制約解消系を備えた関数・論理型言語の処理系とその実装☆

佐々木 重雄† 井田 哲雄††

関数・論理型言語の計算モデルにナローイング計算系がある．ナローイング計算系は，等式を項のドメインにおいて効率良く解くことができるが，算術計算のように特定のドメインに関する等式は数を項にコーディングしないかぎり解くことができない．本論文では，算術の等式を直接解くために，ナローイング計算系に制約解消系を付加した計算系を呈示し，その実装について述べる．まず，ナローイング計算系と制約解消系を組み合わせた計算系を示す．この計算系により，制約付き関数・論理型言語の振舞いが，抽象的な水準で定義される．次に，逐次計算機上での処理系の実装に適した計算系を与える．この計算系ではバックトラックに伴う計算の過程を，状態の遷移により明示的に与える．さらにこの計算系の実装について述べる．特に，ナローイング推論エンジンと制約解消器の通信，および，バックトラックの同期について述べる．我々の実装方式には，1)ナローイングと制約解消を並列に実行することが可能，2)様々な種類の制約解消器を利用可能という特徴がある．

A Constraint Functional Logic Programming Language System and Its Implementation

SHIGEO SASAKI † and TETSUO IDA ††

Recently we have seen a surge of interest in integrating functional and logic programming languages. A promising avenue for the integration is to base the language on a narrowing calculus. The narrowing calculus can solve equations over the domain of terms, but it is not capable of solving arithmetic equations efficiently. To cope with arithmetic equations, we provide an arithmetic constraint solver. This paper discusses a method of combining a narrowing calculus and constraint solver by presenting several new calculi. We start from a calculus that combines a narrowing calculus and constraint solver. We first give the calculus in the form of an inference system, and then elaborate the calculus in order to implement it on a sequential machine. We next describe the implementation of the calculus. Especially we discuss the communication between the narrowing inference engine and the constraint solver and the synchronization of backtracking. The features of our implementation are following. 1) We can develop a parallel implementation which performs narrowing and constraint solving. 2) We can employ various kinds of existing constraint solvers.

1. はじめに

関数型言語と論理型言語を計算モデルのレベルから融合する試みとして，ナローイング計算系に基づく言語の研究が行われている．ナローイングは，等式を効率よく解くための求解方法の一つである．操作的な観点から見ると，ナローイングは，書換え規則による等式の書換えであるが，単一化代入が等式列全体にも適用される[1)~3)]．これにより関数型言語の機能と論理型言語の機能が実現される．

ナローイングは，エルブラン領域で等式を効率よく解くことができる．しかし数のような特定の計算ドメイン上の等式を扱うには適さない．このような等式の求解を効率よく扱うためには，制約解消系を導入するのが有効であることが知られている[4)~6)]．制約は，特定の計算ドメイン上の関係という宣言的な意味を持つため，関数・論理型言語との理論的な親和性も高い．

本論文では，ナローイング計算系と数の求解機構である制約解消系を組み合わせることにより，記号と数を統一的に扱う関数・論理型言語の計算モデルを呈示し，その処理系の実装について述べる．

本論文の構成は以下のとおりである．第2章で，基礎となるナローイング計算系 OINC を示す．第3章で

† 筑波大学工学研究科
Doctoral Program in Engineerings, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electoronics, University of Tsukuba
* 本研究の一部は，文部省 科学研究費 重点領域研究(1)超並列記述系・処理系に関する研究 04235104 および筑波先端学際領域センター(TARA)の支援を受けて行われた．

は，OINCと制約解消系を組み合わせることにより得られる制約ナローイング計算系OINC+Cを示す．この計算系は，制約付き関数・論理型言語の動作を抽象的な水準で記述したものである．第4章では，言語処理系の実現を考慮に入れ，バックトラックの振舞いを記述する計算系（逐次計算系）を示す．ここでは問題を2段階に分けて考える．まずOINCの逐次計算系seq-OINCを示す．次にseq-OINCと制約解消系を組み合せた計算系seq-OINC+Cを与える．第5章では，言語処理系の設計と実装について述べ，第6章で，関連研究との比較を行う．

2. 準　　備

2.1 項

関数記号の集合を$\mathcal{F}$，変数の集合を$\mathcal{V}$とし，項の集合を$\mathcal{T}(\mathcal{F},\mathcal{V})$と表す．$\mathcal{F}$は，ユーザ定義の関数記号$\mathcal{F}_D$，構成子記号$\mathcal{F}_C$，算術記号$\mathcal{F}_A$の直和とする．$\mathcal{F}_A$には，数$(0,1,-1/2$など$)$，演算子$(+,-,\times,\div)$，等号$(=)$，不等号$(>,\geq,\ldots)$が含まれる．$\mathcal{T}(\mathcal{F}_A,\mathcal{V})$に含まれる項を算術項，$\mathcal{T}(\mathcal{F}_C\cup\mathcal{F}_A,\mathcal{V})$に含まれる項をデータ項と呼ぶ．ユーザ定義の関数記号をルートに持つ項を関数項と呼ぶ．制約付き計算系において，数のドメインの項とそれ以外の項を区別するために，項に型をつける．型がβの項tをt^βと表す．aを算術型を表す型定数とする．なお，算術項は算術型であるが，算術型の項は必ずしも算術項ではない．また項の区別はゴールの等式中の項についてのみできればよいので，代入および制約に含まれる項には型を付けない．記述を簡単にするために，項$f(s_1,\ldots,s_n)$を$f(\vec{s})$と略記する．また等式列$s_1=t_1,\ldots,s_n=t_n$を$\vec{s}=\vec{t}$と略記する．型の付いた項の並び$s_1{}^{\alpha_1},\ldots,s_n{}^{\alpha_n}$を$\vec{s}^{\alpha}$と表す．代入$\theta$は，変数$x$から項$t$への束縛$x\mapsto t$の集合で表す．代入$\theta$を項$t$または等式列$E$に適用したものを，それぞれ$t\theta$，$E\theta$と表す．

