データ駆動型プロダクションシステムによる意味ネットワークの探索†

石田 亨††

意味ネットワークの探索方式としては，現在までに Prolog によるもの，専用ハードウェアによるもの等が提案されている．本論文では，エキスパートシステム構築用ツールに広く用いられているデータ駆動型のプロダクションシステムを用いて，効率良く意味ネットワークの探索を行う方法を述べる．データ駆動型のプロダクションシステムは，① 事実データの変更によって，ルールが自動的に起動される，② 以前の探索結果を RETE ネットワーク内に保存し，次回以降の探索に用いることができる等，意味ネットワークの維持や探索に有利な素質を備えている．しかしながら，現時点のプロダクションシステムを用いて意味ネットワークの探索を行おうとすると，①探索を制御する有効な方法がない，②意味ネットワーク中の多重経路がすべて探索されてしまう，③事実データの個数が増大すると RETE ネットワークの計算量が膨大になる等の問題点がある．本論文では上記の問題を克服するために，データ駆動型プロダクションシステムに，① RETE ネットワークに適した前向き推論の制御機能，②冗長な経路の探索を防止するための事実データの非活性化機能，③探索に必要な事実データだけを RETE ネットワークに保持させるための注視点制御機能を導入する．また，これらの機能を備えたプロダクションシステム PLANET の概要を述べ，PLANET によって意味ネットワークの探索が効率良く実行できることを示す．

1. まえがき

意味ネットワークの探索方式としては，これまでに Prolog によるもの[1]，専用ハードウェアによるもの[2]が提案されてきている．本論文では意味ネットワーク探索のもう1つの実現方式として，エキスパートシステム構築用ツールに広く用いられているデータ駆動型のプロダクションシステム（OPS 5[3]，ART[4] 等）を取り上げ，機能を拡張することによって効率の良い探索を実現できることを示す．

プロダクションシステム（以下，PS と略す）を用いて意味ネットワーク（以下，SN と略す）を実現するには，SN のノードやリンクを表す事実データをワーキングメモリに格納し，SN を探索するための規則*をプロダクションルールで表せばよい．この時，データ駆動型の PS は SN 探索に適した以下の素質を備えている．

① データ駆動型の PS では，ワーキングメモリの更新が引き金となって，ルールが自動的に起動される．この性質を用いると，SN の更新を引き金として SN の規則を表したルールを起動し，影響を受ける継承値を自動的に再計算させることができる．この性質は SN の維持に有効である．

② データ駆動型の PS の内部表現である RETE ネットワーク[6]は，ルールの条件部の判定に必要な計算の途中結果をすべて保存するものである．したがって，特別なプログラミングテクニックを要することなく，以前の SN 探索中に行った計算結果を，次の探索に用いることができる．この性質は SN 探索（特に徐々に探索範囲を広げていく探検的な探索）の効率化に有効である．

しかしながら，現状の PS の機能で SN 探索を行おうとすると以下の問題が生じる．

① 前向き推論では，ルールの発火を制御する方法がないため，SN の全探索が行われてしまう．一方，後向き推論では，探索を制御するために多数の goal が生成されるが，この goal が RETE ネットワーク内に保持されるため，探索効率が著しく低下する．

② SN では同一の事実データを導くのに複数の経路が存在することがある．特に SN 中に冗長な知識が存在すると冗長な経路が数多く存在することになる．現状の PS ではこれらをすべて探索してしまうため，無駄なルールの発火が多数生じる．

③ 事実データの量が増大すると，RETE ネットワークの計算量が膨大になる．また，計算の途中結果をすべて主記憶内に保持することが困難となる．

本論文では，上記の問題を解決し，PS が本来備え

† Exploration of Semantic Networks Using Data-Driven Production Systems by TORU ISHIDA (NTT Communications and Information Processing Laboratories).

†† NTT 情報通信処理研究所

* SN の規則には，is-a, has-a, instance-of を用いた規則[1]のほかに，「飼い犬の住所は飼い主の住所と同じである」，「ある人の父の父は，その人にとっては祖父である」等，様々な継承規則が考えられる[5]．ただし，本論文では規則の内容については議論しない．規則を用いて SN を効率良く探索し，継承値を得る方法について論じる．

ている SN 探索に適した素質を活用するために，① データ駆動型の動作メカニズムに適した推論制御機能，② 多重経路の探索を防ぐための事実データの非活性化機能，および ③ 探索に必要な事実データだけを RETE ネットワーク中に保持するための事実データの注視点制御機能を PS に導入する．また，これらの機能を導入して試作したプロダクションシステム PLANET の概要と，PLANET を用いた各種の評価結果を報告する．

2. プロダクションシステムの基本モデル

この章では本論文で提案する PS の基本モデルを定義する．PS は，① 事実データベース（事実 DB と略す），② ワーキングメモリ（WM と略す），および③ プロダクションメモリ（PM と略す）から構成される．なお，SN 探索のために新たに導入する機能については，4章で改めて述べることにする．

2.1 事実データベース

事実 DB は，以下の形式から成る事実の集合である．

（事実 ID　メタ情報　事実情報）

① 事実 ID

事実を同定するためのものである．ID は事実の生成順にふられる整数である．

② メタ情報

事実の生成日時，生成理由，確信度等，事実に付随する様々な情報である．メタ情報の設定は事実 ID とメタ情報の種別を指定することにより行う．参照する場合も同様である．

③ 事実情報

事実の内容である．本論文では事実情報をリスト形式で表すが，どのような表現形式を用いるかは本質的ではない．条件照合の対象はこの事実情報であり，照合が成功すると事実 ID が返却される．事実情報の変更は，旧事実の除去と新事実の生成を意味する*．事実情報の中には未定義な値を表す '?' を含んでよい．また，同一の事実情報を持つ事実は，事実 DB 中に1個しか存在しないこととする．

