Haskellを利用したKnuth-Bendix完備化手続きの実現

草刈圭一朗，酒井正彦

初版：2002年4月15日
改訂：2012年4月6日（西田直樹）
改訂 (Haskell版)：2021年6月9日（橋本健二）

Knuth-Bendix完備化手続きとは普遍代数 (universal algebra) の語の問題 (word problem) を解くためにKnuthとBendixにより提案された手続きである [6]．この手続きは等式の集合を入力すると，論理的に等価である完備な (合流性と停止性を持つ) 項書換え系を出力する．本資料では，Knuth-Bendix完備化手続きをHaskellによって実現するための指針を示す．なお，完備化手続きを理解するために必要な合流性と停止性に関する基本的な概念や結果を付録に紹介しておいたので参考にされたし．

1　最汎単一化子（Unifyモジュール）

変数を共有しない2つの項の最汎単一化子を計算する関数 `unify` を定義する．例えば，関数 `unify` は2つの項 $F(x, x, H(w))$ と $F(G(z), y, z)$ に対応するTerm型の値を入力すると，ブール値と代入の対

$$(True,\ \{x := G(H(w)),\ y := G(H(w)),\ z := H(w)\})$$

に対応する値を返す．ここで，対の第一要素は入力の2つの項が単一化可能かどうかの判定を表現しており，第一要素が `True` のときの対の第2要素の代入は最汎単一化子になるように定義する．最汎単一化子を求めるアルゴリズムについては論理プログラミングに関連する教科書 (例えば文献 [3] や [4]) に載っている．本節では，文献 [5] のアルゴリズムを紹介する[1]．最初に各自に設計して貰うUnifyモジュールについて記しておく．

```
module Unify where

import TRS
import Reduce

usub :: Subst -> Term -> Term -> (Bool, Subst)
usub .....
usublist :: Subst -> [Term] -> [Term] -> (Bool, Subst)
usublist .....

unify :: Term -> Term -> (Bool, Subst)
unify t1 t2 = usub [] t1 t2
```

見て貰えば分かるように関数 `unify` は補助関数 `usub` を用いて定義されている．また，補助関数 `usub` は関数 `usublist` と相互再帰で定義されている．これらの補助関数は以下のように定義すれば良い．

- $t1 = Node\ (f1,\ ts1)$, $t2 = Node\ (f2,\ ts2)$, $alist = [(x_1, u_1), (x_2, u_2), \ldots, (x_n, u_n)]$ とする．
  このとき，usub alist t1 t2 は以下のように定義される．
  - $f1$ が変数のとき，

[1] プログラミングしやすいよう多少変更してある．

    * $f1 = f2$ ならば $(True,\ alist)$ を返す．
    * $f1 \neq f2$ かつ $f1 \in Var(t2)$ ならば $(False,\ [])$ を返す．
    * $f1 \neq f2$ かつ $f1 \notin Var(t2)$ ならば $(True, [(f1, t2), (x_1, u'_1), (x_2, u'_2), \ldots, (x_n, u'_n)])$ を返す[2]．
      ここで，各 u'_i は u_i に代入 $[(f1, t2)]$ を適用して得られる項である．
  - $f1$ は変数でないが，$f2$ は変数のとき，
    * $f2 \in Var(t1)$ ならば $(False,\ [])$ を返す．
    * $f2 \notin Var(t1)$ ならば $(True, [(f2, t1), (x_1, u'_1), (x_2, u'_2), \ldots, (x_n, u'_n)])$ を返す[2]．
      ここで，各 u'_i は u_i に代入 $[(f2, t1)]$ を適用して得られる項である．
  - $f1$ も $f2$ も変数でないとき，
    * $f1 = f2$ ならば usublist alist ts1 ts2 の返り値を返す．
    * $f1 \neq f2$ ならば $(False,\ [])$ を返す．

- $ts1 = [s_1, s_2, \ldots, s_n]$, $ts2 = [t_1, t_2, \ldots, t_m]$ とする．
  このとき，usublist alist ts1 ts2 は以下のように定義される．
  - $n = m = 0$ のときは $(True,\ alist)$ を返す．
  - $n, m > 0$ のときは，$usub\ alist\ s1\ t1$ の返り値を $(b,\ alist1)$ とすると，
    * $b = True$ のときは $usublist\ alist1\ [s'_2, \ldots, s'_n]\ [t'_2, \ldots, t'_m]$ の返り値を返す．
      ここで，各 s'_i は s_i に代入 $alist1$ を適用して得られる項であり，各 t'_i は t_i に代入 $alist1$ を適用して得られる項である．
    * $b = False$ のときは $(False,\ [])$ を返す．
  - それ以外のときは $(False,\ [])$ を返す．

