LA-008

実行可能な代数仕様の停止性証明について
On proving termination of executable algebraic specifications

中村 正樹[1] 二木 厚吉[1]
Masaki Nakamura Kokichi Futatsugi

1. はじめに

等式に基づく代数仕様言語である OBJ 言語族 [1, 2, 3, 4] は，代数をモデルに持つ仕様言語であり，項書換システム（TRS）[5] に基づく実行可能性が大きな特徴である．TRS の重要な性質の 1 つに停止性がある．停止性は，仕様の実行が有限時間内に停止することを保証する性質である．TRS の停止性は決定不能問題であり，数多くの停止性の十分条件，判定ツールが提案されている．本稿では停止性を持つ仕様の作成法に注目する．仕様作成者の目的は，できあがった仕様の停止性判定ではなく，停止性を持つように仕様を作成することである．そのためには，いずれかの停止性判定手法を深く理解し，その手法で停止性を示せるように仕様を作成することが重要となる．本稿では，代数をモデルにもつ代数仕様の作成者が，直観的に理解しやすい停止性判定手法を提案する．

2. 代数仕様

OBJ 言語の 1 つ CafeOBJ[3] の仕様 NAT+を示す．

```
mod! NAT+{
  [Zero NzNat < Nat]
  op 0 : -> Zero
  op s_ : Nat -> NzNat
  op _+_ : Nat Nat -> Nat
  vars M N : Nat
  eq N + 0 = N .
  eq M + s N = s(M + N) .
}
```

ソート Zero, NzNat, Nat, サブソート関係 Zero < Nat, NzNat < Nat が宣言されている．ソート Zero の定数 0, Nat を引数に持ち NzNat を返す演算子 s_, Nat を 2 つ取り Nat を返す演算子_+_が宣言されている．下線は，引数の位置の指定である．M, N はソート Nat の変数であり，続く等式宣言で用いられる．1 つ目の等式は，項（パターン）N + 0 が 0 と等しいことを表す．N は変数なので，任意のソート Nat の項 t に対して，t + 0 = 0 が成り立つことを表している．2 つ目の等式も同様に，任意の Nat 項 t, t' に対して t + s t' = s(t + t') が成り立つことを表している．

代数とモデル ソートと演算子の宣言部分をシグネチャ，等式の宣言部分を公理と呼ぶ．シグネチャ Σ に対し，Σ-代数 M とは，ソート s に対応する台集合 M_s，演算子 op f : $s_1 \cdots s_n$ -> s に対応する演算（関数）$M_f : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ から成り，順序関係にあるソート $s < s'$ に対して $M_s \subseteq M_{s'}$ を満たす代数である．Σ-代数 M，等式 $l = r$ が，任意の変数から M への写像 a に対して，$\bar{a}(l) = \bar{a}(r)$ を満たすとき，M は $l = r$ を満たすといい，$M \models l = r$ あるいは $M_l = M_r$ と書く．ここで $\bar{a}$ は写像 a から自然な拡張で得られる項から M への写像である．公理 E のすべての等式を満たす Σ-代数を (Σ, E)-代数と呼ぶ．

仕様は，(Σ, E)-代数の集合を意味（表示）する．mod! で始まるモジュールは，最小の (Σ, E)-代数（始代数）を意味し，mod*で始まるモジュールは，すべての (Σ, E)-代数を意味する．仕様が意味する代数の集合の要素を仕様のモデルと呼ぶ．また仕様の輸入（import）などを組み合わせることで，大規模・複雑なシステムの仕様が作成できる．輸入を含む仕様の意味に関しては文献 [3] が詳しい．以下で与えられる自然数の代数 N は，仕様 NAT+のモデルである．$N_{\mathtt{Zero}} = \{0\}$, $N_{\mathtt{NzNat}} = \mathcal{N}_+ = \{1, 2, \ldots\}$, $N_{\mathtt{Nat}} = \mathcal{N} = \{0, 1, 2, \ldots\}$, $N_0 = 0$, $N_{\mathtt{s}}(x) = x + 1$, $N_+(x, y) = x + y$.