2.2 ワーキングメモリ

WM は事実 DB の部分集合である．ルールでの条件照合は WM 中の事実（WME：ワーキングメモリエレメント）に対してのみ行われる．すなわち，事実 DB 中の事実であっても，WM 中に含まれなければルールの発火には何ら影響を与えない．事実の大規模化に備え，事実 DB-WM の格納階層を導入したことが，この基本モデルの特徴である*．

* メタ情報の変更は単に事実の一部を変更するだけであり，旧事実の除去と新事実の生成を引き起こすものではない．

```
〈プロダクションルール〉::=
    (defrule 〈ルール名〉 {〈条件〉}+ --> {〈動作〉}+)
〈条件〉::= 〈パターン〉|
           (bind 〈パターン変数〉 〈パターン〉)|
           (test 〈Lisp 式〉)
〈動作〉::= (make 〈事実情報〉)|
           (remove 〈事実 ID〉)
```

＊パターンの先頭要素は bind, test ではないものとする．
＊{ }+ は，{ }内を1回以上繰り返すことを表す．

図 1 プロダクションルールの構文則
Fig. 1 Syntax of production rules.

2.3 プロダクションメモリ

PM は図1の形式を基本とするプロダクションルールの集合である．説明を以下に示す．

① パターンは事実情報と同じ形式のリストである．ただし，パターン変数を任意の場合に含んでよい．パターン変数は $?x, ?y, \cdots$ と表す．？は無名変数である．パターン変数の有効範囲は1ルール内である．

② 条件部（LHS：Left Hand Side）に記述されたパターンは WM と照合される．bind は，パターンと照合が成功した事実の ID をパターン変数に束縛する．test は Lisp 式を評価し，その返却値を条件判定結果とする．

③ 実行部（RHS：Right Hand Side）には事実 DB を（したがってその部分集合である WM を）更新する操作を指定する．事実情報を指定して事実の生成を行う make，事実 ID を指定して事実の除去を行う remove が記述できる．make では事実情報のかわりにパターンを指定してもよい．この場合には，パターン中のパターン変数を束縛値に置換した結果得られる事実情報が指定されたものと見なされる．（この時，未束縛なパターン変数は未定義な値を示す '?' に置換される．）同様に，remove では事実 ID のかわりに事実 IDが束縛されているパターン変数を指定してもよい．

2.4 プロダクションシステムの実行モデル

PS の実行モデルは，文献6)に示されたものを用いる．以下，要点だけを述べる．

* PRISM[7] では，WM と long-term memory（事実 DB のことをこう呼んでいる）の階層が提供されている．ただし，WM は long-term memory の部分集合では必ずしもない．本論文で提案する基本モデルでは，WM は常に事実 DB の部分集合であり，データはコピーされない．すなわち，WM は事実 DB 上の View となっている．

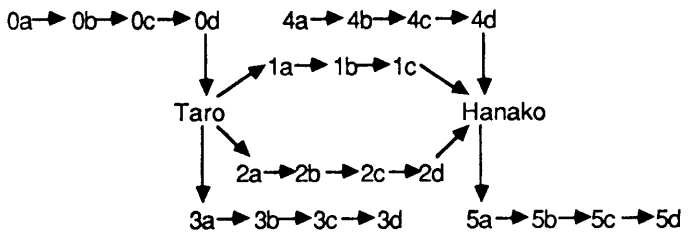
PSの実行は，①Match（すべてのルールについて，LHSとその時点のWMとの照合を行う），②Conflict Resolution（照合が成功したルールの中から1つを定められた戦略に従って選び出す），③Act（選択されたルールを実行し，事実の追加除去を行う）のサイクルを繰り返す．データ駆動型PSではルールの条件部はRETEネットワークと呼ばれるデータフローグラフに変換される．Actフェーズで事実がWMに追加されると，その事実がRETEネットワーク中に流し込まれ，変化に伴うネットワークの更新が行われる．この更新（Matchフェーズに相当する）は，以下の手順で進められる．まず，LHSの各条件に対して1つの条件内で完了するテスト（1-input-test）が行われ，テストを通過した事実がネットワーク内に蓄えられる．その後，条件間でパターン変数の値が矛盾しないかどうかを調べるテスト（2-input-test）が順に行われ，テストを通過した事実の組がネットワーク内に蓄えられていく．LHSのすべての条件を満たした事実の組（インスタンシエーション）は，RETEネットワークの終端（terminal）に到達し，対応するルールを発火可能とする．RETEネットワークの更新が完了すると，処理はConflict Resolutionフェーズに移行する．

```
(defrule relation-chain
 (older-than ?node1 ?node2)
 (older-than ?node2 ?node3)
 -->
 (make (older-than ?node1 ?node3)))
```

（1） 前向き推論ルール

```
(defrule relation-chain
 (goal older-than ?node1 ?node3)
 (older-than ?node1 ?node2)  -----  (a)
 (older-than ?node2 ?node3)  -----  (b)
 -->
 (make (older-than ?node1 ?node3)))
```

（2） 後向き推論ルール




```
(older-than 0a 0b)
(older-than 0b 0c)
(older-than 0c 0d)
(older-than 0d Taro)
:
```

（3） ワーキングメモリ(意味ネットワーク)

```
(goal older-than Taro Hanako)
(goal older-than Taro ?)
(goal older-than 1a   Hanako)
(goal older-than 1a   ?)
:
```

（4） 生成されるgoal

図2 探索制御の問題点
Fig. 2 Problems on controlling SN exploration.

