ソフトウエア開発における協調支援環境Vela
—— (5)ソフトウエア・プロセスの実行機構 ——

2G−5

成松　克己　山口 高平　落水 浩一郎

静岡大学

1. はじめに

実際の開発活動を分析、形式化[1]して得られた知識(ソフトウエアプロセスのルール表現)には、以下のような問題がある。(1)ルールの条件部の値が正確に特定できず、複数のルールが競合する。(2)ルールのアクション部で変数にユニファイされる値(中間生成物)が必ずしも正しいものとは限らない。本稿では、(1)知的バックトラック機構により、過去のルール選択のやり直しをサポートし、非決定性に対処する。(2)Truth Maintenance Systemに基づく仮説推論により、作られる「もの」の質や理解の程度などの不確実なことがらに対して仮説を立て、いきづまった時にこの原因として仮説を変更することによって対処する。という二つの機構をプロセスの実行系に組み込む方法について述べる。

2. 知識利用・仮説管理フェーズのアーキテクチャ

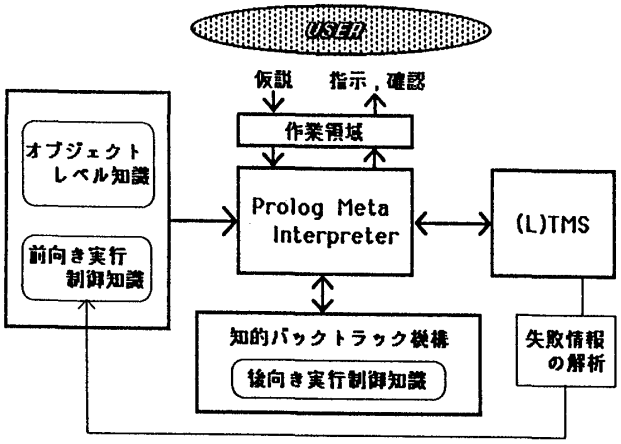

図1　本フェーズのアーキテクチャ

図1に本フェーズのアーキテクチャを示す。オブジェクトレベル知識と前向き実行制御知識、そして後向き実行制御知識は前のフェーズから得られる知識であり、Prologルールで表現されている。メタインタプリタはルール選択時に前向き実行制御知識を利用し推論を進める。その際、ユーザが与えた仮説に対してassumptionノードを割当て、その真理値をTMSが管理する。そして不都合な状況が生じたときには、そのときのコンテクストをnogoodとして記録する。不都合の原因にはルールの選択法とユーザが与えた仮説が考えられるが、ユーザに初心者を想定しているため仮説の方が信頼度が低い。そこでまずnogoodからの依存関係に基づいた逆トレースにより、仮説の中から原因(culprit)を探し出し、その仮説を変更することによって不都合な状況を解消する。それでもだめな時は知的バックトラック機構により、後向き実行制御知識を利用して過去のルール選択のうち間違ったと思われるところまでバックトラックし、ルール選択をやり直す。また、不都合の起きたコンテクスト(nogood)を解析することにより前向き実行制御知識を精錬することも将来的には考えている。

3. 知的バックトラック機構の実装

知的バックトラック機構は、過去のいくつかのルール選択の中から戻るべき地点を決定する機構と、それにより指定された地点へバックトラックする機構によって実現される。前者の機構は、後向き実行制御知識を利用することにより実現されるが、まだ実装されていない。後者については、Prologメタインタプリタにおいて現地点から指定された後戻り地点までの部分実行履歴を破棄することが必要である。ここでは、指定された範囲において代替ルールの実行を阻止することによって逐次的にスタックポインタを戻す方法を採用している。図2にこのメタインタプリタの基本構造を示す。

```
(1) execute(Goal):-
(2)       select(Goal,Rule_id,Body),
(3)       (true;condition(Rule_id),!,fail),
(4)       execute(Body).
(5) condition(Rule_id):-
(6)       bp(X),!,X \ ==Rule_id.
(7) condition(Rule_id):-!,
(8)       ib(X),assert(bp(X)),X \ ==Rule_id.
```

図2　メタインタプリタの基本構造

(2)でルールを選択してGoalとルールのヘッドをユニファイし、ボディ部(Body)を取り出す。そして(3)でまずtrueを実行して(4)へ行き、失敗して処理系が(3)にバックトラックしたとする。すると(3)のcondition/1を実行して(6)か(8)に制御が移る。戻り先地点の情報