仕様の実行 仕様の実行は，実装には向かない両方向性を持つ等式を左から右への方向性を持たせた書換規則と見なすことで行われる．書換規則の集合を項書換システム（TRS）と呼ぶ．TRS による項の簡約は，項に書換規則を順次適用することで行われる．以下は，項 s 0 + s s 0 の簡約の実行例である．

```
CafeOBJ> red in NAT+ : s 0 + s s 0 .
-- reduce in NAT+ : s 0 + s (s 0)
s (s (s 0)) : NzNat
```

より詳しく書換列を見る．項 t_0 = s 0 + s s 0 を考える．等式 e_1 = eq M + s N = s(M + N) に対し，変数 M に s 0, N に s 0 を代入すると，e_1 の左辺と t_0 が一致する．等式を書換規則として適用して，対応する e_1 の右辺のインスタンス t_1 = s (s 0 + s 0) に置き換える．次に，t_1 の部分項 s 0 + s 0 に再び，同じ等式を適用し，t_2 = s (s (s 0 + 0)) に置き換える．最後に等式 e_0 = eq N + 0 = N を適用して，t_3 = s (s (s 0)) が得られる．t_3 はどの等式の左辺のインスタンスにもならないので，簡約命令 red は t_3 を簡約結果として出力する．簡約命令の入力と出力がモデルにおいて等しいという定理が成り立つ [3]．したがって，この実行例は 1 + 2 = 3 の証明である．

3. 停止性

無限長の書換列が存在しないとき，仕様（または TRS）は停止性を持つという [5]．例えば，等式 e_2 = eq s(0 + N) = 0 + s N を NAT+に付け加えると，s(0 + 0) -->$_{e_2}$ 0 + s 0 -->$_{e_1}$ s(0 + 0) -->$_{e_2}$ $\cdots$ というループが存在するため，停止性を持たない．

TRS の停止性の分野は，停止性の自動判定手法およびツールの開発が中心的な話題であるが，本稿では停止性

[1]北陸先端科学技術大学院大学 情報科学研究科, JAIST

を持つ仕様の作成法に注目する．いくつかの停止性判定ツールは，基本的には停止性を持つか否かの判定結果を知らせるツールであり，停止性を持つ仕様を与えてはくれない[2]．停止性を持つ仕様を作成するには，いずれかの停止性判定手法を深く理解し，その手法で停止性を示せるように仕様を作成することが重要であり，停止性判定ツールは確認作業など補助的な役割を担う．停止性判定手法には，大きく構文的な手法と意味的な手法がある．

再帰経路順序　構文的な手法では，再帰経路順序（RPO）が代表的である．RPO は，演算子上の順序関係を種に項の構造に基づいて再帰的に定義される項上の順序関係であり，整礎，すなわち無限下降列を持たないという性質を持つ．また書換規則に順序付けられればそこから生成される書換関係にも順序が付くという特徴も持つ．したがって，すべての書換規則が RPO で順序付けできるならば，停止性が保証される [5]．NAT+は，$+ > s > 0$ という演算子上の順序から生成される RPO により，停止性を示すことができる．

RPO は，実装に向いている反面，適用範囲が狭いという短所がある．RPO を想定した仕様作成は，構文を限定的にし，代数仕様の柔軟性を損なうおそれがある．

多項式順序　意味的な手法は，整礎な代数を用いて停止性を示す．整礎な順序関係を持つ台集合の要素に項を解釈する．書換関係にある項を順序関係のある要素に解釈することで停止性を保証する．代表的な手法に，演算子を自然数上の多項式に解釈する多項式順序がある [5]．NAT+の各演算子を $M_0 = 1$，$M_s(x) = x+1$，$M_+ = x+2y+1$ と解釈する代数 M を考える．等式 e_0 に対して，$M_{N+0} = M_+(M_N, M_0) = M_N + 3 > M_N$ が成り立ち，等式 e_1 に対して，$M_{M+sN} = M_+(M_M, M_s(M_N)) = M_M + 2M_N + 3 > M_M + 2M_N + 2 = M_s(M_+(M_M, M_N)) = M_{s(M+N)}$ が成り立つため，M は NAT+の停止性を保証する．