3. プロダクションシステムによる意味ネットワーク探索の問題点

3.1 探索制御の問題点

（1） 前向き推論での問題点

図2(1)に，2個の関係から新たな関係を導く，SNの規則を表した前向きのプロダクションルールを示す．このルールを図2(3)の意味ネットワークを表すWMに適用すると，(older-than Taro Hanako)を含むすべての組合せが生成され始める．目標とする事実が生成されるまでの探索範囲は，深さ優先の競合解決戦略よりも，幅優先の方が狭い．しかし，いずれにしても，目的の事実が(older-than Taro Hanako)だけであれば，適切な制御を持たないこの方式での多くのルールの発火は無駄となる．この例では，深さ優先で61個の事実が，幅優先で50個の事実が生成される．ルールの発火総数はさらに多く，深さ優先で186回，幅優先で63回である*．

（2） 後向き推論での問題点

図2(2)に後向き推論のルールを示す．このルールに(goal older-than Taro Hanako)が与えられると，(older-than Taro Hanako)を目標として深さ優先で探索が開始される．ルール中の(a)，(b)の条件で，その時点の変数束縛を用いて自動的にsub-goalが生成されるとしよう[4]．この後向き推論には以下に示す問題が存在する．

① goalが多数生成される．

ルール発火前に多数のgoalが生成される．図2(4)に生成されるgoalの一部を示す．この例では28個のgoalが生成された後，ルールの発火が始まる．目標とする事実が導かれるまでに生成される事実は6

* これらの数字は，厳密には意味ネットワークを構成する事実の生成順序に依存する．

個，ルールの発火回数は6回であるが，PS の実行時間を決定するのは発火回数ではなく，むしろ goal を含む RETE ネットワークの変更回数である．PS では，goal も他の事実と同様に RETE ネットワークに保持されるので，goal が多数生成されることは，記憶効率上も実行効率上も好ましくない．

② goal の変更ごとに RETE ネットワークが再計算される．

RETE ネットワークでは，LHS の条件の記述順に従って 2-input-test が計算される．このため，先頭に記述された goal が変更されると以降の 2-input-test がすべて再計算されることになる．すなわち，この方式では計算の途中結果を保存するという RETE ネットワークの利点が生かされないことになる．RETE ネットワークの処理では 2-input-test の計算の占める割合が大きいため，特に事実数が増大すると，goal の変更による再計算は，性能を著しく劣下させる原因となる．

以上，両推論方式の問題点を述べた．この例では，理想的には3個の事実の生成，3回のルールの発火で (older-than Taro Hanako) が得られるはずであるので，これまでに述べた結果から，データ駆動型の計算メカニズムに適した推論制御機能が必要であることが分かる．

3.2 多重経路探索の問題点

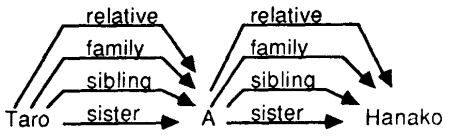
SN 中に同一の事実を導く経路が多数存在する原因としては以下のものが考えられる．

(1) 事実が冗長であるため，経路が多数生じる場合．

図3 に例を示す．この例では，sister, sibling, family, relative の4階層の関係間の is-a が定義されている．したがって，経路の数は 4×4=16 通りである．探索に要する経路が長くなれば経路の数は爆発する．冗長な事実を RETE ネットワークから除去し，多重経路の探索を防ぐメカニズムの導入が必要である．

(2) 事実には冗長性はないが，経路が多数存在する場合．

図2(3)の Taro, Hanako 間の経路がこれにあたる．通常，必要とされるのは事実情報であって，事実が成立するための根拠をすべて数え上げる必要はない．したがって，1つの経路で目標の事実情報が求まれば他の経路の探索を中止することが必要である．




```
(is-a  family  relative)
(is-a  sibling family)
(is-a  sister  sibling)
(relative Taro  A)
(family   Taro  A)
(sibling  Taro  A)
   :
```

図3 冗長な事実による多重経路
Fig. 3 Redundant paths in SN.

3.3 事実の大規模化により生じる問題点

RETE ネットワークはルールの条件部の判定に必要な計算の途中結果をすべて保存するものである．事実の規模が増大すれば，特に 2-input-test の計算量は膨大になる．また，計算結果をすべて主記憶上に保持することも困難になる．しかし，たとえ膨大な事実を対象とする探索であっても，探索を進めるために常に事実 DB 中のすべての事実が必要なわけではない．探索に必要となった時点で事実を RETE ネットワークにロードし，不要となればアンロードする機能（注視点制御機能）が必要である．

4. 意味ネットワーク探索に必要な機能の導入

3章で述べた問題点を解決するために，以下の機能を PS に導入する．

4.1 前向き推論の制御機能

この節では前向き推論を制御するための機能として goal と対照的な gate を導入する．その前に準備としてパターンマッチの分類を行っておく．

(1) パターンマッチの分類

図4 にパターンマッチの例を示す．事実情報中に未定義な値 '?' の存在を許しているためパターンマッチには以下の種別が生じる．

fact-specific： パターンはより詳細 (specific) に定義された事実情報とのみ照合が成功する．(すなわち，

pattern	(a b ?)			
fact	(a b c)	(a b ?)	(a ? c)	(a ? ?)
fact-specific	match	match	no match	no match
pattern-specific	no match	match	no match	match
mutual	match	match	match	match
literal	no match	match	no match	no match

図4 パターンマッチの分類
Fig. 4 Classification of pattern matching.

