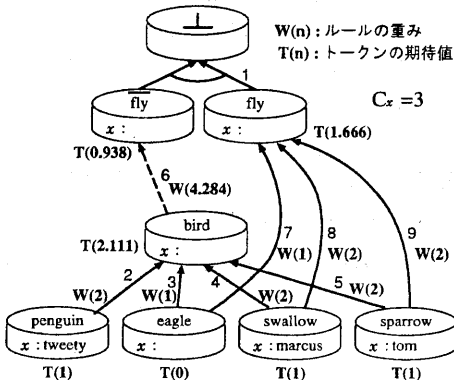


図 4: トークンの期待値とルールの重み

guin は bird への推論パスがある最も一般的なノードではないので, 洗練化には用いない.

3.3 洗練化アルゴリズム

以上の洗練化手法を, 本節ではアルゴリズムとして形式的にまとめる. アルゴリズムへの入力図 4 のように既に各ルールの重みの計算を終了した ADAG 表現の知識である.

本アルゴリズムでは, 次の記号を用いる.

- R_{del_j} : 不整合事例の性質 A_k について削除の対象となるルールの集合
- R_{add_j} : 不整合事例の性質 A_k について追加の対象となるルールの集合
- W_{del_j} : 不整合事例の性質 A_k について削除するルールの重みの和
- W_{add_j} : 不整合事例の性質 A_k について追加するルールの重みの和

[アルゴリズム]

- I. 制約 $\perp \leftarrow A, A_1, \dots, A_m \in R_C$ を選択
 - (1) 不完全ルール $A \leftarrow B_1, \dots, B_n \in R_I$ を選択
 - (a) 各 A_j ($1 \leq j \leq m$) について
 $R_{add_j} := \{A_j \leftarrow B_1, \dots, B_n\},$
 $R_{del_j} := \{A \leftarrow B_1, \dots, B_n\}$
 - (b) $B_k \leftarrow D_1, \dots, D_l \in R_C (1 \leq k \leq n)$ であるとき, $S = \{s | s = \{B_1, \dots, B_{k-1}, D_1, \dots, D_l, B_{k+1}, \dots, B_n\}\}$ をすべて検出. 各 $s \in S$ について
 - if s からある $A_i (1 \leq i \leq m)$ にトークンが伝播不可能.
 - if $A \leftarrow s$ を加えて矛盾しない
 - then 各 j について $R_{add_j} := R_{add_j} \cup (A \leftarrow s)$
 - else 各 j について $R_{add_j} := R_{add_j} \cup (A \leftarrow s)$
 - if s から任意の A_i にトークンが伝播可能.
 - if $A_j \leftarrow s \in R_C$
 - then $R_{del_j} := R_{del_j} \cup (A_j \leftarrow s)$
 - if $A_j \leftarrow s \in R_I$
 - then $R_{del_j} := R_{del_j} \cup (A_j \leftarrow s)$
 - (c) 各 j について
 - $r_{del} \in R_{del_j}$ について
 - $W_{del_j} := \sum W(r_{del})$
 - $r_{del} \in R_{del_j}$ を知識ベースから削除したと仮定し, 各 $r_{add} \in R_{add_j}$ の重みを計算.
 - $W_{add_j} := \sum W(r_{add})$
 - (d) $W_{max} := \max(W_{del_1} - W_{add_1}, \dots, W_{del_m} - W_{add_m})$
 - if ($W_{max} > 0, W_{max} = W_{del_{max}} - W_{add_{max}}$)
 - then 知識ベースに $R_{add_{max}}$ を追加,
 - 知識ベースから $R_{del_{max}}$ を削除.
 - (2) (1) $\rightarrow$ 戻り新たな不完全ルールを選択.
- II. 終了

□

4 実行例

図4の知識を入力として、前章の洗練化アルゴリズムを実行した場合、I. で制約 ID=1 が選択され、I.(1) で不完全ルール6が選択され、さらに、I. (1)(b) でノード集合 {penguin}, {eagle}, {swallow}, {sparrow} が検出される。