代数によって項を解釈する意味的な手法は，代数をモデルに持つ代数仕様に一見，相性がよいように思われる．しかしながら，仕様のモデルとなる代数では，等式 $l = r$ は等しく解釈される（$M_l = M_r$）のに対し，停止性を示すための整礎な代数では，不等式（$M_l > M_r$）に解釈される．すなわち，仕様のモデルとはまったく異なる代数を考える必要があり，仕様作成者には負担となる．

依存対　近年，停止性の分野で，依存対と呼ばれる停止性判定に画期的な概念が提案された [6]．依存対は，等式中の停止性に本質的な部分を抜き出した項の対である．依存対を用いることで，これまで RPO などで示すことのできなかった TRS の停止性も扱えるようになった．現行のほとんどの停止性判定ツールは依存対を用いている．

等式の左辺のルート記号を**定義演算子**と呼ぶ．演算子 f の名前変え演算子を $f^\#$ と書く．$f^\#$ は他のどの既存の演算子とも異なる演算子とする．項 t に対し，項のルート記号を名前変えした項を $t^\#$ と書く．例：$t = 0 + N$ のとき $t^\# = 0 +^\# N$．ある等式 $l = r$ が存在し，r の部分項 u のルート記号が定義演算子であるとき，項の対 $(l^\#, u^\#)$ を**依存対**と呼ぶ [6]．

命題 1 ([6])　項上の擬順序 $\gtrsim$ が，(1) $>$ が整礎，(2) $\gtrsim$ が弱単調，(3) $\gtrsim$ および $>$ が代入に閉じている，(4) 任意の等式 eq $l = r$ に対して $l \gtrsim r$，(5) 任意の依存対 $(l^\#, u^\#)$ に対して $l^\# > u^\#$ を満たすとき，TRS は停止性を持つ．□

ここで，順序 $>$ は，$a > b \Leftrightarrow a \gtrsim b \wedge b \not\gtrsim a$ で定義される．任意の項 t, t'，演算子 f に対して，$t \gtrsim t' \Rightarrow f(\ldots t \ldots) \gtrsim f(\ldots t' \ldots)$ を満たす $\gtrsim$ を弱単調と呼ぶ[3]．

命題 1 で注目すべき点は，順序付けの緩和である．通常，停止性を示す場合，すべての書換規則に対して $l > r$ と真に順序付けを行う整礎な項上の順序 $>$ を構築する．依存対を用いた方法では，依存対には真に順序づけを行う必要があるが，書換規則は真に減じる必要はなく，増えなければよい．

4. 代数モデルに基づく停止性判定手法

代数仕様のモデルを利用した停止性判定手法を提案する．前節で触れたように，多項式順序などの整礎な代数による判定手法は，等式を不等式に解釈するため，仕様のモデルとはまったく異なる代数を用意する必要があった．ところが，依存対を用いた停止性判定（命題 1）では，等式を等式として解釈しても構わない．よって仕様のモデルそのものを停止性判定の擬順序の構築に用いることが可能となる．

定義 1　仕様 SP を輸入した依存対仕様 $SP^\#$ を以下のように定義する．(1) mod*で宣言する．(2) 保護モードの輸入宣言 pr(SP) を宣言する．(3) SP に含まれないソート DP を宣言する．(4) 演算子 op _>_: DP DP -> Bool を宣言する[4]．(5) 定義演算子 op f :$s_1 \cdots s_n$ -> s に対して，演算子 op $f^\#$:$s_1 \cdots s_n$ -> DP を宣言する．(6) 依存対 $(l^\#, u^\#)$ に対して，等式 eq $l^\#$ > $u^\#$ = true を宣言する．□

依存対仕様は以下のような形をしている．

```
mod* SP# {
  pr(SP) [DP]
  op _>_ : DP DP -> Bool
  op f1# : ... -> DP
  op f2# : ... -> DP
  ...
  ep f1#(...) > f1#(...) = true .
  ep f2#(...) > f1#(...) = true .
  ep f2#(...) > f2#(...) = true .
  ...
}
```

ここで省略した $f^\#$ の引数... は，元の f の引数と同じである．$(f_1^\#(\ldots), f_2^\#(\ldots))$ などは依存対である．$SP^\#$

