演繹データベースの問合せ処理

ICOT
第５研究室
三菱電機(株)

世　木　博　久

演繹データベース

知識（事実、ルール）　←　論理式による表現

演繹規則　←　一階述語論理

データベースに明示的に書かれている情報のみならず、
論理的に演繹できる情報も引き出す

←　関係データベースの拡張

いかに効率的に情報を引き出すか？

→　問合せ処理アルゴリズムの研究の重要性

問合せ処理アルゴリズムの満たすべき性質

- 健全性
- 完全性
- 停止性

解が有限個の場合に、全解を求めて停止する一般的なアルゴリズムは存在しない

→ あるクラスのデータベースに対して、全解を求めて停止する効率的なアルゴリズムを見つける（例えばdatalog）

問合せ処理アルゴリズムの種類

- 下降（トップダウン）型アルゴリズム … 後向き推論

G ← G1, . . . , Gn

==⇒ ルールの結論部のゴールを条件部のサブゴールへ分解

問合せから出発、単位節（ファクト）に到達すると成功

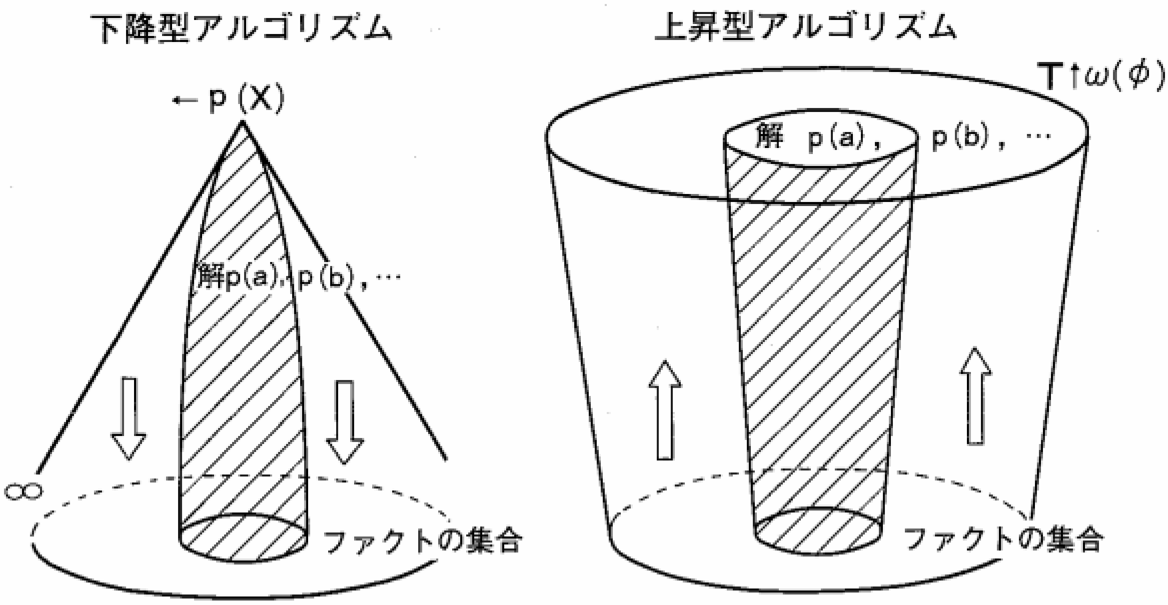
- 上昇（ボトムアップ）型アルゴリズム … 前向き推論

G ← G1, . . . , Gn

⇐== ルールの条件部が成立する時、結論部を導く

ファクトから出発、

ルール集合の不動点が導かれたら終了



goal-directedな計算
無限ループに陥る可能性

必ず停止（datalogの場合）
ゴールと無関係な計算

上昇型アルゴリズムの改良：基本的な考え方

☆ 上昇型計算をgoal-directedにする

→ 下降型計算時の変数の束縛情報の流れを上昇型計算にも導入

… マジックセット法、Alexander法、制約子法

節： p ← a, b, c

＝＝＞ 書き換え

p ← call_p, a, b, c

ルールの発火を制限するアトムを節の本体に付け加える

どのように新アトムcall_p を定義するか？

・ 問合せに無関係な計算を行わず、解を失わない

＝＝＞ 下降型計算で生成されるｐに関する問合せを含むように定義する

ルールの書換え

・　問合せ　：　← p

＝＝＞　ファクト：　call_p　を導入

・　各　プログラム節：p　←　a,　b,　c

＝＝＞　ｃａｌｌアトムを定義するルール：

call_a　←　call_p

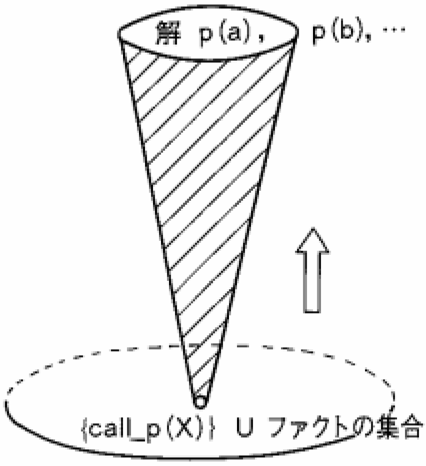
call_b　←　call_p,　a

call_c　←　call_p,　a,　b

及び、解ｐを導くルール：

p　←　call_p,　a,　b,　c

ルールの書換えによる探索空間の制限



制約子callアトムの導入によるルール発火の制限

call_a ← call_p(X)，…

call_b ← call_p(X)，a

…

ルール書換えの最適化　…　共通表現（中間結果）用のアトムの導入

節 r ： p ← a, b, c　　　　改良版

⇒call_a←call_p	call_a←call_p
	cont(id(a), info(a))←call_p