問 1 以下のそれぞれの 2 つの項は単一化可能かどうか判定し，単一化可能ならば最汎単一化子を求めよ．また，各自の作成したシステムで検証せよ．

(i) $F(x, S(O))$ と $F(O, y)$．　(ii) $F(x, x)$ と $F(O, S(O))$．　(iii) $F(x, x, H(w))$ と $F(G(z), y, z)$．

2 危険対（CP モジュール）

危険対を求めるために必要となる変数の名前替えを行なう関数 `uniquevar` を配布したファイル `Rename.hs` において定義されている構造 Rename において用意しておいた．以下に関数 `uniquevar` の利用例を示す．

```
> (r1, r2) = uniquevar (rdrule "F(x,y) -> x", rdrule "G(x,y) -> H(y,x)")
> putStr $ prrule r1 ++ "\n" ++ prrule r2 ++ "\n''
F(x,y) -> x
G(x:1,y:1) -> H(y:1,x:1)
```

さて，以下が各自に実装して貰う CP モジュールである．

```
module CP where

import TRS
import Reduce
import Unify
import Rename

cp_component :: Term -> Rule -> [(Term, Subst)]
```

[2]結果として `x := x` のような場合が生じても取り除かないこと．

```
cp_component.....
cp_componentlist :: [Term] -> Rule -> [([Term], Subst)]
cp_componentlist.....

cp_assemble :: Term -> [(Term, Subst)] -> EquationSet
cp_assemble.....

cpair :: Rule -> Rule -> EquationSet
cpair r1 r2 = cp_assemble r3 (cp_component l3 (l4, r4)) ++ cp_assemble r4 (cp_component l4 (l3, r3))
  where ((l3, r3), (l4, r4)) = uniquevar (r1, r2)
```

見て貰えば分かるように関数 cpair は関数 uniquevar を用いて入力された 2 つの書き換え規則の変数を名前変えし，その後，補助関数 cp_component と cp_assemble を用いて定義される．また，補助関数 cp_component は関数 cp_componentlist と相互再帰で定義されている．各補助関数が以下の性質を持つように設計すれば関数 cpair は 2 つの書き換え規則の間に生成される全ての危険対のリストを計算する．

- 関数 cp_component と cp_componentlist は以下の性質を満たす．ただし，$ts = [t_1, \ldots, t_n]$ とする．

$$\begin{aligned} cp\_component\ l_1\ (l_2, r_2) &= [(C[r_2], \theta) \mid l_1 \equiv C[l_1'],\ l_1' \notin \mathcal{V},\ \theta : l_1' \text{と } l_2 \text{の最汎単一化子}] \\ cp\_componentlist\ ts\ (l,\ r) &= [([t_1, \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_n],\ \theta) \mid (t_i',\ \theta) \in cp\_component\ t_i\ (l,\ r)] \end{aligned}$$

- 関数 cp_assemble は以下の性質を満たす．

$$cp\_assemble\ r'\ [(t_1, \theta_1), \ldots, (t_n, \theta_n)] = [(t_1\theta_1, r'\theta_1), \ldots, (t_n\theta_n, r'\theta_n)]$$

問 2 それぞれの書き換え規則間に生成される全ての危険対を求め，各自の作成したシステムで検証せよ．

1. $A \to O$ と $F(A) \to S(O)$.
2. $F(x, S(O)) \to G(x)$ と $F(O, y) \to H(y)$.
3. $F(G(x), G(x)) \to H1(x)$ と $G(F(x, x)) \to H2(x)$.
4. $H(G(F(A, x)), F(x, B)) \to I(x)$ と $F(y, y) \to y$.

3 簡約化順序（Order モジュール）

項書換え系の停止性を保証する簡約化順序として辞書式経路順序と再帰経路順序を実現する．最初に各自に設計して貰う Orde モジュールを見て貰おう．

```
module Order where

import Util
import TRS

type Mset a = [a]

mkgrter :: Ord a => (a -> a -> Bool) -> a -> a -> Bool
mkeq :: Ord a => (a -> a -> Bool) -> a -> a -> Bool

eq_symbol :: Assoc String Int -> String -> String -> Bool
grtereq_symbol :: Assoc String Int -> String -> String -> Bool
grter_symbol :: Assoc String Int -> String -> String -> Bool

grtereq_lex :: Ord a => (a -> a -> Bool) -> [a] -> [a] -> Bool
grtereq_lpo :: Assoc String Int -> Term -> Term -> Bool
grter_lpo :: Assoc String Int -> Term -> Term -> Bool

grtereq_mset :: Ord a => (a -> a -> Bool) -> Mset a -> Mset a -> Bool
grtereq_rpo :: Assoc String Int -> Term -> Term -> Bool
grter_rpo :: Assoc String Int -> Term -> Term -> Bool
```