[2]停止性を持たない原因を出力する機能を持つツールも存在するが，その情報から直ちに停止性を持つ仕様が得られるわけではない．

[3]ただし，$t, t', f(\ldots)$ は名前変えされた記号 $(g^\#)$ を含まない．

[4]CafeOBJ では，すべての仕様は汎用性の高い論理値の組込モジュール BOOL を暗黙に保護モードで輸入しており，論理値のソート Bool，ソート Bool の定数 true，false，論理演算子_and_，_or_などを明示的に宣言することなしに使用できる．BOOL のモデルは，それぞれの要素を自然に解釈したブール代数である．

のモデルを考える．mod*で宣言することで，DP は任意の集合に解釈できる．$f^{\#}$ も任意の関数に解釈できる．演算子_>_は，依存対に対して真となる関数（述語）に解釈される．SP を保護モードで輸入しているので，$SP^{\#}$ のモデル DP は必ず SP のモデル M を部分モデルとして含まなければならない．すなわち DP 以外の既存のソートおよび $f^{\#}, >$ 以外の既存の演算子は，DP においても M と同じく解釈される．

定義 2 仕様 $SP^{\#}$ のモデル DP に対し，$DP_{>}$ が整礎な順序であるとき，DP を整礎なモデルという．□

$DP_{>}(x, y)$ を $x > y$ と略す．以下の定理が成り立つ．

定理 1 依存対仕様 $SP^{\#}$ の整礎なモデル DP が存在するとき，SP は停止性を持つ．

証明 項上の擬順序 $t \gtrsim t'$ を以下で定義する．(t, t') が依存対のとき $t > t'$，そうでないとき $DP_t = DP_{t'}$．命題 1 を満たすことを示す．(1) DP は整礎なモデルであるため $>$ は整礎．(2) 弱単調では名前変えされた記号は考えない．$t \gtrsim t'$ で (t, t') が依存対でないならば，定義より $DP_t = DP_{t'}$ である．代数モデルの性質から，任意の演算子 f に対して，$DP_{f(\ldots t \ldots)} = DP_{f(\ldots t' \ldots)}$ が成り立ち，$f(\ldots t \ldots) \gtrsim f(\ldots t' \ldots)$ である．(3) 代数モデルの性質から明らかに代入に閉じている．(4) DP は (Σ, E)-代数であるので，等式 eq $l = r$ に対して $DP_l = DP_r$ すなわち $l \gtrsim r$ が成り立つ．(5) DP は $SP^{\#}$ の各等式を満たすため，任意の依存対 $(l^{\#}, u^{\#})$ に対して，$DP_{l^{\#}} > DP_{u^{\#}}$ が成り立つ．よって，$l^{\#} > u^{\#}$ が成り立つ．□

5. 停止性を持つ仕様の作成法

定理 1 に基づいて NAT+の停止性を示す．定義演算子は，_+_のみである．仕様 NAT+#は以下で与えられる．

```
mod* NAT+#{
  pr(NAT+) [DP]
  op _>_ : DP DP -> Bool
  op +# : Nat Nat -> DP
  vars M N : Nat
  eq +#(M, s N) > +#(M, N) = true .
}
```

依存対は 2 つ目の等式 e_1 から生成される (+#(M, s N), +#(M, N)) のみである．依存対の各要素は，ソート DP の項である．NAT+#のモデル DP を以下で与える．NAT+の各要素は，自然数の代数 N と同じく解釈する．ソート DP の解釈 DP_{DP} を自然数 $\mathcal{N}$ とし，定義演算子+#の解釈を $DP_{+\#}(x, y) = y$ とする．このとき，DP は NAT+#の整礎なモデルである．実際，$DP_{+\#(\mathrm{M},\mathrm{sN})} = DP_{\mathrm{N}} + 1 > DP_{\mathrm{N}} = DP_{+\#(\mathrm{M},\mathrm{N})}$ より，DP は等式 eq +#(M, s N) > +#(M, N) = true を満たす．

ここで $N_{+\#}(x, y) = y$ と解釈したのは，NAT+の等式が_+_の 2 引数目が減じるように定義されているためである．依存対を用いない手法では，項全体に注目しなくてはいけなかったのに対し，依存対を用いた本手法では停止性に本質的な部分だけに集中できる．また，仕様のモデルをそのまま停止性判定に用いることができる．仕様作成者は，仕様のモデルを念頭において仕様を作成しているため，そのモデル上に名前変えした演算子の解釈を与える本手法は，代数仕様の作成者に適していると言える．