2.2 ナローイング計算系 OINC

本論文ではいくつかの計算系を提示するが，その基礎となる計算系として，OINC(Outside-In Narrowing Calculus)[7]を用いる．計算系OINCは，ナローイングにより書換え可能な部分項を最左外内(Leftmost Outside-In)[8]の要領で選択する戦略をナローイングに組み込んだものである．この計算系は，簡約における遅延評価に相当する操作をナローイングにおいて実現している．

計算系OINCは，正交な（orthogonal）項書換え系に対して定義され，右辺が変数を含まない正規形である等式の列を操作の対象とする．形式的には，**図1**に示す推論規則の集合として定義される．計算系で操作の対象となる等式の列をゴールと呼ぶ．

[on] 最外ナローイング
$$\frac{f(\vec{s})=t,E}{\vec{s}=\vec{l},r=t,E}\qquad t\notin\mathcal{V},f(\vec{l})\to r\in\mathcal{R}$$
[d] 分解
$$\frac{c(\vec{s})=c(\vec{t}),E}{\vec{s}=\vec{t},E}\qquad c\in\mathcal{F}$$
[v] 変数の除去
$$\frac{t=x,E}{E\theta}\ \text{あるいは}\ \frac{x=t,E}{E\theta}$$
ただし$\theta=\{x\mapsto t\}$

図1 OINCの推論規則
Fig. 1 The inference rules of OINC.

[on-s] 厳格等式の最外ナローイング
$$\frac{f(\vec{s})\equiv t,E}{\vec{s}=\vec{l},r\equiv t,E}\qquad f(\vec{l})\to r\in\mathcal{R}$$
あるいは
$$\frac{s\equiv f(\vec{t}),E}{\vec{t}=\vec{l},s\equiv r,E}\qquad f(\vec{l})\to r\in\mathcal{R}$$
[d-s] 厳格等式の分解
$$\frac{c(\vec{s})\equiv c(\vec{t}),E}{\vec{s}\equiv\vec{t},E}\qquad c\in\mathcal{F}_C$$
[im-s] 厳格等式のイミテーション
$$\frac{c(\vec{s})\equiv x,E}{\vec{s}\theta\equiv\vec{y},E\theta}\qquad c\in\mathcal{F}_C$$
あるいは
$$\frac{x\equiv c(\vec{s}),E}{\vec{y}\equiv\vec{s}\theta,E\theta}\qquad c\in\mathcal{F}_C$$
ただし$\theta=\{x\mapsto c(\vec{y})\}$

[t-s] 自明な厳格等式の除去
$$\frac{x\equiv y,E}{E\theta}$$
ただし$\theta=\begin{cases}\{x\mapsto y\}, & x\not\equiv y\\ \emptyset, & x\equiv y\end{cases}$

図2 s-OINCの推論規則
Fig. 2 The inference rules of s-OINC.

OINCの計算が最左外内ナローイングの戦略を実現していることから，健全性を見るのは容易である．またOINCは完全であることが知られている[7]．ここでナローイング計算系が完全であるとは，任意の正規形の解に対してより一般的な解を計算できるということである．

関数・論理型言語の処理系の計算モデルには，OINCのゴールに厳格等式を扱う機能を付加したs-OINCを用いる．s-OINCは，OINCに**図2**の推論規則を追加したものである．厳格等式$s\equiv t$は，項sとtが同一の変数を含まないデータ項である時に真となる等式である．s-OINCの健全性，完全性についても，文献7)で触れている．

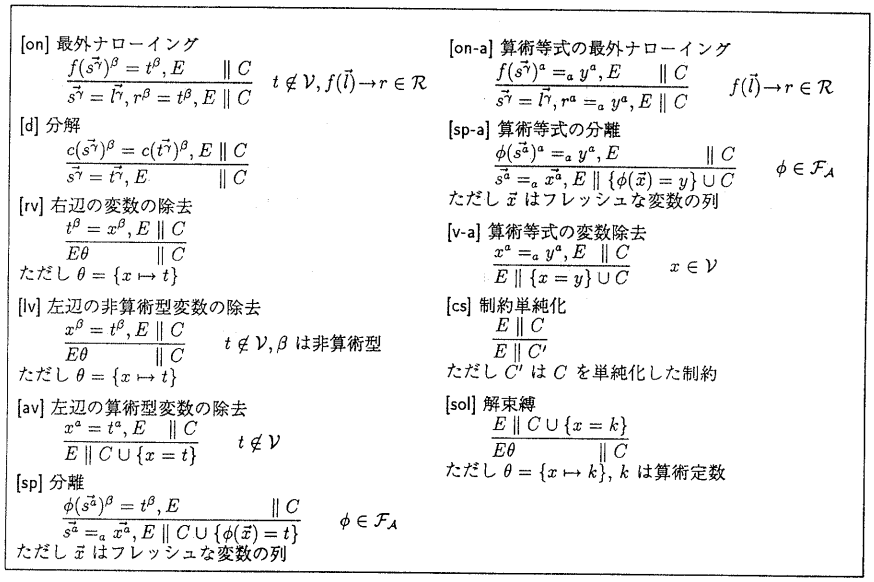


図3 OINC+C の推論規則
Fig. 3 The inference rules of OINC+C.