パターン中の定数と事実中の'?'とは照合が成功しない.)

pattern-specific: パターンはより一般的(general)に定義された事実情報とのみ照合が成功する.(すなわち,パターン中の'?'と事実中の定数とは照合が成功しない.)

mutual: パターンは矛盾しない限り,事実情報と照合が成功する.(すなわち,パターン中および事実中の'?'はすべての定数と照合が成功する.)

literal: パターンは文字どおりに事実情報と照合される.(すなわち,パターン中および事実中の'?'は'?'とのみ照合が成功する.)

同様に 2-input-test は以下のように分類される.

right-specific: 右入力の束縛が,より詳細な場合に照合が成功する.

left-specific: 左入力の束縛がより詳細な場合に照合が成功する.

このほか,1-input-test と同様に mutual, literal な照合を考えることができる.

(2) gate の導入

新たに導入する gate 機能を従来からの goal と対比して述べる.

goal: ルールの条件部に記述されたものを goal パターン,WM 中のものを単に goal と呼ぶ.goal パターンは,WM 中のより specific な goal とのみ照合が成功する.goal パターンは,プログラミングスタイルとしては LHS の先頭に置かれ,戦略的にルールを選別する(rule filtering)のに用いられる.すなわち,詳細に定義された goal ほど(目的がはっきりしているほど),goal パターンを持つルールを発火させやすい.

gate: 一方,gate パターンは,WM 中のより general な gate とのみ照合が成功する.gate パターンはスタイルとしては,LHS の末尾に置かれ,ルールの発火を引き起こすインスタンシエーションを選別する(instantiation filtering)のに用いられる.すなわち,詳細でない gate ほど(広い門ほど),gate パターンを持つルールを発火させやすい.

goal および gate 機能の実現には,以下のようにパターンマッチを組み合わせればよい*.

goal: goal パターンに対応する 1-input-test では fact-specific な照合を行い,goal を右入力とする 2-input-test では right-specific な照合を行う.

gate: gate パターンに対応する 1-input-test では mutual な照合を行い,gate を右入力とする 2-input-test では left-specific な照合を行う.

(3) any の導入

gate の効果をより高めるために,従来からの not と対照的な any を LHS に導入する.

not: 形式は (not 〈パターン〉).

not は条件パターンと照合が成功する事実がまったく存在しなければ満足される.

any: 形式は (any 〈パターン〉).

any は条件パターンと照合が成功する事実が1つでも存在すれば満足される.

not と any は単にチェックに用いられるので,処理中に生じた変数束縛は以降の条件照合に影響しない.(any は Prolog や POPS 2[8] の2重否定と同様にふるまう.)

図5に gate を用いてインスタンシエーションの選別を行う様子を示す.

4.2 事実の非活性化機能

3.2 節で述べた多重経路の探索を防ぐ手段として,事実を非活性化する方法を述べる.まず,事実のメタ情報として,事実の状態と事実の根拠を導入する.次にそれらを自動的にメンテナンスできるよう,make, remove 等の事実操作プリミティブを拡張する.

(1) 事実の状態

事実の活性化のレベルを以下の3状態で表す.

active: 活性化された事実である.探索に用いられる.

sleep: 事実は論理的な根拠を持つが,冗長であるため非活性化されている状態である.事実 DB には存在するが探索には用いられない.(すなわち,その事実は RETE ネットワーク中には存在しない.)

killed: 論理的な根拠がなく,除去されるべき事実であることを示す.しかし,何らかの理由で復活する可能性があるため事実 DB に残されている.例えば,矛盾を解消するために,除去される事実等がこれにあたる.探索には用いられない.(すなわち,RETE ネットワーク中には存在しない.)

(2) 事実の根拠

事実の根拠として以下の2種を導入する.

in: 事実が存在するための"根拠"のリスト(要素の並びは or を表す)である.各"根拠"は事実 ID のリスト(要素の並びは and を表す)である.無条件に

* goal, gate 以外のパターンでは,対応する 1-input-test で fact-specific な照合を行い,そのパターンを右入力とする 2-input-test では literal な照合を行っている.

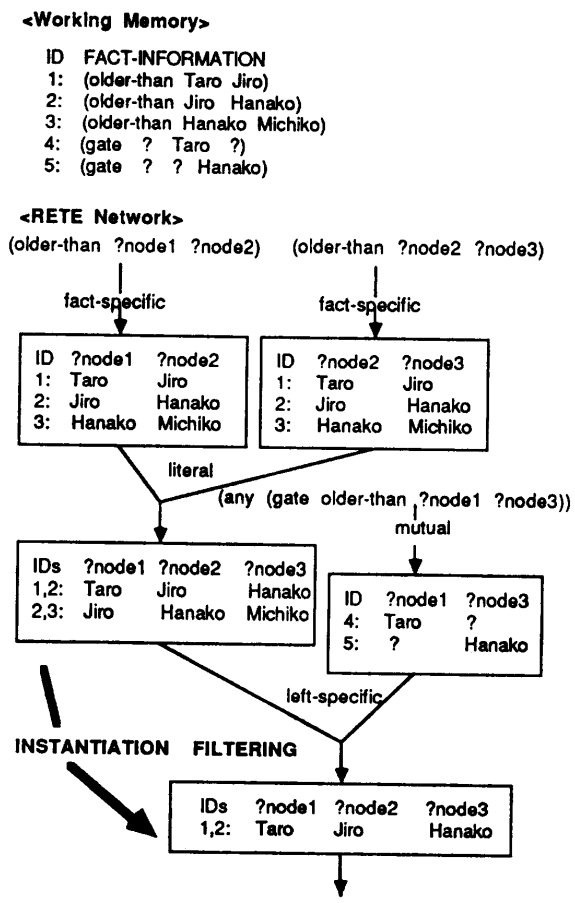


図 5 gate の処理例
Fig. 5 Example of gate processing.