擬順序 $\gtrsim$ から生成される狭義の半順序と同値関係をそれぞれ $\gtrsim$ と $\sim$ とする．関数 mkgrter は $\gtrsim$ から $\gtrsim$ への変換，関数 mkeq は $\gtrsim$ から $\sim$ への変換に対応する関数である．

```
mkgrter gsim x y = gsim x y && not (gsim y x);

mkeq gsim x y = gsim x y && gsim y x;
```

関数 grtereq_lex と grtereq_mset は，擬順序 $\gtrsim$ から辞書式順序 $\gtrsim^{lex}$ への変換と多重集合上の擬順序 $\gtrsim^{mul}$ への変換にそれぞれ対応する関数である．

記号上の順序は，記号と整数の連想リストで [("0",1),("S",1),("P",3),("M",4),("F",5)] のように表現し，記号上の大小関係は対応する自然数の大小関係で定義されるとする．記号上の擬順序 $\gtrsim$ に対応する関数 grtereq_symbol と，$\gtrsim$ から生成される狭義の半順序 $\gtrsim$ と同値関係 $\sim$ にそれぞれ対応する関数 grter_symbol と eq_symbol は以下で定義される．

```
eq_symbol ol x y =
    if x == y then True
    else case (find x ol, find y ol) of
        (Just n, Just m) -> n == m
        _                -> False

grtereq_symbol ol x y =
    if x == y then True
    else case (find x ol, find y ol) of
        (Just n, Just m) -> n >= m
        _                -> False

grter_symbol ol x y =
    case (find x ol, find y ol) of
        (Just n, Just m) -> n > m
        _                -> False
```

上述の関数を利用して，辞書式経路順序 $>_{lpo}$ に対応する関数 grter_lpo は $\gtrsim_{lpo}$ に対応する関数 grtereq_lpo と共に相互再帰で定義できる．また，再帰経路順序 $>_{rpo}$ に対応する関数 grter_rpo は $\gtrsim_{rpo}$ に対応する関数 grtereq_rpo と共に相互再帰で定義できる．

問 3 以下のそれぞれの 2 つの項の間に辞書式経路順序 $>_{lpo}$ で順序付けが成功するか，また，再帰経路順序 $>_{rpo}$ ではどうか答えよ．さらに，解答を各自の作成したシステムで検証せよ．ただし，記号上の順序 $\gtrsim$ は $F \gtrsim G \sim H \gtrsim A \gtrsim B$ とする．

(1) $F(A,A)$ と $F(A,B)$.　(2) $F(A,B)$ と $F(B,A)$.　(3) $G(F(x))$ と $F(x)$.
(4) $F(x)$ と $G(F(x))$.　(5) $G(x)$ と $H(x,x)$.　(6) $G(x,x)$ と $H(x)$.
(7) $F(B,A,A)$ と $F(A,B,B)$.　(8) $F(B,F(A,A))$ と $F(A,A)$.

4 Knuth-Bendix 完備化手続き（KB モジュール）

さて，準備は全て整った．簡約化順序と等式の集合を入力すると Knuth-Bendix 完備化手続きを実行して完備な項書換え系を出力する関数 kb を定義しよう．先ずは各自に設計して貰う KB モジュールを見て貰おう．

```
module KB (kb) where

import Util
import TRS
import Reduce
import CP
import Debug.Trace


orientation :: (Term -> Term -> Bool) -> (RuleSet, EquationSet)

composition :: (Rule, RuleSet, EquationSet) -> (Rule, RuleSet, EquationSet)

deduction   :: (Rule, RuleSet, EquationSet) -> (Rule, RuleSet, EquationSet)

collapse    :: (Rule, RuleSet, EquationSet) -> (Rule, RuleSet, EquationSet)

simplification :: (RuleSet, EquationSet) -> (RuleSet, EquationSet)

deletion       :: (RuleSet, EquationSet) -> (RuleSet, EquationSet)


kbsub :: (Term -> Term -> Bool) -> (RuleSet, EquationSet) -> Int -> RuleSet
kbsub _ (rs, []) n = trace ("Success ("++(show n)++" steps).\n") rs
kbsub grter (rs, es) n =
  trace
    ("=== "++(show (n+1))++" step ===\n"
     ++"R =\n"++(prrules rs1)
     ++"E =\n"++(preqs es1))
    (kbsub grter (rs1, es1) (n+1))
  where (rs1, es1) = kbstep grter (rs, es)
kbstep :: (Term -> Term -> Bool) -> (RuleSet, EquationSet) -> (RuleSet, EquationSet)
kbstep grter (rs, es) =
    (deletion
      (simplification
        ((\(r, rs, es) -> (r:rs, es))
          (collapse
            (deduction
              (composition (orientation grter (rs, es))))))))

kb :: (Term -> Term -> Bool) -> EquationSet -> RuleSet
kb grter es =
  kbsub grter ([], filter (\(t1,t2) -> t1 /= t2) es ) 0
```