call_b←call_p, a	call_b←cont(id(a), info(a)), a
	cont(id(b), info(b))←cont(id(a), info(a)), a
call_c←call_p, a, b	call_c←cont(id(b), info(b)), b
	cont(id(c), info(c))←cont(id(b), info(b)), b
p←call_p, a, b, c	p←cont(id(c), info(c)), c

id(g) ： 節 r のアトム g の識別子

info(g) ： サブゴール← g 以降の計算に必要な変数の束縛情報

例題：　グラフの到達可能性問題

プログラム path(X, Y) ← path(X, Z), path(Z, Y)　　…　（1）

path(X, Y) ← arc(X, Y)　　…　（2）

問合せ←path(a, X) に対するルール変換：

・問合せから　call_path(a, X)

・節（1）から： call_path(X, Z) ← call_path(X, Y)

cont(1-1, [X, Y, Z]) ← call_path(X, Y)

call_path(Z, Y) ← cont(1-1, [X, Y, Z]), path(X, Z)

cont(1-2, [X, Y, Z]) ← cont(1-1, [X, Y, Z]), path(X, Z)

path(X, Y) ← cont(1-2, [X, Y, Z]), path(Z, Y)

・　節（2）から：　path(X, Y) ← call_path(X, Y), arc(X, Y)

上昇型アルゴリズム：Alexander Templates

・マジックセット法、Alexander法の一般化

・任意のホーン節データベースに適用可能

（１）ルールの書き換え

プログラムＰ　→　ルール集合Ｒ

（２）上昇型計算

ルール集合Ｒの不動点$T \uparrow \omega (\phi)$を求める

セミナイーブ評価による効率化

（差分集合を用いて重複計算を除去）

Ａｌｅｘａｎｄｅｒ　Ｔｅｍｐｌａｔｅｓの性質

・　健全性

・　完全性

・　停止性　（任意のｄａｔａｌｏｇプログラムに対して）

☆　下降型アルゴリズムと同じ効率を上昇型の計算で実現している

下降型アルゴリズムとの比較

下降型アルゴリズムをどのくらい良くシミュレートしているか？

下降型	←G	上昇型	T↑1 ∋	call_G
幅優先	/ \			
	←G1,...		T↑2 ∋	call_G1,
	…			cont(id(G1),info(G1))...
	|			…
	□（解 G'）		T↑k ∋	G'

・　ノードと上昇型計算で得られるアトムの間に一対一の対応

・　ループチェック ⇌ セミナイーブ評価による重複除去

☆　下降型＋テーブル化機能の複雑な計算を、単純な上昇型計算で実現

今後の課題

（１）　拡張

ｄａｔａｌｏｇからより広いクラスへ

（２）　応用

全解探索機能を用いたさまざまな応用

…　プログラムの抽象解釈　［金森　９０］

（３）　アルゴリズムの並列化

ＫＬ１による効率的な実現