LCC- 論理コンパイラコンパイラ

5R－11

白川浩美 三浦孝夫 塩谷勇
産能大学 経営情報学部

1 まえがき.

知識処理を効果的に行うためのキーとなる技術は処理効率の向上にある．導出原理は，述語論理系で効率良い推論を実現するために提案されている方法であるが，前方向や後ろ方向の連鎖，深さ優先や幅優先の探索などを通じて，複数の推論経路を幾度となく処理する必要がある．[1]

これまで導出原理には数多くの戦略が提案されている [10]．これらは完全な形式システムでない場合もある．主なものだけでも次のようになろう：

- 冗長なリテラルや項を簡単化する：消去戦略など
- 導出形を得るための節の選択方法の制限：意味導出，線形導出，単位導出など
- 記号やリテラルに順序を設ける：ロック導出，OL導出，OI導出など
- 導出形を得るための節の選択の条件化：SLD導出など
- 途中結果を保存し，重複出現を抑える：OLDT導出など

これらは，共通して第一階述語論理の推論規則として導入されたもので，いずれかが決定的に効率よい結果を生むというものではない．例えば，ホーン節論理プログラムではSLD導出が用いられることが多く，演繹データベースではOLDT導出を意識する．知識処理の操作の向上からは，これらを使い分けることが重要であり，導出原理に基づく一般的な知識処理戦略を指示する基盤が必要となる．

論理コンパイラコンパイラ（Logic Compiler Compiler, LCC）はこのような目的を持つメタ導出原理システムで，次のような特徴を有する：

- 主要な導出原理戦略を記述できる
- 実行方式と独立して記述できる
- 実際の処理系は，項書換に基づく並列モデルを利用する
- メタ知識の記述を利用して，処理の停止性を決定できることがある

これまでの多くの知識処理法では推論過程を手続きと捕え，その制御を手続きの枠の内で扱おうとしている．導出過程は宣言的に捕えられないのであろうか？メタ知識処理を宣言的に表わすことができるならば，操作の意図が透明になり，戦略記述と（戦略を実行するための）実現を独立に論じることができる．例えば並列実行環境がどのような戦略を用いようと適用を探ることが可能である．宣言的な意図の記述は，また戦略のモジュラー化を促進し，例えば単一化の戦略と途中結果の保存方式を別個に表わすことも可能となろう．

ここで提案する方法は，メタ知識戦略を静的なデータ構造として捕えようとする．すなわち，節に対する操作を次のように与えることができる：

- 節の集まりを単位として次の動作を決定するため，代数操作を導入する
- 順序はリストで，情報は順序組で，また候補節は集合で表わされる
- これらの上で定義される操作を用いて導出形を決定する
- 必要なデータ構造は，ADTと見て宣言的に定義することができる [13]

代数的でありながら，処理の宣言性を失わないように定義できる点に注目されたい．筆者らの知る限り，このような代数的アプローチをとる提案はない．

2 LCCの構文

非論理公理と定理は節の形式に変換されているものとする．証明系列 P_n は導出のステップ n と節集合 S_n の組 $\lceil n, S_n \rceil$ の有限列である．

$$P_n = [\lceil 1, S_1 \rceil, \lceil 2, S_2 \rceil, \ldots \lceil n, S_n \rceil]$$

$\lceil i, S_i \rceil$ はある $\lceil j, S_j \rceil$ $(j < i)$ と $\lceil i-1, S_{i-1} \rceil$ のクラスタ演算 $\bowtie$ から定義される．

$$\lceil i, S_i \rceil = \lceil i-1, S_{i-1} \rceil \bowtie \lceil j, S_j \rceil = \{x \circledcirc y \mid x \in S_{i-1}, y \in S_j\}$$

$\circledcirc$ は2項導出を表す．証明過程は証明の戦略 C と証明系列の組

$$< C, P >$$

からなる．どの節集合上に対してクラスタ演算を施すかを決定するものが戦略であり，以下の関数を用いて記述される．

[1] H. Shirakawa, T. Miura and I. Shioya : LCC- Logic Compiler Compiler, Sanno College.