関数 kb の定義の引数にある変数 grter と es はそれぞれ Knuth-Bendix 完備化手続きに入力される簡約化順序と等式の集合に対応する．以下では各補助関数の説明を行う．なお，これらの関数は文献 [1] に書かれている規則に対応している．詳細は文献 [1] を参照されたし．

- `orientation`: (RuleSet, EquationSet) → (Rule, RuleSet, EquationSet)

  引数を (R, E) とする．E の等式のうち両辺の間に順序がついているものを一つ選ぶ[3]．選んできた項の対を (t, s) (ただし $t > s$) とすると，((t, s), R, E − {t=s}) を返す．ただし，$s = t$ も忘れずに考慮すること．

- `composition`: (Rule, RuleSet, EquationSet) → (Rule, RuleSet, EquationSet)

  引数を (r, R, E) とする．{r}∪R を用いて R のすべての規則の右辺を正規形にする．

- `deduction`: (Rule, RuleSet, EquationSet) → (Rule, RuleSet, EquationSet)

  引数を (r, R, E) とする．r と {r}∪R の規則の間にできる全ての危険対を E に加える．

[3]等式の集合 E から (t,s) を選ぶ戦略には注意すること．戦略次第では正規化が成功したはずの入力に対しても暴走することがある．だが逆に，賢い戦略を選ぶと飛躍的な効率の向上につながる．

- `collapse`: (Rule, RuleSet, EquationSet) → (Rule, RuleSet, EquationSet)

  引数を ((l, r), R, E) とする．R の規則のうちで左辺が l の例 (instance) となる部分項を持つ規則を全て取り除く．

- `simplification`: (RuleSet, EquationSet) → (RuleSet, EquationSet)

  引数を (R, E) とする．R を用いて E の等式の両辺を正規形にする．

- `deletion`: (RuleSet, EquationSet) → (RuleSet, EquationSet)

  引数を (R, E) とする．E から両辺の等しい等式を取り除く．

参考文献

[1] L.Backmair, N.Dershowitz, Equational Inference, Canonical Proofs, and Proof Orderings, J.ACM, Vol.41, No.2, pp.236–276, 1994.

[2] N.Dershowitz, Termination of Rewriting, J.Symbolic Computation, Vol.3, pp.69–116, 1987.

[3] J.H.Gallier, Logic for Computer Science: Foundations of Automatic Theorem Proving, John Wiley & Sons inc., 1987.

[4] J.v.Leeuwen 監修, 廣瀬健, 野崎昭弘, 小林孝次郎 監訳, コンピュータ基礎理論ハンドブック II: 形式的モデルと意味論, 丸善株式会社, 1994.

[5] J.P.Jouannaud, C.Kirchner, Solving Equations in Abstract Algebras: a Rule-Based Survey of Unification, in Computational logic: essays in honor of Alan Robinson, eds. J.L.Lassez, G.Plotokin, MIT press, pp.257–321, 1991.

[6] D.E.Knuth, P.B.Bendix, Simple Word Problems in Universal Algebra, In J.Leech Ed., *Computational Problems in Abstract Algebra*, pp.263–297, Pergamon Press, Oxford, U.K., 1970.

A　合流性

合流性を解析するために Knuth と Bendix は危険対の概念を導入した [6]．危険対の概念を用いると，停止性を持った項書換え系が合流性を持つかどうかの判定を非常に効率的に行うことができる[4]．以下では危険対と合流性の関係に関する基本的な結果を紹介する．最初に，危険対の概念を紹介するために必要な最汎単一化子の概念を定義する．

定義 A.1 項 s と t が単一化可能 *(unifiable)* とは，ある代入 θ が存在して $s\theta \equiv t\theta$ となることである．このとき，θ を項 s と t の単一化子 *(unifier)* と呼ぶ．項 s と t の単一化子 θ が最汎単一化子 *(most general unifier)* であるとは，任意の s と t の単一化子 θ' に対してある代入 θ'' が存在して $\theta' = \theta'' \circ \theta$ となることである．