ここで、 $A = \overline{\text{fly}}$, $A_1 = \text{fly}$ とする。検出されたノード集合のうち {penguin} については A_1 に伝播不可能であり、ルール「 $\overline{\text{fly}}(x) \leftarrow \text{penguin}(x)$ 」を加えても矛盾が生じないので、 R_{add_1} にルール「 $\overline{\text{fly}}(x) \leftarrow \text{penguin}(x)$ 」を追加する。{eagle}, {swallow}, {sparrow} については A_1 に伝播可能であり、ルール「 $\text{fly}(x) \leftarrow \text{eagle}(x)$ 」, 「 $\text{fly}(x) \leftarrow \text{swallow}(x)$ 」, 「 $\text{fly}(x) \leftarrow \text{sparrow}(x)$ 」が存在するので、 R_{del_1} にこれらのルールを追加する。すなわち、 $R_{add_1} = \{\text{fly}(x) \leftarrow \text{bird}(x), \overline{\text{fly}}(x) \leftarrow \text{penguin}(x)\}$ となり、 $R_{del_1} = \{\overline{\text{fly}}(x) \leftarrow \text{bird}(x), \text{fly}(x) \leftarrow \text{eagle}(x), \text{fly}(x) \leftarrow \text{swallow}(x), \text{fly}(x) \leftarrow \text{sparrow}(x)\}$ となる。

次に、I.(1)(c)において $W_{del_1} = 9.284$, $W_{add_1} = 5.815$ と計算される。この場合 $W_{del_1} - W_{add_1} > 0$ となるので、I.(1)(d)において R_{del_1} 中の各ルールが削除され、 R_{add_1} 中の各ルールが追加される。図4には、ルール1とルール6の他に不完全ルール、および、制約が存在しないので、以上でアルゴリズムが終了する。図5に洗練化後の知識を示す。

洗練化前にノード eagle, swallow, sparrow から fly に伝播していたトークンについては、洗練化後にはルール3, 4, 5と新たなルール11を用いて fly に伝播する。また、洗練化前にノード penguin から $\overline{\text{fly}}$ に伝播していたトークンについては、洗練化後には新たなルール10を用いて $\overline{\text{fly}}$ に伝播する。つま

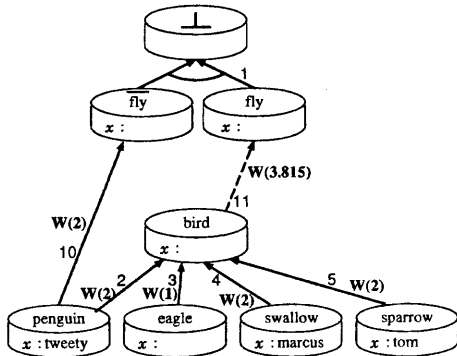


図5: 洗練化後の知識

り、洗練化の前後で推論結果は変化しない。また、洗練化前の知識では、伝播完了状態を求めるために detect を呼び出す回数は19回、洗練化後の知識では14回であるので、洗練化によって推論効率が向上したといえる。

5 まとめ

本稿では知識コンバージョンを用いて、推論効率が向上する場合に知識を再構成する洗練化手法を提案した。知識ベースにおいて推論結果を求める際には、ルールの条件部を満たす代入例の検出が必要ことから、提案手法ではこの検出を行う関数の呼び出し回数を推論効率の指標とし、知識ベース全体でこの回数が減少する知識の洗練化を実現した。本手法ではシステムが自動的に推論効率が向上するよう知識を再構成することから、これは知的なシステムにおける知識の自己組織化方法のモデル化の一つと捉えることができる。今後の課題としては、知識に多重拡張が存在する場合への拡張、モデル化の拡張などが残されている。

なお、本研究の一部は文部省科学研究費・萌芽的研究の補助による。

参考文献

- [1] A.Ginsberg, S.Weiss, and P.Politakis: SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement, Proc. IJ-CAI85, pp367-374(1985).
- [2] 桂田, 大原, 馬場口, 北橋: ルール数の変化を考慮した完全・不完全知識の洗練化, 1996年度人工知能学会全国大会論文集, pp.309-312(1996).
- [3] 桂田, 大原, 馬場口, 北橋: 知識コンバージョンと完全・不完全知識の洗練化の統合について, 情報処理学会第53回全国大会講演論文集(分冊2), pp.229-230(1996).
- [4] 馬場口, 大原, 桂田, 北橋: 矛盾を契機とする知識の質的転化, 1995年度人工知能学会全国大会論文集, pp.53-56(1995).
- [5] M.Ginsberg: Essentials of Artificial Intelligence, Morgan Kaufmann(1993).
- [6] J.D.Ullman: Principles of Database and Knowledge-base Systems, Vol1, Computer Science Press(1988).