仕様獲得支援システムの開発
-仕様デバッガの試作-

5S-4

°西村 一彦　中村 英夫
(株)東芝　システム・ソフトウェア技術研究所
(E-mail nishi@ssel.toshiba.junet, nakamura@ssel.toshiba.junet)

1. はじめに

ワークステーションを用いたソフトウェア開発の要求定義支援技術は,要求分析というよりもむしろ仕様獲得というより積極的な意味での支援技術になりつつある.本稿では,仕様獲得支援システムの一部として仕様の不具合(不完全,矛盾,冗長)を効率良く発見する仕様デバッガの不具合診断部について報告する.本デバッガの特長は,従来のPrologデバッガの技法に計画生成技法を組合せ,ユーザの意図を反映したデバッグを行える点にある.

2. 仕様獲得支援システム

ソフトウェア開発の上流工程支援のためワークステーションを用いたCASEツールが実用化されつつある.しかし,CASEツールの対象範囲はかなり広く現状の支援環境では課題も多い.ここでは,要求定義フェーズとシステム設計フェーズのインタフェース部分について考える.要求定義側は必ずしも充分性,妥当性を考えて要求定義書を作成するとは限らない.一方,これを受ける仕様作成側(提案書,システム設計仕様書などの)は,完全性,実現可能性などを確認した仕様書を作成せねばならない.我々は,このためのインタフェース技術が仕様獲得技術であると考えている(図1).この様なインタフェースはソフトウェア開発工程の各工程にあり,仕様獲得技術は要素技術的なものであるとも言える.

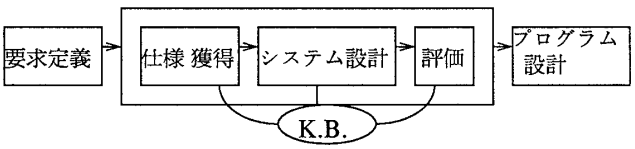


図1 仕様獲得技術とソフトウェア開発モデル

また,図2に示すように仕様獲得技術は仕様の統合・精錬を行う技術である.

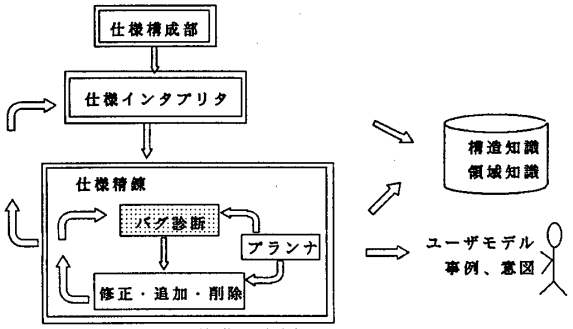


図2 仕様獲得支援システム

従って,仕様獲得支援システムにはの次の三つの機能が必要である.

(1)あいまい,断片的な要求仕様の統合仕様構成
(2)仕様の解釈,実行
(3)統合仕様の実現性,完全性検証・獲得(検証,修正など)

今回は(3)について報告する.

3. バグ診断部

3.1 仕様の不具合

仕様の不具合(不完全,矛盾,冗長)は仕様のバグとして考える.これらを論理プログラムの二種類のバグ(誤りと失敗)[4]に対応させた場合,誤りは矛盾仕様,冗長な仕様にあたり,誤りは不足な仕様,矛盾仕様に該当する.

3.2 Prologのデバッグ環境

Prologのデバッグ環境として論理型言語の特性を生かした知的デバッガが研究されている[3][4].これはバグを発見するためのアルゴリズムを持っており,ユーザはPrologの複雑な実行制御を意識する必要がない.ただし途中の計算が正しいかどうかを判定するアルゴリズムはないのでオラクルというユーザの意図(正しい答え)を入力する.

