知識工学と人工知能　54－10
（１９８７・９・１８）

制約プログラミングのためのネットワークモデルとその上での変更伝搬

沼尾　雅之
日本アイ・ビー・エム株式会社　東京基礎研究所

制約プログラミングのための枠組として、Cell&Relation モデルというネットワークモデルを提案し、その上での制約伝搬機構を開発した。このモデルは、一般的な制約問題を記述するための枠組としても、また、論理型言語の１つの表現形式としても用いることができる。さらに、視覚的プログラミング環境を構成する核言語としても有効である。一方、このモデルに手続き的な意味を与える制約伝搬機構には、ユーザがオン・ラインかつ対話的に処理を進めるのに有利な変更伝搬型のアルゴリズムを用いた。多対多かつ方向性のない双方向伝搬を実現するために、制約伝搬機構は、仮説生成型の制約伝搬部と、その評価部という２つの部分によって構成されている。そして、この２つは、伝搬時の動的な仮説および関係の生成と、その遅延評価によって、協調しながら伝搬を進めていって全体の制約を満たすように動作する。さらに、仮説生成の際には変更の影響を最少限に抑えるようなヒューリスティックスを採用している。

A Network Model and Update Propagation for Constraint Programming

Masayuki NUMAO

Tokyo Research Laboratory, IBM Japan, Ltd.
5-19 Sanbancho, Chiyoda-ku Tokyo 102, Japan

As a framework for constraint Programming, an Update Propagation Network was developed. It is based on the Cell and Relation model (CRM), where the Cell holds a value which can be modified/fixed by a user, and the Relation defines a constraint between cells. Problems are represented in a network of the CRM. Network consistency is preserved by an update propagation mechanism. If any cell is modified, the mechanism modifies other cells so that the relations connecting to the modified cell would be re-satisfied. A prototype version of of the Network interpreter is running on VM/Prolog with an XEDIT interface.

１．はじめに

人工知能の分野における多くの問題は、制約の表現とその解決という、制約プログラミングの枠組として捉えることができる。例えば、スケジューリングの問題においては、機械・人員の数や、ある製品を作るための工程順序等は、制約として表現されるべきであり、こうした条件下でいかに効率良く目標量の製品を作り出すかが問題となっている。また、ＣＡＤの分野でも部品間の関係を制約として定義しておいて、一方の部品の属性をユーザが変えたときに自動的に他方の属性を、その制約が満たされるように変えることが、これからの知的ＣＡＤの機能として必要とされている。このように制約表現に優れた枠組と、それを効率良く解決する機構を提供することは、人工知能用言語を設計する上で極めて重要な位置を占めるであろう。

本論文では、制約を表現する基本的な枠組としてCell & Relation モデルを提案し、その上での制約伝搬の機構を説明する[1]。このモデルでは、値とその間の制約条件を、値を入れるためセルと、その間にはられたｎ項関係によって表現する。従って、多数の制約条件からなる複雑な問題は、このセルと関係からなるネットワークによって表現される。そして、ネットワーク中の関係は全て満たされるようにシステムが管理する。これが、制約伝搬機構と呼ばれるもので、ここで問題にしているのは、変更伝搬に基づいた制約伝搬機構である。これは、ネットワークの初期状態としては、すべての関係が満たされていることを前提として、その中の任意個のセルの値を変えたときに、全体の制約が満たされるように変更の影響を伝搬させていくことを意味している。このような伝搬の方法は、変更の影響を最少限にくいとめることができるために、効率よく全体の制約の保持をすることができる。従って、知的スプレッドシートのような対話的な環境で用いることができる。

このような制約伝搬をするシステムとしては、ThingLab[2]を代表として、対象指向型言語の枠組のなかで実現をしているものが多いが、オブジェクト間にまたがる制約関係の記述に困難があるように思われる。これは、特に伝搬の方向を指定しない双方向伝搬を実現する際に顕著に現われる。つまり、このような制約をどこに記述したらよいかという問題である。部分－全体関係のように、制約に関係するオブジェクトを含むような複合オブジェクトが作れるばあいには、そのなかに記述すればよいが、そうでない場合には、制約オブジェクトという不自然なオブジェクトを無理に作って、そのなかに記述しなければならない。もし、これを避けて、オブジェクト間だけのメッセージパッシングで表現しようとすると、もともとは１つの制約を、関係するオブジェクト全てに重複して記述しなくてはならなくなり、知識表現及び管理の面から考えると、成功したものになっているとは言いがたい。

