制約・関数・論理型言語のための抽象機械の設計*

7F−8

佐々木 重雄　井田 哲雄

筑波大学

1 序論

関数・論理型言語とは、関数型言語と論理型言語を融合した、両者の利点を併せ持つ言語である。我々は、ナローイング計算系を計算機構とする、効率的で実用性の高い関数・論理型言語を設計し、その系統的な実現法を研究している [1]。

ナローイング計算系に基づく関数・論理型言語は、記号計算の問題に対しては十分な能力を持つが、数値計算の問題を解く場合、実行効率の面で不利である。さらにナローイングの機構だけでは、一般的な連立方程式を解くことができない。

関数・論理型言語を実際に応用することを考えると、制約解消機構を付加した制約・関数・論理型言語が必要である。我々は、記号間の等式をナローイング計算系により解き、数値領域上の等式、不等式を制約解消系により解く言語を考えた。この言語では、記号計算、数値計算とも、等式を解くという共通の枠組で行なわれるので、計算モデルが単純で明解なものとなる。また、ナローイングに基づく反駁と制約解消を並列に行なうことにより、性能向上が得られると期待される。

本稿では、抽象機械の実行の対象となる言語の構文および計算モデルを定義し、抽象機械の構成、制約変数の実現法およびバックトラックの動作について述べる。

2 言語と計算モデル

言語の構文を、1階の項の上の条件付等式システムとして定義する。いくつかの関数記号は、算術の組込み関数を表す記号として用意される。また最も外側の関数記号が、算術の等号あるいは不等号である項を算術項と呼ぶ。制約解消系は、有理数領域上の線形制約を扱うものとする。

この言語のプログラムは、基本式変換によって基本式という形式に変換され（変換された等式の集合を R とする）、その上で、次の推論規則によって与えられる計算系にしたがって、計算される。

variable elimination [v]

$$\frac{\theta(t=d,E)}{\theta' E}$$

$$\text{ただし}\begin{cases}\theta t &= x\in\\ \theta' &= \mathrm{simple}(\theta\circ\{x=d\})\end{cases}$$

outermost narrowing [on]

$$\frac{\theta(t=d,E)}{\theta'(F,t[u\leftarrow r]=d,E)}$$

$$\text{ただし}\begin{cases}\text{“}\ell\to r\Leftarrow F\text{”}\in R\\ \theta t/u \text{ は } \theta t \text{ の最左最外可簡約項}\\ \sigma \text{ は } \theta t/u \text{ と } \ell \text{ の最汎単一化子}\\ \theta'=\mathrm{simple}(\theta\circ\sigma)\end{cases}$$

unification [u]

$$\frac{\theta(t=c(x_1,\ldots,x_n),E)}{\theta' E}$$

$$\text{ただし}\begin{cases}\theta t &= c(s_1,\ldots,s_n)\\ \theta' &= \mathrm{simple}(\theta\circ\{x_1=s_1,\ldots,x_n=s_n\})\end{cases}$$

arithmetic term elimination [a]

$$\frac{\theta(t=d,E)}{\theta' E}$$

$$\text{ただし}\begin{cases}\theta t\text{は可簡約項を含まない算術項}\\ \theta'=\mathrm{simple}(\theta\circ\{\psi(s_1,\ldots,s_n)=d\})\end{cases}$$

以上の推論規則で、θ は制約、simple は制約単純化関数である。

3 抽象機械の構成

3.1 全体の構成

我々の設計する抽象機械のは、次の3つのモジュールからなる。

- 反駁エンジン
- インターフェイス
- 制約解消器

ここで、ナローイングと制約解消を並列に実行できる可能性がある。そこで我々は、反駁エンジンとインターフェイスを一つのプロセスとして、制約解消器を別のプロセスとして実現することにした。この利点は、3.2節で述べる。

次に各モジュールについて説明する。

反駁エンジン　これはナローイング計算系に基づく反駁を行なうモジュールで、現行の遅延ナローイング抽象機械全体に相当する。反駁エンジンには、現行の抽象マシンに対し、制約変数の扱いに関する拡張が必要である。

---

*An Abstract Machine Design for a Constraint Functional Logic Language
SASAKI Shige, Ida Tetsuo
University of Tsukuba

インターフェイス　これは、反駁エンジンの下位モジュールで、反駁の結果得られた制約を制約解消器に渡す作業を行なう。この際、簡単な最適化を行なう。例えば、$2\times3+X$ という算術式をそのまま制約解消器に渡すのは、いかにも無駄である。インターフェイスは、制約の中に計算可能な式が含まれれば、その値を計算する。その結果に変数が含まれている場合に限り、それを制約解消器に渡す。

