プログラムの代数的仕様記述法に関する研究

1982年12月

杉山裕二

内　容　梗　概

本論文は，筆者が大阪大学基礎工学部情報工学科高研究室在職中に行った研究のうち，プログラムの代数的仕様記述法に関する研究を3章にまとめたものである．

緒論および各章の第1節では，研究の現状，その工学上の意義，本研究の新しい諸成果について概説している．各章の最後の節および全体の結論では，本研究で得られた主な結果と今後に残された問題について述べている．

第1章では，既存データタイプの拡張という立場での代数的仕様“B-仕様”を定式化し，無矛盾性ならびに関数値の“don't care”に関する諸性質を示すとともに，B-仕様の一種“B-順序機械”を導入し，その“簡約化”および抽象的“表”での一般的な実現方法について述べている．

第2章では，ファイル管理システムの順編成ファイルに関する部分の仕様を2つの抽象化レベルで代数的に記述し，同期機能の表現方法について述べている．次，これらの記述が，無矛盾であり，且つ必要な関数値が定義されているという意味で“準完全”であることを示し，2つのレベルの対応関係“機構化”について論じている．

第3章では，B-仕様の一種“完全順序項書き換え系”を導入し，そこでの効率のよい計算方法について論じている．項を有向アサイクリックグラフ（dag）で表し，最小の書き換え回数で“値”を求めるdag書き換えアルゴリズムを示すとともに，dagの書き換え自体を効率よく行うために，手続き的なプログラムへの変換方法を示している．

プログラムの代数的仕様記述法に関する研究

目　　次

緒　　論

ソフトウェアの大規模化，複雑化に伴い，プログラムの作成過程で仕様を的確に記述することが重要になってきており，形式的な仕様記述法も幾つか提案されている．仕様を形式的に記述することにより，仕様の曖昧性をなくし，無矛盾性，完全性を始めとした仕様自身の正しさ，及び仕様に従って作成されたプログラムの正しさの検証を厳密に行うための基礎が確立する．仕様の形式性はプログラムの信頼性を高めるために不可欠のものであるといえよう．

代数的仕様記述法は，抽象データタイプの形式的な仕様記述法の一つとして提案されたものである．抽象データタイプの概念は，プログラミング方法論の立場から，プログラムの信頼性，理解性，柔軟性を高めるために導入されたものの一つで，データタイプを，それに対する操作の組でもって定義しようとするものである．その仕様記述の問題は，規模，複雑さの差違を除いて，プログラムの仕様記述の問題と原理的に同じであり，抽象データタイプに対して提案された仕様記述法はプログラムの仕様記述にも適用できると考えられる．

抽象データタイプの形式的な仕様記述法には，代数的方法の外に，述語論理による方法，抽象モデルによる方法，状態機械による方法，Scott の方法などがあるが，いずれも一長一短がある[24]．本論文で採用する代数的仕様記述法は，意味の定義が簡明であり，記述すべき対象固有の構造および性質を，書き手が選んだ抽象化のレベルで意味をもつ概念を用いて表現することができる．又，仕様と関数的なプログラムを同じ枠組の中で議論するこ

とができるという利点もあり，最近盛んに研究が行われている．

抽象データタイプの代数的仕様記述法は，抽象データタイプを多ソート代数として捉え，データタイプに対する操作を関数で，又，データタイプに対して行ってきた操作の系列を，対応する関数から構成される項で表し，操作系列間の“等価関係”を項の上の合同関係で表そうとするものである．項の上の合同関係は公理によって定められるが，公理がどのような合同関係を表すかについては，例えば，始代数の立場と終代数の立場がある．始代数の立場は，公理から等価であると導出できないような二つの項は（合同関係による項集合の分割で）異なる同値類に属するというもので，Goguen らによって定式化されている．これに対し，終代数の立場は，既存データタイプの拡張として抽象データタイプを定義し，既存データタイプへの出力値から区別できないような項は同じ同値類に属するというものであり，Guttag らにより提案され，Wand による定式化がある．

始代数の立場は，前提とするデータタイプが不要であるという反面，そのことが記述を煩雑にする．例えば，抽象データタイプの定義においてブール値，整数およびそれらに関する演算を用いる場合，これらの基本的なデータタイプの定義をも問題の抽象データタイプの定義の中に含めねばならない．記述性からも，又，階層性からも，既存のデータタイプを再度定義することなく使用できる方が，すなわち，抽象データタイプを既存データタイプの拡張として定義できる方が望ましいといえよう．又，入出力対応からは全く同じ振舞をするが公理からは等しいことが導出できない二つの状態に対し，始代数の

立場はそれらを異なった状態とし，終代数の立場は同じ状態と定める．これらを同じとするか異なるとするかは実現時の都合で決めればよく，仕様としては指定しない方が望ましい．仕様が書かれている抽象化のレベルにおいて本質的でない出力値，いわゆる“don't care”の値についても同様のことがいえる．

本論文では，既存データタイプの拡張として代数的仕様を記述し，それが表すものを既存データタイプと両立するような合同関係のクラスとする立場を採る．そのクラスは，始代数および終代数の立場の合同関係を共に含んでいる．又，そのような立場から，仕様間の関係としての機構化の定義，及びその特殊な場合として，仕様によって定義されるプログラムというものも自然に定式化される．

代数的手法による仕様の記述例として引合いに出されるものは，スタック，配列，表など，非常に簡単なものばかりであり，代数的手法の実際面の応用について疑問視されていた．又，代数的記述をプログラムとして見たとき，一般には有効な実行方法がないという欠点があった．そこで本論文では，具体的なシステムの記述例として順編成ファイルの管理システムの仕様を代数的に記述し，同期機能の表現方法などを示し，代数的仕様の一種である項書き換え系における効率のよい計算方法について論ずる．

第1章には，文献[1]～[6]として公表した代数的仕様の定式化およびそこで得られた諸性質が示されている．ここでの代数的仕様の理論的枠組は，その簡明さと厳密さから，Goguen らに従う．既存データタイプは具体的な代数（基底代数と呼ぶ）として定義されているとし，

基底代数が代数的仕様の枠組の中に導入されるとき，その公理系は，基底代数で定義されている関数の“定義表”の形で与えられるとする．これは，プログラムとしての実行に対する配慮や，基底部の関数の計算方法には関知しないという階層性の立場から，妥当なものであるといえよう．又，意味ある仕様の公理系は，基底代数を前提とするかぎり，基底代数に対して矛盾しては（すなわち，基底代数の異なる二つの元が公理から等しくなるようなことがあっては）ならないが，それを保証するための十分条件として，Church-Rosserの性質およびそのための既知の十分条件が容易に利用できる．

この章では，このような（基底代数Bを前提とする）代数的仕様，B-仕様，の定式化の下に，基底代数に対する無矛盾性について，基底代数の具体的内容に依存しない形の十分条件，及び“don't care”の関数値の（公理を満たす範囲内での）任意の定め方に対して仕様が無矛盾性を保つためのの十分条件を示し，更に，B-仕様の一種として，順序機械の一般化であり，データ構造やシステムプログラムの記述に有用と考えられるB-順序機械を導入し，その簡約化および抽象的“表”での一般的な実現方法を示している．第2章のファイル管理システムの記述例は，B-順序機械の拡張形として書かれている．

第2章では，文献[7]，[8]として公表したファイル管理システムの順編成ファイルに関する部分の論理レコードレベルでの記述例を示し，そこで生じた機構化の問題について論じている．READ及びWRITEリクエストに対する処理の可分性や同期のための機能と無関係に，順編成ファイル固有の性格から意味が定義できる部分を抽出し，それをSystem-0として表し，System-0に同期

の機能や“プロセス制御”に関する部分を付加したものをFile-systemとして表している．これらの記述については，無矛盾性および“don't care”を除いて値が完全に決まるという準完全性が示されている．後者は，公理に不足がないことを意味する．

File-systemは，より抽象的なSystem-0の論理レコードレベルでの機構化と考えられるが，利用者プロセスから観測可能な部分の機能追加が行われており，両者の関係は，いわゆる“実現”のような“内部の詳細化”のみではない．File-systemとSystem-0との対応については，File-systemにおいて利用者プロセスが利用者バッファを“正しく”使う限り，両者が実効的に等しいことが示されている．

第3章では，文献[9]～[11]に公表した効率のよい計算方法に関する研究結果が述べられている．代数的記述言語の一種であるChurch-Rosserの性質を満たす項書き換え系については，書き換えの“先読み”を必要としないという意味で効率のよい書き換え手順をもつクラスstrongly sequential term rewriting systemsがHuetらによって得られている．この章では，書き換え規則（公理）の左辺に一定の制限を加えた完全順序項書き換え系を導入し，そこでの効率のよい計算方法について述べる．完全順序項書き換え系は，strongly sequential term rewriting systems with constructorsに含まれるが，B-順序機械や，BackusのFP系を含んでいる．

完全順序項書き換え系での計算方法について，まず，同一計算の重複を避けるために項の有向アサイクリックグラフ（dag）表現およびdagの書き換えを導入し，必

須な書き換え場所でのdagの書き換えを用いれば，最小の書き換え回数で値が計算されることを示す．次に，dagの書き換え自体を，適当なデータ構造と関数呼び出しの機構を用いて，効率よく行う方法を示す．又，この種の言語の処理系で一般に問題となる不要になった記憶域の回収について，関数手続の終了時にその関数手続が確保した記憶域を回収することが可能な項書き換え系のクラスを示す．

ここで示す計算方法をstrongly sequential term rewriting systems with constructorsに拡張することはできるが，そのクラスより大きなクラスの項書き換え系をそれと"等価"な完全順序項書き換え系に変換できることと，説明の簡単さから，完全順序項書き換え系について考察を行っている．

関連発表論文

〔1〕 嵩，谷口，杉山，萩原，鈴木，奥井：“プログラム仕様記述の代数的方法について”，電子通信学会技術研究報告，AL78－5（昭和53年5月）．

〔2〕 鈴木，杉山，萩原，谷口，嵩，奥井：“プログラムの仕様とその実現化の代数的記述”，電子通信学会技術研究報告，AL78－46（昭和53年10月）．

〔3〕 奥井，鈴木，杉山，萩原，谷口，嵩：“仕様記述間の対応について”，電子通信学会技術研究報告，AL78－79（昭和54年1月）．

〔4〕 杉山，谷口，嵩：“代数的仕様記述に基づくプログラム設計の一例”，電子通信学会技術研究報告，AL79－13（昭和54年5月）．

〔5〕 杉山，谷口，嵩：“代数的記述言語とその部分言語としての関数的プログラミング言語”，電子通信学会技術研究報告，AL79－99（昭和55年1月）．

〔6〕 杉山，谷口，嵩：“基底代数を前提とする代数的仕様”，電子通信学会論文誌（D），J64－D，4，pp. 324－331（昭和56年4月）．

〔7〕 鈴木，杉山，谷口，嵩：“代数的仕様記述における詳細化－特に抽象的順序機械の場合－”，京都大学数理解析研究所講究録，421，pp. 92－105，（昭和56年3月）．

〔8〕 杉山，奥井，嵩：“順編成ファイル管理システムの代数的記述について－同期機能の表現－”，電子通信学会論文誌（D），（昭和58年2月掲載予定）．

[9] 杉山，鈴木，谷口，嵩：“代数的記述からプログラムへの変換について”，電子通信学会技術研究報告，AL81－12（昭和56年5月）．

[10] 杉山，井上，嵩：“項書き換え系のある標準形への変換”，電子通信学会技術研究報告，AL81－95，（昭和57年1月）．

[11] 杉山，鈴木，谷口，嵩：“あるクラスの項書き換え系の効率のよい実行”，電子通信学会論文誌(D)，J65－D，7，pp. 858－865（昭和57年7月）．

第1章　基底代数を前提とする代数的仕様

1.1　序言

抽象データタイプあるいはプログラムの仕様記述の代数的方法について多くの研究がなされている[1~9]。仕様を書く上で，何らかの方法で既に定義されている一群の関数を，その中で再度定義することなく使用できれば便利であり，仕様に階層性をもたせるなどの点からもその方が望ましいといえよう。ここでは，そのような基本的な関数は，例えば，ブール代数，加減算をもつ整数，あるアルファベット上の系列などといった具体的な“代数”（基底代数と呼ぶ）における基本演算として定義されているとし，それらの基底代数の上で新たな関数を定義する，いわゆる基底代数の拡張の立場で仕様を考える。本論文の理論的枠組は Goguen[1] らに従うが，基本的な考え方は，上述の意味で Guttag[2] に近い。

仕様によって定義しようとする“系”にとって，外から与えられる一次的に意味のあるものは基底代数の元であり，系の外への出力値として最終的に意味をもつのも基底代数の元であると考えられる。このことから，仕様により表わされるものは，その始代数[1]に限定せず，公理による系の入出力の対応（1.3.4の写像 eval で与えられる）が一致するような代数のクラスであると考える。そのクラスのいずれの代数を選ぶかは，“実現”時の裁量にゆだねられる。例えば，“状態の最小化”を考えるならば，終代数[3]を選ばべばよい。

基底代数を前提とする以上，意味ある仕様の公理系は，基底代数に矛盾しては（すなわち，基底代数の異なる元

が公理により等しいとされるようなことがあっては）ならないが，完全である（すなわち，すべての出力値が基底代数の元になるように公理で定められている）必要はない．公理で基底代数のいずれの元とも定められていないような出力値は，（公理系を満たす）代数によって異なってよく，仕様としてはいわゆる“don't care”を表している．

基本関数をいかにして計算するかは，定義すべき系の外の問題であると考えられる．従って，同じ代数的枠組の中へ基底代数を導入するとき，基本関数に関する公理は，引数の値を指定すればその関数値が与えられるという，いわゆる“定義表”の形で与えられているとする．基底代数をこのような形で導入する理由の一つには，公理系が基底代数に対して無矛盾であることの十分条件として，Church-Rosserの性質[10]及びそのための既知の十分条件[10],[11]が容易に利用できることがある．なお，公理系が無矛盾であるか否かは，一般に決定不能である．[2]

本章では，1.2で基礎的な概念を述べ，1.3で基底代数Bをもつ仕様“B-仕様”定式化し，基底代数Bに対する無矛盾性について，Bの具体的内容に依存しない形の議論を行い，1.4で“don't care”の関数値の（公理を満たす範囲内での）任意の定め方に対して，B-仕様が無矛盾性を保つためのの十分条件を示す．最後に，1.5で，B-仕様の一種で，データ構造やシステムプログラムの記述に関連してよく用いられる“B-順序機械”を導入し，その“簡約化”及び抽象的“表”での一般的な実現方法について述べる．

1.2 諸定義

1.2.1 項

整数，ブール値，"スタックの状態" などのような集合の "種類" をソートと呼ぶ．関数 f の値がソート ρ の集合に属するとき，f は（関数値が）ソート ρ の関数であるといい，f の引数のソートが，左から順に，ρ_1，ρ_2，…，ρ_n であるとき，f の引数ソート系列は $\rho_1 \cdot \rho_2 \cdot \cdots \cdot \rho_n$ であるといい，

$$f : \rho_1, \rho_2, \cdots, \rho_n \longrightarrow \rho$$

と書く．引数を全くもたない関数を，特に，定数と呼び，その引数ソート系列を λ（空系列）で表す．

S をソートの集合とするとき，関数値のソート及び各引数ソートが S に属するような関数記号を S - 関数記号と呼ぶ．S - 関数記号の集合 Σ に対し，Σ の部分集合で，引数ソート系列が $\alpha \in S^*$，関数値のソートが $\rho \in S$ であるような関数記号全体の集合を $\Sigma^{\alpha,\rho}$ と書く．又，変数の集合 X に対し，ソート ρ の変数の集合を X^ρ と書く．ソート名，関数記号，変数名は互いに異なった記号とする．

各ソート $\rho \in S$ について，ソート ρ の項の集合 $T_\Sigma^\rho(X)$ を，次の (i)～(iii) を満たす最小の集合と定義する．

(i) $\Sigma^{\lambda,\rho} \subseteq T_\Sigma^\rho(X)$

(ii) $X^\rho \subseteq T_\Sigma^\rho(X)$

(iii) $f \in \Sigma^{\rho_1 \cdots \rho_n, \rho}$ 且つ $t_i \in T_\Sigma^{\rho_i}(X)$ $(1 \leqq i \leqq n)$

$\Rightarrow f(t_1, \cdots, t_n) \in T_\Sigma^\rho(X)$

以下，Σが明らかな場合，$T_\Sigma^s(X)$ の添字Σを省略する．$X=\emptyset$（空集合）のとき，すなわち，変数を含まないソートsの項の集合をT^sと書き，

$$T(X) \triangleq \bigcup_{s\in S} T^s(X)$$

$$T \triangleq \bigcup_{s\in S} T^s$$

と定義する．

項tに対し，tにおける出現の集合 $O(t)$ を，次の(i)及び(ii)を満たす最小の整数系列の集合と定義する．

(i) $\lambda \in O(t)$

(ii) $o \in O(t_i) \Rightarrow i\cdot o \in O(f(t_1, \cdots, t_n))$
$(1 \leqq i \leqq n)$

項t及び出現 $o \in O(t)$ に対し，tの出現oの部分項（t/oと書く）を次のように定義する．

(i) $t/\lambda \triangleq t$

(ii) $f(t_1, \cdots, t_n)/i\cdot o \triangleq t_i/o$
$(n \geqq 1,\ f(t_1, \cdots, t_n) \in T(X),\ 1 \leqq i \leqq n)$

t自身を除くtの部分項を，特に，真部分項と呼ぶ．tの出現oの部分項を同一ソートの項t'で置き換えて得られる項を $t[o \leftarrow t']$ と書く．形式的には次のように定義される．

(i) $t[\lambda \leftarrow t'] \triangleq t'$

(ii) $f(t_1, \cdots, t_n)[i\cdot o \leftarrow t']$

$$\triangleq f(t_1, \cdots, t_{i-1}, t_i[o \leftarrow t'], t_{i+1}, \cdots, t_n)$$
$(1 \leqq i \leqq n)$

代入 σ とは X から $T(X)$ への写像である．但し，

$$x \in X^{\Delta} \Rightarrow \sigma(x) \in T^{\Delta}(X)$$

代入の定義域を拡張し，σ を次のような $T(X)$ から $T(X)$ への写像として用いる．$f(t_1, \cdots, t_n) \in T(X)$ に対し，

$$\sigma(f(t_1, \cdots, t_n)) = f(\sigma(t_1), \cdots, \sigma(t_n))$$

$x_1, \cdots, x_n$ を互いに異なる変数とするとき，$T(\{x_1, \cdots, x_n\})$ に属する項 t を $\tau(x_1, \cdots, x_n)$ の形で表すことがある．このとき，$\sigma(x_i) = t_i$ $(1 \leqq i \leqq n)$ なる代入 σ に対して，

$$\tau(t_1, \cdots, t_n) \triangleq \sigma(\tau(x_1, \cdots, x_n))$$

と定義する．

1.2.2 仕様

仕様 $\mathcal{S}$ を3字組 $(S, \Sigma, \mathcal{E})$ で定義する．ここで，S はソートの有限集合，Σ は S-関数記号の可算集合，$\mathcal{E}$ は同一ソートの（一般に変数を含んだ）項の順序対の可算集合である．$\mathcal{E}$ に属する順序対を公理，$\mathcal{E}$ 全体を公理系と呼ぶ．公理を $l == r$ $(l, r \in T(X))$ で表し，l をその公理の左辺，r を右辺と呼ぶ．Σ，$\mathcal{E}$ は有限集合でなくてもよいとする．

公理系 $\mathcal{E}$ の下での $T(X)$ 上の関係 "$\overset{\mathcal{E}}{\rightarrow}$" を次のように定義する．二つの項 t，$t' \in T(X)$ に対し，公理 $l == r \in \mathcal{E}$，代入 σ 及び t の出現 $o \in O(t)$ の部分項 t/o があって，

$$t/o = \sigma(l) \text{ 且つ } t' = t[o \leftarrow \sigma(r)]$$

が成り立つとき，且つそのときに限り

$$t \xrightarrow[\mathcal{E}]{} t'$$

と書く．関係 $\xrightarrow[\mathcal{E}]{}$ の反射推移閉包を $\xrightarrow[\mathcal{E}]{*}$ と書き，反射対称推移閉包を $\underset{\mathcal{E}}{\equiv}$ と書く．すなわち，

$$t \underset{\mathcal{E}}{\equiv} t' \iff t = t' \text{ または } t \underset{\mathcal{E}}{\leftrightarrow} t'$$

と定義する．以下，公理系 $\mathcal{E}$ が明らかなとき，関係 $\xrightarrow[\mathcal{E}]{}$，$\xrightarrow[\mathcal{E}]{*}$，$\underset{\mathcal{E}}{\equiv}$，$\xrightarrow[\mathcal{E}]{\equiv}$ の $\mathcal{E}$ を省略し，それぞれ $\rightarrow$，$\xrightarrow{*}$，$\equiv$，$\xrightarrow{\equiv}$ と書く．

関係 $\equiv$ は等価関係であり，これを $\mathcal{E}$ の生成する等価関係という．等価関係 $\equiv$ を含み，すなわち，

$$t \equiv t' \Rightarrow t \; R \; t'$$

且つ次の条件 SP を満たす T 上の等価関係 R を $\mathcal{E}$-合同関係という．

〔条件 SP〕 任意の $f : \rho_1, \cdots, \rho_n \to \rho$ 及び $t_i, t_i' \in T^{\rho_i}$ $(1 \leqq i \leqq n)$ に対し，

$$\bigwedge_{i=1}^{n} t_i \; R \; t_i' \Rightarrow f(t_1, \cdots, t_n) \; R \; f(t_1', \cdots, t_n')$$

定義より，$\equiv$ は $\mathcal{E}$-合同関係である．

1.2.3 仕様を満たす代数

S-関数記号の集合 Σ に対し，次のような集合族と関数の集合の対 $A = (\{C_A^\rho \mid \rho \in S\}, \{f^A \mid f \in \Sigma\})$ を Σ-代数と呼ぶ．ここで，C_A^ρ はソート ρ に割り当てられた具体的な（可算）集合（ρ の台と呼ぶ）であり，f^A は，関数記号 $f : \rho_1, \cdots, \rho_n \to \rho$ に割り当てられた，定義域が $C_A^{\rho_1} \times \cdots \times C_A^{\rho_n}$ で，関数値が C_A^ρ に属する

関数である．項 $t \in T$ に対して，代数 A における t の値 t_A を次のように定義する．

(i) t が定数記号（のみからなる項）の場合，t_A は定数関数 t^A の値であり，

(ii) $t = f(t_1, \cdots, t_n)$ のとき，t_A は関数値 $f^A(t_{1A}, \cdots, t_{nA})$ である．

仕様 $\mathcal{S} = (S, \Sigma, \mathcal{E})$ に対して Σ - 代数 A が

$$t \equiv t' \Rightarrow t_A = t'_A$$

を満たすとき，A は仕様 $\mathcal{S}$（又は公理系 $\mathcal{E}$）を満たすという．このとき，関係 R を，

$$t \; R \; t' \Leftrightarrow t_A = t'_A$$

と定義すると，関係 R は $\mathcal{E}$ - 合同関係であり，A に対応する（T 上の）$\mathcal{E}$ - 合同関係と呼ばれる．逆に，T 上の $\mathcal{E}$ - 合同関係 R が与えられたとき，上記の式が成り立つような $\mathcal{S}$ を満たす Σ - 代数 A が存在し，同形の範囲で一意に定まる．A は R に対応する代数（の一つ）と呼ばれる．