生成された事実の根拠は nil とする.

out: 事実を sleep, killed 状態とした "根拠" のリストである. 各 "根拠" は事実 ID のリストである. 無条件に sleep あるいは kill された場合には根拠として nil が設定される.

図 6 に事実の状態と根拠の例を示す.

(3) 事実操作プリミティブ

事実に状態と根拠を導入したので, それらを自動的にメンテナンスできるよう事実操作プリミティブを拡張する*.

make: 形式は (make 〈事実情報〉 [: in 〈根拠〉]).

〈根拠〉はこの事実生成の前提となった事実 ID のリストである. make では, まず同一の〈事実情報〉を持つ事実が既に存在しているかどうかをチェックする. 存在しなければ事実を生成し active 状態とする. 次に, 生成された事実, あるいは既に存在していた事実の in リストに〈根拠〉を追加する. 〈根拠〉が指定されていない場合には nil を in リストに追加する.

remove: 形式は (remove 〈事実 ID〉).

指定された事実を除去する. 事実 DB 中のすべての事実の in リストおよび out リストから, 除去された事実を含む根拠を除去する. この結果, もし in リストが空になる事実があれば, その事実を remove する. out リストが空になる事実があれば, その事実を active 状態とする.

sleep: 形式は (sleep 〈事実 ID〉 [:out 〈根拠〉]).

〈根拠〉はこの事実が sleep させられる原因となった事実 ID のリストである. sleep では, まず指定された事実を sleep 状態とする. 次に〈根拠〉を out リストに追加する. もし〈根拠〉が指定されていなければ nil を追加する.

kill: 形式は (kill 〈事実 ID〉 [: out 〈根拠〉]).

指定された事実を killed 状態とする. 次に〈根拠〉を out リストに追加する. もし〈根拠〉が指定されていなければ nil を追加する. 事実 DB 中のすべての事実の in リスト, および out リストから kill された事実を含む〈根拠〉を除去する. この結果, もし in リストが空になる事実があれば, その事実を remove する. out リストが空になる事実があれば, その事実を active 状態とする.

(4) 事実の状態の変化と RETE ネットワークとの関係

従来の PS では, WM 中にすべての事実が存在することを仮定していた. しかし, 事実に状態を導入したことによってこの仮定はくずれることになる. WM は, 探索に不要な sleep, killed 状態の事実は含まない. 事実の状態の変化が RETE ネットワークにどのような影響を与えるかを以下にまとめる.

```
(1 ((status active) (in nil)) (older-than Taro 1a))
(2 ((status active) (in nil)) (older-than 1a Hanako))
(3 ((status active) (in nil)) (older-than Taro 2a))
(4 ((status active) (in nil)) (older-than 2a Hanako))
(5 ((status active) (in nil)) (younger-than Taro Hanako))
(6 ((status killed) (in (3 4) (1 2)) (out (5)))
                    (older-than Taro Hanako))
```

*事実 6 は 2 つの存在根拠をもっている.
事実 1, 2 が共に存在すれば存在する.
あるいは, 事実 3, 4 が共に存在すれば存在する.
*事実 6 は事実 5 と矛盾したため, killed 状態にある.

図 6 事実の例
Fig. 6 Example of facts.

* 事実の根拠の表現, およびそのメンテナンス機構は, Doyle の TMS[9] を参考にしている. しかし, 事実 DB の冗長度をどのレベルに保つか, あるいはどの事実を誤りと判断するかは利用者が記述するプロダクションルールに任せている. これらの判断は, 一般にアプリケーションによると考えているからである.

① active 状態から active 以外の状態 (remove された状態を含む) に変化した場合にはその事実を RETE ネットワークから除去する.

② active 以外の状態 (存在しない状態を含む) から, active 状態に変化した場合にはその事実を RETE ネットワークに加える.

4.3 事実データの注視点制御機能

前節では冗長な事実を非活性化し, RETE ネットワークに保持しない方法を示した. しかし, 大規模な事実 DB を対象とする場合には, たとえ active な事実であってもすべてを RETE ネットワークに保持することはできなくなる. 探索に必要な事実だけを選択してネットワークに保持するために, さらに以下の3種の機能を導入する.

(1) 事実 DB 中の事実を RETE ネットワークに保持せずに, ルールから直接参照する機能.

access: 形式は (access 〈パターン〉).

LHS に記述する. access が指定された条件を右入力とする 2-input-test が実行されるごとに, その時点でのパターン変数の束縛を用いて, 〈パターン〉と照合が成功する事実を事実 DB から検索する. 単に〈パターン〉を指定した場合とは異なり, 照合が成功した事実を RETE ネットワーク中に保持することはしない. したがって, 〈パターン〉と照合が成功する事実は, ルールの実行中に更新されないというのが前提である. たとえ更新されても, その更新が引き金となってルールが発火することはない.

(2) 事実 DB 中の事実を, 推論に必要となった時点で自動的に RETE ネットワークにロードする機能.

load: 形式は, (load 〈パターン〉).

LHS に記述する. load が指定された条件を右入力とする 2-input-test が実行されるごとに, 〈パターン〉と照合が成功する事実 DB 中の事実を, RETE ネットワークにロードする. 事実 DB の検索結果を RETE ネットワーク中に保持する点が access と異なっている. この機能は探索に必要な事実を on-demand に RETE ネットワークにロードするのに有効である.

(3) 事実 DB から RETE ネットワークへの事実のロード/アンロードを, ルールの実行部できめ細かく制御する機能.

load-wm/unload-wm: 形式は, (load-wm 〈パターン〉)/(unload-wm 〈パターン〉).

RHS に記述する. load-wm は〈パターン〉と照合が成功する事実 DB 中の事実を RETE ネットワークにロードする. unload-wm はその逆の機能である. 前述の load では実現できないきめ細かな制御を記述するのに用いる.