3. 制約ナローイング計算系 OINC+C

3.1 制約解消系

特定の計算ドメイン上の関係（制約）を解くためのシステムを，制約解消系と呼ぶ．制約解消系は，状態として制約の集合 C を持つ．制約解消系は，新たな制約 c が与えられると，C と c をまとめて，解としてより適切な形に変形し（単純化），その結果である C' を新しい状態とする．本論文では，制約解消系は数のドメインを扱うものとする．

3.2 計算系 OINC+C

次に s-OINC と制約解消系を組み合わせることにより，制約ナローイング計算系 OINC+C を作る．OINC+C は，制約付き関数・論理型言語の計算モデルとなる．

制約解消系は算術項の処理専用のシステムであり，その他の処理，例えばユーザ定義関数の処理はできない．そのため，母体となるナローイング計算系は，制約解消系に算術項以外の項が渡されないことを保証しなければならない．OINC+C では，項を型付けすることにより，制約解消系に渡すべき項とそれ以外の項を区別する．

OINC+C では，ゴールを等式列 E と制約の集合 C の対 $E \| C$ とする．OINC+C を，図3に示す推論規則の集合として定義する*．

[on]と[d]は，それぞれ OINC の同名の推論規則に対応する．これらは，型付きの項を扱うという点を除くと，OINC の推論規則と同じである．

[rv], [lv], [av]は，一辺が変数である等式から代入または制約を生成するための推論規則である．右辺が変数の場合，型によらず左辺の評価を保留し代入を生成する．これを行うのが[rv]である．この操作により遅延評価が実現される．左辺が変数の場合，型によって適用する推論規則が異なる．算術型の場合，この等式は制約解消系に渡すべき式なので[av]を適用する．算術型でない場合，代入を生成するために[lv]を適用する．

[sp]は，一般の項から算術項を分離するための推論規則である．算術型の項には，部分項として関数項が含まれている可能性がある．しかし制約解消系は関数項を処理できないので，算術型の項に含まれる関数項を分離する必要がある．[sp]は，算術型の項 $\phi(s_1,...,s_n)$ の部分項 $s_1,...,s_n$ を左辺に持つ新しい等式を作る．[sp]が作る等式は，ルート記号が $=_a$ である特別な等式（算術等式）である．[on-a],[sp-a],[v-a]は，算術等式に関する推論方式を定義している．これらの規則により $\phi(s_1,...,s_n)$ のすべての部分項が再帰的に処理さ

* これに加えて，厳格等式に対する推論規則を与えれば，実際の言語の計算モデルになる．非算術型の項の間の厳格等式に対する推論規則は，冗長になるのでここでは示さない．算術型の項の間の厳格等式 $s^a \equiv t^a$ は，数の等式 $s^a =_a t^a$ = true（ただし $\equiv_a \in \mathcal{F}_\mathcal{A}$）と同一視するので，必要ない．

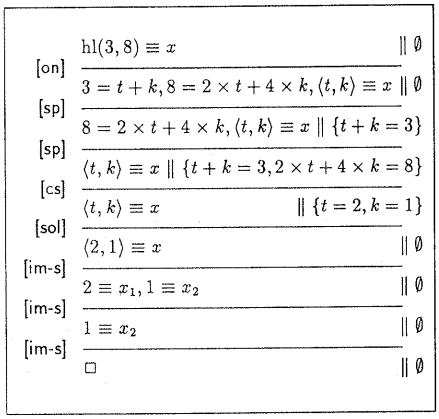


図4 OINC+Cの計算例
Fig. 4 A computation example of OINC+C.

れ，算術項が作られる．

制約解消系に制約を渡す推論規則は，[av],[sp],[sp-a],[v-a]であるが，これらの適用で関数項が制約解消系に渡されることはない．これにより，計算系全体として制約解消系に関数項が渡らないことが保証される．

[cs]は制約解消系が制約を単純化する操作を，[sol]は制約解消系が得た解をナローイング計算系に戻す操作を定義している．

OINC+Cの推論規則のうち，[sp],[sp-a],[cs]を除いた部分は，OINCの推論規則と本質的には同じものである．[sp],[sp-a]も等式の変形として妥当であることは明らかなので，OINC+Cの健全性は[cs]の健全性に依存する．すなわち制約解消系として健全なものを与えればOINC+Cは健全性を持つ．

3.3 OINC+Cの計算例

書換え規則の集合として $\mathcal{R}=\{\mathrm{hl}(t+k,2\times t+4\times k)\rightarrow\langle t,k\rangle\}$ を考える．これは，引数として頭の数と足の数を与えた時，結果として鶴の数 t と亀の数 k の組 $\langle t,k\rangle$ を返す，いわゆる鶴亀算のプログラムである．このとき等式 $hl(3,8)\equiv x$ が与えられると図4に示す計算が行われる．この計算によって解 $\{x\mapsto\langle 2,1\rangle\}$ が得られる．