われわれは、このような制約関係を記述するのには論理型言語が最適であると考えている。これは、例えば多対多かつ方向性のない関係を記述するのに、論理型言語、関数型言語、手続型言語のうちどれが書きやすいかを考えてみれば明らかであろう。Cell & Relation モデルは、論理型言語上で定義が与えられている。即ち、Relationは述語に、Cellはその引数にそれぞれ対応している。従って、このモデルは論理型言語の１つの表現モデルとみなすこともできる。そして、このモデル上での制約伝搬の実行機構は、このモデルに手続き的な意味を与えるものと見ることができる。

本論の構成は以下のようになっている。まず第２章では、Cell & Relation モデルの定義をし、その特長を様々の側面から考察する。第３章では、このモデルに手続き的意味を与える制約伝搬機構について説明する。第４章では、例題をもとにして、変更伝搬がいかに行われるかについて説明する。

２．Cell & Relation モデル

Cell & Relation モデル（以下、CRMと記す）は、Cell、Relation をノードとして持つグラフとして表現される。このモデルは、一般的な制約問題の記述に適しているだけではなく、論理型言語の１つの表現モデルとしても見ることができる。さらに、グラフによる視覚的言語として、図形的なプログラムの表現、及び実行過程の視覚化にも適したものとなっている。本章では、まずモデル記述のための図形的表記法を示し、次にモデルの論理型言語的解釈及び、Prologとの比較を説明する。最後に、視覚的プログラミング環境としての有効性について述べる。

２．１　記号

・Cell

Cell は cell-id, cell-valueの２つの要素を持つ四角いノードで示される。ノードは２種類あり下辺が二重線のものとそうでないものであり、前者を固定セル、後者を自由セルとよぶ。

・Relation

Relation は cell-id を引数として持つ項であり、楕円のノードで示される。Relation には、一重楕円ノードのものと二重楕円ノードのものの２種類があり、前者を関係、後者を定義用関係 とよぶ。

・Link

Link は Cell と Relation を結ぶ線であり、関係の引数部に書かれた cell-id を持つセルとその関係との間は、全て Link で結ばれる。

図１にそれぞれの図形を示す。

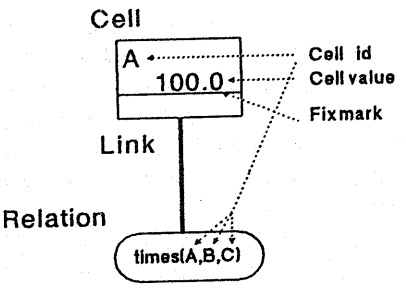


図１．CRMで用いる記号

２．２　宣言的な意味

セルの各要素の意味は以下のようになっている。

・cell-id ： セルの名前であり、関係の引数部にはこの名前が書かれる。

・cell-value ： セルの値であり、関係が実際に満たされるか否かを評価する対象。

また、 固定セルは、制約伝搬の際にその値が変わらないセルを示す。

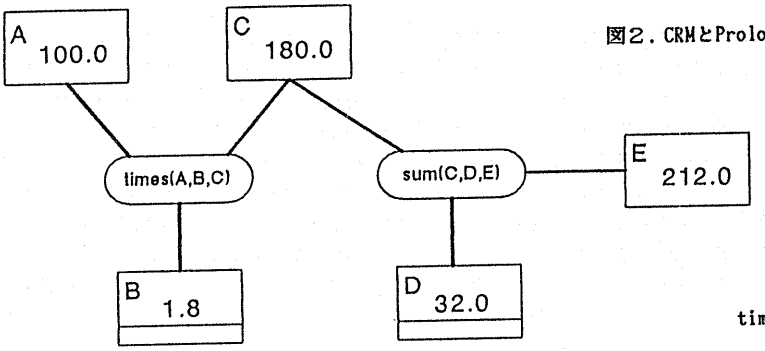
関係はそれにつながれているセルの値の間の制約関係を規定する。また、二重楕円の関係は、ネットワーク中の適当なセルの間を新しく関係として定義するためのものである。