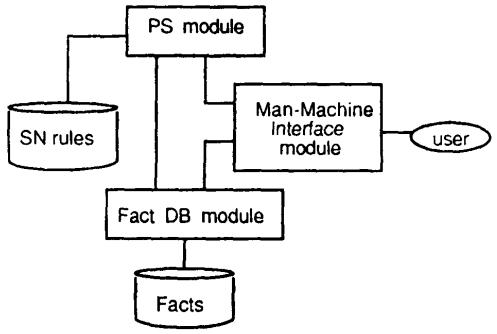


図 7 PLANET の構成
Fig. 7 System configuration of PLANET.

5. 試作・評価

4章で述べた言語機能を備えたデータ駆動型のプロダクションシステム PLANET を試作し評価を行った. PLANET の構成を図7に示す. PLANET は, ①事実 DB モジュール, ②PS モジュール, ③マンマシンインタフェースモジュールから構成されている. 事実 DB モジュールは, PS モジュールからだけではなく, 手続き的なプログラムからも自由に呼び出すことができる. マンマシンインタフェースモジュールはビットマップディスプレーを用いて, ネットワークを図形表示するモジュールである.

5.1 推論制御機能の評価

図 8(1)に gate と any を用いた SN 探索のためのルールを示す. このルールに, (gate older-than Taro ?) が与えられると, Taro と関係する事実だけが生成される. 図 8(2)は goal から gate を生成するルールを表している. 今, (goal older-than Taro Hanako) が与えられると, このルールが発火し, (gate older-than Taro ?) と (gate older-than ? Hanako) が生成される*. 競合解決戦略を幅優先 (LEX[3]の逆順) にしておくと, Taro と Hanako の2点を中心として同心円を描くように探索を進めることができる. PLANET では, goal はその goal を満たす事実が生成された時

* 現状では目標とする事実を求めるために必要な gate の生成は, 利用者にまかされている. SN が複雑になり, 必要十分な gate を生成することが容易でない場合には, 例えば (gate ? Taro ?) のように, 単に端点だけを指定する gate を生成してもよい. SN の規則を解析し, 必要となる gate を自動的に生成することは今後の課題である.

```
(defrule relation-chain
 (bind ?fact1 (older-than ?node1 ?node2))
 (bind ?fact2 (older-than ?node2 ?node3))
 (any (gate older-than ?node1 ?node3))
 -->
 (make (older-than ?node1 ?node3) :in (?fact1 ?fact2)))
```

（1） gate を用いた前向き推論ルール

```
(defrule gate-1
 (bind ?fact1 (goal older-than ?node1 ?node2))
 (test (instantiated-p ?node1))
 -->
 (make (gate older-than ?node1 ?) :in (?fact1)))

(defrule gate-2
 (bind ?fact1 (goal older-than ?node1 ?node2))
 (test (instantiated-p ?node2))
 -->
 (make (gate older-than ? ?node2) :in (?fact1)))
```

（2） goal から gate を生成するルール

＊ instantiated-p は引数が未定義な値を含まないときに，t を返却する関数.

図 8 gate を用いた探索制御の例
Fig. 8 SN exploration under gate control.

に kill される．goal が kill されると，その goal から生成されていた gate が除去され，Taro と Hanako の関係の探索が停止する．実行例を**図 9** に示す．3 章と同じ問題を実行させると，生成される事実は 11 個，ルールの発火総数は 13 回である．

さらにこの後，何らかの理由で目標とした事実が除去されると，killed 状態の goal が再び active 状態となり，gate が再度生成され，以前探索を中止した状態から探索が続行される．この結果，異なる経路により目標とする事実が生成され，探索は再び停止する．

図 10 に PLANET が提供する SN 探索用コマンドの使用例を示す．match は事実 DB の内容検索を行うもので，関係をたどる探索は行わない．一方，ask は SN の規則を用いて探索を行い，match では得られなかった事実を導き出す．すなわち，図 10 の処理の背後では図 9 の処理が実行されている．ask では，必要な事実の個数を指定し，探索を途中で停止させることも可能である．

図 11 は，事実が大規模化した場合の，各種推論制御方式の性能特性を示している．横軸は SN を構成する事実の個数を表している．SN はこれらの事実からメッシュ状に構成されているものとする．縦軸は，SN の中央付近で関係を 4 回たどることによって新たな関係を求める過程で，生成される事実の個数を表している*．（後向き推論で生成される goal の個数を含む．）

* 縦軸にルールの発火回数をとらないのは，ルールの発火回数がデータ駆動型 PS の処理時間の単位を表すのに適切ではないからである．例えば，WM を変更しないルールの発火時間はほとんど無視できる．

```
PLANET> (make (goal older-than Taro Hanako))
  =>a (26 ((status active) (in nil)) (goal older-than Taro Hanako))

PLANET> (run)
1 gate-2 (26)
  =>a (27 ((status active) (in (26))) (gate older-than ? Hanako))
2 gate-1 (26)
  =>a (28 ((status active) (in (26))) (gate older-than Taro ?))
3 relation-chain (5 6)
  =>a (29 ((status active) (in (5 6))) (older-than Taro 3b))
4 relation-chain (9 10)
  =>a (30 ((status active) (in (9 10))) (older-than Taro 1b))
5 relation-chain (11 12)
  =>a (31 ((status active) (in (11 12))) (older-than Taro 2b))
6 relation-chain (13 14)
  =>a (32 ((status active) (in (13 14))) (older-than 1b Hanako))
7 relation-chain (16 17)
  =>a (33 ((status active) (in (16 17))) (older-than 2c Hanako))
8 relation-chain (20 21)
  =>a (34 ((status active) (in (20 21))) (older-than 4c Hanako))
9 relation-chain (7 29)
  =>a (35 ((status active) (in (29 7))) (older-than Taro 3c))
10 relation-chain (13 30)
  =>a (36 ((status active) (in (30 13))) (older-than Taro 1c))
11 relation-chain (15 31)
  =>a (37 ((status active) (in (31 15))) (older-than Taro 2c))
12 relation-chain (10 32)
  =>a (38 ((status active) (in (10 32))) (older-than 1a Hanako))
13 relation-chain (30 32)
  =>a (39 ((status active) (in (30 32))) (older-than Taro Hanako))
k<=a (26 ((status killed) (in nil) (out (39)))
                                  (goal older-than Taro Hanako))
  <=a (27 ((status deleted)) (gate older-than ? Hanako))
  <=a (28 ((status deleted)) (gate older-than Taro ?))
>> 13 productions are fired.
```

