早稲田大学大学院理工学研究科

博 士 論 文 概 要

論 文 題 目

Lattice Programming: Programming Methodology
and Compilation to Logic Programs

束プログラミング：プログラミング方法論と
論理プログラムへのコンパイル

申 請 者

福永 光一
Koichi Fukunaga

平成 3 年 6 月

本論文は、新しく導入されたプログラミング・パラダイム「束プログラミング (Lattice Programming)」の概念と組合せ問題への応用方法についての研究をまとめたものである。その構成は、研究の位置づけと主要概念を述べた第一章、理論的背景を述べた第二章と、本論に相当する第三章、第四章、第五章からなる。以下、各章についての研究成果を記す。

人工知能やオペレーションズ・リサーチといった分野では、数多くの興味深い組合せ問題がある。これらの問題では、ある制約集合を満たす空間の中の点を求めることが目的となる。ある場合には、この点は更に最適条件を満たさなければならない。これらの問題を解くプログラムを書く場合に考えるべきことは、制約の表現方法と解の探索方法である。ここでの研究課題として、問題解決者のアイデアをいかに速くコンピュータに写像し、その検証を行なうかという試行的プログラミングの効率の向上がある。

近年注目されてきている論理プログラミングは、この課題を扱うのに都合の良い性質を持っている。論理プログラムは、宣言的に読めば関係式の集合であり、制約を関係式に翻訳するのは比較的簡単である。また、非決定的手続きを利用することにより、探索手続きを書かずに済ませられる。その結果、これらの性質を利用して書いたプログラムは、修正や拡張が容易なものとなる。しかし、論理プログラミングには重大な表現力上の限界がある。それは、論理プログラムでは、真と偽の二値しか扱えないことである。これは、何等かの基準のもとに最適値を求めようとする際に、著しく不便となる。たとえば、エキスパート・システムでは、真と偽の間の不確定性を表現したいことが多い。van Emden は、この問題を解決するために、論理プログラミングのひとつの拡張を与えたが、それは時には不自然な解釈を与える不十分なものであり、その改善が必要とされている。同様の要求は、組合せ問題にも見られ、O'Keefe や Parker 等が、論理プログラムをこの方向で拡張しているが、いずれも命題に関してだけのものであり、変数を含んだ述語は扱われていない。この制限はプログラミング言語としては致命的なものであり、更なる拡張方法が必要とされている。

本論文では、論理プログラムが取る二値（真と偽）を完備束に拡張するした新しいプログラミング・パラダイム、束プログラミング（ Lattice Programming ）を提案し、それが上述の問題を解決することを示している。また、この拡張により可能となるプログラミング方法論および束プログラムの論理プログラムへのコンパイル技法について述べている。

第一章では、このような状況を明らかにし、その中でのこの論文の位置づけが行なわれ、束プログラミングの概念の導入が行なわれている。そこでは、論理プログラムにおけるホーン節が真理値を定義域とする不等式と見做せ、この不等式の定義域を完備束に拡張しても、論理プログラミングにおける各種の意味論がそのまま成立するので、束プログラミングでも宣言的なプログラミングが行なえることを論じ、この拡張の応用を、グラフの最小木を求める例で示している。完備束への拡張後の意味論の理論的性質の概要は、第二章にまとめられている。

第三章では、束プログラミングが宣言的レベルでプログラム中の組合せに関する質的性質と量的性質をわけて段階的に扱うことを可能にし、この性質が問題解決者の思考の節約に貢献するという本研究の主張を、例題を通じて実証する。束プログラムでは、解は不等式群の最小解で与えられる。この結果、プログラムの効率は、不等式がいかに強く解の下界を狭めるか（いかに強く解を近似するか）に、依存することになる。すなわち、解の近似度の強化が、探索空間の縮小につながる。不等式の解の近似度は宣言的概念であるため、これを利用すると、プログラムのより効率のよいものへの変換が、宣言的に行なえる。このことを、実際にソーティングのアルゴリズムを導出し、類似例と比較することにより確かめている。この導出は宣言的であるため、他の類似のケースに応用可能である。また、変換結果の等価性の証明も非常に簡単になる。

さらに、束プログラムでは質・量のふたつの要素を分離でき、それらを直交したものとして扱えるので、どちらかの要素が共通であるアルゴリズム群をまとめて論じることができる。とくに、質に関わる部分は、制約内の探索に携わりアルゴリズムの工夫が集中するところであり、これを共通とするアルゴリズム群の考察は、有用な情報を与えることを、最小木を求める Prim のアルゴリズムと最短距離を求める Dijkstra 法との比較等で示す。これらを束プログラムで書けば、両者の質的部分は同一で量を扱う束だけが異なることが一目瞭然となり、束プログラミングが、アルゴリズムの比較の基盤を与えることがわかる。

このほかにも、順序構造を扱いながらこれまでそれを明示的に表現する術がなかった問題群に対し、束プログラミングは宣言的記法を与えることがある。最初に挙げたエキスパート・システムや、論理回路の故障診断や時制推論などがその例となることを示し、論理プログラムの性質を分析するアブストラクト・インタープリティションも、部分解群の上限をとる操作が基本であるので、束プログラムが自然な表現手段となることが論じられる。

第四章では、束プログラミングの能率的な実行手段を論ずる。束プログラミングは、関数型プログラミングと同様にトップ・ダウンに実行され、しかも全解探索を基本とするので、本質的に重複計算が多い。これを、プログラムを最初から論理プログラムで書いたのと、同程度の効率で実行できるようにするための論理プログラムへのコンパイル技法は、以下のようになる。まず、ソース・プログラムの各部を入出力毎に分離した表現を作り、それをもとにした全解探索インタープリタを作る。このインタープリタには、部分解をひとつのストリームにまとめる箇所がある。ここで上限値をフィルター演算で求めることにすれば、束プログラムの計算ができる。この修正版インタープリタを部分評価すれば、束プログラムのコンパイルが完了する。重複計算を避けるためには、同一部分解を集めるストリームを共有するテーブル化を行なえばよい。このテーブル化は、Extension Table という名で知られている技法と、本質的に同じものである。なお、この方法は、論理プログラムにおける上田や玉木の方法の一般化になっている。この方法によるコンパイル結果は、大抵の場合人手により最適化した論理プログラムと同等の手間で計算できるものとなる。というのは、フィルターとテーブルの組合せが、Branch and Bound や Dynamic Programming と同等の働きをするからである。

束プログラムのコンパイルについての上述の分析の副次効果として、上田、玉木のトップ・ダウン全解探索法と、Alexander templates や magic sets 等のボトム・アップ法との関係も明らかになる。この比較は、入出力部を分離した表現により可能となったもので、ボトム・アップ法がトップ・ダウン法をシミュレートしていることが、直観的に示せる。この理解は、アブストラクト・インタープリテイションのアルゴリズム設計に有用である。これらは、第五章で論じられる。

以上で述べたように、束プログラミングは、論理プログラミングの宣言的性質を保ちつつその表現力を増加させ、組合せ問題への速やかな取り組みを可能にする。質的要素と量的要素の直交性を利用した宣言的プログラミングは、単にプログラミングを容易にするのみならず、その分野のアルゴリズムの知識の共有化を促進する。論理プログラムへのコンパイル技法は、無駄な計算を除去する最適化操作含んでおり、問題解決者を、この手続きからも解放する。この二重の思考の生産性向上は、人工知能でよく見られるような、制約構造そのものを試行的に確かめる必要がある問題で、大きな効果を発揮する。