バックトラックは、抽象マシン全体で同期して行なわれる必要があるが、この同期のための処理もここで行なわれる。

制約解消器　このモジュールは、与えられた制約を単純化する。その結果変数の値が定まれば、それをインターフェイスを通して反駁エンジンに渡す。現在、有理数を計算領域とし、線形制約を扱う制約解消器を実装することを考えている。

3.2　マルチプロセス化の利点

並列実行による高速化　現行の制約型言語は制約解消に多くの実行時間を費やしているということが、知られている。言語本体の実行（反駁）と制約解消を並列に実行することにより、言語処理系の高速化が期待できる。

制約解消器の部品化　インターフェイスと制約解消器の間の通信プロトコルを設定しておけば、様々な種類の制約解消器を利用することが可能となる。記述する言語も問われない。例えば、lisp で記述した、高次方程式を解くことはできるが比較的低速の制約解消器と、C で記述した、線形方程式しか解けないが高速の制約解消器の両方を利用することができる。また従来の制約型言語では、言語ごとに利用可能な制約解消器が予め固定されたため、制約が扱える領域まで固定されていた。我々の方式では、制約解消器が部品化されるため、扱いたい領域ごとに制約解消器を取りつけて利用することが可能となる。

新しい研究対象　プロセス代数的な観点からプログラムの実行を見ることにより、ナローイングと制約解消の関係を、今までとは異なる角度から明らかにできると考えられる。

4　制約変数

制約解消系によって制約された変数を制約変数と呼び、論理変数と区別する。ここでは、制約変数の抽象機械内における表現について述べる。

制約変数についてまず考慮しなければならないのは、アドレス空間の異なる複数のプロセスによって制約変数が共有されることである。そのため、制約変数はアドレスとは異なる、全プロセスで一意に定まる識別子を持つ必要がある。

制約変数の表現法としてまず考えられるのは、(1)「セルに識別子を埋め込む」方法である。しかしこの方法では、反駁エンジン内で制約変数を表すセルをコピーすると、制約変数とそれを表すセルが1対多の関係になってしまう。するとあるセルを定数に束縛しても、他のセルは依然として変数のままであり、矛盾が生じてしまう。そこで、(2)「制約変数は自己参照のセルによって表現し、そのアドレスから識別子への写像テーブルを用意し、それを通して通信を行なう」方法を考えた。この方法では、変数を表すセルのコピーは、もとのセルを参照するセルを作ることと同じになり、1対1の関係は保たれる。

さらに識別子は、あるプロセスのアドレスであって良い。そこで、反駁エンジンにおけるアドレスを制約変数の識別子とする。これにより反駁エンジンにおける写像テーブルが不要になる。

5　バックトラックの同期

我々の抽象機械は、複数のプロセスによって実現される。そのためバックトラックの機構が複雑なものとなる。具体的には、反駁エンジンが変数の束縛に失敗した時、あるいは、制約解消器が解を求めるのに失敗した時、どのチョイス・ポイントにバックトラックするかを、それぞれが他方に伝達する必要がある。

反駁エンジンと制約解消器は、対応するチョイス・ポイントを互いに別個に持つ必要がある。しかし反駁エンジンがあるチョイス・ポイントを作ってから次のチョイス・ポイントを作るまでに制約を解消器に一つも送らなかったなら、制約解消器は対応するチョイス・ポイントを作る必要はない。制約解消器のチョイス・ポイントは非常に大きくなる傾向があるため、この最適化は必須である。

6　まとめ

6.1　結論

抽象機械の設計において特に重要な、計算モデル、制約変数の表現、バックトラックの同期について述べた。今後さらに、抽象機械の命令セットの設定、コンパイル写像の決定を行なう必要がある。これについては、議論が進んでおり、本稿発表時には決定している予定である。なお、我々のこれまでの研究により、命令セットの設定から比較的短い期間で反駁エンジンの実装が可能であることがわかっている。

6.2　今後の課題

制約解消器としては、現在のところ有理数上の線形制約を扱うものしか考えていない。しかし、プロセス間通信プロトコルをうまく設定することにより、非線形制約やブール値の制約を扱う制約解消器を取りつけることが可能となる。これを行なうためには、どのようなプロトコルを設定すれば良いか考察する必要がある。

参考文献

[1] 鈴木, 中川, 井田 : 遅延ナローイング計算系に基づく言語 Ev とその処理系, 情報処理学会第 44 回全国大会予稿 5F-7, 1992.

[2] Joxan Jaffer, Spiro Michaylov, Peter J. Stuckey, Roland H. C. Yap : The CLP($\mathcal{R}$) Language and System, in Procedings of the 4th International Conference on Logic Programming, May 1987.