＊ k<=a は事実の状態が active から killed に変化したことを表す.

＊ deleted 状態は事実が除去されたことを示す．除去された事実を残しているのは，デバグのためである.

図 9 SN 探索の実行例
Fig. 9 Example of SN exploration.

```
PLANET> (match (older-than Taro ?))
   5 : (older-than Taro 3a)
   9 : (older-than Taro 1a)
  11 : (older-than Taro 2a)
>> 3 facts are found.

PLANET> (ask (older-than Taro Hanako))
   => 39 (older-than Taro Hanako)
>> 1 fact is found.
>> 10 facts are created.
>> 13 productions are fired.

PLANET> (ask (older-than Taro ?) :no 2)
   => 42 (older-than Taro 2d)
   => 43 (older-than Taro 5a)
>> 2 facts are found.
>> 3 facts are created.
>> 9 productions are fired.

PLANET> (more 2)
   => 44 (older-than Taro 5b)
   => 45 (older-than Taro 5c)
>> 2 facts are found.
>> 2 facts are created.
>> 3 productions are fired.
```

図 10 PLANET による SN 探索例
Fig. 10 SN exploration with PLANET.

図11から以下のことが言える.

① 前向き推論では探さ優先深索に比べて幅優先探索が有利である.

② 後向き推論では生成される goal が負担となって,効率が劣化する.探索過程で生成される事実のほとんどは goal であり,その生成個数は意味ネットワークの規模に比例して増大する.

③ gate を用いた前向き推論では,SN が大規模化しても探索過程で生成される事実の個数には変化がない.したがって,他の方式 (SN の規模の増大に比例して,性能が劣化する) に比べて SN の大規模化に極めて強い方式であることが分かる.

5.2 事実の非活性化機能の評価

図 12 は冗長な事実を sleep させるルールの例である.これによって,図3に示した多重経路の探索を防ぐことができる.具体的には以下のように処理が行われる.

① **圧縮**: まず Taro と A, A と Hanako の関係を示す事実のうち,冗長なものが sleep 状態となる (図12参照).したがって,active 状態に残るのは,(sister Taro A) と (sister A Hanako) だけとなる.

② **探索**: (sister Taro Hanako) が (sister Taro A) と (sister A Hanako) から導かれる.

③ **展開**: 利用者からの検索要求等,必要があれば (relative Taro Hanako) 等が,(sister Taro Hanako) から導かれる.ただし,これらの冗長な事実はただちに sleep 状態となる.

この結果ルールの発火回数は,is-a の階層の深さを d, 関係をたどる回数を n とすると,従来 $O(d^n)$ であったものが,$O(d+n)$ に減少する.

5.3 事実データの注視点制御機能の評価

事実データの注視点制御機能を用いると,あらかじめ事実 DB 中の事実をすべて RETE ネットワークに保存する必要はなく,探索している範囲内の事実 (5.1 節の例では同心円内の事実) だけを WM に含むようにすることができる.

図 13 は,探索に必要となった時点で (すなわち on-demand に) 事実を RETE ネットワークにロードするようにした場合の性能を表している.測定対象の SN は図11のものと同一であり,探索方式は gate を用いた前向き推論である.gate を用いた推論では,探索過程で生成される事実の個数が SN の大きさによらず一定であるので,縦軸には CPU 時間の相対値をとっている.事実 DB はあらかじめ主記憶上に存

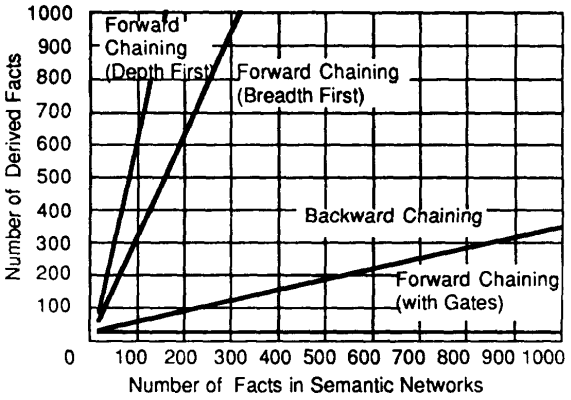


図 11 各種推論制御方式の評価
Fig. 11 Comparison of various control mechanisms.