1.2.4 Church - Rosser の性質

公理系 $\mathcal{E}$ において，任意の $t, t' \in T(X)$ に対し，

$$t \equiv t' \Leftrightarrow \exists t_0 \in T(X) \left(t \xrightarrow{*} t_0 \text{ 且つ } t' \xrightarrow{*} t_0 \right)$$

が成り立つとき，且つそのときに限り公理系 $\mathcal{E}$ は Church-Rosser の性質をもつという．

$\mathcal{E}_1$, $\mathcal{E}_2$ を $T(X)$ 上の公理の集合とする．任意の t, t_1, $t_2 \in T(X)$ に対して，

$$t \xrightarrow[\mathcal{E}_1]{*} t_1 \quad \text{且つ} \quad t \xrightarrow[\mathcal{E}_2]{*} t_2 \quad \Rightarrow$$

$$\exists t' \in T(X) \left(t_1 \xrightarrow[\mathcal{E}_2]{*} t' \quad \text{且つ} \quad t_2 \xrightarrow[\mathcal{E}_1]{*} t' \right)$$

が成り立つとき，且つそのときのみ $\mathcal{E}_1$ と $\mathcal{E}_2$ は可換であるという．可換性と Church-Rosser の性質に関して次の結果が知られている．(10)

〔補題 1.1〕（Rosen） $\mathcal{E}_1, \mathcal{E}_2, \cdots$ を $T(X)$ 上の公理の集合とする．

(i) $\mathcal{E}_1, \mathcal{E}_2, \cdots$ がそれぞれ Church-Rosser の性質をもち，且つすべての i, j（但し $i \neq j$）について $\mathcal{E}_i$ と $\mathcal{E}_j$ が可換であれば，$\bigcup_i \mathcal{E}_i$ も Church-Rosser の性質をもつ．

(ii) すべての $t, t_1, t_2 \in T(X)$ について，

$$t \xrightarrow[\mathcal{E}_1]{} t_1 \quad \text{且つ} \quad t \xrightarrow[\mathcal{E}_2]{} t_2 \quad \Rightarrow$$

$$\exists t' \in T(X) \left(t_1 \xrightarrow[\mathcal{E}_2]{*} t' \quad \text{且つ} \quad t_2 \xrightarrow[\mathcal{E}_1]{=} t' \right)$$

ならば，$\mathcal{E}_1$ と $\mathcal{E}_2$ は可換である． □

二つの項 $\tau(x_1, \cdots, x_n)$，$\tau'(y_1, \cdots, y_{n'}) \in T(X)$ に対して，適当な項 $t_i (1 \leqq i \leqq n)$，$t'_j (1 \leqq j \leqq n')$ が存在して，$t \equiv \tau(t_1, \cdots, t_n)$，$t' \equiv \tau'(t'_1, \cdots, t'_{n'})$ の一方が他方の部分項となるとき（但し，t がいずれかの t'_i の部分項，または t' がいずれかの t_i の部分項の場合を除く），t と t' は重なり得るという．

〔無曖昧性〕 公理系 $\mathcal{E}$ の各公理の左辺が，他のどの公理の左辺とも，又，自分自身のどの真部分項とも重なり得ないとき，$\mathcal{E}$ は無曖昧であるという．

〔左辺線形性〕　公理系 $\mathcal{E}$ のどの公理の左辺にも同じ変数が複数回現れることがないとき，$\mathcal{E}$ は左辺線形であるという．

〔条件 VAR〕　公理の右辺に現れる変数は必ず左辺に現れている．

条件 VAR を満たし，無曖昧性および左辺線形性をもつ公理系が Church-Rosser の性質をもつことは既に知られている．[10] 可換性に関して次の補題が成りたつ．

〔補題 1.2〕　公理の集合 $\mathcal{E}_1$，$\mathcal{E}_2$ は，条件 VAR を満たし，左辺線形であるとする．このとき，$\mathcal{E}_1$ に属する各公理の左辺が $\mathcal{E}_2$ のどの公理の左辺とも重なり得なければ，$\mathcal{E}_1$ と $\mathcal{E}_2$ は可換である．

（証明）以下の証明は Rosen の証明技法[10] を応用したものである．まず，関係 $\overset{\rightarrow}{\mathcal{E}_i}$ $(i = 1, 2)$ を導入し，関係 $\overset{\rightarrow}{\mathcal{E}_i}$ に関して，$\mathcal{E}_1$ と $\mathcal{E}_2$ が可換であることを示す．項 t に対し，公理 $\ell == r \in \mathcal{E}_i$，代入 σ，出現 $o_j \in O(t)$ $(1 \leqq j \leqq m)$ が存在して，$t/o_j = \sigma(\ell)$ とする．このとき，

$$t' \triangleq t[o_1 \leftarrow \sigma(r)][o_2 \leftarrow \sigma(r)] \cdots [o_m \leftarrow \sigma(r)]$$

と定義すると，t' は，$o_1, \cdots, o_m$ の順序の選び方に依らず，一意に定まる．t，t' についてそのような関係が成りたつとき，且つそのときに限り $t \overset{\rightarrow}{\mathcal{E}_i} t'$ と書く．

補題 1.2 の前提条件より，

$$t \overset{\rightarrow}{\mathcal{E}_1} t_1 \text{ 且つ } t \overset{\rightarrow}{\mathcal{E}_2} t_2 \Rightarrow$$

$$\exists t' \ (t_1 \overset{\rightarrow}{\mathcal{E}_2} t' \text{ 且つ } t_2 \overset{\rightarrow}{\mathcal{E}_1} t')$$

が成りえつ．従って，補題1.1(ii)より，$\overset{*}{\underset{\varepsilon_i}{\rightarrow}}$ に関して ε_1 と ε_2 は可換である．すなわち，

$$t \overset{*}{\underset{\varepsilon_1}{\rightarrow}} t_1 \text{ 且つ } t \overset{*}{\underset{\varepsilon_2}{\rightarrow}} t_2 \Rightarrow$$

$$\exists t' \left(t_1 \overset{*}{\underset{\varepsilon_2}{\rightarrow}} t' \text{ 且つ } t_2 \overset{*}{\underset{\varepsilon_1}{\rightarrow}} t' \right)$$

一方，$\overset{*}{\underset{\varepsilon_i}{\rightarrow}}$ $(i = 1, 2)$ の定義より，

$$t \overset{*}{\underset{\varepsilon_i}{\rightarrow}} t' \Leftrightarrow t \overset{*}{\underset{\varepsilon_i}{\rightarrow}} t'$$

であるから，ε_1 と ε_2 は（$\vec{\varepsilon}_1$，$\vec{\varepsilon}_2$ に関して）可換である．（証明終）

1.3 基底代数Bを前提とする仕様

1.3.1 B-仕様と無矛盾性

基底代数Bをもつ仕様を導入するために，まずΣ-代数Bに対応する仕様 spec(B) を導入する．便宜上，代数Bにおける名前eの元を，同じ記号eで表す．

[定義1.1] Σを S-関数記号の集合とする．Σ-代数Bに対して，仕様 spec(B) = $(S_B, \Sigma_B, \mathcal{E}_B)$ を次のように定義する．

(i) $S_B \triangleq S$

(ii) $\Sigma_B \triangleq (\bigcup_{s \in S_B,\ \alpha \in S_B^* - \{\lambda\}} \Sigma^{\alpha, s}) \cup (\bigcup_{s \in S_B} \{e \mid e \in C_B^s\})$

(iii) $\mathcal{E}_B \triangleq \{ f(e_1, \cdots, e_n) == e \mid f \in \Sigma_B^{s_1 \cdots s_n, s},\ e_i \in C_B^{s_i}\ (1 \leqq i \leqq n),\ n \geq 1,\ e$ は $f^B(e_1, \cdots, e_n)$ の値（C_B^s の元）の名前$\}$ □

定義から，代数Bは spec(B) を満たす．二つの仕様 $\mathcal{D} = (S, \Sigma, \mathcal{E})$，$\mathcal{D}' = (S', \Sigma', \mathcal{E}')$ に対して，$S \subseteq S'$，$\Sigma \subseteq \Sigma'$ 且つ $\mathcal{E} \subseteq \mathcal{E}'$ であるとき，$\mathcal{D}$ を $\mathcal{D}'$ の部分仕様という．

[定義1.2] 代数Bに対し，仕様 $\mathcal{D}$ が spec(B) を部分仕様として含むとき，特に，$\mathcal{D}$ をB-仕様と呼ぶ． □

B-仕様において，S_B に属するソートを，基底代数Bのソートであるという意味で，B-ソート，残りの新たに導入されたソートをN-ソートと呼ぶ．N-ソートはB-仕様によって定義しようとするデータ構造，シス

テムの状態集合などに対応する．代数 B のえと一対一に対応する Σ_B の定数記号は代数 B のえを表し，B-定数と呼ばれる．B-定数全体の集合を Σ_B^λ と書く．B-定数は，一般に，可算無限個存在する．えを表すものとして，（定数記号の代りに）幾つかの（通常有限個の）関数記号（構成子と呼ばれる）からなる項（構成する項と呼ばれる）を用いることもできるが，本章では，簡単のため，定数記号を使う．変数を含まない B-ソートの項の集合を T_B と書く．すなわち，

$$T_B \triangleq \bigcup_{s \in S_B} T^s$$

［定義1.3］仕様 $\mathcal{D} = (S, \Sigma, \mathcal{E})$，項の集合 $T_C \subseteq T_\Sigma$ に対して，任意の二項 $t_1, t_2 \in T_C$ が

$$t_1 \equiv t_2 \iff t_1 = t_2$$

を満たすとき，$\mathcal{D}$（又は $\mathcal{E}$）は T_C-無矛盾であるという．特に，$\mathcal{D}$ が B-仕様で，$T_C = \Sigma_B^\lambda$ であるとき，$\mathcal{D}$ は B-無矛盾であるという† □

本章では，B-仕様に対する無矛盾性としては B-無矛盾のみを考える．又，B-仕様としては，公理系が

(B_1) 左辺線形で，

(B_2) 条件 VAR を満たし，且つ

(B_3) 左辺が B-定数のみからなる項であるような公理を含まない

ものみを考える．

項 $t \in T$ に対して，$t \to t'$ なる項 t' が存在しないとき，t を標準項と呼ぶ．$t \equiv t_0$ なる標準項 t_0 が存在

---

† B-無矛盾は文献(2) a consistent に対応する．

するとき，t は標準項 t_0 をもつという．

〔補題 1.3〕 公理系が Church-Rosser の性質をもつなら，標準項 t_0 をもつ項 t に対して

$$t \xrightarrow{*} t_0$$

（証明） 前提条件より，$t \equiv t_0$．Church-Rosser の性質より，

$$\exists t' \ (\ t \xrightarrow{*} t' \text{ 且つ } t_0 \xrightarrow{*} t'\ )$$

t_0 が標準項であることから，$t_0 = t'$． □

〔補題 1.4〕 公理系が Church-Rosser の性質をもてば，任意の項 $t \in T$ に対し，t の標準項は，もし存在するなら，一意である．

（証明） t が標準項 t_1，t_2 をもつとする．$t_1 \equiv t \equiv t_2$ 故，Church-Rosser の性質より，

$$\exists t' \ (\ t_1 \xrightarrow{*} t' \text{ 且つ } t_2 \xrightarrow{*} t'\ )$$

t_1，t_2 が標準項であることから，$t_1 = t' = t_2$． □

B-定数を左辺にもつ公理が存在しないという仮定から，B-定数は標準項である．公理系が Church-Rosser の性質をもつとき，項 $t \in T_B$ の値，すなわち，$t \equiv b$ なる $b \in \Sigma_\beta^\wedge$ が存在するとき，b を求めるには，補題 1.3 より，公理を“書き換え規則”とみなして項を書き換えていけばよいことが分かる．又，補題 1.4 より，異なる二つの B-定数 b_1，b_2 が $b_1 \equiv b_2$ となることはない．従って，次の定理が成り立つ．

〔定理 1.1〕 公理系 $\mathcal{E}$ が Church-Rosser の性質をもてば，$\mathcal{E}$ は B-充分的である． □

基底代数の公理系 $\mathcal{E}_B$ の具体的内容にかかわらず，公理系 $\mathcal{E}$ が Church-Rosser の性質をもつための十分条件を次に示す．

［定理1.2］ B-仕様 $\mathcal{D} = (S, \Sigma, \mathcal{E})$，$spec(B) = (S_B, \Sigma_B, \mathcal{E}_B)$ において，次の条件(i)及び(ii)が成立すれば，$\mathcal{E}$ は Church-Rosser の性質をもつ．

(i) $\mathcal{E} - \mathcal{E}_B$ が Church-Rosser の性質をもつ．

(ii) $\mathcal{E} - \mathcal{E}_B$ に属する公理の左辺が $f(u_1, \cdots, u_n)$ （$f \in \Sigma_B$，u_i $(1 \leq i \leq n)$ は変数または B-定数）の形の部分項を含まない．

（証明） $\mathcal{E}_B$ が Church-Rosser であることは，$\mathcal{E}_B$ が無曖昧であることから明らか．定理の条件(i)より，$\mathcal{E} - \mathcal{E}_B$ もまた Church-Rosser である．更に，条件(ii)より，$\mathcal{E}_B$ の右左辺は $\mathcal{E} - \mathcal{E}_B$ のどの左辺とも重なり得ず，従って補題1.2より，$\mathcal{E} - \mathcal{E}_B$ と $\mathcal{E}_B$ は可換である．故に，補題1.1(i)より，$\mathcal{E}$ は Church-Rosser である． □

1.3.2 $\mathcal{E}_B$ の拡張

基底代数Bは，ブール代数 Boolean を含んでいるとし，そのソートを bool と書く．$\mathcal{E}_B$ の形より，基本関数記号 $f \in \Sigma_B$ に関する公理は，その引数がすべてB-定数に書き換えられなければ適用できない．“条件文”に相当する関数 $if_s : s, s \to s$ に関する公理は，ソート s のすべてのB-定数 b, b' に対して，

$$if(true, b, b') == b \quad \in \mathcal{E}_B$$

$$if(false, b, b') == b' \quad \in \mathcal{E}_B$$

である．従って，一種の“定理”として，

$$if_\rho(true,\ x,\ y) == x \qquad (1.1)$$

$$if_\rho(false,\ x,\ y) == y \qquad (1.2)$$

（但し，$x, y \in X^\rho$）が成りたつ．変数 x, y が表す任意の項に対してこれらの書き換えを適用したい場合は，式(1.1),(1.2)を公理とする必要がある．式(1.1),(1.2)の公理を追加しても公理系がChurch-Rosser性を保つためには，$\mathcal{E}-\mathcal{E}_B$ が定理1.2の条件を満たし，式(1.1),(1.2)の左辺と重なり得る左辺をもつ公理が $\mathcal{E}-\mathcal{E}_B$ に存在しなければよい．

1.3.3 B-仕様の例

B-仕様の例として，検索項目が整数（ソート名をintと書く）で，内容が文字列（string）の抽象的な表の仕様を図1.1に示す．N-ソートとして，表がとり得る“状態”の集合 table が新たに導入されている．関数には，“初期状態”を表す定数 initial-table，“状態”を遷移させるものとして，項目を追加・変更する put，および削除する delete と，“状態”に関する情報を出力させるものとして，項目が登録されているか否かの述語 in，表を引く access がある．

B-ソート bool，int について，“else節”のある条件文と無い条件文（通常のプログラミング言語のIF文の形で書かれている）があり，1.3.2 で述べた公理の拡張が行われているものとする．又，条件節中に用いられている記号“=”は，二つの整数が同じであるか否かを判定する述語を表しており，これは，基本関数とし

```
SPEC Table;

 BASE Integer, Boolean, String;
 SORT table;

 OP initial-table:                -> table,
    put    : int, string, table -> table,
    delete: int,          table -> table,

    in     : int,          table -> bool,
    access: int,          table -> string;

 VAR t      SORT table,
     i, i´  SORT int,
     s      SORT string;

 Ax1: in(i, initial-table) == false;
 Ax2: in(i, put(i´, s, t)) ==
    if i = i´ then true else in(i, t);
 Ax3: in(i, delete(i´, t)) ==
    if i = i´ then false else in(i, t);

 Ax4: access(i, put(i´, s, t)) ==
    if i = i´ then s else access(i, t);
 Ax5: access(i, delete(i´, t)) ==

    if i ≠ i´ then access(i, t);

END;
```

図 1.1　抽象的"表"の記述例

て代数 Integer に含まれているものとする．

例として挙げた表の仕様は，検索項目を整数，内容を文字列としたが，これらに関して公理で利用していることは整数に関する述語“＝”のみである．従って，図 1.1 の抽象的表における基底代数を，ブール代数および二つのえが同じか否かの述語をもつ集合を含む任意の代数とすることができ，“パラメータ化”することができる．

1.3.4 B－仕様を満たす代数

〔定義 1.4〕 B－仕様 $\mathcal{D}$ に対し，（$\mathcal{D}$ を単なる仕様とみたときの）$\mathcal{D}$ を満たす代数 A において，B－定数が互いに異なるえに割り当てられているとき，A を B－仕様としての $\mathcal{D}$ を満たす代数という． □

B－仕様としての $\mathcal{D}$ を満たす代数が常に存在するとは限らないが，もし存在するならば，明らかに，$\mathcal{D}$ は B－無矛盾である．逆に，$\mathcal{D}$ が B－無矛盾ならば，$\mathcal{D}$ の始代数では，互いに異なるえが B－定数に割り当てられているから，B－仕様としての $\mathcal{D}$ を満たす．従って，次の定理が成り立つ．

〔定理 1.3〕 B－仕様としての $\mathcal{D}$ を満たす代数が存在するための必要十分条件は，$\mathcal{D}$ が B－無矛盾であることである． □

以下，$\mathcal{D}$ を満たす代数としては，B－仕様としての $\mathcal{D}$ を満たす代数のみを考える．又，$\mathcal{D}$ を満たす各代数は B と同形の代数を部分代数として含むが，この部分代数のえと，対応する B のえを同一視する．

B-無矛盾な B-仕様 $\mathcal{D} = (S, \Sigma, E)$ を満たす代数のクラス $\mathcal{A}$ に属する代数の性格を表す写像 $eval_E$ を導入する．記号 $\perp$ を，他のどの記号とも異なる新しい記号とする．

$$T_B \triangleq \bigcup_{s \in S_B} T^s, \qquad \Sigma_B^\lambda \triangleq \bigcup_{s \in S_B} \Sigma_B^{\lambda, s}$$

と定義する．

[定義 1.5] 写像 $eval_E : T_B \to \Sigma_B^\lambda \cup \{\perp\}$ を次のように定める．各項 $t \in T_B$ に対して，ある B-定数 b が存在して $t \equiv_E b$ のとき，$eval_E(t) \triangleq b$，そうでないとき，$eval_E(t) \triangleq \perp$． □

以下，E が明らかなとき，$eval_E$ の添字 E を省略する．$eval(t) = b$ なる項 t については，$t \equiv b$ 故，すべての代数 $A \in \mathcal{A}$ における t の値 t_A は b_A であるが，$eval(t) = \perp$ なる t については，($\mathcal{A}$ に属する) 代数によって異なってよい．すなわち，t の値はいわゆる "don't care" と考えられる．なお，B-ソートのすべての項 t に対して $eval(t)$ が B-定数であるとき，その公理系は完全† であるという．

図 1.1 の記述例では，ソート table のすべての項 t，ソート int のすべての B-定数 i について $eval(in(i, t)) \neq \perp$ であるが，$access(i, t)$ については，$in(i, t) \equiv false$ ならば，$eval(access(i, t)) = \perp$．抽象的表を "実現" するとき，これら "don't care" の値を適当に選ぶことができる．

---

† 文献(2) a sufficiently complete に対応する．

1.4 関数の未定義域の解消

B-無矛盾な B-仕様 $\mathcal{D}$ において，$eval(t) = \perp$ であるような t の値として，代数 B のえを割り当てても，又，B を拡張し，新たに導入した元（例えば，"禁止入力"に対するメッセージに相当する値）を割り当てても，$\mathcal{D}$ を満たす代数を作ることができる場合がある．このように，$eval(t) = \perp$ なる項 t に（一般には拡張後の）B-定数を，公理を満たすように対応づけることを未定義域の解消と呼ぶ．ここでは，任意の未定義域の解消に対して，$\mathcal{D}$（又は拡張後の $\mathcal{D}$）を満たす代数が存在するための十分条件を与える．まず，未定義域の解消 $\rho_\mathcal{E}$ を定義する．この定義では，便宜上，$eval(t) \neq \perp$ なる項 t に対しても $\rho_\mathcal{E}(t) = eval(t)$ となるように $\rho_\mathcal{E}$ を拡張している．

［定義1.6］ B-無矛盾な公理系 $\mathcal{E}$ に対し，その（一つの）未定義域の解消とは，次の (i)～(iii) を満たす写像 $\rho_\mathcal{E} : T_B \to \Sigma_B^\wedge$ をいう．

(i) $b \in \Sigma_B^\wedge \Rightarrow \rho_\mathcal{E}(b) = b$

(ii) $t \underset{\mathcal{E}}{\equiv} t' \in T_B \Rightarrow \rho_\mathcal{E}(t) = \rho_\mathcal{E}(t')$

(iii) $t/\sigma = \tilde{t}$, $t, \tilde{t} \in T_B$, $\theta \in O(t)$
$\Rightarrow \rho_\mathcal{E}(t) = \rho_\mathcal{E}(t[\sigma \leftarrow \rho(\tilde{t})])$ □

以下，$\mathcal{E}$ が明らかな場合，$\rho_\mathcal{E}$ の添字 $\mathcal{E}$ を省略する．

［定義1.7］ ρ を，B-無矛盾な B-仕様 $\mathcal{D}$ における一つの未定義域の解消とする．$\mathcal{D}$ を満たす代数 A が，すべての B-ソートの項 $t \in T_B$ に対して $t_A = \rho(t)_A$ を満たすとき，特に，A を ρ に従う代数という． □

〔補題1.5〕 R_1, R_2 を ε-合同関係とするとき，$R_1 \cup R_2$ の反射対称推移閉包 R もまた ε-合同関係である．

（証明） R の作り方より，R は等価関係であり，明らかに $\equiv$ を含む．従って，R が条件 SP を満たすことを示せば十分である．$t_i \; R \; t_i'$ $(1 \leqq i \leqq n)$ とする．

$$t_i = t_{i1} \; R_{i_1} \; t_{i2} \; R_{i_2} \cdots R_{i_{m-1}} \; t_{im} = t_i'$$
$$(i_j = 1 \text{ または } 2, \; 1 \leqq j \leqq m-1)$$

であり，R_1, R_2 が ε-合同関係であることから，

$$\begin{aligned} f(t_{11}, t_2, \cdots, t_n) \; & R_{i_1} \; f(t_{12}, t_2, \cdots, t_n) \\ & R_{i_2} \; f(t_{13}, t_2, \cdots, t_n) \\ & \vdots \\ & R_{i_{m-1}} \; f(t_{1m}, t_2, \cdots, t_n) \end{aligned}$$