さらに、ネットワーク上の任意のセルの値はユーザによって、書換えることができ、その際は制約伝搬機構が働いて、すべての関係が再び満たされるように他の自由セルの値が自動的に書換えられる。この制約伝搬機構はモデルに手続き的な意味を与えるものであるが、これについては次章で説明する。

２．３　論理型言語的解釈

CRMは、関係を述語、セルの値をその引数とすることによって、簡単に論理型言語の枠組で解釈することができる。たとえば、図２(a)で示したCRM表現は、図２(b)のようなPrologゴールとして解釈することができる。ところが、ここで２つの表現の間には、大きな相違がある。まずCRMでは、関係の評価の順番がまったく仮定されていないのに対して、Prologでは、ゴール実行の順番が左から右へ仮定されているように見える。また、CRMでは、２つ以上の関係が参照している値でも、それが同じものであれば、１つのセルだけで表現することができるが、Prologでは、１つの変数が、いくつもの述語にあらわれてしまう。これは、１次元的表現と２次元グラフ表現の本質的な差によるものであるが、Prologの表現は、実行の順番、変数の入出力の区別（どこのゴールで変数が束縛されるか）等の手続き的な意味が見えてしまう危険がある。CRMでは、グラフ表現の特徴を活かして、ユーザによる任意のセルの値の書換えが許され、また、変更されたセルにつながっている関係から順に評価するという動的な実行順序の決定が可能になっている。

一方、Prologでは、変数に任意の構造が書け、構造間のパターンマッチングも簡単にできるが、CRMでは、あるセルの値が構造であり、その一部を別のセルの値としてとっておきたいときには、構造を分解・構成するための新たな関係が必要となる。また、CRMでは、新しい関係の定義はごく簡単なものに限られる。これは、基本的にその新しい関係があらわれたときには、定義したときの元のグラフがマクロ展開されるだけだからである。



(a) CRM表現

図２．CRMとProlog表現の比較（温度の単位変換）

```
times(X, 1.8, Y) & sum(Y, 32.0, Z).
```

(b) Prolog表現

２．４　視覚的プログラミング環境としてみたCRM

ビットマップディスプレイ上で、マルチウィンドウ、マウス、アイコンを用いてプログラムを作成、実行、デバッグできるようなプログラミング環境は、プログラムの生産性を高める上で非常に重要になっている。表計算やデータベース検索など、特定のアプリケーションにおいては、このような視覚的プログラミング環境は成功しているといえる。スプレッドシート型インタフェースをもちいた、MultiPlan, QBE等はこの代表例である。ところが、汎用言語に対する環境を考えてみると、このような環境で実際に作られるプログラム自体は、Lispや、Smalltalk等、特に視覚的な表現形態を持たないために、実際のプログラム作成の過程においては、結局大半をキー・ストロークに依存しなければならなくなっている。

CRMは、視覚的言語としてビットマップディスプレイ上でプログラムを作成、実行、デバッグをするための表現としても適したものである。以下にCRMによるプログラム作成、実行の過程を説明する。

・プログラム作成

プログラムの作成はスクリーン上にネットワークを書くことによってできる。これは、まずマウスによって、適当な位置にセルを配置し、次に既定義関係リストの中から、適当な関係を選んで配置し、この関係から出ているリンクを対応するセルに結び付ける事によってできる。ここまでは、マウス操作だけでできる。新しい関係の定義は二重楕円ノードの関係を作って、セルに結び付ける。このとき始めて新しい関係の名前と引数をキーインすればよい。

・プログラム実行

プログラムの実行は、ユーザが任意のセルの値を変更することによって起動される。変更されセルにつながっている関係が評価されると、その結果が成功、失敗、評価保留という状態に応じて、ノードの色が変り、さらにシステムによって値が変更されたセルのノードも色が変わって、いま制約伝搬がどこまで進んでいるかが、一目でわかるようになっている。また、デバッグの際には、関係が評価の対象となるごとにウェイト状態になるステップ実行もできる。

３．制約伝搬の方法

CRMで記述されたネットワークにおいては、そのなかの全ての関係は満たされていなければならない。これを保証するのが制約伝搬機構である。ここで考えているのは、初期状態のネットワークはすべての関係が満たされていることを前提として、ユーザが任意のセルの値を変更したときに、いかに全体の関係が再び満たされるように他のセルを変更していくかという変更伝搬である。このような変更伝搬の機構は、ネットワーク中のすべての関係が満たされるようにネットワーク全体について一度に計算をするオフ・ライン型のアルゴリズムに対比すると、小さな変更に対しては少ない計算量で済ませようというオン・ライン型のアルゴリズムであるといえる。このアルゴリズムの利点は、ユーザがシステムと対話的に計算を進めるような環境に向いていることである。ユーザはいろいろなパラメータを微調節しながら、それが全体の挙動にどう影響するかを見て、最適な値の組を見つけていくことができるからである。