(1) $\uparrow P$：証明系列 P の先頭を取り出す．$\uparrow^n P$ は $\uparrow(\uparrow^{n-1} P)$ で定義される
(2) $\downarrow P$：証明系列の最後を取り出す．$\downarrow^n P$ は $\downarrow(\downarrow^{n-1} P)$ で定義される
(3) $\Uparrow P$：証明系列 P から先頭を除いた残りの証明系列
(4) $P_1 \parallel P_2$：2つの証明系列のマージ
(5) $S_1 \bowtie S_2$：節集合 S_1 と S_2 のクラスタ演算
(6) ∇T は記号列 T とマッチする節．T には $p(f(g_v(a), g_v(b)))$ のようにパタンの記述ができる [4]．ここで，f_v, x, y は変数である（ランクは固定されていると仮定）．
(7) $[C_1, .., C_n]$ は節 C_i のリスト
(8) $[C\ in\ L]$ は証明系列 L の節 C のリスト
(9) $\{A_1, .., A_n\}$ は節 A_i の集合
(10) $\{A\ in\ L\}$ は証明系列 L の節 A の集合
(11) $foreach\ x\ in$ 証明系列 $L\ do\ E$：L の各要素を x に対して，E を評価する
(12) $$$ は処理中の証明系列を表す
(13) 新たなデータ構造は ADT によって利用者が記述できる

例えば，線形導出は次のように記述される．

$$foreach\ x\ in\ \$\$\ do\ \uparrow \$\$ \bowtie x$$

入力導出は次のように記述される．

$$\uparrow \$\$ \bowtie \downarrow \$\$$$

述語記号 p を持つ節集合間のクラスタ演算は次のように記述される．

$$\{\nabla p(x, y)\ in\ \$\$\} \parallel \{\nabla \neg p(x, y)\ in\ \$\$\}$$

述語記号 p の現れる節集合と最近導出された節集合との間のクラスタ演算は次のように記述される．

$$foreach\ x\ in\ \{p(x)\ in\ \$\$\}\ do\ \uparrow \$\$ \bowtie x$$

$\neg p(f_v(a), b)$ とマッチする節集合と $p(x, y)$ のマッチする節集合の間のクラスタ演算は次のように表される．

$$foreach\ x\ in\ [\neg p(f_v(a), b)\ in \$\$]\ do\ foreach\ y\ in\ [p(x, y)\ in\ \$\$]\ do\ x \bowtie y$$

3 LCCの計算モデル

3.1 節の表現

節 $p_1, .., p_m \leftarrow q_1, .., q_n$ $(m, n \geq 0)$ を考える．ここで，p_i, q_j はアトムである．節は各々環境を持つ（環境は整数を指標として表される）．節の間で導出を試みる毎に新しい環境が作成される．ここでは節を環境とアトムから次の(1),(2)によって再帰的に表す．以下において，式は項またはリテラルのいずれかを表すとする．

(1) s は導出のステップ，e, v が環境，p が式ならば，$\ll s, e, v, p \gg$ は節式である．また，式を全く含まない形式 $\ll s, e, v, \circ \gg$ を特に空節式と呼ぶ．
(2) s が導出のステップ，$e_i,\ e'_j,\ v_i,\ v'_j$ がそれぞれ環境，$p_1, \ldots\ p_m$，$q_1, .., q_n$ がそれぞれ節式または式ならば，$\ll s, e_1, e'_1, p_1 \gg, \ldots \ll s, e_m, e'_m, p_m \gg \leftarrow \ll s, v_1, v'_1, q_1 \gg, \ldots \ll s, v_n, v'_n, q_n \gg$ は節式である．

以下に於て，$\ll s, e, v, (p_1, \ldots p_m \leftarrow q_1, \ldots q_n) \gg$ は $\ll s, e, v, p_1 \gg, \ldots \ll s, e, v, p_m \gg \leftarrow \ll s, e, v, q_1 \gg, \ldots \ll s, e, v, q_n \gg$ に書き換えられるものとする．

節式 $\ll s, e_1, v_1, q_1 \gg, \ldots \ll s, e_m, v_m, q_m \gg$ は各式 q_i の環境が e_i と v_i であり，導出のステップが s であることを表す．証明する節 $p_1, .., p_m \leftarrow q_1, .., q_n$ は節の形式 $\ll 1, e, e, (p_1, \ldots p_m \leftarrow q_1 \ldots q_n) \gg$ で表す．また，節式 $\ll s'', e'', v'', \ll s_1, e_1, v_1, p_1 \gg, \ldots \ll s_m, e_m, v_m, p_m \gg \leftarrow \ll s, e'_1, v'_1, q_1 \gg, \ldots \ll e'_n, v'_n, q_n \gg \gg$ は $\ll s'', e'', v_1, p_1 \gg, \ldots \ll s'', e'', v_m, p_m \gg \leftarrow \ll s'', e'', v'_1, q_1 \gg, \ldots \ll s'', e'', v'_n, q_n \gg$ 書き換えられるものとする．

3.2 導出

節式

$$C_1 = \ll s_1, e_1, v_1, p_1 \gg, \ldots \ll s_1, e_m, v_m, p_m \gg \\ \leftarrow \ll s_1, e'_1, v'_1, q_1 \gg, \ldots \ll s_1, e'_n, v'_n, q_n \gg$$