4. バックトラック付き計算系

次に，処理系の実装のために，より実行レベルに近い計算系を定義する．

OINC+Cは，ナローイングと制約解消が協調して求解を行う系とみなすことができる．本節では，ナローイングと制約解消をそれぞれ独立したプロセスが担当し，この2つのプロセスが互いに協調しつつ計算を進めていく処理系の実装を述べる．ナローイングを担当するモジュールと制約解消を行うモジュールが独立していることには，次の利点がある．(1)本論文で示す通信方式に従う複数の種類の制約解消器が利用可能となる．目的別に制約解消器を交換して用いることができる．(2)ナローイングと制約解消を並列に実行する処理系の実装が可能となる．一般に制約型言語では制約解消に多くの処理時間が費やされることを考えると，このような並列処理は処理系の高速化に有効である．

ナローイング計算系，制約解消系自体が並列処理されるシステムも考えることができるが，本論文では，この2つの系は各々1つのプロセスにより逐次実行されるものとして議論を進める．ナローイング計算系の持つ，書換え規則の選択に関する非決定性は，これをバックトラックにより実現される．

本章では，制約ナローイング計算系にバックトラックの機構を取り込んだ計算系を2段階に分けて設計する．まずバックトラックを扱うナローイング計算系—逐次ナローイング計算系 seq-OINC を与える．これによりバックトラックの振舞いのエッセンスを記述する．続いて目標とする計算系，つまり逐次ナローイング計算系に制約解消系を組み合わせた計算系—逐次制約ナローイング計算系 seq-OINC+C を示す．

4.1 逐次ナローイング計算系 seq-OINC

バックトラックの動作を定めるため，逐次計算系は，計算の状態を明示的に扱う必要がある．逐次計算系 seq-OINC を，状態遷移系として与える．計算系の状態は，選択点情報のスタック S と等式列 E の対によって与えられる．この対を $S\vdash E$ と表記する．特に求解に失敗している状態を $S\vdash\perp$ と表記する．この状態になると，seq-OINC ではバックトラックが引き起こされる．$\psi{:}S$ は先頭が選択点情報 ψ のスタックを表す．選択点情報 ψ は対 $\langle R,E\rangle$ で表される．R は適用可能な書換え規則の集合，E は選択点が作られた時点の等式列である．

逐次ナローイング計算系 seq-OINC を，**図5**に示す，状態 $S\vdash E$ に対する遷移規則の集合として定義する．

OINCの推論規則[on]は，seq-OINCの[ch]と[alt]によって実現される．[ch]は，適用可能な書換え規則を集め，選択点情報を作る推論規則である．[alt]は，集められた書換え規則の中から1つを選び出し，それに従って等式を書き換える．seq-OINCにおいて，[ch]の次には必ず[alt]が選択される．

バックトラックを実現するための推論規則が，[f],[alt],[unw]である．等式の両辺が構成子項で，それらのルートの記号が異なる場合，[f]が適用され，失敗の

[ch] 選択点の生成

$$\frac{S \vdash f(\vec{s}) = t, E}{\langle R_f, [f(\vec{s}) = t, E]\rangle : S \vdash \bot} \quad t \notin \mathcal{V}$$

ただし R_f は、f に関する全ての書換え規則の集合

[alt] 書換え規則の適用

$$\frac{\langle \{f(\vec{l_i}) \to r_i\} \cup R, [f(\vec{s}) = t, E]\rangle : S \vdash \bot}{\langle R, [f(\vec{s}) = t, E]\rangle : S \vdash \vec{s} = \vec{l_i}, r_i = t, E}$$

[v] 変数の除去

$$\frac{S \vdash x = t, E}{S \vdash E\theta} \text{ あるいは } \frac{S \vdash t = x, E}{S \vdash E\theta} \quad x \in \mathcal{V}$$

ただし $\theta = \{x \mapsto t\}$

[d] 分解

$$\frac{S \vdash c(\vec{s}) = c(\vec{t}), E}{S \vdash \vec{s} = \vec{t}, E} \quad c \in \mathcal{F}_c$$

[f] 失敗

$$\frac{S \vdash c(\vec{s}) = c'(\vec{t}), E}{S \vdash \bot} \quad c, c' \in \mathcal{F}_c, c \neq c'$$

[unw] スタックの解放

$$\frac{\langle \emptyset, E\rangle : S \vdash \bot}{S \vdash \bot}$$

図5 seq-OINC の推論規則
Fig. 5 The inference rules of seq-OINC.