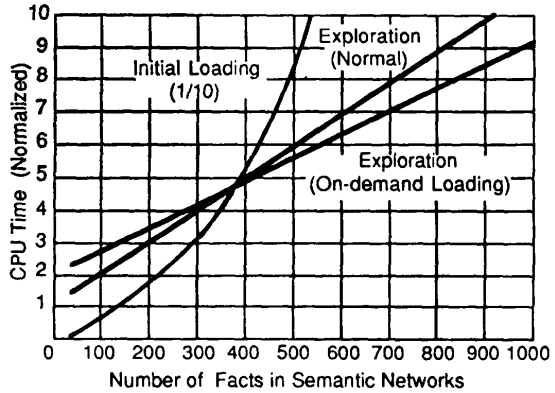
```
PLANET> (show-wm)
  1 : (relative Taro a)
  2 : (family Taro a)
  3 : (sister Taro a)
  4 : (is-a family relative)
  5 : (is-a sister family)
>> 5 facts are found.

PLANET> (defrule sleep-redundant-fact
          (bind ?special-fact (?relation1 ?node1 ?node2))
          (bind ?general-fact (?relation2 ?node1 ?node2))
          (bind ?is-a (is-a ?relation1 ?relation2))
          -->
          (sleep ?general-fact :out (?special-fact ?is-a)))
>> production sleep-redundant-fact is defined.

PLANET> (run)
1 sleep-redundant-fact (2 1 4)
   s<=a (1 ((status sleep) (in nil) (out (2 4))) (relative Taro a))
2 sleep-redundant-fact (3 2 5)
   s<=a (2 ((status sleep) (in nil) (out (3 5))) (family Taro a))
>> 2 productions are fired.

PLANET> (show-wm)
  3 : (sister Taro a)
  4 : (is-a family relative)
  5 : (is-a sister family)
>> 3 facts are found.
```

図 12 事実の非活性化の例
Fig. 12 Example of inactivating facts.



初期ロードに要する CPU 時間は図の 10 倍である.
図 13 注視点制御方式の評価
Fig. 13 Effectiveness of on-demand fact loading.

在するものと仮定し，ファイルアクセス時間は考慮していない．縦軸が CPU 時間であるため，グラフの詳細は処理系に依存するが，以下のことが言える．

① 探索時間は事実 DB が大規模化すると，on-demand に事実をロードする方式が有利となる．これは on-demand に事実をロードすることによって，総計算量に占める割合の大きい 2-input-test が必要最小限におさえられるからである．

② あらかじめ事実をすべてロードする場合には，事実 DB が大規模化すると RETE ネットワークの初期ロードに膨大な時間が必要となる．一方，on-demand に事実をロードすると初期ロードそのものが不要となる．

上記以外に，より現実に近い応用例として，物語中に登場する人物の関係を SN で表現し評価を行った．SN のノード（人物）数は約 50，リンク数は夫婦，親子を表現するもの等，約 80 である．この SN に対して関係を数回たどる探索を行い，探索時間を測定した．その結果，この程度の規模の SN においても本論文で提案した各種手法を用いた探索は，従来の前向き，後向きの探索に比べ，数〜10 数倍性能が良いことを確認している．

6. むすび

データ駆動型のプロダクションシステムに，① 前向き推論の制御機能，② 事実の非活性化機能，および ③ 事実データの注視点制御機能を導入することにより，意味ネットワーク探索を効率化できることを示した．

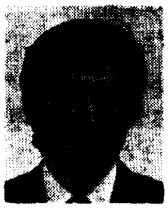
データ駆動型のプロダクションシステムはエキスパートシステム構築用ツールとして広く用いられているものであるが，事実データの数が増大すると性能が極端に劣化することが問題とされている．本論文で述べた手法は意味ネットワークの探索という応用に限らず，事実データの個数が多いプロダクションシステムの応用に幅広く適用できると考えている．今後の課題としては，提案した機能を用いて，大規模な事実データベースを対象とする応用システムを構築すること[10]，および既存のデータベースシステムとの結合を図ることが残されている．

謝辞 本研究の機会を与えていただいた堀内敬之主幹研究員，服部文夫主幹研究員，並びに日頃討論いただく研究室各位に感謝します．

参考文献

1) 田中穂積，小山晴生，奥村 学：知識表現形式 DCKR とその応用，ソフトウェア科学会，論理と自然言語ワークショップ (1986)．
2) 半田剣一，樋口哲也，古谷立美，国分明男：意味記憶システム IX —IXL による知識表現—，知識工学と人工知能研究会，39-7 (1985)．
3) Forgy, C. L.: OPS5 User's Manual, Tech. Report CS-79-132, Dept. of Computer Science, Carnegie-Mellon University (1979).
4) ART Reference Manual, Inference Corp. (1986).
5) Fox, M. S.: On Inheritance in Knowledge Representation, *6th IJCAI* (1979).
6) Forgy, C. L.: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol. 19, pp. 17-37 (1982).
7) Brownston, L., Farrell, R., Kant, E. and Martin, N.: *Programming Expert System in OPS5: An Introduction to Rule-Based Programming*, Addison-Wesley (1985).
8) 広瀬紳一：強力なパターンマッチング機能を持ったプロダクションシステム記述言語 POPS2，日本ソフトウェア科学会，知識プログラミングシンポジウム資料，KP-85-8 (1985)．
9) Doyle, J.: A Truth Maintenance System, *Artif. Intell.*, Vol. 12, pp. 231-272 (1979).
10) 石田 亨：ファクト管理を強化したプロダクションシステムとその応用，人工知能学会全国大会論文集，pp. 115-118 (1987)．

(昭和 61 年 11 月 6 日受付)
(昭和 62 年 9 月 9 日採録)



石田 亨（正会員）

昭和 28 年生．昭和 51 年京都大学工学部情報工学科卒業．昭和 53 年大学院修士課程修了．同年日本電信電話公社に入社．昭和 58〜59 年，コロンビア大学客員研究員．現在，NTT 情報通信処理研究所知識処理研究部勤務．プログラミング環境，知識ベースシステム，並列処理に興味を持つ．電子情報通信学会，人工知能学会，ソフトウェア科学会，AAAI 各会員．