従って，$f(t_1, t_2, \cdots, t_n) \; R \; f(t_1', t_2, \cdots, t_n)$．以下，$i = 2, \cdots, n$ について同様の考察を繰り返せば，

$$f(t_1, t_2, \cdots, t_n) \; R \; f(t_1', t_2', \cdots, t_n')$$

が得られる．故に，R は条件 SP を満たす． □

一般に，指定された P に対し，P に従う代数が存在するとは限らない．そのような代数が存在するとして，R_1, R_2 を，それぞれ，P に従う代数 A_1, A_2 に対応する ε-合同関係とする．このとき，補題1.5より，$R_1 \cup R_2$ の反射対称推移閉包 R は ε-合同関係であり，従って，R に対応する代数 A が存在する．又，定義より，B-ソートの項に関しては，R_1, R_2, R は（P に従って）同じであり，従って，A は P に従う．以上から，P に従う代数が一つでも存在すれば，対応する ε-合同関係が最大の

（ρ に従う）代数が存在する．その代数は，ρ に従う任意の代数からその上への準同形写像が存在する（いわゆる終代数）という意味で，ρ に従う最簡な代数である．以下，ρ に従う代数が存在するための十分条件について考える．

$eval(t) = \bot$, $t \in T_B$ なる項 t を $\rho(t)$ に書き換える公理を追加した公理系 $\mathcal{E}_\rho \triangleq \mathcal{E} \cup \mathcal{E}_\rho^\bot$ を考える．ここで，

$$\mathcal{E}_\rho^\bot \triangleq \{ t == \rho(t) \mid eval(t) = \bot,\ t \in T_B \}$$

ρ の定義と $\mathcal{E}_\rho$ の作りかたより，任意の項 $t \in T_B$ に対して，$t \equiv_{\mathcal{E}_\rho} \rho(t)$．従って，$\mathcal{E}_\rho$ が B－定有限ならば，B－仕様 $(S, \Sigma, \mathcal{E}_\rho)$ を満たす代数 A が存在し，A は ρ に従う．以上から，次の補題が成りたつ．

［補題1.6］ $\mathcal{E}_\rho$ が B－定有限ならば，ρ に従う代数が存在する． □

［定理1.4］ B－仕様 $\mathcal{D} = (S, \Sigma, \mathcal{E})$ において，

(i) $\mathcal{E}$ が Church-Rosser の性質をもち，且つ，

(ii) 公理の左辺の真部分項で，ソートが B－ソートの項は B－定数か変数のみである

ならば，$\mathcal{D}$ の任意の未定義域の解消 ρ に対して，ρ に従う代数が存在する．

（証明）　補題1.6より，$\mathcal{E}_\rho$ が Church-Rosser であることを示せば十分である．定義1.6 (iii) より，$\mathcal{E}_\rho^{\perp}$ が Church-Rosser であることは明らか．従って，$\mathcal{E}$ と $\mathcal{E}_\rho^{\perp}$ が可換であるための補題 1.1(ii) の十分条件：任意の B-ソートの項 $t, t_1, t_2 \in T_B$ に対し，

$$t \xrightarrow{\mathcal{E}} t_1 \quad \text{且つ} \quad t \xrightarrow{\mathcal{E}_\rho^{\perp}} t_2 \quad \Rightarrow$$

$$\exists t' \in T_B \left(t_1 \xrightarrow{*}_{\mathcal{E}_\rho^{\perp}} t' \quad \text{且つ} \quad t_2 \xrightarrow{=}_{\mathcal{E}} t' \right) \tag{1.3}$$

が成りたてば，補題 1.1 より，$\mathcal{E}_\rho$ も Church-Rosser である．以下，式(1.3)が成立することを示す．$t \xrightarrow{\mathcal{E}} t_1$ において，t の部分項 $\tilde{t}_1 \triangleq \tau_\ell(u_1, \cdots, u_n)$ が $\tilde{t}_1' \triangleq \tau_r(u_1, \cdots, u_n)$ で置き換えられ，$t \xrightarrow{\mathcal{E}_\rho^{\perp}} t_2$ において，t の部分項 $\tilde{t}_2$ が $\rho(t_2)$ で置き換えられたとする．$\tilde{t}_1$ と $\tilde{t}_2$ の関係により，(イ)～(ハ) の場合に分かれる．

(イ)　$\tilde{t}_1$ が $\tilde{t}_2$ の部分項．$\tilde{t}_2$ において，その部分項 $\tilde{t}_1$ を $\tilde{t}_1'$ で置き換えた項を $\tilde{t}_2''$ と書くと（図 1.2 参照），($\tilde{t}_2 \to \tilde{t}_2''$ 故) $val(\tilde{t}_2'') = val(t_2) = \perp$ であるから，$\tilde{t}_2'' = \rho(\tilde{t}_2'') \in \mathcal{E}_\rho^{\perp}$．定義 1.6 (ii) より，$\rho(\tilde{t}_2) = \rho(\tilde{t}_2'')$ 故，式(1.3)が成立する．

(ロ)　$\tilde{t}_2$ が $\tilde{t}_1$ の真部分項．定理の条件 (ii) より，$\tilde{t}_2$ はいずれかの u_i の部分項である．u_i において，その部分項 $\tilde{t}_2$ を $\rho(\tilde{t}_2)$ で置き換えた項を u_i' と書くと（図 1.3 参照），明らかに，$\tau_\ell(\cdots, u_i', \cdots) \xrightarrow{\mathcal{E}} \tau_r(\cdots, u_i', \cdots)$ 且つ $\tau_r(\cdots, u_i, \cdots) \xrightarrow{*}_{\mathcal{E}_\rho^{\perp}} \tau_r(\cdots, u_i', \cdots)$ が成りたち，式(1.3)が成立する．

(ハ)　$\tilde{t}_1$, $\tilde{t}_2$ のいずれも他の部分項ではない．この場合，式(1.3)が成立することは明らか．　□

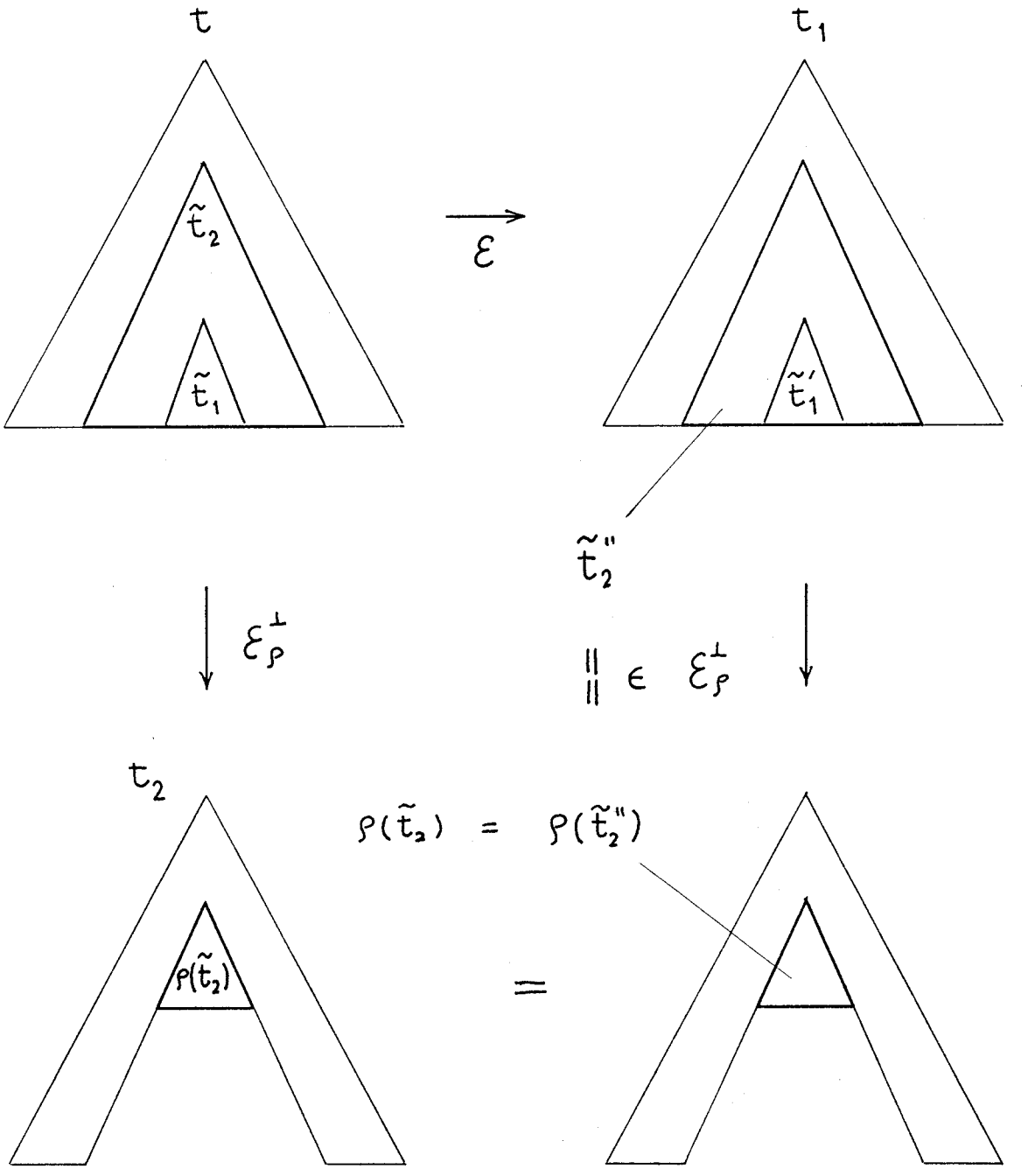

図 1.2　定理 1.4 証明中の関係.
— $\tilde{t}_1$ が $\tilde{t}_2$ の部分項の場合 —

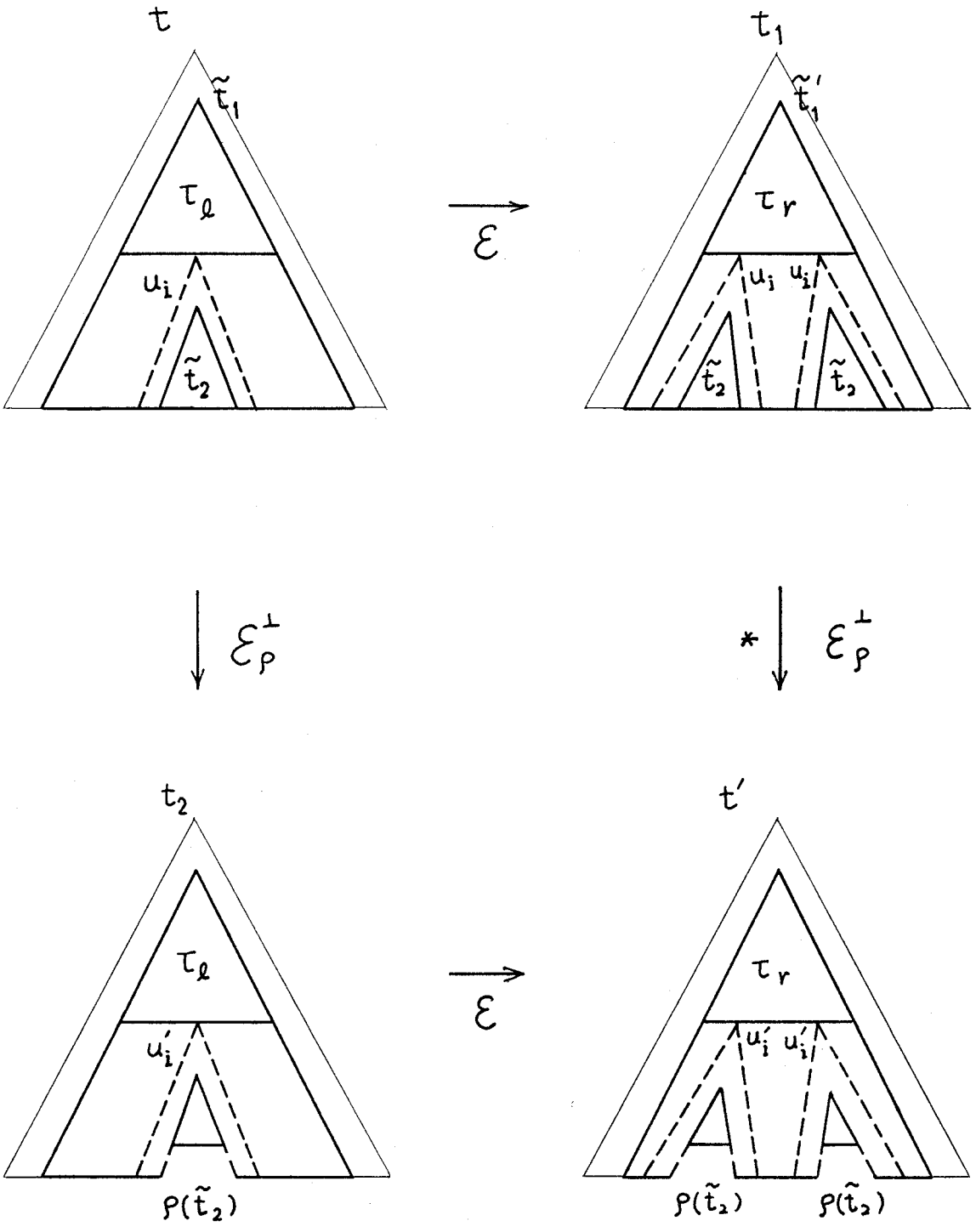

図 1.3　定理 1.4 証明中の関係
— $\tilde{t}_2$ が $\tilde{t}_1$ の真部分項の場合 —

1.5 B-順序機械

順序機械の一般化であり，データ構造とかシステムプログラムの記述などに用いられるB-仕様の部分クラスを定義する．

〔定義 1.8〕 B-仕様 $\mathcal{D}=(S, \Sigma, \mathcal{E})$ が次の条件(i)～(iv)を満たすとき，$\mathcal{D}$を，特に，B-順序機械と呼ぶ．

(i) N-ソートは一つ（それを ρ_N と書く）．

(ii) 引数にソート ρ_N を二つ以上もつ関数記号は存在しない．便宜上，本節では，ρ_N を引数にもつ関数記号の引数ソート列を $\rho_1 \cdots \rho_n$ とするとき，左端の ρ_1 が ρ_N であるとする．

(iii) $x_1, x_2, \cdots$ を互いに異なる変数とする．$\mathcal{E}-\mathcal{E}_B$ に属する公理の左辺は，

(1) $f(x_1, \cdots, x_n)$, $f \in \Sigma - \Sigma_B$, のは

(2) $f(g(x_1, \cdots, x_\ell), x_{\ell+1}, \cdots)$, $f \in \Sigma^{\rho_N \cdot \beta, \rho}$, $g \in \Sigma^{\alpha, \rho_N}$, $\ell \geq 0$, $\rho \in S$, $\alpha \in S^*$, $\beta \in S_B^*$

のいずれかの形をしている．

(iv) 同じ左辺の公理は複数個存在しない． □

ソート ρ_N は状態集合に，B-定数は入出力記号に，ソート ρ_N の関数（N-関数と呼ぶ）は状態遷移関数に，引数に ρ_N をもつB-ソートの関数（NB-関数と呼ぶ）は出力関数に類比される．

〔定理 1.5〕 B-順序機械の公理系は Church-Rosser の性質をもつ．

（証明）　B－順序機械の定義 (iii), (iv) から，$\mathcal{E}-\mathcal{E}_B$ は，左正線形かつ無膠味であり，又 $\mathcal{E}_B$ と可換である．従って，補題 1.1 (i) より，$\mathcal{E}$ は Church-Rosser の性質をもつ．　□

〔系〕　B－順序機械について以下が成り立つ．

(1)　B－順序機械は B－無矛盾である．

(2)　任意の半定義域の解消 ρ に対し，ρ に従う代数が存在する．　□

$\mathcal{A}=(S, \Sigma, \mathcal{E})$ を B－順序機械，ρ をその一つの半定義域の解消とする．T_Σ 上の関係 "$\overset{\sim}{\rho}$" を次のように定める．

(I)　$\forall t, t' \in T_B \quad (t \overset{\sim}{\rho} t' \iff \rho(t) = \rho(t'))$

(II)　$\forall t, t' \in T^{\Delta_N} \{ t \overset{\sim}{\rho} t' \iff$

（任意の NB－関数 $f : \Delta_N, \Delta_2, \cdots, \Delta_n \to \Delta$ と $t_i \overset{\sim}{\rho} t_i'$ なる任意の項 $t_i, t_i' \in T^{\Delta_i}$ $(1 \leq i \leq n)$ に対し
$f(t, t_2, \cdots, t_n) \overset{\sim}{\rho} f(t', t_2', \cdots, t_n')$) $\}$

条件 (II) は "状態の出力面と" に類比される．

〔補題 1.7〕　関係 $\overset{\sim}{\rho}$ は，ρ に従う任意の代数 A に対応する $\mathcal{E}$－合同関係 R を含む等価関係である．

（証明）　$\overset{\sim}{\rho}$ が等価関係であることは明らかであろう．以下，任意の項 $t, t' \in T$ に対して，

$$t \; R \; t' \quad \Rightarrow \quad t \overset{\sim}{\rho} t' \qquad (1.4)$$

を示す．A が ρ に従うことから，$t, t' \in T_B$ については，$t \; R \; t' \iff \rho(t) = \rho(t')$ 故，条件 (I) より，R と $\overset{\sim}{\rho}$

は同じである．従って，t，t' がB-ソートの項である場合，式(1.4)は成立する．t，$t' \in T^{\rho_N}$ の場合を考える．R は条件SPを満たすから，$t\,R\,t'$ ならば，任意のNB-関数 f，$t_i\,R\,t_i'$ なる項 t_i，t_i' に対し，

$$f(t, t_2, \cdots, t_n)\ R\ f(t', t_2', \cdots, t_n')$$

t_i，t_i' $(2 \leqq i \leqq n)$，$f(\cdots)$ はB-ソートの項であるから，（R と $\tilde{\equiv}_\Gamma$ は同じであり）

$$t_i \tilde{\equiv}_\Gamma t_i'$$

$$f(t, t_2, \cdots, t_n)\ \tilde{\equiv}_\Gamma\ f(t', t_2', \cdots, t_n')$$

従って，条件(II)より，$t \tilde{\equiv}_\Gamma t'$． □

関係 $\tilde{\equiv}_\Gamma$ がもし条件SPを満たせば，$\tilde{\equiv}_\Gamma$ はε-合同関係であり，$\tilde{\equiv}_\Gamma$ に対応する代数は Γ に従う．すなわち，$\tilde{\equiv}_\Gamma$ は，Γ に従う最簡な代数に対応するε-合同関係である．$\tilde{\equiv}_\Gamma$ が条件SPを満たすための十分条件を示す．

［定理 1.6］ B-順序機械 $\mathfrak{M}$ において，すべてのNB-関数記号 $f: \rho_N, \rho_{\ell+1}, \cdots, \rho_m \to \rho$，N-関数記号 $g: \rho_1, \cdots, \rho_\ell \to \rho_N$ の対について，次のような項 $\tau_{fg}(x_1, \cdots, x_m)$ が存在すると仮定する．

(i) ソート ρ_i の任意の項 t_i $(1 \leqq i \leqq m)$ に対し，

$$f(g(t_1, \cdots, t_\ell), t_{\ell+1}, \cdots, t_m) \equiv \tau_{fg}(t_1, \cdots, t_m)$$

(ii) 引数としてソート ρ_N をもつN-関数記号は τ_{fg} に現れない．

このとき，任意の未定義域の解消 Γ に対して，関係 $\tilde{\equiv}_\Gamma$ は条件SPを満たし，従って，$\tilde{\equiv}_\Gamma$ は Γ に従う代数の中で最簡な代数に対応するε-合同関係である．

(証明) $\tilde{\equiv}_P$ が条件 SP を満たすことを示す. t_i, t_i' $(i=1, 2, \cdots)$ を $t_i \tilde{\equiv}_P t_i'$ なる任意の項とする. まず次の補題を示す.

[補題 1.8] すべての t_i $(1 \leqq i)$ が B-ソートの項なら, t_i を部分項として含む B-ソートの項 t において, 各 t_i を t_i' で置き換えて得られる項を t' とおくと, $t \tilde{\equiv}_P t'$.

(証明) t' の作りかより, $t \equiv_P t'$. B-順序機械は, 定理 1.4 の前提条件を満たすことから, $\equiv_P$ は B-無矛盾であり, 従って, B-ソートの項については $\tilde{\equiv}_P$ と $\equiv_P$ は同じ関係であり, $t \tilde{\equiv}_P t'$. □

この補題 1.8 を用いて, $\tilde{\equiv}_P$ が条件 SP を満たすことを示す. B-ソートの関数記号 f について,

$$f(t_1, \cdots, t_n) \tilde{\equiv}_P f(t_1', \cdots, t_n')$$

が成りたつことは, t_1 のソートが B-ソートなら補題 1.8 より, ソート ρ_N なら $\tilde{\equiv}_P$ の定義の条件 (II) より明らか. N-関数記号 g について,

$$g(t_1, \cdots, t_n) \tilde{\equiv}_P g(t_1', \cdots, t_n') \quad (1.5)$$

を示すには, $\tilde{\equiv}_P$ の定義 (II) より, 任意の NB-関数記号 f に対して,

$$b \triangleq P(f(g(t_1, \cdots, t_\ell), t_{\ell+1}, \cdots, t_m))$$
$$b' \triangleq P(f(g(t_1', \cdots, t_\ell'), t_{\ell+1}', \cdots, t_m'))$$

とおくと, $b = b'$ を示せばよい.