Vela:A Software Development Environment for Cooperation Support

Katsumi NARIMATSU, Takahira YAMAGUCHI and Koichiro OCHIMIZU

Shizuoka University

である bp(X) がデータベースに書かれていれば (6) で戻り先のルールと現在の Rule_id と比較し、異なっていれば (3) の condition/1 が成功し、!,fail するので (2) の select/3 による代替ルールの選択実行を阻止し、execute/1 を失敗させる。execute/1 は再帰的に何回も呼び出されているので、戻り先と現在の Rule_id が異なっている間はメタインタプリタのバックトラックが行なわれる。戻り先と現在のルールが同じならば (3) の condition/1 が失敗し、処理系は (2) の select/3 で代替ルールを選択実行することによって、指定された地点まで後戻りすることになる。ただし、戻り先 bp(X) は、はじめて condition/1 が実行されたときに (8) の ib/1 で後向き実行制御知識を利用して戻り先を決定し、データベースに assert される。また (8) の X \ ==Rule は、戻り先を決定した結果が現在失敗したルールだった場合に (2) へ戻るための処理であり、年代順後戻りに対処するものである。こうして、知的バックトラック機構が実現される。

4. Logic-based Truth Maintenance System の実装

ＴＭＳのうち、一番良く知られているのはＪＴＭＳ（Justification-based TMS）であるが、これはノードの否定が導出できないなど論理的記述力に欠ける。そこで本アーキテクチャではＬＴＭＳ（Logic-based TMS）[2] を採用する。

4.1. LTMS の概要

LTMS では各ノードは True、False、Unknown のいずれかの値をもち、ノードの否定を積極的に信じることができる。そして LTMS は論理式で与えられる各ノード間の依存関係（制約）を保持し、この制約によってノードの真理値を管理するので Justification 以外の制約をも加えることができるという点で記述力が高くなっている。LTMS の基本的な仕事は、制約の集合と仮定（enable assumption）の集合から導出されるノードに対して真理値を割り当てる（ラベルを振る）ことである。例えば、制約A→Bと仮定A：true が与えられたとき、Bのラベルを true にする。本来の LTMS は、任意の論理式で表された制約を受け取り、それを内部で連言標準形に変換して選言の形をした節の集合として扱う。例えば、前述のA→Bは、$\neg A \vee B$ となる。LTMS はこれらの節をすべて満たすように BCP(Boolean Constraint Propagation) アルゴリズムを用いてラベル付けを行う。

4.2. BCP アルゴリズム

BCP は単調な動作をする。unknown になっているノードを true または false に変える操作をするが、一旦 true または false になったノードは、仮定集合に変更がない限りそのラベルを変更されない。BCP は調べるべき節の行列を保持し、この中の節を一つずつ処理してラベリングを広げるが、そのときのふるまいは次のようになる。a. 現在のラベリングが節を満たしていれば、この節を行列から外す。b. 現在のラベリングが節で与えられる制約を破っているときは、推論エンジンに通知する。c. 節中に現れるノードのうち一つだけが unknown であり、さらにそのほかのノードによるリテラルがすべて偽の値をとるときはそのノードのラベルを書き替え、その節を行列から外す。d. そのほかの場合には何もしない。そして、すべての節が満たされるか、行列中の節を一通り評価し終わった時点で新しいラベルがつけられていなくなるまで、行列中の節を繰り返し評価する。

4.3. インプリメンテーション

BCP は健全であるが、完全ではない。間違ったラベルをつけることはないが、仮定と制約からラベル付けが可能なノードに対して unknown を返す事がある。これは、一度に一つの節しか処理できないためである。LTMS の完全性を高めるために BCP を拡張する技法があるが、BCP だけでもある程度の LTMS の機能が実現できる。そこで、BCP アルゴリズムと簡単な周辺機能を Prolog でインプリメントした。およそ２３０行、節の数が６０程の簡単なものである。ノードと節の書式は次の通りである。

tms_node（ノード名，ラベル，備考）
tms_clause（clause 番号，[true-nodes], [false-nodes], bcp-label)

備考は、ラベル付けが enable-assumption によるものか節によるものかを区別するためのものであり、節によってラベルづけされていた場合にはその節と、前件となるノードの集合が記録される。bcp-label は、BCP アルゴリズム中で節の状態を保持し、処理すべき節を識別するのに使う。

5. おわりに

LTMS は単体では機能せず、対象世界における推論エンジンとの組み合せによって利用される。仮説推論を行う際のノードの設計を具体的におこない、実際にシステムとしてどのような働きができるのかシミュレートして、評価することが必要である。この点については、JSP を例題に、ユーザが生成するオブジェクトを仮定としてとらえる方向でノードを設計し、評価する研究を進めている。[3]

謝辞　本研究はＳＤＡコンソーシアムの補助金と、科研費重点領域研究 (課題番号 02249109) の一部の援助のもとにおこなわれた。記して謝意を表する。

参考文献

[1] 中尾, 山口, 落水: ソフトウエアプロセスの分析と形式化, 第 41 回情報処理学会全国大会予稿集 (1990).

[2] J.de.Kleer, 他: *Truth Maintenance Systems*, tutorial program of 11th IJCAI(1989).

[3] 太田, 山口, 落水: ソフトウエアプロセスの動的特性に対応するための一機構, ソフトウエアシンポジウム '90(1990).