プログラムのデバッキングについて

北上　始，古川康一
財団法人）新世代コンピュータ技術開発機構

１．はじめに

Ｐｒｏｌｏｇで記述されたプログラムに対する虫とりを、半自動化するために、backtrace という述語が提案されている［２］。

本報告では、そのbacktrace 述語に対する解説を、Quick Sortプログラムの虫とりを例にして行う。なお、本報告書は、文献［２］の部分的要約であり、さらに、著者らの解釈を付け加えたものである。

２．　Ｑｕｉｃｋ　Ｓｏｒｔ　プログラム

Quick Sortプログラム（図３．を参照）は、partition とappend述語から構成される再帰的なプログラムである。

Quick Sortプログラムの呼び出し形式は，qsort(Input-List,Output-List) である。第１引数に，ソートすべきアトム列をリストの形で入力すると，qsort プログラムは，第２引数にソート済みのアトム列をリストの形で出力する。図１は，Quick Sortプログラムの実行イメージ図である。

partition プログラムは，あるアトムＸを基準に，与えられた１本のリスト中のアトムを左右に振り分けて，２本のリストを作るプログラムである。partition の呼び出し形式をpartition(Input-List,X,Output1,Output2) とすると，この左右の振り分けはX 未満のアトムをOutput1 に格納し，X 以上のアトムをOutput2 に格納することで実施される。