$$\begin{array}{ll}
 & \emptyset \vdash f(x,g) = b \\
[ch] & \hline \langle \{f(a,y) \to y, f(b,y) \to b\}, E_1\rangle \vdash \bot \\
[alt] & \hline \langle \{f(b,y) \to b\}, E_1\rangle \vdash x = a, g = y, y = b \\
[v] & \hline \langle \{f(b,y) \to b\}, E_1\rangle \vdash g = y, y = b \\
[v] & \hline \langle \{f(b,y) \to b\}, E_1\rangle \vdash g = b \\
[ch] & \hline \langle \{g \to a\}, E_2\rangle : \langle \{f(b,y) \to b\}, E_1\rangle \vdash \bot \\
[alt] & \hline \langle \emptyset, E_2\rangle : \langle \{f(b,y) \to b\}, E_1\rangle \vdash a = b \\
[f] & \hline \langle \emptyset, E_2\rangle : \langle \{f(b,y) \to b\}, E_1\rangle \vdash \bot \\
[unw] & \hline \langle \{f(b,y) \to b\}, E_1\rangle \vdash \bot \\
[alt] & \hline \langle \emptyset, E_1\rangle \vdash x = b, g = y, b = b \\
[v] & \hline \langle \emptyset, E_1\rangle \vdash g = y, b = b \\
[v] & \hline \langle \emptyset, E_1\rangle \vdash b = b \\
[d] & \hline \langle \emptyset, E_1\rangle \vdash \Box
\end{array}$$

ただし E_1 は $[f(x,g) = b]$, E_2 は $[g = b]$

図6 seq-OINC の計算例
Fig. 6 A computation example of seq-OINC.

状態になる．この次に適用される推論規則は，[alt]または[unw]である．選択点情報に含まれる選択可能な書換え規則の集合が空でないならば，[alt]が適用され，他の書換え規則に従ったナローイングが続行される．空ならば，[unw]の適用により選択点情報が捨てられ，より深いバックトラックが行われる．

次に seq-OINC の計算例を示す．書換え規則として

$$\mathcal{R} = \begin{Bmatrix} f(a,y) \to y, \\ f(b,y) \to b, \\ g \to a \end{Bmatrix}$$

が与えられた時，状態 $\emptyset \vdash f(x,g) = b$ で始まる計算は，図6のようになる．この計算は，状態 $\langle \emptyset, E_1\rangle \vdash \Box$ で終了し，解として $\{x \mapsto b\}$ を得る．

4.2 バックトラック付き計算系のための制約解消系

バックトラック付き計算系の制約解消系は，ナローイングによる求解に失敗した時，あるいは解消系自身による求解に失敗した時にバックトラックを行い，以前の選択点の状態に戻す機構を必要とする．この実現方法を述べる．

バックトラックの振舞いを簡潔に記述するため，制約の集合を選択点情報のスタック S に対応する形で管理する．選択点 ϕ_{i+1} が作られるまでに生成された制約の集合を κ_i と表す．κ_{i+1} には，κ_i に含まれるすべての制約と ϕ_{i+1} が作られた後（もし ϕ_{i+2} が存在すれば，それが作られるまで）の制約がすべて含まれる．対 $\kappa_i \diamond \phi_i$ の列を制約スタック $\mathcal{K}$ と呼び，これを制約解消系の持つ状態とする．

制約解消系に対する操作は，制約の追加，制約単純化，選択点の生成およびバックトラックである．これらを以下のように定めると，逐次計算系のための制約解消系が得られる．制約スタック $\kappa \diamond \psi : \mathcal{K}$ に制約 c を**追加**することにより，新しい制約スタック $(\{c\} \cup \kappa) \diamond \psi : \mathcal{K}$ を得る．制約スタックが $\mathcal{K}$ のとき，選択点 ϕ の**生成**により，制約スタックは $\emptyset \diamond \phi : \mathcal{K}$ になる．**制約単純化**は，$\mathcal{K}$ に含まれる制約の集合 κ_i それぞれに対し，制約単純化を行うこととして定式化される．制約スタックが $\mathcal{K}' : \kappa \diamond \phi : \mathcal{K}$ のとき，選択点 ϕ への**バックトラック**により，制約スタックは $\emptyset \diamond \phi : \mathcal{K}$ になる．

4.3 逐次制約ナローイング計算系 seq-OINC+C

OINC と制約解消系から OINC+C を得るのと同じ考え方で逐次ナローイング計算系 seq-OINC と 4.2 節の制約解消系を組み合わせる．これにより目標とする計算系 seq-OINC+C を得る．

seq-OINC+C の計算の状態は，選択点情報のスタック S，制約のスタック $\mathcal{K}$，ゴール E によって表される．この組を $S \vdash E \,\|\, \mathcal{K}$ と表記する．スタック S は，選択点情報の列である．選択点情報は，選択点識別子 ϕ，選択可能な書換え規則の集合 R，選択点が作られた時の等式列 E の組 $\langle \phi, R, E\rangle$ である．選択点識別子 ϕ が付いている点が，seq-OINC と異なる．これの存在により，選択点 ϕ に対応する制約の単純化に失敗した時，ナローイング計算系と制約解消系の双方が同時に

[ch] 選択点の生成

$$\frac{S \vdash f(\vec{s^\gamma})^\beta = t^\beta, E \quad \| \mathcal{K}}{\langle \psi, R_f, [f(\vec{s^\gamma})^\beta = t^\beta, E] \rangle : S \vdash \bot \| \emptyset \diamond \psi : \mathcal{K}}$$