$t_1 \in T_B$ なら, 補題 1.8 より, $b = b'$. 以下, t_1, t_1' のソートを ρ_N とする. 定理の前提条件 (i) より,

$$b = \rho(\tau_{fg}(t_1, t_2, \cdots, t_m))$$

$$b' = \rho(\tau_{fg}(t_1', t_2', \cdots, t_m'))$$
$$= \rho(\tau_{fg}(t_1', t_2, \cdots, t_m)) \quad (\text{補題 1.8 より})$$

又，$\tau_{fg}(x_1, \cdots)$ 中，変数 x_1 は NB-関数記号の第1引数としてのみ現れる．x_1 は，その出現の長い（すなわち，関数のネストの深い）ものから順に，

$$f_i(x_1, t_{i2}, \cdots, t_{in_i}), \quad i = 1, 2, \cdots$$

の形で現れているとし，$t \equiv \tau_{fg}(t_1, t_2, \cdots, t_m)$，$t' \equiv \tau_{fg}(t_1', t_2, \cdots, t_m)$ 中の対応する項を，それぞれ，$f_i(t_1, \tilde{t}_{i2}, \cdots, \tilde{t}_{in_i})$，$f_i(t_1', \tilde{t}_{i2}', \cdots, \tilde{t}_{in_i}')$ と書く．t と t' は，部分項の t_1 と t_1' が互いに入れ換っているだけで，他は同一であり，t_1 の出現の最も長い $i = 1$ については，$\tilde{t}_{ij} = \tilde{t}_{ij}'$ $(2 \leq j \leq n_i)$．従って，仮定の $t_1 \overset{\sim}{\rho} t_1'$ 及び $\overset{\sim}{\rho}$ の定義(II)から，

$$\rho(f_i(t_1, \tilde{t}_{i2}, \cdots)) = \rho(f_i(t_1', \tilde{t}_{i2}', \cdots)) \quad (1.6)$$

t' において，$f_1(t_1', \tilde{t}_{12}', \cdots)$ を $f_1(t_1, \tilde{t}_{12}, \cdots)$ で置き換えて得られる項を t'' とすると，補題 1.8 より，$\rho(t') = \rho(t'')$．t と t'' では $i = 2$ について同様のことがいえる．以下，この議論の繰り返しで，すべての i について式(1.6)が成り立ち，従って，$b = b'$．すなわち，式(1.5)が成立する．　（定理 1.6 証明終）

定理 1.6 の前提条件を満たす B-順序機械 $\mathcal{D}$ は，その一つの未定義域の解消 ρ が与えられたとき，次のように表される．NB-関数全体を

$$\{ f_i : \rho_N, \rho_{i2}, \cdots, \rho_{in_i} \to \rho_i \mid 1 \leqq i \leqq p \}$$

とし，関係 $\overset{\sim}{\rho}$ による T^{ρ_N} の分割を $\{ U_k \mid k = 1, 2, \cdots \}$ と置く．各 U_k に対し，次のような p 個の関数 $F_i^k : \rho_{i2}, \cdots, \rho_{in_i} \to \rho_i$ $(1 \leqq i \leqq p)$ を考える．

$$F_i^k(x_{i2}, \cdots, x_{in_i}) = \rho(f_i(u, x_{i2}, \cdots, x_{in_i}))$$

但し，$u \in U_k$ （仮定から u の選び方に依存しない）．$\mathcal{D}$ の（"最簡形"における）一状態に相当する U_k は，$\overset{\sim}{\rho}$ の同値類であることから，p 個の関数の組

$$\mathcal{F}^k \triangleq \langle F_1^k, F_2^k, \cdots, F_p^k \rangle$$

と一対一に対応し，$\mathcal{F}^k$ で表される．

更に，各 F_i^k について，F_i^k のとり得る値の種類が有限で，そのうち一つの値を除いて他の値をとる引数の組合せの数が有限ならば，F_i^k は，ソート $\rho_{i2}, \cdots, \rho_{in_i}$ の B-定数の並びを検索項目とし，内容がソート ρ_i の B-定数であるような（登録されている値の数が）有限の表 $table_i^k$ で，従って，$\mathcal{F}^k$ はそのような表の p 個の並び $table^k$ で表される．このとき，"状態" $u \in U_k$ での各 NB-関数 $f_i(u, b_{i2}, \cdots, b_{in_i})$ は

$$access(\langle b_{i2}, \cdots, b_{in_i} \rangle, table_i^k)$$

で表され，N-関数の適用 $g(u, b_2, \cdots, b_n)$ は，

$$f_i(g(u, b_2, \cdots, b_n), b_{i2}, \cdots, b_{in_i})$$

$$\neq f_i(u, b_{i2}, \cdots, b_{in_i})$$

であるような（すなわち，N－関数の適用により値が変化するような）項目 $\langle b_{i2}, \cdots, b_{in_i} \rangle$ について，その内容を

$$p(\overline{\tau_{fg}}(table^k, b_2, \cdots, b_n, b_{i2}, \cdots, b_{in_i}))$$

に変更する操作に対応する．ここで，$\overline{\tau_{fg}}(table^k, \cdots)$ は，$\tau_{fg}(u, \cdots)$ における $f_j(u, \cdots)$ の形の項をすべて $access(\langle \cdots \rangle, table_j^k)$ で置き換えて得られる項である．定理 1.6 の条件(ii)より，$\overline{\tau_{fg}}(table^k, \cdots)$ の値は，$table^k$ の内容に基底代数上の関数を施した値として得られる．

1.6 結言

B－仕様を定式化し，そこでの“don't care”の取扱い等の諸性質を示し，更に，比較的よく用いられるB－仕様の部分クラスB－順序機械について，簡単化および“表”での実現手法について述べた．1.3.3の記述例に関して述べたように，基底代数は，必ずしもその全部が具体的に与えられていなくてもよく，パラメータ化されていると見ることができる．

1.3.3の記述例およびHDLC手順の一部の記述[9]がB－順序機械として得られている．又，状態を階層的に表している場合，階層化の下のレベルの状態（N－ソート）を上のレベルのB－ソートとみなすなどのようにB－順序機械の定義を拡張すれば，次章のファイル管理システムの記述もそのクラスに属す．そのような拡張を行っても1.5の議論には支障を来さない．

仕様が“正しく”書かれているかという問に対する形式的な立場からの解答に，公理系の無矛盾性を示すことと，“don't care”を除くすべてのB－ソートの項について $eval(t) \neq \perp$ を示すことがある．後者は公理が不足していないことを意味する．B－順序機械では，無矛盾性は定理1.5系より保証されており，公理が足りていることも，定理1.6の前提条件が満たされていれば，項の構造に関する帰納法で容易に示される場合が多い．

B－仕様の定義は文献(4)に見られるような関数形のプログラムを含んでおり，B－仕様をプログラムとみなすこともできる[7]．B－仕様は，この意味で，仕様とプログラムを同一の枠組で定式化するものである．又，“don't care”も自然な形で取扱うことができる．

第2章　順編成ファイル管理システムの記述
—同期機能の表現—

2.1　序言

代数的記述の対象として，ファイル管理システムのうち，順編成ファイルに関する部分を選んだ．それが含む同期の問題をいかに記述するか，又，その機構化の定式化も興味ある問題と考えたからである．通常の高級言語による記述では，その正確な意味が言語の形式的な意味の定義書（それ自身極めて複雑である）を参照して初めて定まる．これに対して，本方式による代数的な記述では，項間の等式の組が生成する合同関係の概念，及び論理演算，整数の加減算と大小関係，文字列，系列と記号の連接などのみを前提とする以外，すべて明示されている．

利用者プロセスから見て，レコードレベルで完結した一つの記述 File-system を 2.2 に示している．記述はかなり長くなるので，できるだけ階層化し，“モジュール化”している．READ 及び WRITE リクエストは，利用者プロセスから見ても不可分の操作でなく，システムとの同期のためにWAIT リクエスト，あるいは READ，WRITE リクエスト自身が使われる．このような READ，WRITE リクエストの可分性と同期のための機能とに無関係に，順編成ファイル固有の性格から意味が定義できる部分をまず抽出して，System-0 として 2.2.4 に記述した．そこでは，READ，WRITE リクエスト操作は不可分として表現されている．この記述自体，4つの部分仕様から成り立っている．一方，File-system では，READ，

WRITE リクエストに対する処理を，受け付けて外部記憶装置へ依頼する前処理と，外部記憶装置からの応答に対する後処理に分けて表し，これらの前処理に関連する関数と公理を System-0 に付加した部分仕様 System-1 及び利用者プロセスと管理プロセス間の“プロセス制御”に関する部分から成りたっている．これらの記述について，“無矛盾性”と“準完全性（必要な関数の値が定義されている）”が示される．

File-system は，より“抽象的な” System-0 のレコードレベルでの“機構化”と考えられるが，利用者プロセスから見て，WAIT リクエスト，更に READ, WRITE リクエストの同期機能が追加されていることからも分かるように，“外”すなわち利用者プロセスから見えない内部の詳細化ではない．（ファイルの整合性を保つ機能は管理プロセスによって提供されているが，）利用者バッファの“整合性”を保つのは利用者プロセスの責任に委ねられている．この整合性を保つための，利用者バッファの使用と WAIT リクエストなどの使い方に関する制限を表す述語 wellB を図 2.11 に示す．利用者プロセスがこの制限に従って操作を行う限り，利用者プロセスから見て，File-system は System-0 と実効的に等しいことを 2.3 に示す．この意味で，System-0 の仕様と wellB に関する公理を合せたものは，File-system の利用者プロセスから見て直接意味のある部分を集約した一つの形式的な手引書と考えられよう．

2.2 ファイル管理システムの記述

2.2.1 記述の対象

ファイル管理システムのうち，順編成ファイルに関する部分を記述する．記述が長くなるので，次の(1)～(7)のリクエスト，利用者操作のみに限定する．

(1)	ファイルfの作成のために開く	OPENW(f, r)
(2)	ファイルfを読み出し用に開く	OPENR(f, r)
(3)	1レコード書き加える	WRITE(r, b)
(4)	1レコード読み取る	READ(r, b)
(5)	ファイルを閉じる	CLOSE(r)
(6)	先に出したREAD(r, b)又は WRITE(r, b)の完了を待つ	WAIT(r)
(7)	利用者バッファbにレコードrecを書き込む	PUTB(b, rec)

ここで，fはファイル名，rはファイルにアクセスするときの参照名，bは利用者バッファの識別子である．簡単のため，ファイルの保護規定に関する記述を省略し，又，実在する利用者プロセスが一つしかない単一利用者システムを考える．

2.2.2 基底代数と if 関数

File-systemでは，次の(1)～(4)を基底代数として採用している（図 2.1 参照）．

(1) Presburger Arithmetic. 論理値および整数のソート名をそれぞれ bool 及び int と書く．これらに関する演算は通常の記法に従う．

(2) ファイル名・参照名・利用者バッファ識別子の集合．それぞれの集合を表すソートを fid, rid, bid と書く．各集合において，元が等しいか否かの述語が定義されていることを以外仮定しない．

(3) 文字列の集合． 文字列集合を表すソートを string と書く．図 2.1 (3) の5つの文字列と，相等の述語のみ用いる．

(4) 論理レコードとその系列の集合． 論理レコード（以下，単にレコードと呼ぶ）及びレコードの系列（ファイルの内容）の集合を表すソートを，それぞれ，record 及び seqREC と書く．レコードの空系列を nullR，レコード系列 R の後尾にレコード rec を連接する演算を R.rec，R の前から第 i 番目（先頭を0番目とする）のレコードを select(R, i)，R の長さ（レコード数）を length(R) と書く．レコードの特殊な元 “eof” は，読み出すべきレコードがなくなった旨のシステムメッセージを表すとする．

なお，元の相等に関する述語とその否定を各ソート毎に異なった記号を用いず，共通の記号 =，≠ で表す† 各ソートにつき，図 2.1 (5) の公理をもった if 関数が，

---

† 一般に，複数の関数を同じ記号で表すことを許す．但し，現れているそれぞれの場所での関数記号が引数ソートその他から，どの関数を表しているかが識別できなくてはならない．

(1) [Presburger Arithmetic]

```
SORT bool, int;

OP   true, false:              -> bool,
     not         : bool        -> bool,
     and,  or    : bool, bool  -> bool,

      0,    1    :             -> int,
     +1,   -1    : int         -> int,
     =0          : int         -> bool,
     >=          : int.  int   -> bool;
```

(2) [identifiers]

```
SORT fid, rid, bid;

OP   FOR EACH S IN {fid, rid, bid},

     =, ≠: S, S -> bool;
```

(3) [character strings]

```
SORT string;

OP   "OP/PT", "R/W", "CL",
     "WT",    "AC" :       -> string,

     =, ≠: string, string -> bool;
```

(4) [records and record sequences]

```
SORT record, seqREC;

OP   nullR :                    -> seqREC,
     .     : seqREC, record     -> seqREC,
     select: seqREC, int        -> record,
     length: seqREC             -> int,

     "eof" :                    -> record;
```

(5)
```
A: if true  then x else y == x;
B: if false then x else y == y;

C: if true  then x        == x;
```

図 2.1 基底代数のソート・関数と if 関数の公理

暗黙のうちに，宣言されているものとする．(5) Cの形のif関数は，条件節が真のときのみ関数値を定義し，そうでないとき未定義とする場合に用いられる．

2.2.3 リクエストおよび利用者操作に関する仕様

リクエストおよび利用者操作（以下，単にリクエストと呼ぶ）のインスタンスとそれに関する演算の仕様を図2.2に示す．図2.2文(1)は一つの仕様の始まりとその仕様名を宣言し，文(14)はその仕様の終りを表す．

文(2)は，この仕様が前提とする基底代数が（2.2.2に示した）Base-Algebraであることを表し，文(3)は，新たに導入する（リクエスト・インスタンスの集合を表す）ソート名uopを宣言している．

文(4)は，リクエスト・インスタンスを構成する関数記号とそれらの引数ソート系列および値域ソートを宣言している．CONSTRUCTOR文で宣言される関数記号を構成子と呼ぶ．構成子の値は公理から直接定義されない．これに対し，公理で関数値が定義されるような関数記号は(5)のOP文で宣言する．文(5)は，リクエストを処理内容の類似性から分類するための関数type，およびリクエスト・インスタンスから（参照名が構成要素の一部であるとき）参照名を取り出す関数refIDを宣言しており，これらがO-function（以下，O-関数と書く）と呼ばれることを表している．

文(6)～(13)は関数typeおよびrefIDの定義を与えている．これらの文中の記号f，r，b，recは（公理における）変数記号であり，そのソートは，（引数として）使われている場所により決定される．文(13)は，略記法

```
(1)  SPEC Requests-and-user-operations;
(2)   BASE Base-Algebra;
(3)   SORT uop;
(4)   CONSTRUCTOR
         OPENW, OPENR: fid, rid     -> uop,
         WRITE, READ : rid, bid     -> uop,
         CLOSE, WAIT : rid          -> uop,
         PUTB        : bid, record -> uop;
(5)   OP [O-function]
         type        : uop          -> string,
         refID       : uop          -> rid;

(6)   T1: type(OPENW(f,r))   == "OP/PT";
(7)   T2: type(OPENR(f,r))   == "OP/PT";
(8)   T3: type(WRITE(r,b))   == "R/W";
(9)   T4: type(READ (r,b))   == "R/W";
(10)  T5: type(CLOSE(r))     == "CL";
(11)  T6: type(WAIT (r))     == "WT";
(12)  T7: type(PUTB(b,rec)) == "OP/PT";
(13)  RF1*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r)
         WRITE(r,b) READ(r,b) CLOSE(r) WAIT(r)},
            refID(Q) == r;
(14) END;
```

図 2.2　リクエストおび利用者操作に関する仕様

の一つで，EACH に続く記号（ここでは Q）を，“{”から“}”までの空白で区切られた（空白を含まない）文字列（ここでは，“OPENW(f, r)”，…，“WAIT(r)”）で置き換えて得られる公理（ここでは6個）を表す．なお，太字は見易さのために用いているだけである．PUTB(b, rec) に対しては，（参照名が構成要素でないから）refID の値を定義していない．

2.2.4 System-0 の記述

すべてのリクエストが不可分であるとして，順編成ファイル固有の性格から定義されるそれらの意味の仕様 System-0 及びその4つの部分仕様を図 2.3 ～ 2.6 に示す．この場合，WAIT リクエストは意味をもたないのでリクエストから除く．

(1) System-0 の階層構成． System-0 は，利用者バッファ（以下，単にバッファと呼ぶ）の集合，ファイルの集合，ファイルディレクトリ，データセット制御部からなる．これらに関する仕様をそれぞれ Buffers, Files, File-directory, Dataset-control-blocks と呼ぶ．File-directory 及び Dataset-control-blocks を，それぞれ FD 及び DCB と略す．図 2.3 の 2，3 行目の INCLUDE 文は，仕様 System-0 が上記4つの仕様を部分仕様として含むことを示す．なお，一つの仕様内で，同じ仕様が複数回 INCLUDE 文で包含されても一回の包含と同等である．

システム状態の集合を導入し，それを表すソートを sys と呼び，その初期状態を Initials で表す．状態 s

でリクエストqに対する処理が行われた直後の状態を Execute(q, σ) と書く．部分仕様 Buffers, Files, FD, DCBにおいてもそれぞれ状態集合を導入し，それらを表すソートをそれぞれ bf, fl, fd, dcb と呼び，それぞれの初期状態を initialB, initialF, initialFD, initialDCB で表す．System−0 の状態σは4つの部分仕様に関する状態成分 stateB(σ), stateF(σ), stateFD(σ), stateDCB(σ) よりなると考えている．stateB, stateF, stateFD, stateDCB は，System-0 の状態からそれぞれの成分への射影関数である．仕様 System-0 では，それぞれの状態成分を σB, σF, σFD, σDCB と略記しており，それを表すのが図2.3の4つの LET文である．

PUTB 以外のリクエストqには，System-0 の各状態σにおいて，それが意味をもつか否かの適用条件が定められている．その条件を表す述語を invalid(q, σ)（真のとき，状態σでqは無効）と書く．invalid は，図2.3 V1 ~ V5 の公理で定義されている．公理 B1 ~ B3* は，バッファが PUTB 及び有効な READ リクエストによってのみ変化すること，及びその変化を示す．公理 B2 では，読み易くするため，WHERE節（適用範囲はその文に限定される）による略記を行っている．以下，公理の説明を省くが，それらはすべて，状態遷移後 O-関数がどのように変化するかを示すことで，その O-関数を定義している．次に4つの部分仕様について説明する．

(2) 仕様 Buffers では，状態 σB において，識別子 b のバッファにレコード rec を書き込んだ直後の状態を putB(b, rec, σB)，バッファ b の内容を contentB(b, σB)，バッファの初期内容を initialR と表している．

```
SPEC System-0;

 INCLUDE Buffers, Files, File-directory,
         Dataset-control-blocks;
 SORT    sys;

 CONSTRUCTOR
    InitialS:           -> sys,
    Execute : uop, sys -> sys;

 OP [O-function]
    invalid : uop, sys -> bool,
    stateB  :      sys -> bf,
    stateF  :      sys -> fl,
    stateFD :      sys -> fd,
    stateDCB:      sys -> dcb;

 LET sB := stateB(s);  LET sFD  := stateFD (s);
 LET sF := stateF(s);  LET sDCB := stateDCB(s);

 V1: invalid(OPENW(f,r), s) ==
    exist(f, sFD) or used(r, sDCB);
 V2: invalid(OPENR(f,r), s) ==
    if not exist(f, sFD) then true
    else used(r, sDCB) or openedW(f, sFD);
 V3: invalid(WRITE(r,b), s) ==
    if not used(r, sDCB) then true
    else not openedW(fname(r, sDCB), sFD);
 V4: invalid(READ(r,b), s) ==
    if not used(r, sDCB) then true
    else openedW(fname(r, sDCB), sFD);
 V5: invalid(CLOSE(r), s) == not used(r, sDCB);
```

図 2.3 (a)　仕様 System-0 (その1)

```
B0: stateB(InitialS) == initialB;
B1: stateB(Execute(PUTB(b,rec), s)) ==
   putB(b, rec, sB);
B2: stateB(Execute(READ(r,b), s)) ==
   if invalid(READ(r,b), s) then sB
   else putB(b, READREC, sB)
   WHERE READREC :=
      if count(r, sDCB) >= length(FILE) then "eof"
      else select(FILE, count(r, sDCB))
      WHERE FILE := file(fname(r, sDCB), sF);
B3*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r)
   WRITE(r,b) CLOSE(r)},
   stateB(Execute(Q, s)) == sB;

F0: stateF(InitialS) == initialF;
F1: stateF(Execute(WRITE(r,b), s)) ==
   if invalid(WRITE(r,b), s) then sF
   else appendF(fname(r,sDCB), contentB(b,sB), sF);
F2*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r)
   READ(r,b) CLOSE(r) PUTB(b,rec)},
   stateF(Execute(Q, s)) == sF;


FD0: stateFD(InitialS) == initialFD;
FD1: stateFD(Execute(OPENW(f,r), s)) ==
   if invalid(OPENW(f,r), s) then sFD
   else openFD(f, sFD);
FD2: stateFD(Execute(CLOSE(r), s)) ==
   if invalid(CLOSE(r), s) then sFD
   else if openedW(fname(r, sDCB), sFD) then
      closeFD(fname(r, sDCB), sFD)
   else sFD;
FD3*: FOR EACH Q IN {OPENR(f,r) WRITE(r,b)
   READ(r,b) PUTB(b,rec)},
   stateFD(Execute(Q, s)) == sFD;

DCB0: stateDCB(InitialS) == initialDCB;
DCB1*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r)
   WRITE(r,b) READ(r,b) CLOSE(r)},
   stateDCB(Execute(Q, s)) ==
   if invalid(Q,s) then sDCB else execDCB(Q,sDCB);
DCB6: stateDCB(Execute(PUTB(b,rec), s)) == sDCB;

END;
```

図 2.3 (b)　仕様 System-0 (その 2)

```
SPEC Buffers;

 BASE Base-Algebra;
 SORT bf;

 CONSTRUCTOR
    initialR:                    -> record,
    initialB:                    -> bf,
    putB     : bid, record, bf -> bf;

 OP [O-function]
    contentB: bid,          bf -> record;

 C0: contentB(b, initialB) == initialR;
 C1: contentB(b, putB(b', rec, s)) ==
    if b = b' then rec else contentB(b, s);

END;
```

```
SPEC Files;

 BASE Base-Algebra;
 SORT fl;

 CONSTRUCTOR
    initialF:                    -> fl,
    appendF  : fid, record, fl -> fl;

 OP [O-function]
    file     : fid,          fl -> seqREC;

 F0: file(f, initialF) == nullR;
 F1: file(f, appendF(f', rec, s)) ==
    if f = f' then file(f,s).rec else file(f,s);

END;
```

図 2.4 仕様 Buffers, Files

```
SPEC File-directory;

 BASE Base-Algebra;
 SORT fd;

 CONSTRUCTOR
    initialFD      :          -> fd,
    openFD, closeFD: fid, fd -> fd;

 OP [O-function]
    exist,  openedW: fid, fd -> bool;

 EX0: exist(f, initialFD) == false;
 EX1: exist(f, openFD(f', s)) ==
    if f = f' then true else exist(f, s);
 EX2: exist(f, closeFD(f', s)) == exist(f, s);

 OW1: openedW(f, openFD(f', s)) ==
    if f = f' then true else openedW(f, s);
 OW2: openedW(f, closeFD(f', s)) ==
    if f = f' then false else openedW(f, s);

END;
```

図 2.5　仕様 File-directory

```
SPEC Dataset-control-blocks;

 INCLUDE Requests-and-user-operations;
 SORT    dcb;

 CONSTRUCTOR
    initialDCB:           -> dcb,
    execDCB    : uop, dcb -> dcb;

 OP [O-function]
    used : rid, dcb -> bool,
    fname: rid, dcb -> fid,
    count: rid, dcb -> int;

 INI: used(r, initialDCB) == false;

 OP1*: FOR EACH Q IN {OPENW OPENR},
   {used(r, execDCB(Q(f,r´), s)) ==
       if r = r´ then true else used(r, s);
    fname(r, execDCB(Q(f,r´), s)) ==
       if r = r´ then f else fname(r, s);
    count(r, execDCB(Q(f,r´), s)) ==
       if r = r´ then 0 else count(r, s)};

 WR1*: FOR EACH Q IN {WRITE READ},
   {used (r, execDCB(Q(r´,b), s)) == used (r, s);
    fname(r, execDCB(Q(r´,b), s)) == fname(r, s);
    count(r, execDCB(Q(r´,b), s)) ==
       if r = r´ then count(r,s)+1 else count(r,s)};

 CL1: used(r, execDCB(CLOSE(r´), s)) ==
    if r = r´ then false else used(r, s);
 CL2*: FOR EACH F IN {fname count},
    F(r, execDCB(CLOSE(r´), s)) ==
    if r ≠ r´ then F(r, s);

END;
```