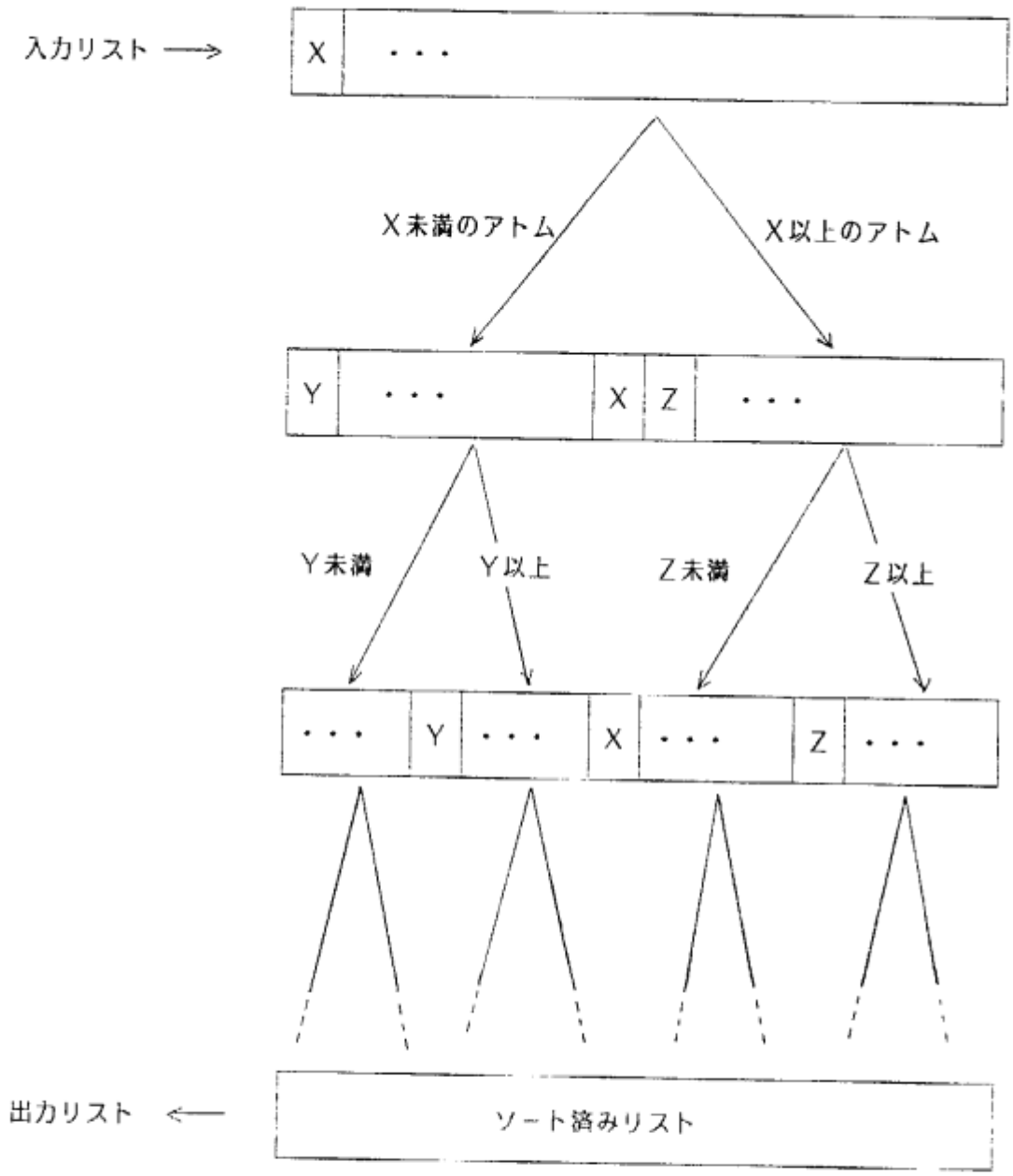

図１．　Ｑｕｉｃｋ　Ｓｏｒｔ
　　プログラムの実行イメージ

```
qsort([X|L],LO):-
      partition(L,X,L1,L2),
      qsort(L1,L3),qsort(L2,L4),
      append(L3,L4,LO).
qsort([],[]).

partition([Y|L],X,L1,[Y|L2]):-
      partition(L,X,L1,L2).
partition([Y|L],X,[ Y| L1],I2):-
      Y@=<X,
      partition(L,X,L1,L2).
partition([],X,[],[]).

append([ X| L1],I2,L3):-
      append(L1,L2,L3).
append([],L,L).
```

図２．　プログラム１

```
qsort([X|L],LO):-
      Partition(L,X,L1,L2),
      qsort(L1,L3),qsort(L2,L4),
      append(L3,[ X|  L4],LO).              % Revise2 & 4
qsort([],[]).

partition([Y|L],X,L1,[Y|L2]):-
      Y@>X,                                 % Add1
      partition(L,X,L1,L2).
partition([Y|L],X,[ Y|  L1],L2):-
      Y@=<X,
      partition(L,X,L1,L2).
partition([],X,[],[]).

append([ X|  L1],L2,[X|L3]):-               % Revise3
      append(L1,L2,L3).
append([],L,L).
```

図３．　プログラム５

３．　プログラムのデバッキング

プログラム１（図２）は，虫をもつ Quick Sort プログラムである。デバッキングを実施しながら最終的なプログラム５（図３）を得るまでのプロセスを、以下説明する。図３のＡｄｄ１，Ｒｅｖｉｓｅ２，Ｒｅｖｉｓｅ３，Ｒｅｖｉｓｅ４は，それぞれプログラム１，２，３，４に対する修正を意味する。具体的な内容は，表１を参照の事。この例では，ゴールの証明プロセスで導出されたファクトの正当性を，インタラクティブにユーザーに問い合せる事で，虫の発見（誤りをもつクローズの発見）を行っている。

この虫の発見はbacktrace という述語で実施される。しかし，虫のクローズが発見されたとき，それをどの様なクローズに直せば良いかという問題は，この例では，ユーザの判断にまかせられている。Shapiro の Algorithmic Program Debugging Systemでは、Inductive な Model Inferenceの一環としてこの問題をとらえている。 これによると、新しい仮説は，精密化オペレータ(Refinement Operator) を使って生成された仮説群の中から探すことにより発見される。