ただし R_f は、f に関する全ての書換え規則の集合、ψ はフレッシュな選択点識別子

[alt] 書換え規則の適用

$$\frac{\langle \psi, \{f(\vec{l_i^\gamma})^\beta \to r_i{}^\beta\} \cup R, [f(\vec{s^\gamma})^\beta = t^\beta, E] \rangle : S \vdash \bot \| \kappa \diamond \psi : \mathcal{K}}{\langle \psi, R, [f(\vec{s^\gamma})^\beta = t^\beta, E] \rangle : S \vdash \vec{s^\gamma} = \vec{l_i^\gamma}, r_i{}^\beta = t^\beta, E \| \emptyset \diamond \psi : \mathcal{K}}$$

[rv] 右辺の変数の除去

$$\frac{S \vdash t^\beta = x^\beta, E \| \mathcal{K}}{S \vdash E\theta \quad \| \mathcal{K}}$$

ただし $\theta = \{x \mapsto t\}$

[lv] 左辺の非算術型変数の除去

$$\frac{S \vdash x^\beta = t^\beta, E \| \mathcal{K}}{S \vdash E\theta \quad \| \mathcal{K}} \quad \beta \text{ は非算術型}$$

ただし $\theta = \{x \mapsto t\}$

[av] 左辺の算術型変数の除去

$$\frac{S \vdash x^a = t^a, E \quad \| \kappa \diamond \psi : \mathcal{K}}{S \vdash E \| \{x = t\} \cup \kappa \diamond \psi : \mathcal{K}}$$

[d] 分解

$$\frac{S \vdash c(\vec{s^\gamma})^\beta = c(\vec{t^\gamma})^\beta, E \| \mathcal{K}}{S \vdash \vec{s^\gamma} = \vec{t^\gamma}, E \quad \| \mathcal{K}}$$

[f] 失敗

$$\frac{S \vdash c(\vec{s^\gamma})^\beta = c'(\vec{t^\gamma})^\beta, E \| \mathcal{K}}{S \vdash \bot \quad \| \mathcal{K}} \quad c, c' \in \mathcal{F}_\mathcal{C}, c \neq c'$$

[unw] スタックの解放

$$\frac{\langle \psi, \emptyset, G \rangle : S \vdash \bot \| \kappa \diamond \psi : \mathcal{K}}{\sigma : S \vdash \bot \quad \| \mathcal{K}}$$

[sp] 分離

$$\frac{S \vdash \phi(\vec{s^a})^\beta = t^\beta, E \quad \| \kappa \diamond \psi : \mathcal{K}}{S \vdash \vec{s^a} =_a \vec{x^a}, E \| \{\phi(\vec{x}) = t\} \cup \kappa \diamond \psi : \mathcal{K}} \quad \phi \in \mathcal{F}_\mathcal{A}, t \notin \mathcal{V}$$

ただし $\vec{x}$ はフレッシュな変数

[ch-a] 算術等式の選択点生成

$$\frac{S \vdash f(\vec{s^\gamma})^a =_a x^a, E \quad \| \mathcal{K}}{\langle \psi, R_f, [f(\vec{s^\gamma})^a =_a x^a] \rangle : S \vdash \bot \| \mathcal{K}}$$

ただし R_f は、f に関する全ての書換え規則の集合、ψ はフレッシュな選択点識別子

[alt-a] 算術等式における書換え規則の適用

$$\frac{\langle \psi, \{f(\vec{l_i^\gamma})^a \to r_i{}^a\} \cup R, [f(\vec{s^\gamma})^a =_a x^a, E] \rangle : S \vdash \bot \quad \| \kappa \diamond \psi : \mathcal{K}}{\langle \psi, R, [f(\vec{s^\gamma})^a =_a x^a, E] \rangle : S \vdash \vec{s^\gamma} = \vec{l_i^\gamma}, r_i{}^a =_a x^a, E \| \emptyset \diamond \psi : \mathcal{K}}$$

[sp-a] 算術等式の分離

$$\frac{S \vdash \phi(\vec{s^a})^a =_a x^a, E \quad \| \kappa \diamond \psi : \mathcal{K}}{S \vdash \vec{s^a} =_a \vec{x^a}, E \| \{\phi(\vec{x}) = x\} \cup \kappa \diamond \psi : \mathcal{K}} \quad \phi \in \mathcal{F}_\mathcal{A}, t \notin \mathcal{V}$$

ただし $\vec{x}$ はフレッシュな変数

[v-a] 算術等式の変数の除去

$$\frac{S \vdash x^a =_a y^a, E \quad \| \kappa \diamond \psi : \mathcal{K}}{S \vdash E \| \{x = a\} \cup \kappa \diamond \psi : \mathcal{K}}$$

図7 seq-OINC+C の推論規則(1)
Fig. 7 The inference rules of seq-OINC+C(1).

選択点 ψ に戻ることができる.

seq-OINC+C を, 図7, 8 に示す, 状態 $S \vdash E \| \mathcal{K}$ の遷移規則の集合として定義する.

[ch],[alt],[rv],[lv],[d],[f],[unw] は, seq-OINC の推論規則に対応する. [av],[sp],[ch-a],[alt-a], [sp-a],[v-a] は, OINC+C の推論規則に対応するもので, これらによって生成された制約は, スタック $\mathcal{K}$ の先頭に追加される.