図 2.6　仕様 Dataset-control-blocks

(3) 仕様 Files では，状態 sF において，ファイル名 f のファイルの後尾にレコード rec を書き加えた直後の状態を appendF(f, rec, sF)，ファイル f の内容（レコード系列）を file(f, sF) と表している．

(4) FD は，状態 sFD において，ファイル名 f が登録されているか否か（exist(f, sFD)），及びそのファイルが書き込みのために開かれているか否か（openedW(f, sFD)）を記憶している部分である．ファイル f の作成開始直後の FD 状態を openFD(f, sFD)，終了直後の状態を closeFD(f, sFD) と書く．

(5) DCB は，状態 sDCB において，参照名 r が使われているか否か（used(r, sDCB)），使用されている r に対し，対象となるファイルの名前（fname(r, sDCB)）及びファイルが開かれている間に出された READ，WRITE（以下，R/W と書く）リクエストの回数（count(r, sDCB)）を記憶している部分である．リクエスト g の処理が行われた直後の DCB 状態を execDCB(g, sDCB) と書く．

2.2.5 File-system の記述

R/W リクエストに対する処理が前処理と後処理に分かれている場合を考える．後処理が終了したとき，完了したという．

(1) File-system の構成．File-system（以下，FS と略す）の仕様を図 2.7 に示す．FS の状態集合を表すソートを fsys，その初期状態を InitialFS と書く．

FS 状態 s で，リクエスト q が出された直後の状態を Issue(q, s)，参照名 r の R/W リクエストの一連の後処理が完了した直後の状態を Complete(r, s) と書く．FS は部分仕様 System-1（図 2.8）を含む．FS 状態 s における System-1 の状態を stateS1(s)，利用者プロセスが実行（可能な）状態にあるか否（すなわち，待ち状態）かの述語を UP_executing(s)，待ち状態のとき，利用者プロセスの出した最後のリクエストを last_uop(s) と書く．

(3) 部分仕様 System-1．　System-1 は，System-0 を部分仕様として含み，新たに，状態遷移関数として，R/W リクエスト q の前処理に対応する Initiate(q, s)，及び 0-関数として，参照名 r の R/W リクエストの前処理は行われたが後処理が完了していないことを表す述語 busyR(r, s)，そのときの未完了のリクエストを表す initiated-uop(r, s) を追加している．なお，新しいソートは導入していない．R/W リクエストに対する状態遷移関数 Execute は，（System-1 では）その後処理が完了した直後の状態を表す．

(4) System-1 の状態変化．　FS においてリクエスト q が出されると，System-1 の状態は次のように変化する．（図 2.7 公理 ST1）

(a) q が R/W，CLOSE，WAIT リクエストで，且つ q で指定される参照名と同じ参照名で未完了のものがあれば，（その完了まで q の処理は待たされ）変化しない．

(b) そうでなく，q が R/W リクエストなら，その前処理が開始される．

```
SPEC File-system;

 INCLUDE System-1;
 SORT    fsys;

 CONSTRUCTOR
    InitialFS:              -> fsys,
    Issue    : uop, fsys -> fsys,
    Complete: rid, fsys -> fsys;

 OP [O-function]
    stateSl      :  fsys -> sys,
    UP-executing:  fsys -> bool,
    last-uop     :  fsys -> uop;

 LET sSl := stateSl(s);

 ST0: stateSl(InitialFS) == InitialS;
 ST1: stateSl(Issue(q, s)) ==
    if UP-executing(s) then
      (if type(q) = "OP/PT" then Execute(q, sSl)
       else if busyR(refID(q), sSl) or
          (type(q) = "WT") then sSl
       else if type(q)="R/W" then Initiate(q, sSl)
       else Execute(q, sSl));
 ST2: stateSl(Complete(r, s)) ==
    if busyR(r, sSl) then
      (if UP-executing(s) then Execute(Q, sSl)
       else if (refID(Q´) ≠ r) or (type(Q´) = "WT")
          then Execute(Q, sSl)
       else if type(Q´) = "R/W" then
          Initiate(Q´, Execute(Q, sSl))
       else Execute(Q´, Execute(Q, sSl)))
       WHERE Q  := initiated-uop(r, sSl) AND
             Q´ := last-uop(s);

 UE0: UP-executing(InitialFS) == true;
 UE1: UP-executing(Issue(q, s)) ==
    if UP-executing(s) then
      (if type(q) = "OP/PT" then true
       else not busyR(refID(q), sSl));
 UE2: UP-executing(Complete(r, s)) ==
    if busyR(r, sSl) then
      (if UP-executing(s) then true
       else (refID(last-uop(s)) = r));

 LU1: last-uop(Issue    (q, s)) ==
    if not UP-executing(Issue(q, s)) then q;
 LU2: last-uop(Complete(r, s)) == last-uop(s);

END;
```

図 2.7　仕様 File-system

```
SPEC System-1;

 INCLUDE System-0;

 CONSTRUCTOR
    Initiate      : uop, sys -> sys;

 OP [O-function]
    busyR         : rid, sys -> bool,
    initiated-uop: rid, sys -> uop;

 BS0: busyR(r, InitialS) == false;
 BS1: busyR(r, Initiate(q, s)) ==
    if invalid(q, s) or (type(q) ≠ "R/W") then
       busyR(r, s)
    else if r = refID(q) then true
    else busyR(r, s);
 BS2: busyR(r, Execute(q, s)) ==
    if type(q) ≠ "R/W" then busyR(r, s)
    else if r = refID(q) then false
    else busyR(r, s);

 IOP1: initiated-uop(r, Initiate(q, s)) ==
    if invalid(q, s) or (type(q) ≠ "R/W") then
       initiated-uop(r, s)
    else if r = refID(q) then q
    else initiated-uop(r, s);
 IOP2: initiated-uop(r, Execute(q, s)) ==
    if busyR(r, Execute(q, s)) then
       initiated-uop(r, s);

 WT1*: FOR EACH X IN {B F FD DCB},

    stateX(Initiate(q, s)) == stateX(s);

END;
```

図 2.8　仕様 System-1

(c) その他の場合，Sの処理が行われる．

FSで参照名rのR/Wリクエストが完了すると，System-1では，その後処理の完了後，待たされている参照名rのリクエストがあれば，その処理（R/Wリクエストの場合前処理）が行われる（公理ST2）．

なお，FSでリクエストが出されるのは利用者プロセスが実行状態のときに限られ，参照名rのR/Wリクエストの完了は，rに関して未完了のものがある場合に限られる．従って，公理ST1，ST2では，そのような場合にのみSystem-1の状態変化を定義している．

2.2.6 無矛盾性と準完全性

基底代数のすべての定数を構成子とする．構成子のみからなる項を構成子項と呼び，構成子項全体の集合をT_Cと書く．図2.2～2.8の公理では，左辺が構成子項であるような公理は存在しないから，任意の構成子項$t \in T_C$に対して，

$$\nexists t' \quad (t \rightarrow t')$$

すなわち，構成子項は標準項である．

図2.2～2.8の公理では，右辺に現れている変数は左辺に必ず現れており，無曖昧かつ左辺線形なものとして書かれているから，それらはChurch-Rosserの性質をもつ．又，左辺に基底部の関数記号が現れてなく，従って，定理1.2の条件を満たす．以上より，仕様FSは，（全体として）Church-Rosserの性質をもつ．このことと，構成子が標準項であることから，異なる二つの構

成る項が公理の下で等価となることはなく，従って，FSは T_C-無矛盾である．（もちろん，基底部の定数についても無矛盾である．）

実引数が与えられた O-関数の“値”を，それと等価な構成子項と定義すると，T_C-無矛盾性より，FSの O-関数の値は，存在するならば，一意に決まる．FSでは，各 O-関数の値は次のような場合に存在する．

〔準完全性〕 t を，t の任意の部分項 $\tilde{t}$ が次の(a)，(b)を満たすようなソート f_{sys} の構成子項とする．

(a) $\tilde{t} = Issue(s, \tilde{t}') \Rightarrow UP\_executing(\tilde{t}') \equiv true$

(b) $\tilde{t} = Complete(r, \tilde{t}') \Rightarrow busyR(r, stateS1(\tilde{t}')) \equiv true$

FSにおいて，stateS1 及び UP_executing は，上記のようなすべての t に対して値が定義されており，last_uop は，UP_executing(t) ≡ false なる t に対して定義されている．

(2) System-0，System-1 において，state X（X ∈ {B, F, FD, DCB}）なる4つの射影関数は，ソート s_{ys} のすべての構成子項に対して値が定義されている．busyR(r, t) は，すべての参照名 r，構成子項 t に対して値が定義されており，initiated_uop(r, t) は，busyR(r, t) ≡ true なる参照名 r，構成子項 t に対して定義されている．invalid はすべての構成子項 s，t に対して定義されている．

(3) f，r，b をそれぞれファイル名，参照名，バッファ識別子，t_B，t_F，t_{FD}，t_{DCB} をそれぞれソート bf，fl，fd，dcb の構成子項とする．contentB(b, t_B)，file(f, t_F)，exist(f, t_{FD})，used(r, t_{DCB}) はすべての f，r，b，t_B，t_F，t_{FD}，t_{DCB} に対して定義されており，openedW(f, t_{FD}) は exist(f, t_{FD}) ≡ true

なるとき，∀，fname(r, t_{DCB}) および count(r, t_{DCB}) は used(r, t_{DCB}) ≡ true なるとき定義されている．

（証明）　状態を表す項の構造的帰納法による．

（I）　まず，準完全性（3）を示す．

（i）　公理 C0，F0（図2.4），EX0（図2.5），INI（図2.6）より，contentB，file，exist，used は各々の初期状態 initialX（X ∈ {B, F, FD, DCB}）および任意のファイル名 f，参照名 r，バッファ識別子 b において値が定義されている．openedW は公理 EX0 より，fname および count は公理 INI より，初期状態においては（3）における前提条件が満たされない．

（ii）　状態遷移関数が k 段ネストしている構成子項 t_B^k，t_F^k，t_{FD}^k，t_{DCB}^k においては（3）が成立していると仮定する．

まず Buffers について，k+1 段ネストした構成子項 $t_B^{k+1} \triangleq putB(b', rec, t_B^k)$ を考える（但し，b′はバッファ識別名，rec はレコード）．contentB(b, t_B^{k+1}) は，公理 C1 より，b = b′なら，値 rec をもち，b ≠ b′の場合も，contentB(b, t_B^k) に等しくなり，帰納法の仮定より，値をもつ．file，exist，used についても同様にして，$t_F^{k+1} \triangleq appendF(f', rec, t_F^k)$，$t_{FD,1}^{k+1} \triangleq openFD(f', t_{FD}^k)$ および $t_{FD,2}^{k+1} \triangleq close(f', t_{FD}^k)$，$t_{DCB}^{k+1} \triangleq execDCB(g, t_{DCB}^k)$ において値をもつことがいえる（但し，f′はファイル名，g はソートuopの構成子項）．

openedW(f, $t_{FD,i}^{k+1}$)（i = 1, 2）については，f = f′なら，公理 OW1，OW2 より，値をもつ．f ≠ f′の場合，

$$openedW(f, t_{FD,i}^{k+1}) \equiv openedW(f, t_{FD}^k)$$

ここで，exist(f, $t_{FD,i}^{k+1}$) ≡ true 且つ f ≠ f′なら，公理

EX1 より，

$$true \equiv exist(f, t_{FD,i}^{k+1}) \equiv exist(f, t_{FD}^{k})$$

故，帰納法の仮定より，$openedW(f, t_{FD}^{k})$ は値をもち，従って，$openedW(f, t_{FD,i}^{k+1})$ も値をもつ．CLOSE リクエスト以外の f に対する $fname(r, t_{DCB}^{k+1})$，$count(r, t_{DCB}^{k+1})$ も，同様にして値をもつことがいえる．

$t_{DCB}^{k+1} = execDCB(CLOSE(r'), t_{DCB}^{k})$ の場合，$r \neq r'$ ならば上述の議論と同様にして $fname(r, t_{DCB}^{k+1})$ 及び $count(r, t_{DCB}^{k+1})$ が値をもつことがいえる．$r = r'$ の場合，公理 CL1 より，$used(r, t_{DCB}^{k+1}) \equiv false$ となり，fname，count に関する前提条件が満たされない．

以上より，準完全性 (3) が成り立つ．

(II) 準完全性 (2) 及び次の命題を示す．

(4) $used(r, stateDCB(t)) \equiv true \Rightarrow exist(fname(r, stateDCB(t)), stateFD(t)) \equiv true$

(i) 初期状態 InitialS において，stateX ($X \in \{B, F, FD, DCB\}$) 及び busyR は，図 2.3 の公理 BO，FO，FDO，DCBO 及び図 2.8 公理 BSO より，値が定義されている．initiated-uop は，公理 BSO より，又，上記 (4) も公理 DCBO 及び図 2.6 の公理 INI より前提条件が満たされない．

(ii) 状態遷移関数が k 段ネストしているソート sys の構成子項 t^{k} について準完全性 (2) 及び上記 (4) が成立していると仮定する．$k+1$ 段ネストした項

$$t_1^{k+1} \triangleq Initiate(f, t^{k})$$
$$t_2^{k+1} \triangleq Execute(f, t^{k})$$

について考える．以下，$t_X^{k} \triangleq stateX(t^{k})$，$t_{X,i}^{k+1} \triangleq$

$state X(t_i^{k+1})$ $(X \in \{B, F, FD, DCB\}, i = 1, 2)$ と略記する．

(a) まず，t_1^{k+1} について考える．射影関数 state X については，公理 WT1* (図 2.8) より，

$$state X(t_1^{k+1}) \equiv state X(t^k)$$

故，帰納法の仮定より，t_1^{k+1} に対して値をもつことがいえる．又，このことから (4) も帰納法の仮定に帰着でき，$t = t_1^{k+1}$ に関して成り立つことがいえる．

上の結果と先に示した準完全性 (3) より，$exist(f, t_{FD,1}^{k+1})$，$used(r, t_{DCB,1}^{k+1})$，及び $exist(f, t_{FD,1}^{k+1}) \equiv true$ の場合の $openedW(f, t_{FD,1}^{k+1})$，$used(r, t_{DCB,1}^{k+1}) \equiv true$ の場合の $openedW(fname(r, t_{DCB,1}^{k+1}), t_{FD,1}^{k+1})$ もまた値をもつことがいえ，従って，公理 V1 ~ V5 (図 2.3) より，ソート uop の任意の構成子項 ξ に対して $invalid(\xi, t_1^{k+1})$ がブール値 (true 又は false) を値としてもつことがいえる．

公理 T1 ~ T7 (図 2.2) より，$type(\xi)$ はすべての構成子項 ξ に対して値をもつ．このことと帰納法の仮定より，$C \equiv invalid(\xi, t^k)$ or $(type(\xi) \neq$ "R/W") も値をもつ．$C \equiv true$ ならば，公理 BS1, IOP1 (図 2.8) より，

$$busyR(r, t_1^{k+1}) \equiv busyR(r, t^k) \qquad (2.1)$$

$$initiated\_uop(r, t_1^{k+1}) \equiv initiated\_uop(r, t^k) \qquad (2.2)$$

従って，帰納法の仮定より，busyR, initiated_uop は，t_1^{k+1} に対して準完全性 (2) を満たす．$C \equiv false$ のとき，$type(\xi) \equiv$ "R/W" 故，$refID(\xi)$ は値をもつ．$r \equiv refID(\xi)$ ならば，公理 BS1, IOP1 より，$busyR(r, t_1^{k+1})$, $initiated\_uop(r, t_1^{k+1})$ は値をもつ．$r \neq refID(\xi)$ なら，上式 (2.1), (2.2) が成り立つ．

(b) 次に t_2^{k+1} = Execute(ξ, t^k) について考える．まず，stateB について．帰納法の仮定より invalid(ξ, t^k) は値をもつから，ξ = READ(r, b) なら，公理 B2（図 2.3）より，

stateB(t_2^{k+1}) ≡ stateB(t^k) または

stateB(t_2^{k+1}) ≡ putB(b, READREC, t_B^k)

のいずれかが成り立つ．従って，READREC が値をもてば，帰納法の仮定より，stateB(t_2^{k+1}) も値をもつ．READREC について考えると，公理 V4 より，invalid(ξ, t^k) ≡ false なら，used(r, t_{DCB}^k) ≡ true が必要であり，従って，c ≜ count(r, t_{DCB}^k)，f ≜ fname(r, t_{DCB}^k)，FILE ≜ file(f, t_F^k) が値をもつ．また，c < length(FILE) なら，select(FILE, c) も値をもつ．従って，公理 B2 中の READREC の定義より，READREC も値をもつ．

ξ が READ リクエストでなければ，公理 B1，B3* より，

stateB(t_2^{k+1}) ≡ stateB(t^k) または

stateB(t_2^{k+1}) ≡ putB(b, rec, t_B^k)

のいずれかが成り立つ．従って，帰納法の仮定より，この場合も stateB(t_2^{k+1}) が値をもつことがいえる．

他の射影関数 stateF，stateFD，stateDCB についても，同様にして，t_2^{k+1} に対して値をもつことがいえる．また，命題 (4)，invalid，busyR，initiated-uop に関しても，(a) の（すなわち，t_1^{k+1} の）場合と同様の議論で，t_2^{k+1} に対する準完全性を示すことができる．

以上より，準完全性 (2) および命題 (4) が成り立つ．

〔III〕 最後に準定全性(1) および次の命題を示す.

(5) UP_executing(t) ≡ false ⇨
type(last_uop(t)) ≠ "OP/PT"

(i) 初期状態 InitialFS に対しては, 公理 ST0, UE0 (図 2.7) より, stateS1, UP_executing は値をもつ. last_uop および命題(5)については, 公理UE0より, UP_executing(InitialFS) ≡ true 故, 前提条件が満たされない.

(ii) 状態遷移関数が k 段ネストしたソート fsys の構成子項 t^k について, 準定全性(1)および命題(5)が成立していると仮定する. 仮定より, $t^k_{S1} \triangleq$ stateS1(t^k) は値をもつ.

(a) まず, $t_1^{k+1} \triangleq$ Issue(ξ, t^k) (但し, UP_executing(t^k) ≡ true)について考える. type(ξ), および type(ξ) ≠ "OP/PT" のときの refID(ξ) が値をもつことと帰納法の仮定, 公理 ST1, UE1 より, stateS1(t_1^{k+1}) および UP_executing(t_1^{k+1}) は値をもつ. 又, UP_executing(t_1^{k+1}) ≡ false なら, 公理LU1より, last_uop(t_1^{k+1}) ≡ ξ. 更にそのとき, 公理UE1より, type(ξ) ≠ "OP/PT" でなければならず, 従って命題(5)も成りえつ.

(b) 次に, $t_2^{k+1} \triangleq$ Complete(r, t^k) (但し, busyR(r, t^k_{S1}) ≡ true)について考える. 準定全性(2)と busyR(r, t^k_{S1}) ≡ true より, Q ≜ initiated_uop(r, t^k_{S1}) が値をもち, 帰納法の仮定より, Q' ≜ last_uop(t^k), type(Q'), refID(Q') (∵ type(Q') ≠ "OP/PT") が値をもつ. このことと公理ST2, UE2から, stateS1(t_2^{k+1}), UP_executing(t_2^{k+1}) は値をもつ.

公理 UE2 より，UP-executing(t_2^{k+1}) ≡ false なら，UP-executing(t^k) ≡ false であることが必要である．公理 LU2 より，

$$last\_uop(t_2^{k+1}) \equiv last\_uop(t^k) \quad (2.3)$$

故，last_uop についても，t_2^{k+1} に対して導出全性が成り立つ．又，式 (2.3) より，命題 (5) は帰納法の仮定に帰着できる．

以上より，導出全性 (1) 及び命題 (5) が成り立つ．

（導出全性証明終）

2.3 System-0 と File-system の関係

System-0 においても，又 File-system (FS) においても，利用者プロセスから直接観測できるのは，利用者プロセスが初期状態以降出してきたリクエストの系列とバッファの内容である．ここでは，この点に着目して両者の関係を述べる．

(1) リクエスト系列．利用者プロセスが“いつ”，“どのバッファを”見たかを明示するため，バッファ b の内容を見るという利用者操作 ACCB(b) を追加して考える．リクエスト系列の仕様を図2.9に示す．リクエストの空系列を nullUOP，リクエスト系列 α にリクエスト ξ を連接したもの（α の次に ξ が出されることを意味する）を $\xi\cdot\alpha$ と書く．又，二つのリクエスト ξ, ξ' が同じか否かの述語を eqUOP(ξ, ξ') と書く．

(2) リクエスト系列と状態との関係．System-0 では，リクエスト系列 α に対して，初期状態以降，α 中のリクエストを右から（但し，WAIT と ACCB は除く）順次出した結果得られる（System-0 の）状態は一意に決まる．その状態を state0(α) と書くと，state0 は図2.10(a)のように定義される．一方，FS では，（リクエストが出される順は α によって指定されるが）R/W リクエストの完了のタイミングにより，一意ではない．初期状態以降 α 中のリクエストを右から（但し，ACCB は除く）順に出すことにより，FS 状態が ρ になり得るか否かの述語を reachable(α, ρ) と書く．reachable(α, ρ) は，ρ に至るまでの状態遷移において，待ち状態のときリクエストを出したり，未完了の R/W リクエス

```
SPEC Sequence-of-UOP

 INCLUDE Requests-and-user-operations;
 SORT    seqUOP;

 CONSTRUCTOR
    nullUOP:                -> seqUOP,
    .       : uop, seqUOP   -> seqUOP,

    ACCB    : bid           -> uop;

 OP [O-function]
    eqUOP   : uop, uop      -> bool;

 LET BINARY-UOP := {OPENW OPENR WRITE READ PUTB};
 LET UNARY-UOP  := {CLOSE WAIT ACCB};

 EQ1*: FOR EACH Q IN BINARY-UOP,
   {eqUOP(Q(i,j), Q(i',j')) == (i=i') and (j=j');
    FOR EACH Q' IN BINARY-UOP - {Q},
       eqUOP(Q(i,j), Q'(i',j')) == false;
    FOR EACH Q' IN UNARY-UOP,
       eqUOP(Q(i,j), Q'(i')) == false};
 EQ41*: FOR EACH Q IN UNARY-UOP,
   {eqUOP(Q(i), Q(i')) == (i = i');
    FOR EACH Q' IN UNARY-UOP - {Q},
       eqUOP(Q(i), Q'(i')) == false;
    FOR EACH Q' IN BINARY-UOP,
       eqUOP(Q(i), Q'(i',j')) == false};

 T8: type(ACCB(b)) == "AC";

END;
```