の p_i と節式

$$C_2 = \ll s_2, f_1, w_1, p'_0 \gg, \ldots \ll s_2, f_{m'}, w_{m'}, p'_{m'} \gg \\ \leftarrow \ll s_2, f'_1, w'_1, q'_1 \gg, \ldots \ll s_2, f'_{n'}, w'_{n'}, q'_{n'} \gg$$

の q'_j が最汎統合可能ならば導出によって得られる新たな節式 $C_1 \otimes C_2$ は次のように定義する (factoring がないとき).

$$\ll s', e', e', (\ll e_1, v_1, p_1 \gg, \ldots \ll e_{j-1}, v_{j-1}, p_{i-1} \gg, \\ \ll e_{j+1}, v_{j+1}, p_{i+1} \gg, \ldots \ll e_m, v_m, p_m \gg, \\ \ll s_2, f_1, w_1, p'_0 \gg, \ldots \ll s_2, f_{m'}, w_{m'}, p'_{m'} \gg \\ \ll s_1, e_1, v_1, p_1 \gg, \ldots \ll s_1, e_m, v_m, p_n \gg \\ \leftarrow \ll s_1, e_1, v_1, p_1 \gg, \ldots \ll s_1, e_m, v_m, p_m \gg, \\ \ll s_2, f'_1, w'_1, q'_1 \gg, \ldots \ll s_2, f'_{j-1}, w'_{j-1}, q'_{j-1} \gg \\ \ll s_2, f'_{j+1}, w'_{j+1}, q'_{j+1} \gg, \ldots \ll s_2, f'_{n'}, w'_{n'}, q'_{n'} \gg) \gg$$

ここで, s' は $max(s_1, s_2) + 1$, e' は新たな環境である.

3.3 証明系列の計算

証明系列の計算は導出のステップ毎に節集合に対するクラスタ演算を行う必要がある. クラスタ演算は各節の間との 2 項導出である. 各節がそれぞれの環境を保持しているために, 節の導出を同期を取ってステップ毎に行う必要がない.

並列に計算可能な節を表現するために, 並列実行可能節と呼ばれるものを定義する. 並列実行可能節は節式 $G_1, \ldots G_n$ の集合 $\{G_1, \ldots G_n\}$, 代入 θ(束縛の有限集合, 束縛の表現形式は次に述べられる), 束縛参照表 $\ll rt, rp \gg$ の 3 組 $(\{G_1, \ldots G_n\}, \theta, \ll rt, rp \gg)$ からなる. rt は 2 次元の半無限長の表, rp は rt の位置を示す番号であり, $rt(i, 1) =$ "$*$" $(i = 1, 2)$ とする. 束縛参照表は各節式からある束縛が参照可能であるか否かを判定するために用いられる).

並列実行可能節は複数の節式を含み, これに含まれる各節式は並列に導出行うことができる. 新しい環境 rp を生成 $(rp \leftarrow rp + 1)$ し, 束縛参照表 rt の $rt(j, rp)$ に e_i を格納 $(rt(j, rp) \leftarrow e_i)$ し, 並列に節の導出が行なわれる.

3.4 束縛の表現

束縛は 5 組 $\ll x, i, k, t, j \gg$ で表す [5]. x は変数, t は項, i, j, k は環境である. この束縛は, ある節式を環境 k の導出で適用し, 環境 i をもつ導出時に適用した節式中の現われた変数 x に環境 j をもつ導出時に適用した節式中に現われた項 t が束縛されたことを表す. ただし, 与えられた節に現われた変数または項は環境 1 をもつものとする.

並列実行可能節 $(\{G_1, \ldots G_n\}, \theta, \ll rt, rp \gg)$ の各節式 G_i から代入 θ の各束縛が参照可能であるか否かは束縛参照表を用いて決定される. $G_n = \ll w, e'_1, v'_1, g_1 \gg, \ldots \ll w, e'_m, v'_m, g_m \gg$ とするならば, e'_s から束縛参照表でたどることのできる環境 k を持つ束縛 $\ll x, i, k, t, j \gg$ は $\ll e'_s, v'_s, g_s \gg$ より参照可能である. すなわち, 束縛 $\ll x, i, k, t, j \gg$ の k が e'_s, $e = rt(i, e'), \ldots (i = 1, 2)$ のように e'_s からたどることのできる環境をもつ束縛はすべて参照可能である.

$G_h = \ll s, e, v, (p_1, \ldots p_m \leftarrow q_1, \ldots q_n) \gg$ を節式とするとき, G_h から θ の束縛 $\ll x, i, k, t, j \gg$ が参照可能であるか否かは次のアルゴリズムによって決定される.

(1) $i = 1, 2$ について以下を繰り返す.

(2) $e' \leftarrow e$

(3) $e' = k$ ならば参照可能である.