本章では、このような制約伝搬機構に必要とされる機能をあげ、そのためのアルゴリズムを説明する。アルゴリズム自体は、Constraint Prolog[3]と呼ばれる、Prolog上に仮説推論と遅延評価の機能を持たせたメタ・インタプリタによって記述されている。

３．１　変更伝搬の機構

ネットワーク中のあるセルの値が変更されると、そのセルに接続している関係は、満たされなくなっている可能性がある。従って、その関係を再評価してみて、もしその関係が満たされなくなっているならば、それを満たすように他のセルの値を変更しなければならない。これが、１ステップの変更伝搬である。そして、この変更がまた、新しい変更伝搬を生じさせて、全体として制約がすべて満たされるように伝搬が進んでいく。

ところで、このような、制約伝搬を実現するにあたっては、３つの大きな問題がある。１つは、２項以上の関係の場合、伝搬の経路が一意には決まらないことである。たとえば、Ａ＋Ｂ＝Ｃという関係があって、そのなかのＡの値が変更されたばあい、Ｂを変更すべきか、Ｃを変更すべきか、両方を変更すべきかという問題である。２つ目は、関係によっては、変更する値が一意には決められないことである。上の例でいえば、ＢとＣの両方を変更することになったときに、どちらをどれだけ変更するかは、その時点では一意には決められない。このように、伝搬の自由度が大きいときには、伝搬の仕方によって全く異なる解が多数存在することになる。従って、３つ目の問題として伝搬の戦略がある。

本システムでは、第１の問題に対しては、伝搬の際にぶつかったセルの値を固定するか変更するかを、仮説をつくりながら伝搬を進めていく、仮説生成型伝搬アルゴリズムによって解決し、第２の問題に対しては、ゴールの遅延評価機能によって解決している。また、第３の問題に対しては仮説生成用ヒューリスティックスによって、２つ以上のセルの中から、どのセルを優先して変更すれば効率的な伝搬をすることができるかを決定している。

制約伝搬機構は大きく２つの部分に分かれている。それは、仮説生成型伝搬機構と、仮説・関係評価機構であり、前者が伝搬を進めながら動的に生成する仮説と関係に対して、後者が評価できるものから評価して前者を制御する 。実現にあたっては、後者が、Prolog上で仮説推論と遅延評価機能を持たせたメタ・インタプリタであり、前者はその上に、仮説生成型伝搬アルゴリズムとして記述されている。

３．２ 仮説生成型伝搬アルゴリズム

アルゴリズムを実現するためのプログラムを図３に示す。プログラムの基本的な動作は、まず、今までにユーザまたはシステムによって値が変更されたセル（ModCellList)につながっていて、まだ評価の対象になっていなかった関係（Ridをもつ Relation）があったならば （6行目までのif条件部)、その関係につながっている他のセル（FocusCells）について、その値を変更するか（NewModCells)、固定するか（NewFixCells）仮説をたてて、その仮説と関係を評価すべきゴールとして生成する（10行目のregister-relation)。これが、１ステップの伝搬である。そして、あたらしく変更されたセルを変更セルのリストに加えてこの伝搬を繰り返していく。

```
propagate(ModCellList,UsedRelationList) <-
  ( relation(Rid,Relation) &
    ~member(Rid,UsedRelationList) &
    Relation =.. (Predicate.ArgList) &
    intersect(ModCellList,ArgList,ModCells) &
    ModCells ~= {} )
  ->
  ( set-difference(ArgList,ModCells,FocusCells) &
    subset(NewModCells,FocusCells) &
    set-difference(FocusCells,NewModCells,NewFixCells) &
    register-relation(Relation,NewFixCells) &
    append(ModCellList,NewModCells,ModCellList1) &
    propagate(ModCellList1,Rid.UsedRelationList) ).
```