例えば，二つの項 $F(x, x, H(w))$ と $F(G(z), y, z)$ は単一化可能であり，その最汎単一化子 θ は以下のようになる．

$$\theta = \{x := G(H(w)),\ y := G(H(w)),\ z := H(w)\}$$

なお，任意の単一化可能な項 s と t に対し，最汎単一化子が (変数の名前変えを除いて) 一意に存在することが知られている．

[4] 与えられた項書換え系が合流性を持つかどうか，という問題は一般には決定不能問題である．

定義 A.2 $l_1 \to r_1$ と $l_2 \to r_2$ を書き換え規則とする．ここで，これらの書き換え規則は変数を共有しないと仮定して良い．ある文脈 $C[\,]$ が存在して $l_2 \equiv C[l_2']$ かつ l_2' は変数でなく l_1 と l_2' が単一化可能であるとき，書き換え規則 $l_1 \to r_1$ は書き換え規則 $l_2 \to r_2$ に重なるという．このとき l_1 と l_2' の最汎単一化子を θ とすると，項の対 $\langle C[r_1]\theta, r_2\theta\rangle$ は $l_1 \to r_1$ の $l_2 \to r_2$ に対する危険対 *(critical pair)* と呼ばれる．

項書換え系 R の危険対とは R の 2 つの書き換え規則 *(*同一の書き換え規則をとってきても良い*)* の間の危険対である．このとき必要ならば，すなわち取り出した 2 つの書き換え規則が変数を共有していた場合，変数の名前変えを行っても良い．ただしこの定義では，任意の書き換え規則 $l \to r$ に対し，明らかに $l \to r$ は $l \to r$ に根位置で重なり危険対 $\langle r, r\rangle$ を持つのだが，この場合は特例として危険対とは呼ばないことにする．項書換え系 R の危険対全体の集合を $CP(R)$ で記す．

項書換え系 R_1, R_2, R_3 における危険対の集合 $CP(R_1), CP(R_2), CP(R_3)$ は以下のようになる．

$$R_1 = \left\{ \begin{array}{lcl} F(G(x)) & \to & F'(x) \\ G(F(x)) & \to & G'(x) \end{array} \right. \qquad CP(R_1) = \left\{ \begin{array}{l} \langle F(G'(x)), F'(F(x))\rangle \\ \langle G(F'(x)), G'(G(x))\rangle \end{array} \right.$$

$$R_2 = \left\{ \begin{array}{lcl} F(x, G(x)) & \to & G'(x) \\ F(H(y), z) & \to & H'(y, z) \end{array} \right. \qquad CP(R_2) = \left\{ \begin{array}{l} \langle G'(H(y)), H'(y, G(H(y)))\rangle \\ \langle H'(y, G(H(y))), G'(H(y))\rangle \end{array} \right.$$

$$R_3 = \left\{ \begin{array}{lcl} F(F(x)) & \to & G(x) \end{array} \right. \qquad CP(R_3) = \left\{ \begin{array}{l} \langle F(G(x)), G(F(x))\rangle \end{array} \right.$$

定義 A.3 R を項書換え系とする．

- R が局所合流性 *(locally confluence property)* を持つとは，任意の項 t,t_1,t_2 に対し，$t_1 \leftarrow t \to t_2$ ならばある項 s が存在して $t_1 \stackrel{*}{\to} s \stackrel{*}{\leftarrow} t_2$ となることである．
- R が合流性 *(confluence property)* を持つとは，任意の項 t,t_1,t_2 に対し，$t_1 \stackrel{*}{\leftarrow} t \stackrel{*}{\to} t_2$ ならばある項 s が存在して $t_1 \stackrel{*}{\to} s \stackrel{*}{\leftarrow} t_2$ となることである．

定理 A.4 項書換え系 R が合流性を持つならば局所合流性を持つ，しかし逆は一般には成立しない．

局所合流性は合流性を保証しない事に注意して欲しい．局所合流性を持つにも関わらず合流性を持たない項書換え系の構成，などというのは非常に良い鍛練になるだろう．気骨ある若者は挑むべし．他方で，停止性の仮定の下では逆も成立する．

定理 A.5 停止性を持つ項書換え系 R において，合流性を持つことと局所合流性を持つことは必要十分である．

危険対の概念を用いると局所合流性に関する必用十分条件が以下のように与えられる．

定理 A.6 項書換え系 R が局所合流性を持つことと，全ての R の危険対 $\langle p, q\rangle$ が会同関係 *(joinability relation)* にある $(\exists t.\ p \stackrel{*}{\to} t \stackrel{*}{\leftarrow} q)$ ことは必要十分である．