ところで、哲学用語では、バックトレースのように、理論を反ばくする試みを、“決定的な実験(Crucial Experiments) ”と呼んでいる。もし、この試みが成功しないとき、われわれは、実験によって理論の確からしさが増したと言う[１]。
しかし、Shapiro のAlgorithmic Program Debugging Systemでは、この理論の確からしさをControl する情報が含まれていない。現在のところ、ユーザーの手によってControl されているとみて良い。確からしさをControl するためには、システムのメタインタプリタ（ICOTでは、 demo 述語と呼んでいる）にfuzzy の概念を導入しなければならない[４]。
APPENDIX-1は、その実現例であり、APPENDIX-2は、その実行トレースである。この例では、確信度を、０から１００までの自然数で、表現されている。

```
backtrace((P,Q),CE):-!,
     backtrace(P,CE),backtrace(Q,CE).
backtrace(P,CE):-
     clause(P,Q),backtrace(Q,CE),resolve((P:-Q),CE).
backtrace(P,CE):-P.

resolve((P:-Q),CE):-
     var(CE),!,ask(P,V),(V=false,CE=(P:-Q);V=true).
resolve((P:-Q),CE).

ask(P,V):-
     fact(P,V),!.
ask(P,V):-
     repeat,write(P),put(63),nl,read(V),
     (V=true;V=false),assert(fact(P,V)),!.
```

図４．　バックトレースのPrologプログラム

The refinement operation, which takes a clause"P"(alternatively,generating hypothesis) as an input and generetes a clause "q"(alternatively,generated hypothesis) as an output, is one of the following four cases:

(1) q ::= a(X1,X2,X3, ……,Xn) if p ::= □,
where"a"is an inductive generalized predicate symbol with n arguments.

(2) q ::= unify Xi and Xj for
P(X1,X2,X3,……Xn),where "Xi" and "Xj" are distinct variables.

(3) q ::= instantiate Xi to t(Y1,Y2,……,Ym) for P(X1,X2,X3,……, Xn),where
"t" is a functor with m arguments.

(4) q ::= (Head:-Goals,Gn) and
P ::= (Head:-Goals),where "Gn" is a user defined predicate, or recursively difined predicate whose symbol name is the same as that for the given "Head's" symbol name.

図5. Refinement Operator [5]

表　１．　Quick Sortプログラムの
変更プロセス

プログラム名	変　更　事　項
プログラム　1	第１番目のpartition プログラムのゴール節に，Y@>Xを追加する。
プログラム　2	第１番目のqsort プログラムでappendの使い方を， append([X¦ L3],L4,L0)に修正する。
プログラム　3	第１番目のappendプログラムのヘッドの形式を次のように変更する。 append([X¦ L1],L2,[X¦ L3]):-同一ゴール節
プログラム　4	第１番目のqsort プログラムでappendの使い方を， append(L3,[X¦ L4],L0)に修正する。
プログラム　5	な　し

４．　プログラム１に対するデバッキング

qsort のプログラムを動かしてみたところ、出力に正常な値が出力されなかったので、backtrace により、虫さがしを行う。partition の第１番目のプログラムに異常がある事を、backtrace が発見（ CE にそのクローズが示されている）したので、そのクローズを修正しなければならない。そのクローズは、あるアトムＸの値未満のアトムを振り分ける手続きであるはずなのに、それを振り分けるための判定処理がないことに、気がつくだろう。これにより、ユーザは、その partition プログラムのゴール節に、“Ｙ@＞Ｘ”を付け加えることになる。　……Ａｄｄ１の適用。

```
| ?- qsort([2,1,2],X).
X = []      --- 誤り

yes
| ?- backtrace(qsort([2,1,2],X),CF).
partition([2],2,[],[2])?
| : true.
partition([2],2,[],[2])?
| : false.

X = [],
CE = (partition([2],2,[],[2]):-partition([],2,[],[]))

yes
```