例題 リストの反転関数を定義した仕様を考える．

```
mod! LIST{
 pr(NAT+) [List]
 op nil : -> List
 op _;_ : Nat List -> List
 op rev : List -> List
 op revApp : List List -> List
 vars L L' : List
 var N : Nat
 eq revApp(nil, L) = L .
 eq revApp(N ; L, L') = revApp(L, N ; L') .
 eq rev(L) = revApp(L, nil) .
}
```

以下は，リスト $[2,1,0]$ への反転関数 rev の適用の実行例である．$[0,1,2]$ が返されている（一部省略）．

```
LIST> red rev(s s 0 ; s 0 ; 0 ; nil) .
...
0 ; (s 0 ; (s (s 0) ; nil)) : List
```

LIST の停止性を示す．LIST の依存対仕様 LIST#は以下の通りである．

```
mod* LIST#{
 pr(LIST) [DP]
 op _>_ : DP DP -> Bool
 op +# : Nat Nat -> DP
 op rev# : List -> DP
 op revApp# : List List -> DP
 vars M N : Nat
 vars L L' : List
 eq +#(M, s N) > +#(M, N) = true .
 eq revApp#(N ; L, L') > revApp#(L, N ; L')
  = true .
 eq rev#(L) > revApp#(L, nil) = true .
}
```

NAT+の定義演算子も考慮しなくてはならないことに注意する．LIST のモデル L を与える．輸入されている NAT+の各要素は自然数のモデル N と同じく解釈する．List を自然数のリストの集合 $L(\mathcal{N})$ と解釈する．nil は空リスト，_;_はリストに要素を加える関数，rev はリストの反転関数，revApp は，1 引数目のリストの反転リストと 2 引数目のリストを結合したリストを返す関数とする（例：$L_{\mathtt{revApp}}([3,4],[2,1]) = [4,3,2,1]$）．$L$ は LIST のモデルである．すなわち各等式を等式に解釈する．LIST#のモデル DP を与える．$DP_{\mathrm{DP}} = \mathcal{N}$ とする．各依存対の演算子を以下で解釈する．

$N_{+\#}(x, y) = y$,
$N_{\mathtt{rev}\#}(l) = ln(l) + 1$,
$N_{\mathtt{revApp}\#}(l, l') = ln(l)$.

ここで ln はリストの長さを返す関数とする．revApp は，1引数目のリストの長さに関して再帰的に定義されているため，このような解釈とした．rev は revApp より真に大きくする必要があるため，リストの長さに1を足した解釈とした．この DP は，LIST#の整礎なモデルであり，定理1より LIST は停止性を持つ．

依存対サイクル [6] という概念を用いることで，さらに停止性に本質的な依存対を絞り込むことができる．依存対サイクルの解析により，上記の例では，3番目の依存対 (rev#(L), revApp#(L, nil)) は考慮しなくてもよいことがわかる．つまり1番目と2番目の依存対に真に順序を付ければ十分となる．よって，上記の例で多少工夫が必要であった rev#の解釈を与えることなしに停止性を示すことができる．詳細は [6] を参照．

6. まとめ

本論文では，依存対の概念を応用して，モデルに基づく仕様の停止性判定手法を提案した．仕様作成者は，仕様 SP からどのような依存対仕様 $SP^{\#}$ が生成されるかを念頭におき，各依存対に真に順序づけができるように等式を記述することで，停止性のある仕様 SP を作成する．仕様のモデルを停止性判定に利用しているため，通常の多項式順序などと異なり，停止性のために通常の演算子に新たな（モデルとは異なる）解釈を与える必要がなく，名前変えされた定義演算子の解釈に集中することができる．