しかし,これまでのオラクルはプログラムが大規模になった場合,ユーザは簡単に提供できないという問題を抱えている.また,計算木のどのノードを選ぶかを直接ユーザが示すことができない[3].その場合,計算の前後や過程を示したり,実行ことを推論する機能が必要である.これは後にユーザの実現手段に関する意図を獲得する機能に反映する.

こうしたデバッグの順序に関するユーザの意図は様々なものがある.[1]ではデバッグに関する経験的知識(デバッグ履歴)を活用するデバッガを提案している.これはプログラムの実行過程を規則化し,似たような状況に対して再利用することにより効率的にバグを発見するという方法である.

3.3 バグ診断部

3.3.1 構成

バグ診断部は,デバッグの順序に関するユーザの意図を反映させるためのデバッグ手順解析部を持つ.

このデバッグ手順の推論は計画生成問題としてとらえることができる.

デバッグ手順解析部(プランナ)はトップゴール(初期状態)とユーザの意図に反する箇所(目標状態)を入力として,初期状態から目標状態へ至る最適な実行系列,すなわち,デバッグの追跡手順(プラン)を立案する.プランナはSTRIPS[2]の方法を採用した.なお,ユーザの要求はバグの存在範囲を絞り込むためにPrologのリテラル単位ではなく項を単位とする.

Development of the Software Speification Acquisition Support System
Kazuhiko NISHIMURA,Hideo NAKAMURA
Systems and Software Engineering Laboratory,Toshiba Corporation

3.3.2 プランの生成

まず,仕様の実行結果からユーザの要求を得る.プランナは入力から要求へ至るプランを推論する.プランは計算木のノード(ゴール)の列で示される.プラン生成の原則として入力,要求部分に対して依存度が高い順(指示された項を含んでいるものを優先)に行なう.また,ユーザはある要求に対するプランを持つならば,目標:プランというように表現する.

入力条件(inp)から行動(goal)を探すプランは

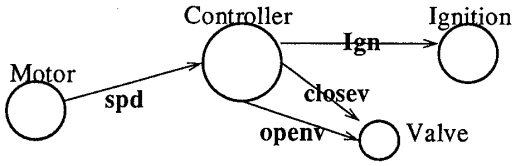
find(inp,_,goal) : fc(inp,_,goal)

バグ診断部はプランに従って仕様を解析し,ユーザはシステムから発せられる仕様の途中結果についてyes/noで答える.

もし,その手順にそってバグが見つからないときは別なプランが作成される.ユーザの答えはデータベースに蓄えられ,重複する質問は避けられる.

3.4 例

実行例の一部を示す.仕様を獲得する対象はヒーティングシステムのコントローラの仕様である[5].仕様の中に「コントローラはモータの速度を監視し,十分なスピードになったら(spd),点火装置とバルブに作動命令(ign,openv)を出す.点火装置は命令を受けると内部状態をoffからonへ変化し,バルブはopenの状態となる.」という制約がある.この制約についてデータフロー解析を行い,以下のProlog言語による仕様を得たとする.




```
controller(F,S,E) :- % コントローラは事実(F)と現在の状態(S)
  find(F,S,[],A), exec(A,S,E). % から次状態(E)を決定
find([],_,X,X).      % 事実から行動を見つける
find([X|Y],S,Z,A) :- fc(X,S,Z,Z1),find(Y,S,Z1,A).
exe([],S,S).         % 行動系列の実行
exe([A1|A2],S,E) :- action(A1,S,S1),exe(A2,S1,E).
fc(spd,S,A1,A2) :-  % スピードが十分な場合の行動
  mem(spd(ok),S),app([ign,openv],A1,A2).
fc(stp,S,A1,A2) :-  % 停止状態の時の行動
  mem(stp(ok),S),app([closev],A1,A2).
action(ign,S,S1) :-   % 点火装置を作動する
  mem(ign(off),S),del(ign(off),S,RS),app([ign(on)],RS,S1).
action(openv,S,S1) :- % バルブを開ける
  mem(v(cl),S),del(v(cl),S,RS),app([v(op)],RS,S1).
app([],X,X).      % 補助述語
app([X|L1],L2,[X|L3]) :- app(L4,L2,L3).
                  % 正しくはapp(L1,L2,l3)
mem(X,[X|_]).
mem(X,[_|Y]) :- mem(X,Y).
del(X,[X|Y],Y).
del(X,[Y1|Y],[Y1|Z]) :- del(X,Y,Z).
```