Ａｄｄ１　を適用

図６．　プログラム１に対するデバッキング

５．　プログラム２に対するデバッキング

プログラム２を、プログラム１　＋　Ａｄｄ１と定義する。このプログラムも又、誤った結果を出すので虫とりを実施しなければならない。qsort([2,1,2],X)をゴールとして、証明プロセスを追っていくと、qsort([2],[]) が真であるというまちがった動作をしていることが発見される。これにより、 qsortプログラムの第１番目のクローズに誤りがある事がわかる（ CE に示されている）。この表示のつじつまがあうように、その中に組み込まれている append 述語の使い方を、append([X｜L3],L4,L0)に変えてみる。

……Ｒｅｖｉｓｅ２の適用

```
| ?- qsort([2,1,2],X).
X = []     --- 誤り

yes
| ?- backtrace(qsort([2,1,2]X),CE).
partition([2],2,[2],[])?
| : true.
partition([1,2],2,[1,2],[])?
| : true.
partition([],1,[],[])?
| : true.
partition([2],1,[],[2])?
| : true.
qsort([],[])?
| : true.
append([],[],[])?
| : true.
qsort([2],[])?
| : false.

X = [],
CE = (qsort([2],[]):-
      partition([],2,[],[]),
      qsort([],[]),qsort([],[]),
      append([],[],[]))

yes
```

Ｒｅｖｉｓｅ２　を適用

図７．　プログラム２に対するデバッキング

6．　プログラム３に対するデッバッキング

　プログラム３を、プログラム２　＋　Ｒｅｖｉｓｅ２と定義する。
ここでは、appendの定義にあやまりがあることが示されている。appendのプログラムで第１番目のクローズに虫がある。このクローズの第３引数に、値を渡す変数Ｘがない事が、わかる。
　…… Ｒｅｖｉｓｅ３の適用。

7．　プログラム４に対するデバッキング

　プログラム４を、プログラム３　＋　Ｒｅｖｉｓｅ３と定義する。
ここでは、qsort の定義にあやまりがあることが、示されている。partition で分割の基準に使われている変数Ｘをアペンドする処理に誤りがあるようである。
これは、プログラム２において修正したappendの使い方に原因があるので、それを正しく修正しなければならない。　…… Ｒｅｖｉｓｅ４を適用

```
| ?- qsort([2,1,2,],X).
X = []        --- 誤り

yes
| ?- backtrace(qsort([2,1,2],X),CE).
append([2],[],[])?
| : false.

X = [],
CE = (append([2],[],[]):-append([],[],[]))

yes
```

Ｒｅｖｉｓｅ３　を適用

図８．　プログラム３に対するデバッキング

```
| ?- qsort([2,1,2]X).

X = [2,1,2]  ---  誤り

yes
| ?- backtrace(qsort([2,1,2],X),CE).
append([2],[],[2])?
| : true.
qsort([2],[2])?
| : true.
append([],[2],[2],)?
| : true.
append([1],[2],[1,2])?
| : true.
qsort([1,2],[1,2])?
| : true.
append([1,2],[],[1,2])?
| : true.
append([2,1,2],[],[2,1,2])?
| : true.
qsort([2,1,2],[2,1,2])?
| : false.

X = [2,1,2],
CE = (qsort([2,1,2],[2,1,2]):-
      partition([1,2],2,[1,2],[]),
      qsort([1,2],[1,2]),qsort([],[]),
      append([2,1,2],[],[2,1,2]))

yes
```

Ｒｅｖｉｓｅ４　を適用

図９．　プログラム４に対するデバッキング

８．　プログラム５の検証

プログラム５を、プログラム４　＋　Ｒｅｖｉｓｅ４と定義する。
このプログラムを使用すると、今までの入力リスト［２，１，２］ばかりでなく、複雑な入力リスト［５，２，４５，……］に対しても正常なリスト（ソート済み）を出力するので、一応ユーザーは安心し、虫とりを止める。

```
| ?- qsort([2,1,2],X).
X = [1,2,2]  ---  正しい！

yes
| ?- qsort([5,2,45,3,24,5,43,2,3,1],X).
X = [1,2,2,3,3,5,5,24,43,45]  ---  正しい！

yes
```