依存対と多項式順序の組み合わせは，停止性の自動判定の分野でも現在最も有効な組み合わせとして注目されており [7]，AProVE[5], CiME[6] TTT[7]など多くの停止性判定ツールがこの組み合わせを実装している．本稿で提案した手法は，依存対と多項式順序の組み合わせの一般化と言える．多項式順序はすべての演算子を自然数上の多項式に解釈するが，本手法では通常の演算子は仕様のモデルで解釈し，依存対の名前変えされた演算子を整礎な集合上に解釈する．残念ながら本手法は実装には向いていない．仕様ごとに異なるモデルが存在するため，それらの実装を用意しておくことは事実上不可能である．すでに存在するシステムの検証として，代数仕様を用いる場合，そのシステムを拡張する形で停止性判定ツールを構築することも考えられる．しかし，本手法の目的は停止性の自動判定ツールを構築することではなく，停止性を持つ仕様を作成するための手法を提案することである．この目的のためには，仕様のモデルを用いた停止性判定手法は有効である．

今後の課題は，結合律，交換律を持った演算子，条件付き等式など，形式手法によるシステム検証で用いられる実際的な仕様の停止性を扱うことである．二項演算子 $\circ$ の結合律 $((a \circ b) \circ c = a \circ (b \circ c))$，交換律 $(a \circ b = b \circ a)$ は，よく用いられる基本的な性質であるが，そのまま等式として記述すると停止性や合流性などの性質を成り立たなくさせる．そのため，OBJ 言語では演算子の属性として結合律，交換律を宣言し，陽に等式として記述しない手法が採用されている．そのような演算子を含む仕様の実行では，結合律，交換律は書き換えではなく，マッチングによって処理される．その実行モデルとして，AC-TRS と呼ばれる TRS の拡張が存在する．依存対による AC-TRS の停止性に関する研究 [8] は，本手法を結合律，交換律を持つ演算子を含む仕様の停止性に適用する助けとなるであろう．条件付き等式は Bool ソートの項を条件部 c として持つ等式 $(l = r \texttt{ if } c)$ である．条件が真のときのみ等式が成立する．仕様の実行では，条件付き等式の適用は条件部が true に簡約されたときのみ行われる．したがって仕様実行の停止性を議論するには，条件部の呼び出しから生じる無限ループも考慮する必要がある．対応する条件付き TRS（CTRS）の分野では，条件呼び出しを考慮した実効停止性 (Effective termination) と呼ばれる性質がある [9]．実効停止性の証明は，条件付き等式を含む仕様（CTRS）から条件付き等式を含まない仕様（TRS）へ変換し，得られた仕様の停止性を示すことで行われる．変換後の停止性が変換前の実効停止性を保証するような変換になっている [9]．変換を仕様のモデルを保存するように与えることができれば，同様な手法をモデルに基づく依存対法に応用することが可能である．

[5] http://aprove.informatik.rwth-aachen.de/
[6] http://cime.lri.fr/
[7] http://colo6-c703.uibk.ac.at/ttt/

参考文献

[1] K.Futatsugi, J.A.Goguen, J.-P.Jouannaud, and J.Meseguer, Principles of OBJ2., *Proceedings of the 12th ACM Symposium on Principles of Programming Languages, POPL*, pp. 52-66, 1985.

[2] J.A.Goguen, T.Winkler, J.Meseguer, K.Futatsugi and J.-P.Jouannaud, *Software Engineering with OBJ: Algebraic Specification in Action*, Kluwers Academic Publishers, 2000, chapter Introducing OBJ*.

[3] R.Diaconescu and K.Futatsugi, "CafeOBJ report," World Scientific, 1998.

[4] *Maude*, http://maude.cs.uiuc.edu/.

[5] Terese:, Term Rewriting Systems, Vol. 55 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2003.

[6] T.Arts and J.Giesl, Termination of term rewriting using dependency pairs, Theor. Comp. Sci. 236 (2000), 133-178.

[7] E.Contejean, C.Marche, A.P.Tomas, X.Urbain, Mechanically Proving Termination Using Polynomial Interpretations, Journal of Automated Reasoning, Vol.34, No.4, 325 -363, 2005.

[8] K.Kusakari and Y.Toyama, On Proving AC-Termination by AC-Dependency Pairs, IEICE Transactions on Information and Systems, Vol.E84-D, No.5, pp.604-612, 2001.

[9] Ohlebusch, E.: *Advanced topics in term rewriting*, Springer, 2002.