図3 コントローラのデータフローと機能仕様

この誤り仕様に対して

```
?-controller([spd],[spd(ok),ign(off),v(cl)],X).
```

を実行すると *X = [ign(on),spd(ok),v(cl)]* となり,ユーザの要求 *X=[spd(ok), ign(on), v(op)]* を満たさない.デバッガはこの原因を探すために,ユーザの注目する部分を指摘してもらう.今,ユーザは入力(+spd)と要求(*X)に注目している.デバッグ手順解析部は*項を目標とし,+項からのプランを探索する.上の例ではまず(a)のプランが得られる.

バグ診断部はプランに沿ってユーザに質問を行なう.診断部はバグがapp/3にあることを示している.(図4:下線部 入力)

```
S:controller(+[spd],[spd,ign(off),v(cl)],*X).
   find - fc - exe      (a)プラン
S:find([spd],[spd,gn(off),v(cl)],[],[ign])is true? no. (b)質問
S:fc(spd,[spd,ign(off),v(cl)],[ign])is true? no.
S:exe([ign],[spd,ign(off),v(cl)],[ign(ok)])is true? yes.
S:バグはfcの中にあります.
S:fc(+spd,[spd,ign(off),v(cl)],*[ign])
   app - app        % *項を生成するプラン
S:app([ign,ovalve],[],[ign(ok)]) is true ? no.
S:app([],[],[]) is true? yes.
S:誤り:app([X1|X2],X3,[X1|X4]) :- app(X5,X3,X4).
```

図4 実行例

3.5 考察

例題をトップダウン診断で実行する.(図5)

```
Q:find([spd],[spd,ign(off),v(cl)],[],[ign])true? no.
Q:fc(spd,[spd(ok),ign(off),v(cl)],[],[ign])true? no.
>Q:mem(spd(ok),[spd(ok),ign(off),v(cl)])true? yes.
Q:app([ign,openv],[],[ign])true? no.
Q:app([],[],[])true? yes.
BUG : app([ign,openv],[],[ign]) :- app([],[],[]).
```

図5 トップダウン診断実行例[3]

トップダウン診断は実行順序を忠実に反映しているが,ユーザの要求に直接には関係ない質問(>部)を行なうことになる.これは問題が大規模になった場合,デバッグに対するユーザの視点をずらすことになる.その点,本システムはユーザの要求を反映したデバッグ順序であるためにそのような問題は起きない.また,プランはユーザの要求を得るための実行系列を反映しており,動作仕様を獲得する上での支援ともなる.

4.まとめ

仕様獲得支援システムにおいて,仕様の誤り部分を指摘するバグ診断部について述べた.今後はプランナに階層性を導入し,より大規模な問題への適用を試みていく方針である.

[参考文献]

[1]M.Eisenstadt,"Retrospective Zooming :A Knowledge Based Tracing and Debugging Methodology For Logic Programming",IJCAI-85,717-719,1985.

[2]R.E.Fikes and N.J.Nilsson,"STRIPS:A New Approach to the Application of Theorem Proving to Problem Solving",Artificial Intelligence 2,189-208,1971.

[3]J.W.Lloyd,"Declarative Error Diagnosis",New Generation Computing,133-154,1987.

[4]E.Shapiro,"Algorithmic Program Debugging",MIT Press,1983.

[5]Problem Sets for the 4th IWSSD,Proc. of 4th IWSSD, 1987.