図３．制約伝搬プログラム

この仮説生成を３項の関係、r(A,B,C)を例にとると以下のようになる（図４参照）。 ここで、｜はセルの値の変更を、－は値の固定を示す。

(1) 関係のなかの１つのセル(ここではA)がすでに変更されていた場合。
仮説としては、B,Cを変更するか固定するかで４通りある（図4 a,b,c,d)。 また、a,b,cの場合は関係 r もすぐに評価できる。

(2) 関係のなかの２つのセル(ここではA,B)がすでに変更されていた場合。
仮説としては、Cを変更するか固定するかで２通りある（図4 e,f)。関係 r はすぐに評価できる。

(3) 関係のなかのセルが３つともすでに変更されていた場合。
仮説は生成されない。関係 r はすぐに評価できる（図4 g)。

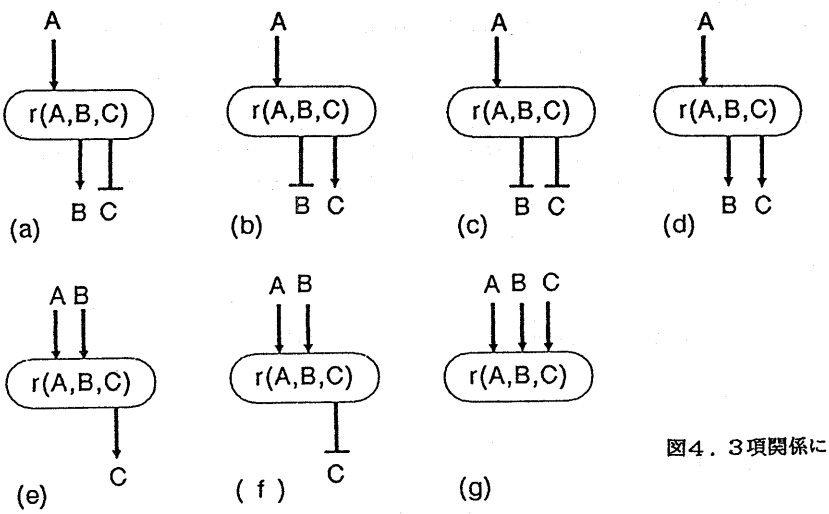


図４．３項関係における仮説生成

３．３ 仮説・関係評価機構

前節で述べたアルゴリズムによって、伝搬と同時に動的に生成される仮説および関係は、仮説については主にユーザがあらかじめ設定しておいた固定セルや、伝搬の途中にシステムによって固定されたセルの値を変更していないかどうかチェックし、関係については、その引数をセルの名前から、もしそのセルが固定されているならば、その値に置き換え、もし変更されているならば未束縛変数に置き換えて評価を試みる。評価は一般に、遅延評価になる。

```
register-relation(Relation,FixCells) <-
    Relation =.. (Predicate.ArgList) &
    set-difference(ArgList,FixCells,ModCells) &
    register-fix(FixCells) &
    register-mod(ModCells) &
    make-goal(Predicate,Arglist,FixCells,ModCells,Goal) &
    register-goal(Goal).
```

図５．仮説・関係生成プログラム

そして、仮説が矛盾していた場合や、関係が評価できてその結果が失敗だった場合には、仮説が誤りだったわけであるから、仮説生成型伝搬機構に対して別の仮説を要求する。この部分のプログラムを図５に示す。

プログラムはまず、セルを固定するか変更するかの仮説を Constraint Prolog の仮説パターンとして生成し(register-fix, register-mod)、次に関係中の引数を仮説に応じて置き換えてゴールとして実行可能な形にして(make-goal)、それを Constraint Prolog の遅延評価パターンとして生成する(register-goal)。ここで、Constraint Prolog側で失敗が検出されるとバックトラックが起動されて、他の仮説を求めるように探索が進んでいく。

仮説パターンは、セル Id を固定するか変更するかに応じて ≪fix(Id)≫, ≪mod(Id)≫ であらわされる。そして Constraint Prolog は、同一セルに対して異なる時点で、固定と変更の両方をしていないかどうかをチェックする。これは、ネットワーク内にループができているときには特に必要である。また、関係 R に対する遅延評価パターンは、≪eval(R)≫ によって表現される。これは、たとえば ≪eval(times(X,Y,Z))≫ の場合には、３個の引数のうち２個以上が束縛されたときに始めて評価される。そして評価の結果が成功だった場合には、変数に対する束縛の結果だけが残され、パターン自体は消去される。この新しい変数束縛の結果、他の遅延評価パターンが評価可能になって、その時点で決定可能なセルの値が次々に決まっていく。