[f-c] は, seq-OINC+C に固有の推論規則である. この規則は, 制約解消系で求解に失敗した時の動作を定義している. [cs]を適用した結果, 制約 κ_l が解を持たないことがわかった場合, ψ_{i+1} が作られてから以降の計算は無効であるので, 選択点情報のスタック S と制約スタック $\mathcal{K}$ の両者について, スタックの先頭から ψ_{i+1} までが捨てられる. また等式列を $\bot$ とするので, 続いて[alt]または[unw]が適用され, バックトラック

[cs] 制約単純化

$$\frac{S \vdash E \parallel \mathcal{K}' : \kappa \diamond \psi : \mathcal{K}}{S \vdash E \parallel \mathcal{K}' : \kappa' \diamond \psi : \mathcal{K}}$$

ただし κ' は κ を単純化した制約

[sol] 解束縛

$$\frac{S \vdash E \parallel \{x = k\} \cup \kappa \diamond \psi : \mathcal{K}}{S \vdash E\theta \qquad \parallel \kappa \diamond \psi : \mathcal{K}}$$

ただし $\theta = \{x \mapsto k\}$

[f-c] 制約解消系における失敗

$$\frac{S' : \langle \psi, R, G \rangle : S \vdash E \parallel \mathcal{K}' : \bot \diamond \psi : \mathcal{K}}{\langle \psi, R, G \rangle : S \vdash \bot \qquad \parallel \bot \diamond \psi : \mathcal{K}}$$

図8 seq-OINC+C の推論規則(2)
Fig. 8 The inference rules of seq-OINC+C(2).

が行われる.

5. 処 理 系

seq-OINC+C に基づいて実装された処理系について述べる. 先に述べたように, ナローイングと制約解消をそれぞれ独立したプロセスが行い, この2つが協調して計算を進めるという考えに基づいて, 処理系を実装した.

各プロセスの逐次的動作は, seq-OINC+C によって定められる. これに加えてプロセス間の通信を定めれば, 処理系の動作を定めたことになる.

5.1 プロセス間の通信

実装した処理系の, ナローイングに基づく求解を行うプロセスを推論エンジン, 制約解消を行うプロセスを制約解消器と呼ぶ. 両プロセス間の通信内容は次のようになる.

seq-OINC+C に基づく計算を行う時, 推論エンジンから制約解消器に渡すデータは, (1)制約, (2)選択点情報, (3)バックトラック指令の3つであり, 処理系の実装を考えると, さらに(4)解の要求指令, (5)初期化指令が必要である. 逆に制約解消器から推論エンジンに渡すデータは, (1)解, (2)バックトラック指令の2つである.

5.2 バックトラックの同期

推論エンジンと制約解消器のスタックの一貫性を保つために, バックトラックは両プロセスの間で同期して行われる必要がある.

バックトラックは, 推論エンジンでも制約解消器でも発生する可能性がある. そのため, 二相コミット (two phase commit) に似た機構を用いて, バックトラックを同期する. すなわち, 第1のフェーズでバックトラックを行うことを他方のプロセスに知らせ, バックトラックの競合の無いことがわかった後, 第2のフェーズで実際にバックトラックを行う. もし第1のフェーズでバックトラックの競合があることがわかった場合, 両プロセスが必要としているバックトラックの深さを比較し, より深い方のバックトラックを行う. これにより, 両プロセスのスタックの一貫性が保たれる.

```
hl(T + K, 2*T + 4*K) = crane_turtle(T, K).

fib(N) = fib1(N, 0, 1).
fib1(0, X, Y) = Y.
fib1(N + 1, X, Y) = fib1(N, Y, X + Y).
```

図9 プログラム例
Fig. 9 A program example.

```
| ?- hl(10, 30) = crane_turtle(CR, TU).
CR = 5
TU = 5 ?
yes

| ?- fib(9) = X.
X = 55 ?
yes

| ?- fib(X) = 55.
X = 9 ?
yes
```

図10 実行例
Fig. 10 An execution example.

5.3 処理系の実装

ここまでで示した計算系および通信規約に基づいて, 言語処理系を設計し, 実装した. 実装の目的は, 計算系の正しさ, およびプロセス間の通信やバックトラックの同期の動作の正しさを確認することである. そこで言語処理系の構成を次のようにした. ナローイングを行う推論エンジンと制約解消器を, それぞれ UNIX の別個のプロセスとして実現し, この間は TCP/IP に基づく通信を行う. 推論エンジンは, Prolog で記述されている. 制約解消器は, 制約論理型言語 CLP(ℛ) の処理系に, スタック管理および通信を行うインタフェースを被せたものとして実装した. 図9にこの処理系で実行可能なプログラムの例を示す. これは, 鶴亀算を行うプログラムとフィボナッチ数列を計算するプログラムである. このプログラムの実行

例を図10に示す.

6. 関連研究

関連研究として, Jaffer と Lassez の制約論理型言語 CLP(ℛ)がある[5),9)]. CLP(ℛ)では, SLD レゾリューションと制約解消系を組み合わせることにより, 論理型言語と制約型言語の融合を図っている. Jaffer と Lassez は, CLP(ℛ)の言語処理系について, 推論エンジンと制約解消器の間のデータのやりとり, バックトラックの同期, および, 制約解消器の動作に関する, 技術的な問題点とその解決を述べている.

一方我々は, 推論エンジンと制約解消器を別個のプロセスとみなし, 各プロセスの動作およびプロセス間の通信を抽象的な水準で形式的に定義した. これにより, 複数の種類の制約解消系を使い分けることが可能となる. また並列処理により, 実行性能の向上が期待できる.