定理 A.5 と A.6 を組み合わせることにより以下の定理が得られる．

定理 A.7 停止性を持つ項書換え系 R において，R が合流性を持つことと全ての R の危険対が会同関係にあることは必要十分である．

問 4 定理 *A.7* を参考にして，与えられた停止性を持つ項書換え系に対し，合流性を持つかどうかを判定するアルゴリズムを提案せよ．

B　停止性

本節では，停止性に関する基本的な概念・用語・結果を紹介する．

B.1　関係と順序

定義 B.1 集合 A 上の 2 項関係 *(binary relation)* とは直積集合 $A \times A$ の部分集合である．2 項関係 Υ にたいし，慣習に従い $(a_1, a_2) \in \Upsilon$ を $a_1 \Upsilon a_2$ で略記する．集合 A 上の 2 項関係 Υ に対し，以下の概念が定義される．

- 各元 $a_1, a_2, a_3 \in A$ にたいし $a_1 \Upsilon a_2 \wedge a_2 \Upsilon a_3 \Rightarrow a_1 \Upsilon a_3$ が成立するならば，Υ は推移的 *(transitive)* であるという．
- 各元 $a \in A$ にたいし $a \Upsilon a$ が成立するならば，Υ は反射的 *(reflexive)* であるという．
- 各元 $a \in A$ にたいし $a \Upsilon a$ が成立しないとき，Υ は非反射的 *(irreflexive)* であるという．
- 各元 $a_1, a_2 \in A$ にたいし $a_1 \Upsilon a_2 \Rightarrow a_2 \Upsilon a_1$ が成立するならば，Υ は対称的 *(symmetric)* であるという．
- 各元 $a_1, a_2 \in A$ にたいし $a_1 \Upsilon a_2 \wedge a_2 \Upsilon a_1 \Rightarrow a_1 = a_2$ が成立するならば，Υ は反対称的 *(antisymmetric)* であるという．
- $a_0 \Upsilon a_1 \Upsilon a_2 \Upsilon \cdots$ のような無限列が存在しないならば，Υ は整礎 *(well-founded)* であるという．

反射的かつ推移的かつ対称的な 2 項関係を同値関係 *(equivalence relation)* と呼ぶ．

定義 B.2 反射的かつ推移的な 2 項関係を擬順序と呼ぶ．反射的かつ推移的かつ反対称的な 2 項関係を半順序と呼ぶ．推移的かつ非反射的な 2 項関係を狭義の半順序と呼ぶ．慣習に従い，擬順序, 半順序, 狭義の半順序をそれぞれ $\gtrsim, \geq, >$ で記す．

定義 B.3 $\gtrsim$ を擬順序とする．2 項関係 $\gtrnsim$ を $\gtrsim \setminus \lesssim$ で定義すると $\gtrnsim$ は狭義の半順序となる．このとき $\gtrnsim$ を $\gtrsim$ から生成される狭義の半順序と呼ぶ．また，2 項関係 $\sim$ を $\gtrsim \cap \lesssim$ で定義すると $\sim$ は同値関係となる．このとき $\sim$ を $\gtrsim$ から生成される同値関係と呼ぶ．

問 5 $\gtrsim$ を擬順序とすると $\sim$ $(= \gtrsim \cap \lesssim)$ が同値関係であり，$\gtrnsim$ $(= \gtrsim \setminus \lesssim)$ が狭義の半順序であることを証明せよ．

B.2　辞書式順序

本節ではリスト上への順序の拡張である辞書式順序を紹介する．

定義 B.4 $>$ を集合 A 上の狭義の半順序とする．辞書式順序 $>^{lex}$ とは以下で再帰的に定義される A のリスト上の順序の事である．

$$\begin{cases} [a_1, a_2, \ldots, a_n] & >^{lex} & [\,] & \textit{if } \ n > 0 \\ [a_1, a_2, \ldots, a_n] & >^{lex} & [a'_1, a'_2, \ldots, a'_m] & \textit{if } \ a_1 > a'_1 \\ [a_1, a_2, \ldots, a_n] & >^{lex} & [a'_1, a'_2, \ldots, a'_m] & \textit{if } \ a_1 = a'_1 \text{ かつ } [a_2, \ldots, a_n] >^{lex} [a'_2, \ldots, a'_m] \end{cases}$$