(4) $rt(i, e') =$ "$*$" ならば, 参照不可能である.

(5) $e' \leftarrow rt(i, e')$ として, (3) へ.

3.5 最汎統合

$(\{G_1, \ldots G_m\}, \theta, \ll rt, rp \gg)$ を並列実行可能とするとき, 節式 $\ll s, e, v, t \gg$ の値 (例 (instance) に対応する) は環境 v' と式 t' の 2 組 $\ll v', t' \gg (= val \ll e, v, t \gg)$ からなり, 次の (1)-(4) から定義される.

(1) t が定数, 関数項またはアトムならば $\ll v', t' \gg \leftarrow \ll v, t \gg$ とする.

(2) t が変数, 環境 e から参照可能な θ の束縛 $\ll t, v, k, u, j \gg$ が存在するならば, $\ll v, t \gg \leftarrow \ll j, u \gg$ として, (1) へ. さもなければ, $\ll v', t' \gg \leftarrow \ll v, t \gg$ とする.

(3) t が目標 $\ll e'', v'', (g''_1, \ldots g''_m) \gg$ ならば $\ll v', t' \gg \leftarrow \ll e, (g''_1, \ldots g''_m) \gg$ とする.

(4) 上記以外ならば $\ll v', t' \gg \leftarrow \ll v, t \gg$ とする.

節式 $\ll s_1, e_j, v_j, g_j \gg$ と節式 $\ll s_1, e', v', p_i \gg$ との (出現検査を略されている) 最汎統合は次の (1)-(8) によって計算され, 成功するならば代入 ξ によって最汎統合可能であると言われる. e' は新たな環境とする.

(1) $\xi \leftarrow \{\}$

(2) $\ll v'', t'' \gg \leftarrow val \ll e', e', p_i \gg$

(3) $\ll v', t' \gg \leftarrow val \ll e_j, v_j, g_j \gg$

(4) t'' が変数ならば $\xi \leftarrow \xi \cup \{\ll t'', v'', e', t', v' \gg\}$ とし, 成功して終了する.

(5) t' が変数ならば $\xi \leftarrow \xi \cup \{\ll t', v', e', t'', v'' \gg\}$ とし, 成功して終了する.

(6) t'', t' が定数であり, $t'' = t'$ ならば成功して終了する. さもなければ, 失敗して終了する.

(7) $t'' = f(s_1, \ldots s_r)$, $t' = f(t_1, \ldots t_r)$ ならば, 各 i に対して $\ll v'', t'' \gg \leftarrow val \ll e', v'', s_i \gg, \ll v', t' \gg \leftarrow val \ll e', v', t_i \gg$ として, (4)-(8) を再起的に呼び出してすべての i について成功するならば成功して終了する. さもなければ失敗して終了する.

(8) 上記以外ならば失敗して終了する.

最汎統合は, Hsiang の方法によって項書き換え規則で表現できることが知られている [3].

4 むすび

この論文では代数操作の演算が導出に対応し, その代数演算は書き換えに対応し, 各導出は並列の動作が可能であることが示された.

参考文献

[1] Boyer, R.S. and Moore, J. : The sharing of structure in theorem-proving programs. DAI Research Report 47, Department of Artificial Intelligence, University of Edinburgh (1972).

[2] Conery: Parallel Execution of Logic Programs, Kluwer Academic Press (1987).

[3] Hsiang, J. : Refutational Theorem Proving using Term-Rewriting Systems, Artificial Intelligence, 25, pp. 255-300 (1985).

[4] Shioya, I. : A Logic Database Language with Tree Patterns, E-J Conf. (1993).

[5] Nakamura, K. : Associative Concurrent Evaluation of Logic Programs, Proc. of the Second International Logic Programming Conference, Uppsala, pp. 321-331 (1984).

[6] Perlis, D.: Languages with Self-Reference 1, Artificial Intelligence 25(1-2), 301-322, 1987

[7] Maes, P.: Issues in Computational Reflections, Meta-Level Architectures and Reflection(Maes et al.ed.),1988, Elsevier

[8] Pettorossi, A. ed.: Meta-Programming in Logic, LNCS 649, 1992, Springer Verlag

[9] Kowalski,R. : Algorithm = Logic + Control, CACM -8. 1979

[10] Genesereth, M.R., Nilson, N.J.: Logical Foundation of Artificial Intelligence, Morgan Kaufmann, 1987

[11] Bowen, K.A., Kowalski, R.: Amalgamating Language and MetaLanguage, Logic Programming (Tarnlund ed.), Academic Press, 1982

[12] Genesereth, M.R. : Introspective Fidelity, Meta-Level Architectures and Reflection(Maes et al.ed.),1988, Elsevier

[13] Miura, T.: A Logical Framework for Deductive Objects, Information Systems 17-5, 1992