図 2.9　リクエスト系列の仕様

```
SPEC Relations-between-SeqUOP-and-System0;
 INCLUDE Sequence-of-UOP, System-0;
 OP      state0: seqUOP -> sys;
 S00: state0(nullUOP) == InitialS;
 S01: state0(q.seq) ==
    if (type(q) = "WT") or (type(q) = "AC")
       then state0(seq)
    else Execute(q, state0(seq));
END;
```

(a) 関数 state0 の定義

```
SPEC Relations-between-SeqUOP-and-FileSystem;
 INCLUDE Sequence-of-UOP, File-system;
 OP      reachable: seqUOP, fsys -> bool;
 RC0: reachable(nullUOP, InitialFS)       == true;
 RC1: reachable(nullUOP, Issue(q, s))     == false;
 RC2: reachable(nullUOP, Complete(r, s)) == false;
 RC3*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r) WAIT(r)
    WRITE(r,b) READ(r,b) CLOSE(r) PUTB(b,rec)},
   {reachable(Q.seq, InitialFS) == false;
    reachable(Q.seq, Issue(q, s)) ==
       UP-executing(s) and eqUOP(Q, q)
       and reachable(seq, s);
    reachable(Q.seq, Complete(r, s)) ==
       busyR(r,stateS1(s)) and reachable(Q.seq,s)};
 RC24*: reachable(ACCB(b).seq, s) ==
    UP-executing(s) and reachable(seq, s);
END;
```

(b) 関数 reachable の定義

図2.10 リクエスト系列とシステム状態との関係

トがないのに"完了"が行われたことがないときに限り真となるように定義されている（図2.10(b)）

(3) バッファの使い方．バッファの"整合性"を保つためには，例えば，リクエスト READ(r, b) を出した後，WAIT(r) または READ(r, b') ($b \neq b'$) を出して，バッファ b の内容が確定していることを確認してからそのバッファにアクセスしなければならない．リクエスト系列 α において，整合性が保たれるようなバッファの使い方がなされているか否かの述語を wellB(α) と書く．

R/W リクエストの処理のための管理システムによるバッファへのアクセスが，そのリクエストの前処理開始時から後処理完了までの期間随時行われると考える．wellB(α) は，reachable(α, s) ≡ true なる任意の FS 状態 s に対して，初期状態以降 s に至るまで，各 b を引数としてもつ R/W リクエストが出されて以降，そのリクエストが完了するまでの期間に，b を引数とする R/W リクエスト，PUTB，ACCB が行われない（すなわち，同じバッファへのアクセスが常に直列的である）ときに限り真となるように定義されている†（図2.11 参照）．図2.11 で，freeB(b, α) は，reachable(α, s) ≡ true なる任意の s において，バッファ b を引数とする未完了の R/W リクエストが存在しないときに限り真となる述語である．また，FREED(b, r, α) は，freeB(b, WAIT(r)·α) が真となるための必要十分条件を略記したものである．rname(b, α) は，α 中で b を指定した

---

† 整合性保存のためならバッファからの読み出し同志は並列的でもよいが，ここでは簡単のため，すべて直列的としている．

```
SPEC How-to-use-buffers;

 INCLUDE Sequence-of-UOP;

 OP wellB:       seqUOP -> bool,
    freeB: bid, seqUOP -> bool,
    rname: bid, seqUOP -> rid;

 WB0: wellB(nullUOP) == true;
 WB1*: FOR EACH Q IN {WRITE(r,b) READ(r,b)
    PUTB(b,rec) ACCB(b)},
    wellB(Q.seq) == wellB(seq) and freeB(b, seq);
 WB5*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r) WAIT(r)
    CLOSE(r)},  wellB(Q.seq) == wellB(seq);

 LET FREED(b, r, seq) :=
    if freeB(b, seq) then true
    else (r = rname(b, seq));

 FR0: freeB(b, nullUOP) == true;
 FR1*: FOR EACH Q IN {WRITE(r,b') READ(r,b')},
    freeB(b, Q.seq) ==
    if b = b' then false else FREED(b, r, seq);
 FR3*: FOR EACH Q IN {CLOSE(r) WAIT(r)},
    freeB(b, Q.seq) == FREED(b, r, seq);
 FR5*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r)
    PUTB(b,rec) ACCB(b)},
    freeB(b, Q.seq) == freeB(b, seq);

 RN1*: FOR EACH Q IN {WRITE(r,b') READ(r,b')},
    rname(b, Q.seq) ==
    if b = b' then r else rname(b, seq);
 RN3*: FOR EACH Q IN {OPENW(f,r) OPENR(f,r)
    CLOSE(r) WAIT(r) PUTB(b,rec) ACCB(b)},
    rname(b, Q.seq) == rname(b, seq);

END;
```

図 2.11　関数 wellB の定義

最後の（最もたの） R/W リクエストの参照名を表す．

(4) FS では，同じファイル名または参照名を引数にもつリクエストについては，出された順に完了していく．FD および DCB の 0-関数は，各ファイル名および参照名について，出されたリクエストの“いずれ”が“どの順に”完了したかで値が決まる．一方，前述の wellB の定義より，wellB(α) ≡ true なら，同じバッファ識別子を引数とする R/W リクエスト，PUTB，ACCB は出された順に完了するから，これらがその完了時に参照するバッファの内容は，従って，ファイルの内容も（完了のタイミングに依らず）常に一致している．このことと，wellB(ACCB(b)·α′) ≡ true ならば α′ で b を引数とする最後のリクエストは reachable(ACCB(b)·α′, s) ≡ true なるすべての FS 状態で完了していることから，そのようなすべての s における contentB(b, stateB(stateS1(s))) の値は一致する．すなわち，α = ACCB(b)·α′ とおくと，

wellB(α) ≡ true 且つ reachable(α, s_i) ≡ true (i = 1, 2) ならば，

contentB(b, stateB(stateS1(s_1))) ≡ contentB(b, stateB(stateS1(s_2)))　(2.4)

(5) FS 状態において，R/W リクエストが出されるといつも直ちに完了するような状態を考える．リクエスト系列 α に対するそのような FS 状態を stateFS(α) と書く．stateFS は，形式的には図 2.12 で定義される．定義より明らかに，

reachable(α, stateFS(α)) ≡ true　(2.5)

```
SPEC Sequential-state-of-FS;
 INCLUDE Sequence-of-UOP, File-system;
 OP      stateFS: seqUOP -> fsys;
 FS0: stateFS(nullUOP) == InitialFS;
 FS1: stateFS(q.seq) ==
    if type(q) = "AC" then stateFS(seq)
    else if type(q) = "R/W" then
       Complete(refID(q), Issue(q, stateFS(seq)))
    else Issue(q, stateFS(seq));
END;
```

図 2.12 関数 stateFS の定義

stateFS(α) では，生じめ，リクエストはその順に処理が完了していくから，各射影関数 stateX (X ∈ {B, F, FD, DCB}) について，

$$stateX(stateS1(stateFS(\alpha))) \equiv stateX(state0(\alpha)) \quad (2.6)$$

(2.6) より，System-0 の 0-関数はすべて，FS 状態 stateFS(α) と System-0 状態 state0(α) とで同じ値をとることがいえる．このことと，(2.4) および (2.5) から，次の性質がなり立つ．α = ACCB(b)·α′ とする．

$$wellB(\alpha) \equiv true \text{ 且つ } reachable(\alpha, s) \equiv true$$

$$\Rightarrow contentB(b, stateB(stateS1(s))) \equiv contentB(b, stateB(state0(\alpha))) \quad (2.7)$$

(2.7) は，wellB に従ってバッファにアクセスする限り，利用者プロセスの見るバッファの内容は，System-0 と FS で常に等しいことを表している．なお，FS で未完了のリクエストがすべて完了すれば，射影関数を除く System-0 のすべての 0-関数の値も，System-0 と FS とで一致することもいえる．

2.4 結言

ファイル管理システムの論理レコードレベルでの2つの記述例，System-0 及び File-system (FS) を示し，それらについて無矛盾性および準完全性が成りえることを示した．4つの部分仕様，Buffers，Files，FD 及び DCB は前章定理 1.6 の前提条件を満たす B-順序機械として記述されているが，System-0 又は FS 全体としては，状態成分を表す N-ソートがあり，そのままでは B-順序機械ではない．しかし，N-ソートが1つという条件を除いて定理 1.6 の前提条件を満たしており，その記述の階層性を利用して，1.5 に示したと同様の方法で"表"による実現を行うことができる．又，記述の階層化をやめ，公理系を，システム状態の遷移後の 0-関数値の変化という形に書き直せば，比較的簡単な変換で B-順序機械に直すことができる．

System-0 と FS の関係は，"外"から見えない内部の詳細化のみではなく，R/W リクエストへの同期機能の追加や新たなリクエスト (WAIT) の追加など，機構化の都合による ("外"から観測可能な部分の) 仕様の変更を含んでいる．データタイプ等の抽象的な記述とその機構化という関係において，このような例も少なくないと思われる．

第3章　あるクラスの項書き換え系の効率のよい実行

3.1　序言

B-仕様を始め，一般に代数的記述言語には，意味の定義が簡潔であり，検証が言語と同じ枠組の中で行えるなどの利点があるが，有効な実行方法がないという欠点がある．本章では，Church-Rosser の性質を満たし，項書き換え系とみなせるようなB-仕様の一種，“完全順序項書き換え系”における効率のよい計算方法について議論する．

完全順序項書き換え系は，N-ソートに関する制限がないという点を除いて，1.5に述べたB-順序機械に等しい（その意味でB-順序機械を含む）．又，“strongly sequential term rewriting systems with constructors[12]”に含まれる．完全順序項書き換え系では，書き換え規則（公理）の左辺に一定の制限が加えられており，そのため，項を標準項に書き換えていくための“必須書き換え場所”を容易に決定することができる．一方，例えばBackus の FP 系[16]に比べて，書き手がデータ構造を自由に定義できる．2章に示したファイル管理システムを始め，HDLC手順の記述[9]が完全順序項書き換え系として得られており，他の応用にも十分答えられると思われる．ここで述べる計算方法を strongly sequential term rewriting systems with constructors のクラスに拡張することはできるが，そのクラスより大きなクラスの項書き換え系を，それと等価な完全順序項書き換え系に変換できる[23]こと，説明の簡単さから，完全順序項書き換え

系について考察を行う．

項書き換え系の基底部の定数記号全体の集合を Σ^0 と書く．項書き換え系が Church-Rosser の性質をもち，且つ基底部の定数記号を書き換える規則を含まないとすれば，項書き換え系 $\mathcal{D}$ と変数 $x_1, \cdots, x_n$ を含む項 t との対に対して，$(\Sigma^0)^n$ から Σ^0 への n 変数（部分）関数 $COMP[\mathcal{D}, t]$ が次のように定義される．

> $(c_1, \cdots, c_n) \in (\Sigma^0)^n$ に対して，項 t に現れている変数 x_i $(1 \leq i \leq n)$ に c_i を代入して得られる項（その項を $t(c_1, \cdots, c_n)$ と書く）に $\mathcal{D}$ の書き換え規則を可能な限り適用して，ある定数記号 $c \in \Sigma^0$ に書き換えられるとき（仮定から一意），すなわち，$t(c_1, \cdots, c_n) \xrightarrow{*} c$ なる $c \in \Sigma^0$ が存在するとき，
>
> $$COMP[\mathcal{D}, t](c_1, \cdots, c_n) \triangleq c$$
>
> と定義し，それ以外は未定義とする．

なお，ここでは簡単のため単一ソートの場合で議論する．多ソートの場合への拡張は容易である．

完全順序項書き換え系 $\mathcal{D}$ の非基底部と項 t との対を，関数 $COMP[\mathcal{D}, t]$ を定義するための（代数的な）プログラムと考え，$\mathcal{D}$ と t から，$COMP[\mathcal{D}, t]$ の計算を行うような（手続き的）プログラムへ変換する（それをコンパイルと呼ぶ）方法を示す．基底部の定数および関数は，目的プログラムを記述するプログラミング言語に組み込まれている基本データタイプの値や基本演算を表しているとする．

$COMP[\mathcal{D}, t]$ の効率のよい計算は次の方針に従って行われる．

(1) 同一項の計算の重複を回避するため，項の有向アサイクリックグラフ（dagと略記する）表現を利用し，dagの書き換えを導入する．

(2) dagの書き換えは必須な場所でのみ行う．

(3) 必須書き換え場所をみつけるための計算など，書き換えを実際に行わなくても，dagから分かることはコンパイル時に行う．

(4) dagをそのまま複写することは避け，適当なデータ構造と，書き換え規則に対応した関数手続を用意して，書き換えを関数手続の実行という形で行う．

上記(1)及び(2)を3.3に，又(3)及び(4)を3.4に述べる．一般に，この種の言語の処理系では，不要になった記憶域の回収方法が問題となるが，関数手続の終了時にその関数手続が使用した記憶域を回収できる，すなわち，記憶域を（関数手続の）内部変数化することが可能な項書き換え系のクラスを3.5に示す．

3.2 定義

3.2.1 項書き換え系

Σ^n を n 変数関数記号の可算集合，X を変数記号の可算集合とし，Σ を $\bigcup_{n\geq 0}\Sigma^n$ と置く．Σ^0 に属する記号を定数記号（又は単に定数）と呼ぶ．Σ と X から構成される項の集合を $T_\Sigma(X)$ と書く．

代入 σ は，$T_\Sigma(X)$ から $T_\Sigma(X)$ への，次のような写像である．

$$\sigma(f(t_1, \cdots, t_n)) = f(\sigma(t_1), \cdots, \sigma(t_n))$$

項書き換え系 $\mathcal{R}$ を対 $(\Sigma, \mathcal{E})$ で表す．ここで，$\mathcal{E}$ は $\alpha == \beta$ $(\alpha, \beta \in T_\Sigma(X))$ の形をした書き換え規則（単に規則と呼ぶこともある）の可算集合である．規則の右辺 β に現れる変数は左辺 α に現れていなければならない．

$T_\Sigma(X)$ 上の関係 $\rightarrow_{\mathcal{R}}$ 及び $\overset{*}{\rightarrow}_{\mathcal{R}}$ を 1.2.2 のように定義する．$t \rightarrow t'$ は，規則の適用により t が t′ に書き換えられることを表す．$\overset{*}{\rightarrow}$ は $\rightarrow$ の推移反射閉包である．項 t に対して，$t \rightarrow t'$ なる項 t′ が存在しないとき，t を標準項と呼ぶ．$t \overset{*}{\rightarrow} t_0$ なる標準項 t_0 が存在するとき，t は標準項 t_0 をもつという．項書き換え系が Church-Rosser の性質をもてば，t の標準項は（もし存在すれば）一意である．

3.2.2 完全順序項書き換え系

$x_1, x_2, \cdots, y_1, y_2, \cdots$ を異なる変数記号とする．項書き換え系 $\mathcal{R} = (\Sigma, \mathcal{E})$ において，$\Sigma - \Sigma^0$ が有限集合であり，Σ が二つの集合 Σ_C および Σ_D に分割され（$\Sigma = \Sigma_C \cup \Sigma_D$，$\Sigma_C \cap \Sigma_D = \emptyset$），$\Sigma_D$ に属する関数記号が次の3つのタイプに分割されるとき，$\mathcal{R}$ を完全順序であるという．

(1) タイプ1： 左辺が $f(\cdots)$ の形の規則はちょうど一つあり，その左辺は $f(x_1, \cdots, x_n)$ の形をしている．

(2) タイプ2： 左辺が $f(\cdots)$ の形の規則は $m_f \geq 1$ 個あり，その左辺はそれぞれ

$$f(g_j(y_1, \cdots, y_{n_j}), x_2, \cdots, x_n)$$

$n_j \geq 0$，$1 \leq j \leq m_f$，$g_j \in \Sigma_C$，$g_i \neq g_j$ $(i \neq j)$ の形をしている．

(3) タイプ3： 左辺が $f(\cdots)$ の形の規則の左辺は，$f(c_1, \cdots, c_n)$，$c_i \in \Sigma^0$ $(1 \leq i \leq n)$ の形をしており，その右辺も定数であり，且つ同じ左辺をもつ規則は複数個存在しない．

Σ_D の関数記号は，その関数値が規則によって定義されており，Σ_C の記号の関数値は規則から直接定義されない．後者を構成子と呼ぶ．以下，定数はすべて構成子であると仮定する．

左辺の形が上記タイプ1又は2であるような規則を非基底部，タイプ3の規則を基底部という．基底部の規則は可算無限個あってもよいが，非基底部の規則は有限個でなければならない．基底部の関数記号は，既に何らかの方法で定義された関数を表し，基底部の規則はその定義表であると考える．簡単のため，すべての $(c_1, \cdots, c_n)$

$\in (\Sigma^0)^n$ について，$f(c_1, \cdots, c_n)$ を左辺にもつ規則が存在するものとする．

タイプ2に関する条件は，変数でない引数を第 i_f 引数に（$1 \leq i_f \leq n$），すなわち，左辺を

$$f(x_1, \cdots, x_{i_f-1}, g_j(y_1, \cdots, y_{n_j}), x_{i_f+1}, \cdots, x_n)$$

の形に拡張することができるが，関数記号の引数位置の交換で上記条件(2)を満たすように変換できることから，簡単のため，第1引数に限定している．又，変数でない引数の数が一つという制限も本質的なものではない．例えば，

$$f(g_1, g_2, y) == t$$

という規則は，新たな関数記号 f_{g_1} を導入して，次の二つの規則に変換することができる．

$$f(g_1, x, y) == f_{g_1}(x, y)$$

$$f_{g_1}(g_2, y) == t$$

この種の変換に関して，strongly sequential systems with constructors[12] を真に含むクラスの項書き換え系を完全順序項書き換え系に変換する方法が既に知られている．[23] なお，完全順序項書き換え系自体は，基底部が無限でもよいという点を除いて，strongly sequential systems with constructors のクラスに含まれる．

3.3 必須書き換え節点における dag の書き換え

$\mathcal{R} = (\Sigma, E)$ を完全順序項書き換え系とする．最も外側の関数記号が構成子であるような項の集合を $T_\Sigma^C(X)$ と書く．すなわち，

$$T_\Sigma^C(X) \triangleq \{ f(\cdots) \in T_\Sigma(X) \mid f \in \Sigma_C \}$$

$\mathrm{COMP}[\mathcal{R}, t]$ を計算するために，より一般化された問題である，$t \xrightarrow{*} t'$ なる $t' \in T_\Sigma^C(X)$ を（もしあるなら）一つ求める問題に対する効率のよいアルゴリズムを示す．

3.3.1 項の dag 表現と dag の書き換え

項を表現するために，唯一つの根をもつラベルつきの連結有向アサイクリックグラフ（以下，dag と呼ぶ）$D = (V, A)$ を用いる．各節点 $v \in V$ に記号 $f \in \Sigma \cup X$ がラベルとして付けられ（v のラベルを $label_D(v)$ と書く），v の出射枝の数は f の引数の数 n（$f \in \Sigma^0 \cup X$ のとき $n = 0$）に等しく，出射枝には 1 から n までの整数が付けられ，i が付けられた枝の終点を v の第 i 子と呼び，$son_D(v, i)$ と書く．v 自身および v から到達可能なすべての節点とそれらの間の枝からなる部分グラフを $dag_D(v)$ と書く．$dag_D(v)$ が表す項（$term_D(v)$ と書く）は以下のように定義される．$label_D(v) = f$ とする．

(1) $f \in \Sigma^0 \cup X$ なら，$term_D(v) \triangleq f$．

(2) $f \in \Sigma^n$ $(n \geq 1)$ なら，$t_i \triangleq term_D(son_D(v, i))$ $(1 \leq i \leq n)$ とおくと，

$$term_D(v) \triangleq f(t_1, \cdots, t_n)$$

dag D および変数に対する定数の代入 σ（すなわち，$\sigma(x) \in \Sigma^0$）に対して，D のラベルを $\sigma(x)$ で置き換えて得られる dag を $\sigma(D)$ と書く．D の根を v とすると，

$$term_{\sigma(D)}(v) = \sigma(term_D(v))$$

が成立する．以下，混同のない限り，上記の記号中の添字 D を省略する．

各規則 $r = \alpha_r == \beta_r \in \mathcal{E}$ に対して，根 v_{r0} をもち，$term(v_{r0}) = \beta_r$ なる dag D_r が与えられているものとする．dag D の節点 v に対して，$term(v) = \sigma(\alpha_r)$ なる代入 σ が存在するとき，節点 v に規則 r が適用可能である，あるいは v は（規則 r の）書き換え節点であるという．完全順序項書き換え系では，定義から明らかに，v に適用可能な規則は一意に定まる．α_r が基底部の規則でない場合，α_r の左から i 番目の変数 x_i に代入されるべき節点を $sub(v, i)$ と書く．$\alpha_r = f(x_1, \cdots, x_n)$ なら，

$$sub(v, i) \triangleq son(v, i),$$

$\alpha_r = f(g(x_1, \cdots, x_k), x_{k+1}, \cdots, x_n)$ の場合，$1 \leq i \leq k$ なら，

$$sub(v, i) \triangleq son(son(v, 1), i),$$

$k < i \leq n$ なら，

$$sub(v, i) \triangleq son(v, i-k+1).$$

節点 v における dag D の書き換えとは，次のようにして新しい dag D' を作る過程をいう．（図 3.1 参照）

(1) D_r と同形の dag D_r'（根を v_0' とする）を作る．

(2) v の各入射枝 (w, v) に対して枝 (w, v_0') を作り，v および v を端点とする枝をすべて消す．

(3) 各 x_i について，ラベル x_i をもつ D_r' の各節点 v' に対し，v' と v' のすべての入射枝 (w', v') を消

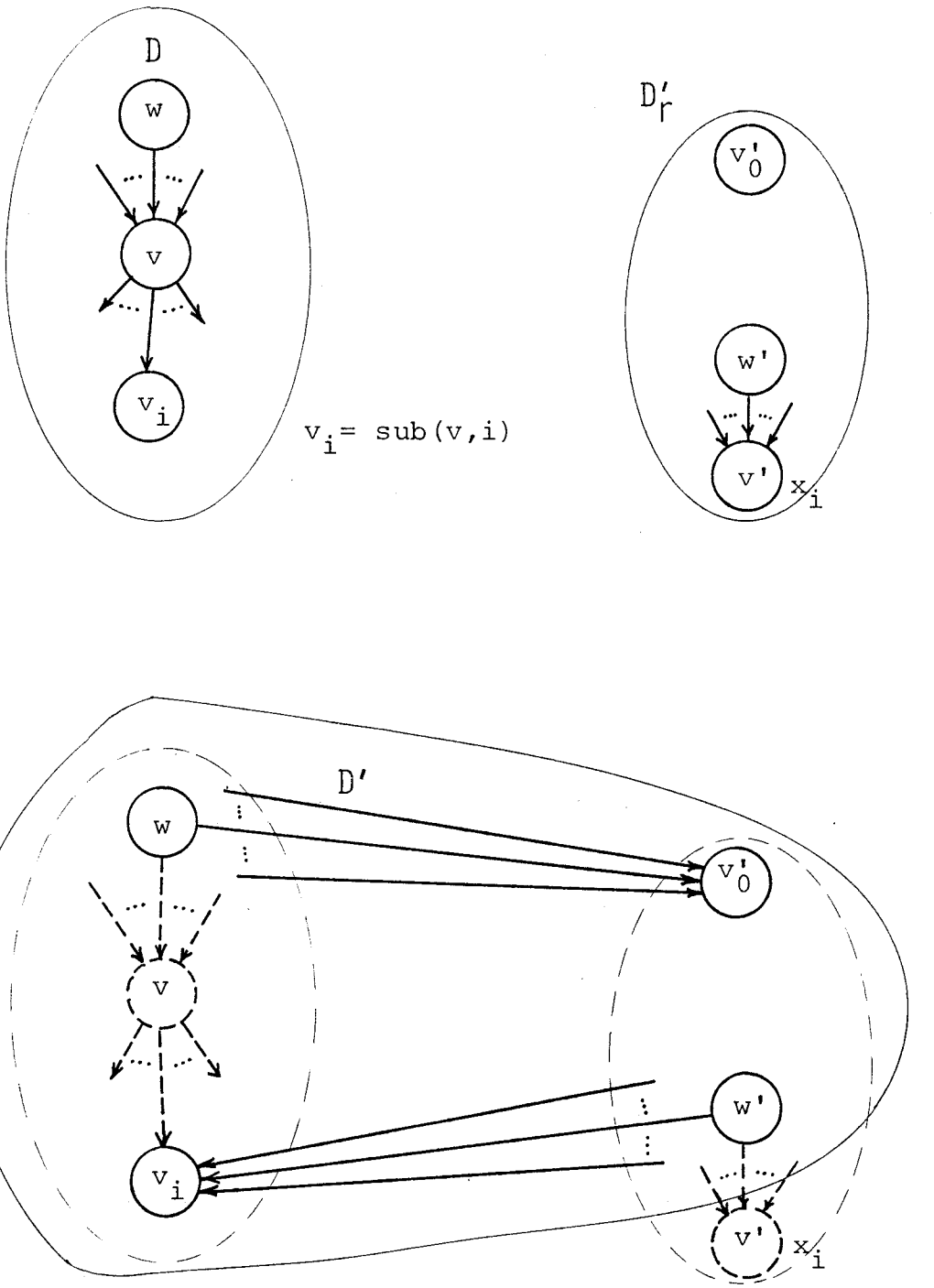

図 3.1　dag の書き換え

し，代りに，枝 $(w', sub(v, \iota))$ を作る．

簡単のため，D' の節点 v' を（その D の）対応する節点名 v で表す．上記の D，D' の関係を

$$D \xrightarrow{v} D' \quad (\text{又は} \quad D \to D')$$

と書き，$\to$ の反射推移閉包を $\xrightarrow{*}$ で表す．$D \xrightarrow{*} D'$ が成立するとき，D は D' に書き換えられるという．