残念ながら，$>$ の整礎性は $>^{lex}$ の整礎性を保証しない．実際，集合 $A = \{a, b\}$ 上の順序 $a > b$ に対し，$[a] >^{lex} [b, a] >^{lex} [b, b, a] >^{lex} \cdots$ の無限減少列が存在する．この問題はリストの長さに制限を加えることで回避することができる．

定理 B.5 $>$ が集合 A 上の狭義の半順序であることと $>^{lex}$ が集合 A のリスト上の狭義の半順序であることは必要十分である．さらに，任意の正整数 n に対し，$>$ が整礎であることと $>^{lex}$ が長さ n 以下のリスト上で整礎であることは必要十分である．

辞書式順序は，上述のように狭義の半順序を基にした場合のみならず擬順序を基にしても同様に定義できる．

定義 B.6 $\gtrsim$ を集合 A 上の擬順序とし，$\gtrsim$ と $\sim$ をそれぞれ $\gtrsim$ から生成される狭義の半順序と同値関係とする．辞書式順序 $\gtrsim^{lex}$ とは以下で再帰的に定義される A のリスト上の順序の事である．

$$\begin{cases} [a_1, a_2, \ldots, a_n] & \gtrsim^{lex} & [\,] & \\ [a_1, a_2, \ldots, a_n] & \gtrsim^{lex} & [a'_1, a'_2, \ldots, a'_m] & \textit{if } \ a_1 \gtrsim a'_1 \\ [a_1, a_2, \ldots, a_n] & \gtrsim^{lex} & [a'_1, a'_2, \ldots, a'_m] & \textit{if } \ a_1 \sim a'_1 \text{ かつ } [a_2, \ldots, a_n] \gtrsim^{lex} [a'_2, \ldots, a'_m] \end{cases}$$

定理 B.7 $\gtrsim$ が集合 A 上の擬順序であることと $\gtrsim^{lex}$ が集合 A のリスト上の擬順序であることは必要十分である．さらに，$\gtrsim$ と $\gtrsim^{lex}$ から生成される狭義の半順序をそれぞれ $\gtrsim$ と $\gtrsim^{lex}$ とし，n を任意の正整数とすると，長さ n 以下のリスト上で，$\gtrsim$ が整礎であることと $\gtrsim^{lex}$ が整礎であることは必要十分である．

B.3 多重集合上への拡張

多重集合 (multiset) とは $\{a, a, b\}$ のように要素の重複を許した集合のことである．集合 A の元からなる多重集合の全体を $M(A)$ で記す．本節では多重集合上への順序の拡張法を紹介する．

定義 B.8 $>$ を集合 A 上の狭義の半順序とする．$M(A)$ 上の順序 $>^{mul}$ は以下で再帰的に定義される．

$$\begin{cases} M & >^{mul} & \emptyset & \textit{if } \ M \neq \emptyset \\ M \cup \{a\} & >^{mul} & M' \cup \{a\} & \textit{if } \ M >^{mul} M' \\ M \cup \{a\} & >^{mul} & M' & \textit{if } \ a \notin M' \text{ かつ } M \geq^{mul} \{a' \in M' \mid a \not> a'\} \end{cases}$$

定理 B.9 $>$ が集合 A 上の整礎な狭義の半順序であることと $>^{mul}$ が $M(A)$ 上の整礎な狭義の半順序であることは必要十分である．

辞書式順序の場合と同様に擬順序を多重集合上へ拡張することも可能である．

定義 B.10 $\gtrsim$ を集合 A 上の擬順序とし，$\sim$ を $\gtrsim$ から生成される同値関係とする．$M(A)$ 上の順序 $\gtrsim^{mul}$ は以下で再帰的に定義される．

$$\begin{cases} M & \gtrsim^{mul} & \emptyset & \\ M \cup \{a\} & \gtrsim^{mul} & M' \cup \{a'\} & \textit{if } \ a \sim a' \text{ かつ } M \gtrsim^{mul} M' \\ M \cup \{a\} & \gtrsim^{mul} & M' & \textit{if } \ \forall a' \in M'.\, a' \not\sim a \text{ かつ } M \gtrsim^{mul} \{a' \in M' \mid a \not\gtrsim a'\} \end{cases}$$

定理 B.11 $\gtrsim$ が集合 A 上の擬順序であることと $\gtrsim^{mul}$ が $M(A)$ 上の擬順序であることは必要十分である．さらに，$\gtrsim$ と $\gtrsim^{mul}$ から生成される狭義の半順序をそれぞれ $\gtrsim$ と $\gtrsim^{mul}$ とすると，$\gtrsim$ が整礎であることと $\gtrsim^{mul}$ が整礎であることは必要十分である．