図１０．　プログラム５の検証

９．　おわりに

本報告では、文献［２］を中心に、Quick Sortプログラムを例にして、backtrace 述語のふるまいを考察した。この述語は、哲学用語で知られている“決定的な実験（Crucial Experiments）”の一部分を、実現したものであり、Fuzzy 推論などを含む、さらに進んだ述語として拡張可能である。

[REFERENCES]

(1) K. R. Popper: Conjectures and Refutations: The Growth of Scientific knowledge, Harper Torch Books, New York, 1968.

(2) E.Y. Shapiro: Inductive Inference of Theories from Facts, Yale Univ. Research Report 192, Feb. 1981

(3) E.Y. Shapiro: Algorithmic Program Debugging, An ACM Distinguished Dissertation 1982, MIT Press.

(4) E.Y. Shapiro: Logic Programs with Uncertainties: A Tool for Implementing Rule-based Systems, IJCAI '83.

(5) H.Kitakami, S.Kunifuji, T.Miyachi and K.Furukawa: A Methodology for Implementation of A Knowledge Acquisition System, ICOT Technical Report TR-037 (1983).

[APPENDIX-1] Fuzzy type meta-interpreter.

```
%% A specialized interpreter that accepts
  a certainty threshold(T).

%% Author : H.Kitakami··· '83.9/27.

* solve(KBN,A,T,C).

 • KBN   : Knowledge Base Name.   [input]
 • A     : Goals.                 [input]
 • T     : Certainty Threshold.   [input]
 • C     : Result(C=>T).          [output]

 % T,C are all less than or equal to 100
   and greater than 0.
```

```
/* Fuzzy type solve */

solve(KBN,true,T,100).
solve(KBN,(A;B),T,C):-!,
     (solve(KBN,A,T,C);solve(KBN,B,T,C)).
solve(KBN,(A,B),T,C):-
     solve(KBN,A,T,X),solve(KBN,B,T,Y),
     min(X,Y,C).
solve(KBN,A,T,C):-
     clause-kb(KBN,A,B,F),times(TT,F,T),
     solve(KBN,B,TT,CC),times(CC,F,C).

clause-kb(KBN,A,B,F):-
     X=..[KBN,Y,F],X,
     (Y=(A:-B)->true:Y=A,B=true).
```

```
min(X,Y,X):-X=<Y,!.
min(X,Y,Y).

times(X,Y,z):-
     var(X),integer(Y),integer(Z) ,!,
     divide(Z,Y,X),X>0,X=<100.
times(X,Y,Z):-
     integer(X),integer(Y),var(Z),!,
     multiply(X,Y,Z),Z>0,Z=<100.

divide(T,F,TT):-T1 is T*100,TT is T1/F.
multiply(CC,F,C):-C1 is CC*F,C is C1/100.

kb(parent(f1,f2),50).
kb(parent(f2,f3),50).
kb(parent(f1,m2),100).
kb(parent(m2,m3),50).
kb(parent(f2,f1),0).
kb((ancestor(X,Y):-parent(X,Y)),100).
kb((ancestor(X,Y):-parent(X,z),ancestor(Z,Y)),100).
```

[APPENDIX-2] Execution traces.

```
| ?- solve(kb,ancestor(X,Y),100,C).

X = f1,
Y = m2,
C = 100 ;

no
| ?- solve(kb,ancestor(X,Y),50,C).

X = f1,
Y = f2,
C = 50 ;

X = f2,
Y = f3,
C = 50 ;

X = f1,
Y = m2,
C = 100 ;

X = m2,
Y = m3,
C = 50 ;

X = f1,
Y = f3,
C = 50 ;

X = f1,
Y = m3,
C = 50 ;

no
```