3.3.2 必須書き換え節点

節点 v に対して，$term(v) \xrightarrow{*} t'$ なる項 $t' \in T_\Sigma^C(X)$ が存在しないとき，v を停止節点と呼ぶ．ラベルが構成子でない節点 v が停止節点とならないためには，次の(1)～(3)のいずれかが成立する必要がある．v の子を（存在すれば）$v_1, \cdots, v_n$ と書く．

(1) v は書き換え節点である．

(2) v のラベルがタイプ2の関数記号 f の場合，$dag(v_1)$ は構成する g（但し $f(g(\cdots)\cdots)$ の形の左辺が存在する）を根のラベルとしてもつ dag に書き換えられなければならない．

(3) v のラベルがタイプ3の関数記号 f a 場合，すべての v_i について，$dag(v_i)$ は根のラベルが定数の dag に書き換えられねばならない．従って，$label(v_i) \in \Sigma_C - \Sigma^0$ なる v_i が存在してはいけない．

$dag(v)$ を，根のラベルが構成子の dag に書き換えるためには，上記(2)の v_1 又は(3)の v_i のラベルが構成子または定数でなければ，まずそれらの節点で dag の書き

換えを行う必要がある．そのような節点（v自身も含む）をvの必須節点と呼ぶ．vの必須節点の集合 $need_D(v)$（添字Dは明らかな場合省略される）は以下のように定義される．

(1) vのラベルが構成子ならば，$need(v) \triangleq \emptyset$.

(2) vのラベルが変数またはアトム1の関数記号ならば，$need(v) = \{v\}$.

(3) vのラベルがアトム2の関数記号の場合．(a-1) $label(v_1) = g \in \Sigma_C$ 且つ左辺が $f(g(\cdots)\cdots)$ の形の規則が存在しないか，(a-2) $need(v_1) = \emptyset$ 且つ $label(v_1) \notin \Sigma_C$ ならば，$need(v) \triangleq \emptyset$. (b) どもなくば，

$$need(v) \triangleq \{v\} \cup need(v_1)$$

(4) vのラベルがアトム3の関数記号の場合．(a-1) $label(v_i) \in \Sigma_C - \Sigma^0$ 又は (a-2) $need(v_i) = \emptyset$ 且つ $label(v_i) \notin \Sigma_C$ なる v_i がなければ，$need(v) \triangleq \emptyset$. (b) どもなくば，

$$need(v) \triangleq \{v\} \cup \left(\bigcup_{i=1}^{n} need(v_i)\right)$$

need(v)に属する節点が書き換え節点であるとき，その節点を（vの）必須書き換え節点という．

〔例〕 次のような階乗関数fの規則を考える．

$$f(x) == \text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1)$$

ここで，=，*，−は通常の記法に従う比基底部の関数，if then elseは，図2.1(5)に示されるような規則をもつif関数である．上記規則の右辺が図3.2で与えられるとき，$need(v_0) = \{v_0, v_1, v_2\}$，$need(v_5) = \{v_2, v_5, v_6\}$.

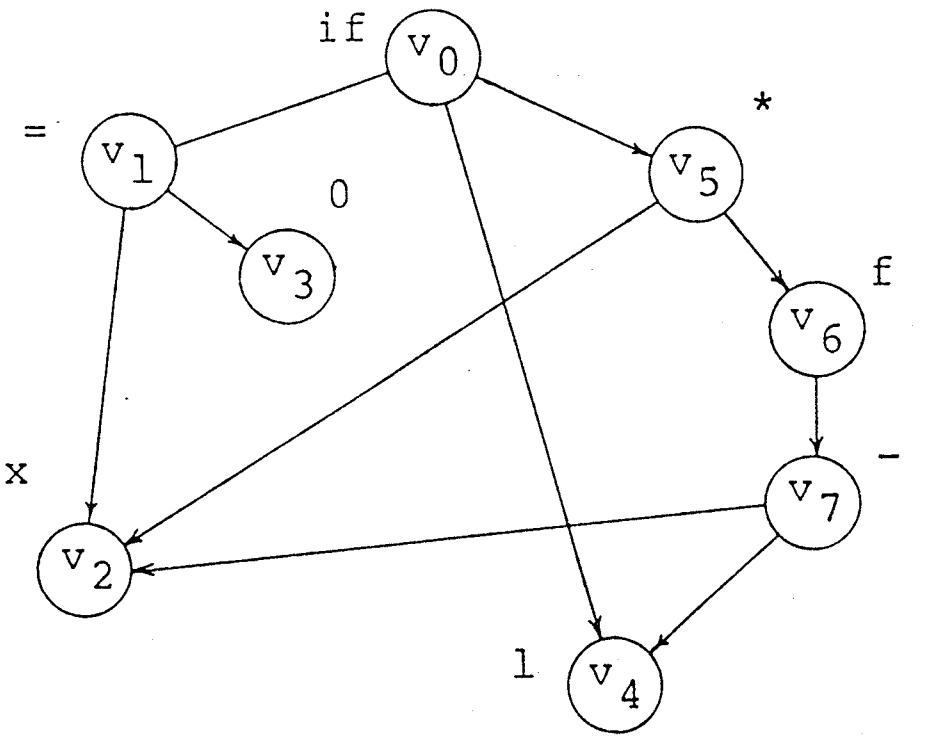

図 3.2　項 if x = 0 then 1 else x * f(x-1)
を表す dag

〔補題 3.1〕　Dを，その根がvの dag とする．

(1)　$need(v) = \emptyset$ 且つ $label(v) \notin \Sigma_C$ ならば，vは停止節点である．

(2)　v'をvの必須書き換え節点とする．Dから，根のラベルが構成子であるような dag へ至るどのような書き換えにおいても，v'における換き換えが行われねばならない！†

(3)　vの必須節点が停止節点ならばvも停止節点である．

(4)　v'をvの必須書き換え節点，$D \xrightarrow{v'} D'$とする．D'でv'が停止節点なら，Dでもv，v'は停止節点であり，さもなくば，

$$need_{D'}(v) = (need_D(v) - \{v'\}) \cup need_{D'}(v')$$

（証明）　前述の議論より，(1)～(3)は明らかであろう．$t_i = term_{D_i}(v_i)$（v_iはD_iの根）$i = 1, 2$とするとき，dag の書き換えの定義より，

$$t_1 \xrightarrow{*} t_2 \quad \Leftrightarrow \quad D_1 \xrightarrow{*} D_2$$

従って，完全順序項書き換え系が Church-Rosser の性質をもつことから，このような dag の書き換えも Church-Rosser の性質をもつ．このことと，一度根のラベルが構成子に書き換えられた dag は，以降どのような書き換えが行われようと根のラベルは変わらないということから，Dを書き換えていって，v，v'のラベルが構成子に書き換わるのなら，D'に対しても同じことがいえる（図 3.3）．従って，D'でv'が停止節点なら（D'でvも停止節点であり），Dでもv，v'は停止節点である．

---

† 必須書き換え節点は strongly needed redex[12] に対応する．

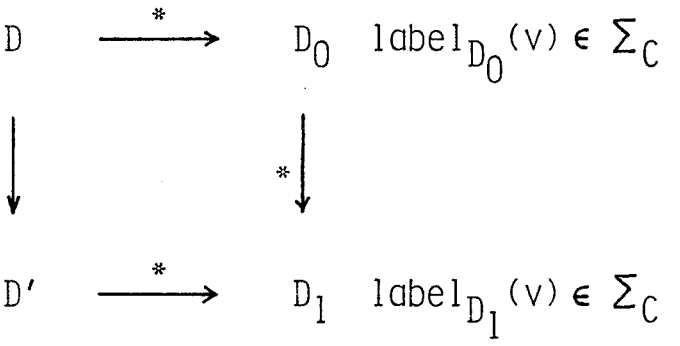

図 3.3　補題 3.1 証明中の状況

v'は停止節点でないとする．書き換えの結果，消えにものは，えの（Dの）節点v'とそれに隣接する枝のみであるから，v'を必須節点にもにない節点については，その必須節点の集合はDからD'への書き換えによって変化しない．又，v'がDで書き換え節点であることから$need_D(v') = \{v'\}$が導出されるから，v'を必須節点にもつ節点については，v'のかわりに$need_{D'}(v')$の各節点が必須節点として追加される．従って次式が成りたつ．

$$need_{D'}(v) = (need_D(v) - \{v'\}) \cup need_{D'}(v')$$

（証明終）

3.3.3 C－必須書き換え節点でのdagの書き換え

節点$v' \in need(v)$に対して，v'から到達可能な節点が（v'の外に）$need(v)$に存在しないとき，v'を（vの）C－必須節点と呼ぶ．$need(v) \neq \emptyset$ならば，vのC－必須節点が少なくとも一つ存在する．C－必須節点v'は，そのうべルが変数でなければ，書き換え節点である．このとき，v'をC－必須書き換え節点と呼ぶ．

d_0を，変数が含まれない項を表すdag，v_0をその一つの節点とする．d_0を，$label_{d^*}(v_0) \in \Sigma_C$なるdag d^*に書き換えるアルゴリズムを示す．

$label_{d_0}(v_0) \in \Sigma_C$なら$d^* \triangleq d_0$．そうでなく，$v_0$の必須節点が存在しなければ（補題1(1)より，$v_0$は停止節点であり）$d^*$は存在しない．以下，$label_{d_0}(v_0) \notin \Sigma_C$且つ$need_{d_0}(v_0) \neq \emptyset$を仮定する．$v_1$を，$d_0$における$v_0$のC－必須節点（変数を含まないことから書き換え節点でもある）の一つとすると，補題1(2)より，d_0はv_1で書き換え

らねばならない．d_1 を $d_0 \xrightarrow{v_1} d_1$ なる dag とする．

補題1(4)より，d_1 で v_1 が停止節点と分かれば，d^* は存在しない．そうなければ，

$$need_{d_1}(v_0) = (need_{d_0}(v_0) - \{v_1\}) \cup need_{d_1}(v_1) \qquad (3.1)$$

$need_{d_1}(v_1)$ が空（すなわち，$label_{d_1}(v_1) \in \Sigma_C$）なら，次の書き換えのために $need_{d_1}(v_0) = need_{d_0}(v_0) - \{v_1\}$ から v_0 の C－必須節点を選ぶ．$need_{d_1}(v_1)$ が空でなければ，v_1 の C－必須節点（(3.1)より，v_0 の C－必須節点でもある）の中から一つを選ぶ．

$$d_0 \xrightarrow{v_1} d_1 \xrightarrow{v_2} \cdots \xrightarrow{v_i} d_i \to \cdots$$

を，上述のようにして選ばれた C－必須節点での書き換えの系列とする．d^* が存在するなら，上記書き換え系列中に次の(1)および(2)を満たす dag d_k が存在する．

(1) v_i $(2 \leq i \leq k)$ は d_{i-1} での v_1 の C－必須節点である．

(2) $label_{d_k}(v_1)$ は構成子である．

各書き換え $d_{i-1} \xrightarrow{v_i} d_i$ $(1 \leq i \leq k)$ は $dag_{d_{i-1}}(v_1)$ の内部にのみ影響を与える．v_1 が dag d_0 の v_0 に関する C－必須節点であることから，$need_{d_0}(v_0) - \{v_1\}$ に属する節点は $dag_{d_{i-1}}(v_1)$ には含まれない．従って，$need_{d_k}(v_1)$ が空であることから，

$$need_{d_k}(v_0) = need_{d_0}(v_0) - \{v_1\}$$

以上より，d^* は（もし存在するなら）次に示す関数 Reduce(d_0, v_0) で与えられる．Reduce では，補題1(1)または(2)の十分条件から，根が停止節点である（d^* は存在しない）ことが分かった dag を上で表す．

[関数 Reduce (d, v)]

(1) $d = \perp$ ならば, Reduce := $\perp$. 以下, $d \neq \perp$ とする.

(2) 節点 v のラベルが構成子ならば, (d が目標の dag であり) Reduce := d.

(3) 節点 v のラベルが構成子でない場合, まず, $need_d(v)$ を求める.

(3.1) $need_d(v) = \emptyset$ ならば, (補題 3.1 より v は停止節点であり) Reduce := $\perp$.

(3.2) $need_d(v) = \{v\}$ ならば, (v は書き換え節点であり) v に規則を適用し, 得られた dag を d' (すなわち, $d \xrightarrow{v} d'$) とすると,

Reduce := Reduce (d', v).

(3.3) それ以外の場合 ($need_d(v)$ に複数個の要素がある場合), $need_d(v)$ のすべての節点からなる次のような系列 $v^{(1)}, \cdots, v^{(m)}$ を一つ選ぶ.

dag d には, $v^{(i)}$ ($1 \leq i < m$) から $v^{(j)}$ ($i < j \leq m$) へ至る道はない.

$d^{(0)} := d$ と置き, 各 $i = 1, 2, \cdots, m$ について,

$$d^{(i)} := \text{Reduce}(d^{(i-1)}, v^{(i)})$$

とおくと,

Reduce := $d^{(m)}$.

(関数 Reduce 終)

C－必須書き換え節点での書き換えは，（必須書き換え節点の中では“下”のものから書き換えが行われるが，一般の書き換え節点を考えると）“上”優先の書き換えといえる．“下”の書き換え節点での書き換えが優先されると，“値”が存在するのにその値を求めることができないことがある．項書き換え系では，“上”を先に書き換えるために，“下”の（書き換え可能な）部分項の数が増えることがあり，“上”優先の書き換えが最適であるとは限らない．しかし，dagの書き換えの場合，そのようなことがなく，以下に示すように最適なものになる．

変数を含まない項を表すdag dに対して，$d \xrightarrow{*} d^*$ 且つ $label_{d^*}(v_0) \in \Sigma^0$（$v_0$は$d$の根）ならば，$d^*$は，一般に，節点$v_0$だけからなる連結成分と他の幾つかの連結成分からなるdagである．v_0だけからなる連結成分を考えると，明らかに，一意であるが，他の連結成分は，dからd^*へ至る書き換えの系列により，一意ではない．以下，最終的に得られたd^*については，（v_0のみからなる他の連結成分を除く）他の連結成分の違いを無視し，d^*は一意であると考える．

〔定理3.1〕　dを，その根がv_0で，変数を含まない項を表すdagとする．

$$d \xrightarrow{*} d^* \text{ 且つ } label_{d^*}(v_0) \in \Sigma^0$$

なるdag d^*が存在するならば，d^*は$Reduce(d, v_0)$として得られ，且つ$Reduce(d, v_0)$で行われるdagの書き換えの回数は，dからd^*へ至るすべての書き換え系列中で最も少ない．

(証明)　d^*が得られるまでの最小の書き換え回数に関する帰納法による．

(i)　書き換え回数が0，すなわち，$label_d(v_0) \in \Sigma^0$なるdag dについては，明らかに，Reduce(d, v_0)でも書き換えが行われず，直ちに$d^* = d$が得られる．

(ii)　最小k回の書き換えでd^*が得られるすべてのdag d'について，Reduce(d', v_0)により，k回の書き換えでd^*が得られると仮定する．最小$k+1$回の書き換えでd^*が得られるdag dについて考える．

$$d_{k+1} \xrightarrow{v_{k+1}} d_k \xrightarrow{v_k} \cdots \xrightarrow{v_{i+1}} d_i \xrightarrow{v_i} d_{i-1} \longrightarrow \cdots \xrightarrow{v_1} d_0 \qquad (3.1)$$

を$d = d_{k+1}$から$k+1$回の書き換えで$d^* = d_0$へ至る一つの書き換え系列とする．Reduce(d, v_0)で最初に書き換えられた節点をvとすると，vが必須書き換え節点であることから(3.1)の書き換え系列中にvは現れている．最初に現れるvをv_i（$v_i = v$でiが最大）とする．v_iがdag d_{k+1}で書き換え節点であることから，v_iのうベルはdag d_j（$k+1 \geq j \geq i$）で構成するでなく，従って，d_{k+1}からd_iへ至る書き換えはv_iに依存しない．又，v_iでの書き換えも節点v_iとそれに隣接する枝が変わるだけである．このことから，$d_{k+1} \xrightarrow{v_i} d'_k$なるdag d'_kについても，

$$d_{k+1} \xrightarrow{v_i} d'_k \xrightarrow{v_{k+1}} d'_{k-1} \xrightarrow{v_k} \cdots \xrightarrow{v_{i+1}} d'_{i-1}$$

なる書き換え系列が存在し，且つ$d_{i-1} = d'_{i-1}$．従って，d'_kはk回の書き換えでd^*へ至り，又，d_{k+1}が最小$k+1$回の書き換えでd^*に至るという仮定から，d'_kがk回以下の書き換えでd^*に至ることはない．帰納法の仮定より，Reduce(d'_k, v_0)は最小k回の書き換えでd^*を得る．

Reduce(d'_k, v_0) が，（3.3）で $need_{d'_k}(v_0)$ の節点の系列を選ぶとき，$need_{d'_k}(v_i)$ に属する節点をその節点より先に並べれば，以下の操作は，Reduce(d_{k+1}, v_0) で最初の（v_i における）書き換えが行われた後と同じである．従って，Reduce(d_{k+1}, v_0) は $k+1$ 回の書き換えで d^* を得る．

以上より，Reduce(d, v_0) は最小の書き換え回数で d^* を得る．　　（証明終）

3.4 dag からプログラムへの変換

完全順序項書き換え系 $\mathcal{R} = (\Sigma, \mathcal{E})$ 及び項 $t \in T_\Sigma(X)$ を表す dag D_0 が与えられたとき，それらを通常の（手続き的）プログラムに変換（コンパイル）する方法について述べる．もに現れる変数はプログラムの入力変数に対応しており，各変数に入力データ（定数）が与えられたとき（それを表す代入を ρ とする）のプログラムの実行は，dag $d_0 = \rho(D_0)$ の必須書き換え節点における書き換えの繰り返しに対応する．

3.4.1 項書き換え系の基底部分

定数（構成子）は，整数，ブール値，文字など通常のプログラム言語に組み込まれている基本データタイプの値に対応しているとし，有限長の配列の一状態も一つの定数であると考える．書き換え規則の基底部は，プログラミング言語に組み込まれている加算，比較演算などの基本演算の定義表であると考える．規則 $f(c_1, \cdots, c_n) = c$ を適用して書き換えることは，定数値 $c_1, \cdots, c_n$ に対して，f に対応するプログラミング言語の基本演算 OP_f を実行することに対応する．簡単のため，定数構成子と対応する基本データタイプの値を同一視する．以下，目的プログラムの記述のためには PASCAL 風の表現を用いる．

3.4.2 目的プログラムの実行過程

Σの非基底部の規則 r の右辺は dag $D_r = (V_r, A_r)$ で表されているとし，D_r の根を v_{r0}，

$$V_r^C \triangleq \{ v \in V_r \mid label(v) \in \Sigma_C \}$$
$$V_r^D \triangleq V_r - V_r^C - \{ v_{r0} \}$$
$$V^C \triangleq \bigcup_r V_r^C$$
$$V^D \triangleq \bigcup_r V_r^D$$

と置く．なお，同じ変数をラベルとしてもつ節点は同一 dag に複数個存在しないものとする．

$$d_0 \rightarrow d_1 \rightarrow \cdots \rightarrow d_k \rightarrow \cdots$$

を，必須書き換え節点における一つの書き換えの系列とする．d_k の節点 w は，えから d_0 に存在した節点か，あるいは $d_{k'}$ $(0 \leq k' < k)$ の節点 w' に規則 r が適用され，D_r が複写されたとき新たに作られた節点である．そこで，複写された各 dag に（複写ごとに異なった）名前をつけ，節点 w を，w が作られたときの D_r の複写の名前 p と D_r の対応する節点名 v との対 (p, v) で表す．dag の書き換えでは，えの節点 w' を消し，複写された D_r の根を（便宜上）w' で表したが，ここではえの w' および新たな D_r の根が共に存在するものとし，両者を異なった節点として扱う．ラベルが変数の節点も，同様に，消されないものとする．

d_k の必須書き換え節点の一つに非基底部の規則 r を適用するとき，D_r の複写を次のように行う．

(I) 次のような構造（レコード構造）をもつ変数Cを導入する．V^C の各節点 v（g 含 label(v)，g の引数の数を n とする）に対応する要素があり（v をそのセレクタとして使う），C.v はプログラムの実行開始時に

"$g($son$(v,1), \cdots,$ son$(v,n))$"

に初期化される．

(II) 規則 r に対して，V_r^D の各節点 v に対応する要素†（そのセレクタを v とする）からなるレコード型のデータタイプ W_r を導入する．規則 r を適用するとき，まず，タイプ W_r の変数のために記憶域を新たに割り当てる．それへのポインタ p を，前述の D_r の複写の名前として用いる．便宜上，dag D_0 にも，同様に，データタイプ W_0 が定義され，プログラムの実行開始時にタイプ W_0 の変数が割り当てられるものとする．