ナローイング計算系と制約解消系を組み合せることによって制約付き関数・論理型言語の計算モデルを与える試みは, Darlington と Guo らによっても行われている[6)]. 彼らは, 制約付き関数・論理型言語の計算モデルを形式的に考察しているが, 処理系の実装を考慮した計算モデルを与えてはいない.

Saraswat は, 並列論理プログラミングに制約の概念を導入した, 並列制約プログラミングを提案している[10)]. Saraswat の与える計算モデルは, 並列に実行される複数のプロセスが, 1つの制約解消系と協調しながら計算を進めていくというものである. 制約解消系が独立したプロセスとして実行される点が, 我々の方式と共通である. しかしこれもまた, 処理系の実装を考慮した計算モデルを与えてはいない.

Aït-Kaci らは, 並列論理型言語を拡張し, ソート, 関数, 高階の項などの特徴を持った言語 LIFE を提案している[11)]. LIFE は, 順序付きソートおよび項の間の等式を制約として扱う. 制約, 関数, 論理型プログラミングの機能を統一的な枠組の中で実現しようとする試みは, 我々と共通である. しかし, 基礎となる計算モデルは異なるものであり, また制約として数を特別に扱うことは考えられていない.

7. まとめ

記号と数の求解を行う計算系である制約ナローイング計算系を設計し, それに基づくプログラミング言語の処理系を実装した.

OINC+C は, 遅延ナローイング計算系 OINC と制約解消系を組み合わせて作られた計算系で, 制約付き関数・論理型言語の動作を抽象的な水準で定義したものである. seq-OINC は, バックトラックの振舞いを記述するために計算の状態を明示的に扱うようにした計算系である. seq-OINC に制約解消系を組み合せた計算系が seq-OINC+C である. この2つの計算系は, バックトラックを行いながら逐次的に解の探索を行う処理系の実現に有効である.

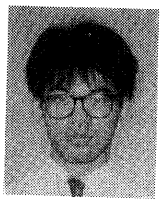
さらに seq-OINC+C に基づく言語の処理系の実装について述べた. 特に, 推論エンジンと制約解消器の間の通信, バックトラックの同期について考察した. 処理系の実装により, 計算系と通信の動作を確認できた.

参考文献

1) Slagle, J. R.: Automatic Theorem Proving in Theories with Simplifiers, Commutativity and Associativity, *J. ACM*, Vol. 21, pp. 622-642 (1974).
2) Fay, M.: First-Order Unification in Equational Theories, *Proceedings of the 4th Coference on Automated Deduction*, pp. 161-167 (1979).
3) Hullot, J.-M.: Cannonical Forms and Unification, Proceedings of the 5th Conference on Automated Deduction, *Lecture Notes in Computer Science*, Vol. 87, pp. 318-334 (1980).
4) Jaffer, J. and Lassez, J.-L.: Constraint Logic Programming, *Proceedings of 14th Symposium on Principles of Programming Languages*, pp. 111-119 (1987).
5) Jaffer, J., Michaylov, S., Stuckey, P. J. and Yap, R. H. C.: The CLP(ℛ) Language and System, *Transactions on Programming Languages and Systems*, Vol. 14, No. 3, pp. 339-395 (1992).
6) Darlington, J., Guo, Y.-K. and Pull, H.: A New Perspective on Integrating Functional Logic Programming Languages, *Proceedings of International Conference on Fifth Generation Computer Systems*, pp. 682-693 (1992).
7) Ida, T. and Nakahara, K.: Leftmost Outside-In Narrowing Calculi, Technical Report ISE-TR-94-107, Institute of Information Sciences and Electronics, University of Tsukuba (1994).
8) Huet, G. and Lêvy, J.: Computations in Orthogonal Rewriting Systems I, *Computational Logic: Essays in Honor of Alan Robinson*, Lassez, J.-L. and Plotkin, G. (eds.), pp. 395-414, The MIT Press (1991).
9) Jaffer, J., Michaylov, S., Stuckey, P. J. and Yap, R. H. C.: An Abstract Machine for CLP(ℛ),

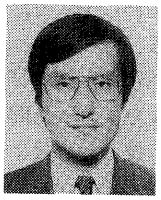
ACM SIGPLAN PLDI, pp. 128-139 (1992).
10) Saraswat, V.: *Concurrent Constraint Programming*, The MIT Press (1993).
11) Aït-Kaci, H. and Podelski, A.: Function as Passive Constraints in LIFE, *Transactions on Programming Languages and Systems*, Vol. 16, No. 4, pp. 1279-1318 (1994).

(平成6年11月24日受付)
(平成7年6月12日採録)



佐々木重雄（学生会員）

1967年生．1990年筑波大学第3学群情報学類卒業．現在筑波大学大学院博士課程工学研究科在学中．工学修士．記号処理，プログラミング言語，分散システムに興味を持つ．ソフトウェア科学会会員．



井田　哲雄（正会員）

1947年生．1971年東京大学教養学部基礎科学科卒業．理学系研究科博士課程物理学専攻中退．理学博士（東京大学）．筑波大学教授．電子・情報工学系および先端学際領域センター．記号処理，宣言型プログラミング，計算モデルの研究に従事．