B.4 簡約化順序と単純化順序

項書換え系 R が停止性を持つとは書き換え関係 $\underset{R}{\rightarrow}$ が整礎であることである．本節では，簡約化順序の概念を用いた項書換え系の停止性証明法と，簡約化順序の設計において重要な役割を持つ単純化順序の概念，そして代表的な単純化順序である辞書式経路順序と再帰経路順序を紹介する．

定義 B.12 簡約化順序 *(reduction order)* とは項上の狭義の半順序 $>$ であり以下の性質を満たすものである．

- $>$ は整礎である．
- $>$ は文脈に閉じている，すなわち，$\forall C[\,].\ s > t \Rightarrow C[s] > C[t]$.
- $>$ は代入に閉じている，すなわち，$\forall \theta.\ s > t \Rightarrow s\theta > t\theta$.

定理 B.13 R を項書換え系とし $>$ を簡約化順序とする．もし全ての書き換え規則 $l \to r \in R$ にたいし $l > r$ ならば R は停止性を持つ．

問 6 定理 *B.13* を証明せよ．

定義 B.14 単純化順序 *(simplification order)* とは項上の狭義の半順序 $>$ であり以下の性質を満たすものである．

- $>$ は文脈に閉じている，すなわち，$\forall C[\,].\ s > t \Rightarrow C[s] > C[t]$.
- $>$ は代入に閉じている，すなわち，$\forall \theta.\ s > t \Rightarrow s\theta > t\theta$.
- $>$ は部分項性 (subterm property) を持つ，すなわち，全ての項 t と空でない文脈 $C[\,]$ に対し $C[t] > t$.

定理 B.15 単純化順序は整礎である．

単純化順序と簡約化順序を比べると，単純化順序は部分項性を要求する代わりに整礎性を要求していない．だがしかし，上記の定理より単純化順序は整礎性を持ち，それゆえに簡約化順序となる．単純化順序の設計による簡約化順序の設計は，直接簡約化順序を設計するより非常に容易である場合が多い．なぜならば一般に，部分項性の証明は整礎性の証明に比べて非常に容易であるためである．以下では，代表的な単純化順序として辞書式経路順序 (lexicographic path ordering) と再帰経路順序 (recursive path ordering) を紹介する．詳細は文献 [2] などを参照されたし．

定義 B.16 $\gtrsim$ を記号上の擬順序とする[5]．ただし，各変数 x, y に対し $x \gtrsim y \iff x = y$ であるものとする．このとき，辞書式経路順序 $>_{lpo}$ と再帰経路順序 $>_{rpo}$ は以下で定義される．

- $>_{lpo}$ は $\gtrsim_{lpo}$ から生成される狭義の半順序であり，$\gtrsim_{lpo}$ は以下で定義される．
  - $x \in Var(s)$ ならば $s \gtrsim_{lpo} x$.
  - 以下のいずれか一つが成立すれば $f(s_1, \ldots, s_n) \gtrsim_{lpo} g(t_1, \ldots, t_m)$.
    - $f \sim g$ かつ $[s_1, \ldots, s_n] \gtrsim_{lpo}^{lex} [t_1, \ldots, t_m]$ かつ各 j で $f(s_1, \ldots, s_n) >_{lpo} t_j$.
    - $f \gtrsim g$ かつ各 j で $f(s_1, \ldots, s_n) >_{lpo} t_j$.
    - $s_i \gtrsim_{lpo} g(t_1, \ldots, t_m)$ を満たす i が存在．
- $>_{rpo}$ は $\gtrsim_{rpo}$ から生成される狭義の半順序であり，$\gtrsim_{rpo}$ は以下で定義される．
  - $x \in Var(s)$ ならば $s \gtrsim_{rpo} x$.
  - 以下のいずれか一つが成立すれば $f(s_1, \ldots, s_n) \gtrsim_{rpo} g(t_1, \ldots, t_m)$.
    - $f \sim g$ かつ $\{s_1, \ldots, s_n\} \gtrsim_{rpo}^{mul} \{t_1, \ldots, t_m\}$.
    - $f \gtrsim g$ かつ各 j で $f(s_1, \ldots, s_n) >_{rpo} t_j$.
    - $s_i \gtrsim_{rpo} g(t_1, \ldots, t_m)$ を満たす i が存在．

定理 B.17 辞書式経路順序 $>_{lpo}$ と再帰経路順序 $>_{rpo}$ は共に単純化順序であり，それゆえに簡約化順序である．

[5] 狭義の半順序として定義する方が一般的であるが，今回はあえて擬順序として定義した．