各 p↑.v（$v \in V^d$）の内容は，(1) "null"，(2) 基本データタイプの定数‡ a，又は (3) 対 (p', v')（p' はポインタ，$v' \in V$）のいずれかである．(1) は節点 (p, v) のラベルが構成子に書き換わってないことを，(2) はラベルが定数 a に等しいことを表し，(3) の場合，v のラベルが変数でなければ，$v' \in V^C$ であり，節点 (p, v) が構成子をラベルにもつ節点 (p', v') に書き換えられたことを表す．C.v' = "$g(v_1, \cdots, v_n)$" とすると，(p', v') のラベルは g で，その第 i 子は (p', v_i) である．v のラベルが変数の場合，（dag の書き換えで）その変数に代入されるべき節点（そのラベルが定数 a 場合，その定

---

† V_r^D の全節点に対して準備する必要はない（3.4.3 参照）．

‡ 配列など一つのセルに格納することが妥当でないような定数もあるが，基本データタイプの実現には触れない．

数）を表す．後述の関数列 F_v（が返す値）のタイプは上記(2)または(3)である．

(III) ラベルが構成子でない根 v_{r0} および入口節点（後述）と呼ばれる節点 $v \in V_r^D$ に対して，3.3.3に述べた Reduce(d, (p, v))（dは現時点のdag）にほぼ対応する† 関数列 $F_v(p)$ を作る．効率のため，実行時に $need_d((p, v))$ を求める代りに，コンパイル時に $need_{D_r}(v)$ を計算しておく．$need_{D_r}(v) = \phi$（$label(v) \notin \Sigma_c$ 故，vは停止節点）の場合，F_v は停止命令のみからなる．$need_{D_r}(v) \neq \phi$ の場合，$need_{D_r}(v)$ に属する節点を $v^{(1)}, \cdots, v^{(m)}$（但し，$v^{(i)}$（$1 \leq i < m$）から $v^{(j)}$（$i < j \leq m$）へ至る道はないとする．F_v は，$i = 1, \cdots, m$ について，この順番で CODE(p, $v^{(i)}$) を実行する．以下，CODE(p, $v^{(i)}$) が行う操作を説明する．

(i) $v^{(i)}$ のラベルが変数でない場合，もし $p\uparrow.v^{(i)} \neq$ "null" なら‡，節点 $(p, v^{(i)})$ のラベルは既に構成子に書き換わっており，何もしない．$p\uparrow.v^{(i)} =$ "null" なら，（CODE(p, $v^{(i)}$) に制御が移った時点では $(p, v^{(i)})$ は必須書き換え節点になっており）以下述べるように，適用規則の選択と実行を行う．$f = label(v^{(i)})$ と置く．

(1) fがタイプ3の関数記号の場合．fに対応するプログラミング言語の基本演算を OP_f と書く．簡単のため，OP_f は，引数に定数でないものが一つでもあれば停止すると仮定する．このとき行うべき操作は，$OP_f(c_1, \cdots, c_n)$

---

† Reduceでは常にC－必須節点が書き換わるが，F_v ではそうとは限らない．

‡ ここでは，簡単のため，すべての $v^{(i)}$ について "null" 判定を行う．"null" 判定の省略については3.4.3に述べる．

（ここで，c_k（$1 \leq k \leq n$）は $v^{(i)}$ の第 k 子 v_k のラベルが定数ならその定数を，さもなくば，$p\uparrow.v_k$ の内容を表す）を求め，$p\uparrow.v^{(i)}$ に代入することである．但し，$i=m$，すなわち，$CODE(p, v^{(i)})$ が F_v の行う最後の処理である場合，$p\uparrow.v^{(i)}$ の代りに F_v に代入する（F_v への代入は，その値を F_v の関数値として返すことを意味する）．以下，$i=m$ の場合の処理は同様とする．

(2) f がタイプ1の関数記号であるか，又は f がタイプ2の関数記号で且つ $v^{(i)}$ が D_r で書き換え節点になっている場合．$v^{(i)}$ に適用可能な規則は一意であり，それを規則 r' とする．このとき F_v は，まず (II) に述べたタイプ $W_{r'}$ の変数を新たに割り当て（それへのポインタを p' とする），ラベルが変数の節点を除くすべての $u \in V_{r'}^D$ について，$p'\uparrow.u$ を "null" に初期化する．次に，規則 r' の左辺の左から j 番目の変数をラベルにもつ $D_{r'}$ の節点を u_j とすると，各 j について，

$$p'\uparrow.u_j := param(p, sub(v^{(i)}, j))$$

ここで，$param(p, v)$ は，v のラベルが定数ならその定数を表し，ラベルが定数以外の構成子であるか，又は構成子でなく $p\uparrow.v$ = "null" ならば (p, v) を，さもなくば，$p\uparrow.v$ の内容を表す．

その後，$D_{r'}$ の根 $v_{r'0}$ のラベルが定数ならその定数を，定数以外の構成子なら $(p', v_{r'0})$ を $p\uparrow.v^{(i)}$ に代入し，それ以外の場合，関数手続 $F_{v_{r'0}}(p')$ を呼び出し，$F_{v_{r'0}}$ から返された値を $p\uparrow.v^{(i)}$ に代入する．

(3) f がタイプ2且つ $v^{(i)}$ が D_r では書き換え節点でない場合．適用規則は，実行時に $(p, v^{(i)})$ の第1子（(p, v_1) とする）のラベルが構成子に書き換わるまで

分からない．左辺が $f(\cdots)$ の形の規則を $r_1, \cdots, r_k$，規則 r_k の左辺を $f(g_k(x_1, \cdots, x_{n_k}), x_{n_k+1}, \cdots, x_n)$ とする．CODE$(p, v^{(i)})$ に制御が移った時点では，既に (p, v_1) のラベルは構成子に書き換わっており，$p\uparrow.v_1$ の内容は定数 a 又は (p'', v') $(v' \in V^c)$ の形になっている．前者の場合，cons := a とし，後者の場合，$C.v'$ の内容を "$g(v_1', \cdots, v_{n'}')$" とすると，cons := g と置く．cons がどの g_k とも等しくなければ，適用規則はなく，F_v は停止する．cons $= g_k$ ならば，規則 r_k が適用可能であり，(2) と同様に（但し，$r' = r_k$），ラベル W_{r_k} の変数の割り当て及びその初期化，$F_{v_{r_k}0}(p')$ の呼び出しなどを行う．(2) と異なる所は，$p'\uparrow.u_j$ の値が，$1 \leq j \leq n_k$ $(n_k = n'$ である$)$ については param(p'', v_j') に設定され，$n_k < j \leq n$ については param$(p, son(v^{(i)}, j - n_k + 1))$ に初期化されることである（図 3.4 参照）．

(ii) $v^{(i)}$ のラベルが変数 x の場合．$v^{(i)}$ が D_0 の節点ならば，入力 x を $p\uparrow.v^{(i)}$ に代入する．$v^{(i)}$ が D_0 の節点でない場合，(i)(2)，(3) で述べたように，F_v が呼び出される直前に，$p\uparrow.v^{(i)}$ に定数または (p', v') の形の値が代入されている．$p\uparrow.v^{(i)} = (p', v')$ 且つ $v' \in V^D$ なら，F_v はまず $F_{v'}(p')$ を呼び出し，返された値を $p'\uparrow.v'$ 及び $p\uparrow.v^{(i)}$ に代入する．それ以外の場合（節点 $(p, v^{(i)})$ は既にラベルが構成子になっており），CODE$(p, v^{(i)})$ では何もしない．

(iv) 手続 F_v は，各 D_r の根（但し，ラベルが構成子でないもの），及び dag の書き換えでいずれかの変数に代入される可能性のある節点 v について準備すればよい．従って，以下に定義する入口節点についてのみ用意すれ

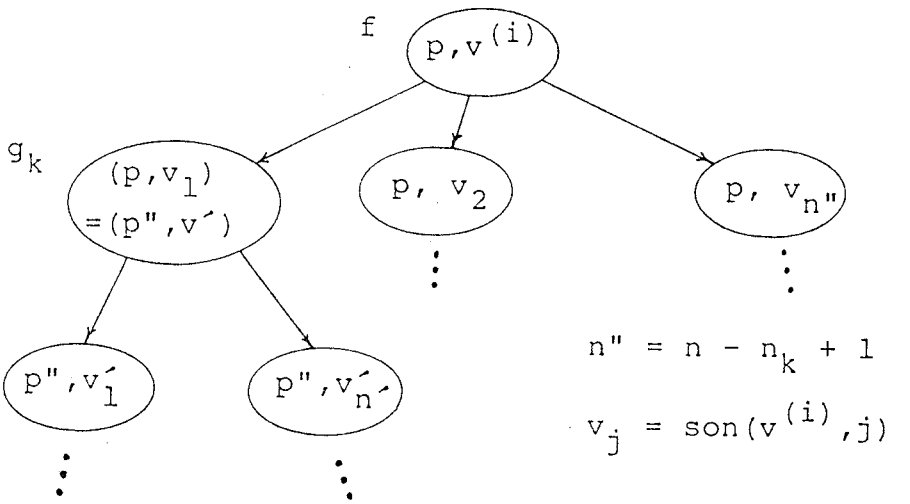

図 3.4　(III)(i)(3) の場合の状況

ば十分である．入口節点は，$label(v) \notin \Sigma_C$ 且つ次の(1)～(3)のいずれかを満たす節点 v として定義される．

(1) $v = v_{r0}$ （D_r の根）

(2) v は，$label(v') \in \Sigma_C$ なる節点 v' の子

(3) 入口節点 v'，枝 (v'', v) が存在して，
$v'' \in need(v')$ 且つ $v \in need(v')$

3.4.3 目的プログラムの改良点

まず，各節点での“null”判定を省略する方法について述べる．D_r の根から（D_r の）節点 v' へ至るすべての道が節点 v を通っているとき，v を v' の支配節点と呼ぶ．入口節点 v が $need(v)$ に属する節点 v' の支配節点ならば，

$$p\uparrow.v = \text{“null”} \Rightarrow p\uparrow.v' = \text{“null”}$$

従って，v が $need(v)$ のすべての v' の支配節点であれば，（$F_v(p)$ は，$p\uparrow.v =$ “null” のときのみ呼び出されるから）各節点 v' で $p\uparrow.v' =$ “null” か否かの判定は不要である．従って，次の(1)及び(2)が成立する．

(1) $F_{v_{r0}}$ では，各節点での“null”判定はいらない．

(2) v をその支配節点としないような節点が $need(v)$ に存在するとき，$need(v)$ を次のような節点の集合 $N_1, N_2, \cdots$ に分割する．

N_k の中に，N_k に属するすべての節点の支配節点 v_k が存在する．

各 N_k について，3.4.2で $need(v)$ から F_v を作ったときと同様にして関数手続 F_{v_k} を作る．F_v と異なる点は，$v' \in N_k$ から N_ℓ の支配節点 v_ℓ への枝が

存在するとき，F_{v_k} において CODE(p, v′) より前に，p↑.v_e の内容が "null" か否かを判定し，"null" のときのみ $F_{v_e}(p)$ を呼び出す操作を行うことと，これら以外では "null" 判定を行わないことである．このようにすれば，"null" 判定を，F_{v_k} の呼び出しのときのみに限定することができる．

上記(2)における分割は，分割のブロック数が小さい程よい．そのような分割を得るためには，まず v を支配節点とする節点の集合 N_1 をもめ，次に N_1 に属さず，且つ N_1 の節点からの入射枝が存在する各節点 v′ について，v′を支配節点とする節点集合 N′, … をもめ，以下，各 N′, … について同様の操作を繰り返せばよい．

又，v_k 以外の節点 $v' \in N_k$ （但し，変数となべルにもつ節点を除く）に対して，p↑.v′ は $F_{v_k}(p)$ の一時記憶用としてのみ使われており，F_{v_k} の内部変数とすることができる．

CODE(p, $v^{(i)}$) (label($v^{(i)}$) $\notin X$) において，適用規則 r′の右辺が，"if関数" の規則のように，一つの変数 x だけからなる場合，タイプ $W_{r'}$ の変数の割り当てから関数手続 $F_{v_{r'0}}$ の呼び出しまでの一連の操作を省略し，直ちに，x へ代入される値（それが節点 (p′, v′), $v' \in V^D$ なら $F_{v'}(p')$ が返す値）を p↑.$v^{(i)}$ に代入することができる．例えば，図3.2において，F_{v_D} は図3.5のようなプログラムになる．図3.5で，(1)のプログラム文は，3.4.2 (iii)(ii)に述べられている操作を表す．

```
function F_V0(p);
   begin
(1)   p↑.V_2 := EVAL(p↑.V_2);
(2)   p↑.V_1 := (p↑.V_2 = 0);
(3)   if p↑.V_1 then F_V0 := 1
                else F_V0 := F_V5(p)
   end
```

図 3.5　改良後の目的プログラム

3.5 記憶域の割り当て制御

関数手続 $F_{v_{r0}}(p)$ は，定数または (p', v) $(v \in V^C)$ を値として返す．C.v の内容を "$g(v_1, \cdots, v_n)$" とすると，$F_{v_{r0}}(p)$ の終了後も $F_{v_k}(p')$ （但し，$v_k \in V^D$）が呼び出される可能性があり（3.4.2 (iii)(ii)参照），従って，$p' = p$ の場合，$F_{v_{r0}}(p)$ が呼び出される直前にタイプ W_r の新たな変数に割り当てられた（p で指される）記憶域を $F_{v_{r0}}(p)$ の終了時に解放することはできない．もし，構成子が定数のみとすると，$F_{v_{r0}}$ が返す値も定数となり，上述のようなことは起こらない．この場合，$F_{v_{r0}}(p)$ の終了時に p が指す記憶域を解放でき，これらの記憶域が割り当てられている変数を $F_{v_{r0}}$ の内部変数とすることができる．このクラスの項書き換え系は，新たに抽象データタイプを定義せず，基本データタイプのみを用いた新たな関数群（プログラム群）の記述に対応する．

今までは，簡単のため，単一ソートの項書き換え系について議論してきたが，以下，多ソート完全順序項書き換え系の次のようなクラスを考える．ソートには，基本データタイプに対応する B-ソートと，新たに導入する抽象データタイプに相当する N-ソートの2種類があるとする．

(1) N-ソートには，そのソートを値域とする定数以外の構成子があってよいが，その ソートの関数 f を定義する（左辺が $f(\cdots)$ の形の）規則は（もしなくても）右辺が変数または定数のもののみとする．

(2) B-ソートについては，規則の右辺に制限はないが，構成子は定数に限られる．

このクラスの項書き換え系では，N-ソートの規則 r については，3.4.3に述べたように，タイプ W_r の変数の記憶域割り当てを行わなくて済み，B-ソートの規則については，前述のように，$F_{V_{r0}}$ の内部変数とすることができる．従って，不要になった記憶域の回収を特別に行う必要はない．

上記のクラスの項書き換え系として，HDLC手順の記述などが得られており[9]，2章のファイル管理プログラムの記述も，階層化をやめるなどの簡単な変更で，上記の2条件を満たすようにできる．

3.6 結言

代数的記述の一つの標準形として，完全順序項書き換え系を導入し，そこでの効率的な計算方法を，効率的なdagの書き換えと，dagの書き換えの効率的な実現の2点に分けて論じた．

最初に，項の“値”が存在するとき，最小の（dagの）書き換え回数でその値を求める書き換えアルゴリズムを示した．そこでは，最も“下”の必須書き換え節点（C-必須書き換え節点）から書き換えを行えば，以前に求めた必須節点の集合は，書き換え後もそのまま使えることなどを利用し，必須書き換え節点も効率よく求めている．

次にdagの構造を，固定のデータ，動的に変化していくデータ，目的プログラム中に埋め込まれた命令で表されるものに分離して表し，dagをそのまま複写することを避け，更に，関数手続の呼び出し機械を有効に利用した“dagの書き換え”方法を示した．そこでは，コンパイル方式を用いて，コンパイル時の計算を増し，実行時の計算を押えている．

最後に，空間効率の点で問題となる不要記憶域について，記憶域を関数手続の内部変数化することでその回収が効率よく行える完全順序項書き換え系の部分クラスを示した．HDLCr順の記述がそのクラスのものとして得られており[9]，2章に示したファイル管理システムの記述も容易にそのクラスの記述に変更できるなど，通常の記述に十分答えられるクラスであると思われる．

結　　論

基底代数Bを前提とし，その拡張としての代数的仕様，B－仕様，を定式化し，“表を用いて表すことが可能な部分クラスとしてB－順序機械を，又，効率のよい実行が可能なクラスとして完全順序項書き換え系を導入した．両者は，その論点の違いから別々に導入されたクラスであるが，B－順序機械のためにN－ソートとB－ソートの区別およびN－ソートが一つという制限があることを除いて，実質的に同じクラスである．これらのクラスは公理あるいは書き換え規則の左辺に一定の制限を加えたものであるが，HDLC手順，順編成ファイル管理システムがそのクラスの記述として得られているなどの点から，実際の応用面において記述性が劣るとは思われない．寧ろ，無矛盾性が保証され，項の構造的帰納法などによる証明が行われ易く，且つ比較的効率的な実行が可能であることなどから，記述の一つの“望ましい”形を与えるものと思われる．

第3章に示した方法は，項の書き換えアルゴリズムとしては効率のよいものであるが，通常の手続き的なプログラムでの実行と比較すると，効率は一般に劣る．例えば，配列など一部要素の値を変更しただけで，配列全体を新たに作らなければならない．一般にはこのような処理が必要であるが，変更後の配列を参照しなければ，あるいはそのように参照順を定めることができれば新たな配列を作り出す必要はなく，元の配列の指定された要素の内容を変更するだけでよい．この外にも，実行効率の改善のために，この種の言語特有の“最適化”手法の開発が必要であるが，代数的記述では，意味の定義が簡明

で明示的であるので，最適化も比較的容易である．

代数的記述言語のもう一つの欠点として，すべてを陽に示さねばならないことによる記述の煩雑さがある．この対策として，第2章の記述例に一部導入したようなif_then_else，インフィックス記法を始めとした項の自由な表現方法を許す枠組および種々の簡略記法の開発が今後の課題として残されている．

謝　　辞

本研究の全過程を通じ，直接理解ある御指導を賜わり，更に論文提出の機会を与えて戴いた大阪大学基礎工学部高忠雄教授に衷心より感謝申しあげます．

本研究に当たり，終始御指導御鞭撻を賜わり，常に励まして戴いた基礎工学部谷口健一助教授に心から深謝申しあげます．

数々の御教示と御助言を戴いた基礎工学部都倉信樹教授，荒木俊郎講師，大阪大学大型計算機センター藤井護助教授，滋賀大学経済学部森将豪助教授，ならびに電子技術総合研究所島居克次博士に心から感謝します．

また，種々の御討論を戴いた基礎工学部萩原兼一助手，大阪大学情報処理教育センター鈴木一郎助手，三菱電機計算機製作所中西通雄氏，高研究室井上克郎氏，東野輝夫氏，関浩之氏に心から感謝します．

文　献

(1) Goguen, J.A., Thatcher, J.W. and Wagner, E.G.: "An initial algebra approach to the specification, correctness, and implementation of abstract data types," IBM Research Report, RC-6487 (1976).

(2) Guttag, J.V.: "The specification and application to programming of abstract data types," Univ. of Tront, Technical Report, CSRG-59 (Sep. 1975).

(3) Wand, M.: "Final algebra semantics and data type extensions," Indiana Univ., Computer Science Dept., Technical Report, No. 65 (July 1978).

(4) Hoffmann, C.M. and O'Donnell, M.J.: "Programming with equations," ACM Trans. on Programming Languages and Systems, Vol. 4, No. 1, pp.83-112 (Jan. 1982).

(5) 和田，荻原，荒木，都倉：“代数的仕様記述に関する一考察”，電子通信学会技術研究報告，AL79-111 (1980-02).

(6) 坂部，稲垣，本多：“部分関数を演算とする抽象データタイプの仕様記述と実現”，電子通信学会技術研究報告，AL80-6 (1980-05).

(7) 杉山，谷口，嵩：“代数的記述言語とその部分言語としての関数的プログラミング言語”，電子通信学会技術研究報告，AL79-99 (1980-01).

(8) 鈴木，杉山，荻原，谷口，嵩，奥井：“プログラムの仕様とその実現化の代数的記述”，電子通信学会技術研究報告，AL78-46 (1978-10).

(9) 森，東野，杉山，谷口，嵩：“HDLC手順の代数的記述”，電子通信学会論文誌（D），J64-D，2，pp. 124－131（86 56－02）．

(10) Rosen, B.K.: "Tree-manipulating systems and Church-Rosser theorems," J. Assoc. Comput. Mach., Vol. 20, No. 1, pp. 160-187 (Jan. 1973).

(11) Huet, G.: "Confluent reductions: abstract properties and applications to term rewriting systems," J. Assoc. Comput. Mach., Vol. 27, No. 4, pp. 797-821 (Oct. 1980).

(12) Huet, G. and Lévy, J-J.: "Call by need computations in non-ambiguous linear term rewriting systems," IRIA Research Report, No. 359 (Aug. 1979).

(13) Ehrich, H-D. and Lipeck, U.: "Proving implementations correct – two alternative approaches," Information Processing 80, pp. 83-88 (1980).

(14) 奥井，鈴木，杉山，萩原，谷口，嵩：“仕様記述間の対応について”，電子通信学会技術研究報告，AL78－79（1979－01）．

(15) 鈴木，杉山，谷口，嵩：“代数的仕様記述における詳細化－特に抽象的順序機械の場合－”，京大数理解析研究所講究録，421号，pp. 92－105（1981－03）．

(16) Backus, J.: "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs," Commun. ACM, Vol. 21, No. 8, pp. 613-641 (Aug. 1978).

(17) Hoffman, C.M. and O'Donell, M.J.: "An interpreter generator using tree pattern matching," Proc. 6th Ann. Sympo. on Principles of Programming Languages, pp. 169-179 (Jan. 1979).

(18) O'Donnell,M.J.: "Computing in systems described by equations," Lecture Notes in Computer Science, No. 58 (1977).

(19) Burstall, R.M.: "Design considerations for a functional programming language," Proc. Infotech State of the Art Conf., pp. 45-57 (1977).

(20) Berry, G. and Lèvy, J-J.: "Minimal and optimal computations of recursive programs," J. Assoc. Comput. Mach., Vol. 26, No. 1, pp. 148-175 (Jan. 1979).

(21) Vuillemin, J.: "Correct and optimal implementations of recursion in a simple programming language," J. Comput. & Syst. Sci., Vol. 9, No. 3, pp. 332-354 (Dec. 1974).

(22) Raoult, J-C. and Vuillemin, J.: "Operational and semantic equivalence between recursive programs," J. Assoc. Comput. Mach., Vol. 27, No. 4, pp. 772-796 (Oct. 1980).

(23) 杉山，井上，高："項書き換え系のある標準形への変換"，電子通信学会技術研究報告，AL81-95 (1982-01).

(24) 鳥居，二木，真野："プログラミング方法論の展望"，情報処理，Vol. 20，No. 1，pp. 22-43 (昭54-01).