ところが、伝搬がすべて終わった時点でまだ評価できない遅延評価パターンが残っていることがある。このようなパターンを集めてみると、例えば、X+Y=3, X-Y=1 というような、連立方程式になっていることが多い。これに対しては、現在は式のまま出力しているが、将来的には、CLP[4]のように等式の処理系を付加することが必要であろう。

３．４ 仮説生成用ヒューリスティックス

多数のセルから構成されるネットワーク中の、数個のセルの値が変更されたときの、伝搬の影響はなるべく小さい範囲に抑えることが望ましい。これは、効率の上からも必要であるし、またユーザも小さな変更に対しては局所的な変化を期待しているからである。このため、伝搬時の仮説生成に際して、以下のような変更の範囲を局所的に抑えるためのヒューリスティックスを用いている。

・固定セルを生成する仮説を優先する。

伝搬の際にぶつかったセルに対しては、まずそのセルの値を固定できるものとして仮説をたてる。これは、固定されたセルから先は、伝搬をしなくてよいからである。

・リンクの数の少ないセルの値の変更を優先する。

ある関係につながっているセルのうち、どれかのセルの値を変更しなければならない場合には、つながっているリンクの数の少ないセルの値を変えるような仮説を優先する。これは、リンクの数が少ないほど、そこから先に再評価しなければならない関係も少なくてすみ、結果として伝搬も広範囲にならずにすむという予想に基づく。

４．例題

ここでは例題として、物理における物の落下運動の式を CRM で表現して、その上で変更伝搬について説明する。

４．１ 方程式の CRM によるモデル化

式　$y = (1/2)gt^2 + v_0t$

この式は、3つの変数　y，t，v_0の間の関係を規定したものであるから、まずこれらに対応するセルが必要である。さらに、定数セルとして、(1/2)とgを保持するものと、中間結果を入れるためのものとして、(1/2)g，t^2，(1/2)gt^2，v_0tに対応するセルが必要である。次にこれらのセルの間を関係で結べばネットワークが完成される。ここでは、関係として、加算・乗算等の単純なものに限定したために、セルの数が多くなっているが、複雑な関係を新たに定義することによって、セルの数は減らすことができる。例えば、時間tと(1/2)gt^2を直接結ぶ関係を定義することによって、上記の定数セルや、中間結果保存用セルは不要になる。図6に、このネットワークモデルを示す。

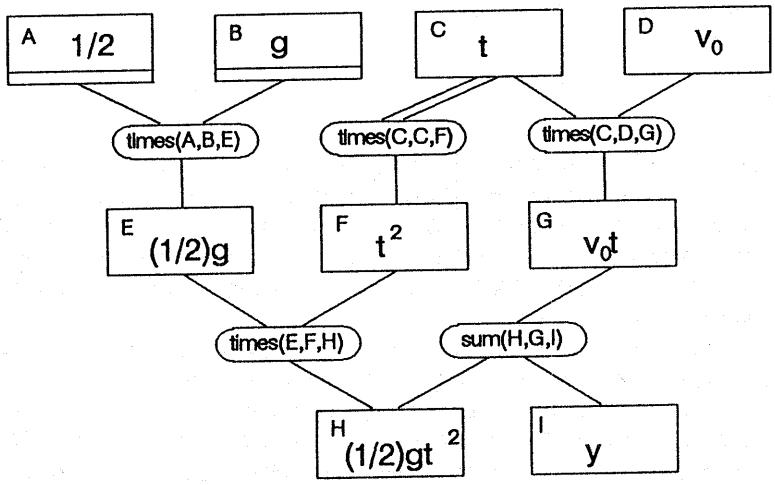


図6．y　＝ (1/2)gt^2　＋　v_0t　の CRM表現

4．2　変更伝搬

上記のモデルにおいて、ユーザが、各変数に対応するセルの値を変更したときにどのような伝搬が行われるかについて説明する。

(a) tを変更した場合

セル C の値 t が変更されると、まず、それに接続されていた2つの関係 times(C,C,F) と times(C,D,G)が再評価の対象となる。 times(C,C,F) については、セル F を変更する仮説の下で、 times(t',t',X) が計算されて、セル Fの新しい値が決まる。 times(C,D,G)については、セル D か、セル G を変更するかであるが、仮説生成用ヒューリスティクスによって、つながっているリンク数の少ないセル D の変更が選ばれ、セル G は固定される。セル D の値は times(t',Xd,v_0t)の計算によって決まる。次に、変更されたセル F に接続されている関係 times(E,F,H) が再評価の対象になるが、これは結局セル E を固定して、セル H を変更する仮説のもとで times((1/2)g,Xf,Xh)が計算され、セル H の値が決まる。最後にセル H に接続している関係 sum(H,G,I) が、セル G は固定、セル I は変更となって、sum(Xh,v_0t,Xi) が計算されて、y の値が決まる。

結局tを変更した結果、v_0とyの両方が変更されたが、ユーザがセル D を固定することによって、yだけに伝搬させることも可能である。

(b) v_0を変更した場合

セル D の値 v_0が変更されると、関係 times(C,D,G)について、セル C か、セル G を変更するかであるが、この選択の優先順位はない。従って、セル C が固定され、セル G が変更されたとすると、times(t,v_0',Xg) が計算されてセル G の新しい値が決まる。つぎに、セル G に接続している関係 sum(H,G,I) について、セル H か、セル I を変更するかであるが、仮説生成用ヒューリスティックスによって、セル I を変更することが選ばれ、sum((1/2)gt^2,Xg,Xi) が計算されて、y の値が求まる。

もし、最初の選択でセル C の変更が選ばれたとすると、結果は異なって、t と y の両方が変更されることになる。これもユーザがセル C を固定することによって、明示的にyだけに伝搬させることができる。

(c) yを変更した場合

セル I の値 y が変更されると、関係 sum(H,G,I) について、セル H か、セル G を変更しなければならないが、これもまた、優先順位がない。セル H が固定され、セル G が変更されたとすると、伝搬の影響は v_0だけにいく。また、セル G が固定され、セル H が変更されたとすると、伝搬の影響は t とv_0の両方になる。ところで、ここでユーザがセル D を固定すると、関係 sum(H,G,I) について、セル H とセル G の両方を変更しなければならなくなり、その結果は、遅延評価パターンが最後まで評価できずに残る。このパターンを整理してみると、

$(1/2)\,g\,Xc^2 + v_0 Xc = y'$

という Xc に関する二次方程式となっている。

5．おわりに

Cell&Relation モデルは、制約プログラミングと論理型プログラミングに対して、共通のモデルを与えるものである。これによって、いままでアドホックに考えられてきた制約プログラミングに対して、論理型プログラミングの立場から１つの枠組を与えることができた。一方、制約伝搬のメカニズムを実現するために、論理型プログラミング側の機能としては、変数同士のユニフィケーション、バックトラックはもとより、ゴールの実行順の動的な再配置、遅延評価、仮説の取扱い等の機能が必要不可欠となっており、この制約プログラミングは論理型プログラミングの非常に高度な応用例としてみることもできる。

このように、CRM は制約プログラミングに対して、１つの基盤を与えたものであるが、現実の問題を記述する際には様々な拡張が必要であろう。例えば、セルに関していえば、その値を変更できるか否かだけではなく、変更しにくさを重さとして与えることによって、中間的な状態を指定したい場合もある。また、伝搬のヒューリスティックスに関しても、本論中であげた変更範囲の局所化の他に、あるグループのセルに対して一様に変更を分散したい場合や、与えられた評価関数を最大にするように変更したい場合も考えられる。従って、今後の課題としては、現実の問題を記述するためにはどのような拡張が必要かを分類して、そのための一般的な枠組を開発することである。

本システムは、VM/CMS上のVM/Prologによってメタインタプリタの形で実現されており、様々な制約プログラムの作成を通じて、その有効性を検証中である。

参考文献

[1] 沼尾 "Update Propagation Network", The Logic Programming Conference '87, ICOT, 1987.

[2] Alan Borning, "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", ACM TOPLAS, Vol.3, No.4, Oct. 1981.

[3] 沼尾 "Constraint Prolog", 情報処理学会第34回全国大会、1987.

[4] J.Jaffer and S.Michaylov,"Methodology and Implementation of a CLP System",4th International Logic Programming Conference, 1987.