情報処理学会 SIGSYM 94.9, Ver. 1.0, Printed 95.8.21

創発的計算のための言語 SOOC: その特徴と実装
— 魔方陣を例題として —

金田　泰*
新情報処理開発機構 (RWCP)　つくば研究センタ

たえず変化する環境に対してひらかれた創発的計算にもとづく問題解決法の確立をめざして，著者は CCM という計算モデルを提案している．CCM は局所的な情報だけで計算される評価関数をともなう，ランダムに動作する一種のプロダクション・システムである．その実験をおこなうために計算言語 SOOC-94 とその処理系を開発し使用している．ここでは SOOC-94 の特徴と逐次計算機の Lisp 上の実装について，魔方陣の問題を例として説明する．この言語の特徴としては，規則適用時に評価関数の自動計算や局所的な自動バックトラックをすること，規則の両辺におけるパタンの構文がほぼ統一されていること，規則の適用順序に関する 2 種類のスケジューリング戦略とくにランダム戦略の存在，データ (構造体) の要素として複数の同名の要素の存在をゆるしていることなどがある．

The Features and Implementation of SOOC: A Language for Emergent Computation
— Using the Magic Square Problem for An Example —

Yasusi Kanada*
Tsukuba Research Center, Real-World Computing Partnership

A computation model called CCM was proposed by the author. CCM is developed toward establishing a problem solving methodology based on emergent computation, which is open to continually varying environment. CCM is a production system with evaluation functions, which are computed using only local information, and CCM works randomly. A computational language called SOOC-94 is used for experiments based on CCM. The features and implementation of SOOC-94 are explained using the magic square problem as an example. The features of SOOC-94 are that the automatic computation of evaluation functions and automatic local backtracking are taken place when applying a rule, that the syntax of the patterns in LHS and RHS are almost unified, the existence of two scheduling strategies, especially the random strategy, on the order of rule applications, that the existence of same name elements in a datum (structure) is allowed, and so on.

* E-mail: kanada@trc.rwcp.or.jp

1. はじめに

実世界の計算システムは刻々と変化する環境に対してひらかれた複雑なシステムである．実世界においてはつねに予測不能な変化がおこる可能性があるため，閉じた完全なシステム仕様を記述することはできない．また，複雑であるということは問題が非線形である，またはうまくモジュール分解や分割統治ができないことを意味する．ところが，従来のシステム開発法とくにソフトウェア開発法は閉じた仕様の存在を仮定していて，かつ本来は線形 (単純) なシステムにだけ適用できるはずのモジュール分解を基本としている．したがって実世界のシステム開発には限界があるとかんがえられる [Kan 92, Kan 94a]．

このような状況のもとでは創発的計算 [For 91] が重要になるとかんがえて，金田 [Kan 92, Kan 94a] は局所的・部分的な情報による創発的計算のための計算モデルである化学的キャスティング・モデル (Chemical Casting Model, CCM) という計算モデルを提案している．上記のような状況のもとでは問題解決のために必要な情報をあらかじめ完全にあつめることはのぞめない．また，とじた完全な情報をもとにした従来のシステムは環境の変化によわいとかんがえられる．そこで，CCM は局所的・部分的な情報だけによるひらかれた計算をめざしている．

CCM の研究はまだ初期段階にあるので，これまで静的な問題を中心にといてきた．古典的な制約充足問題として N クウィーン問題 [Kan 93c, Kan 94a] やグラフ彩色問題 [Kan 93d]，最適化問題として巡回セールスマン問題 [Kan 93a] や整数計画問題 [Kan 94b] などをあつかい，動的な問題への適用も検討してきた [Kan 94c]．さらに，これらの問題をとく際に開発した計算の局所性や局所最大値へのおちいりやすさ等の制御法についてものべた [Kan 94d]．

これらの問題について実験するために計算言語族 SOOC とその処理系を使用している．この報告では，SOOC の最新版 SOOC-94 の特徴と実装についてまとめる．第 2 章では CCM とその特徴についてかんたんにのべる．第 3 章では CCM にもとづく計算言語 SOOC-94 について，魔方陣の問題を例として説明する．第 4 章では SOOC-94 の特徴を説明する．第 5 章では逐次計算機のための SOOC-94 処理系の実装についてのべる．そして第 6 章で結論をのべる．

2. 計算モデル CCM とその特徴

この章では化学的キャスティング・モデル (CCM) とその特徴について説明する．ただし，CCM についてはこれまでの報告においても説明しているので，ここでは最小限の説明にとどめる．

CCM はエキスパート・システムの開発につかわれる前向き推論のプロダクション・システムにもとづいている．プロダクション・システムは if-then 型の規則の集合 (長期記憶) を作業記憶 (短期記憶) にふくまれるデータに適用することによって動作する．古典的なプロダクション・システムは決定論的に動作し，規則の条件部がみたされればそれだけで動作する．しかし CCM においてはマクロな動作は確率的であり，局所的な情報 (少数のデータ) だけで計算される評価関数 (局所秩序度) で動作がきまる．

CCM は化学反応系とのアナロジーにもとづいている (図 1 参照)．したがって作業記憶内の単位データを原子とよぶ．原子は内部状態をもち，原子どうしを化学結合に似たリンクによって結合できる．システムの動作をきめるのは，化学反応式に相当する反応規則 (if-then 規則) である．局所秩序度は一種の評価関数であり，作業記憶の局所的な状態が “よりよい” ほどおおきな値をとるように定義される．局所秩序度は負号をつけた一種のエネルギー (原子間の結合エネルギーのようなもの) とかんがえることができる．

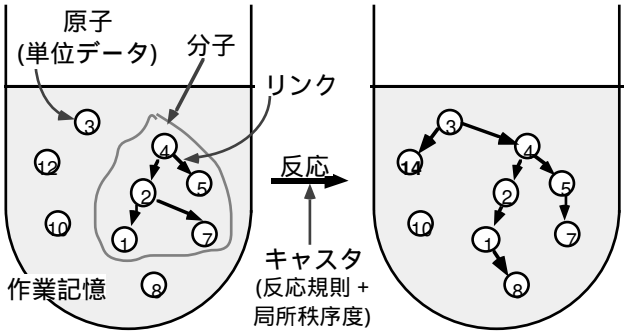

図 1　化学的キャスティング・モデルの構成要素

反応すなわち反応規則の適用は，それに関係する (規則の両辺にあらわれる) 原子の局所秩序度の和が反応によって減少しないときだけおこると定義されている．したがって，CCM はミクロにみれば決定的に動作する．反応しうる原子のくみあわせが存在しなくなると実行は中断する．反応しうる原子の組が複数あるときの反応順序は非決定的であり，通常はランダムにきめられる．ある条件をみたせば並列に反応させることもできる．したがって，システムはマクロにみれば局所秩序度の平均値すなわち平均秩序度 [Kan 94d] が増加するほうにむかって非決定

的あるいはランダムに動作する．

CCM はプロダクション・システムを基本としているが，規則をランダムに適用するという点と，規則の適用にあたって評価関数を使用するという点が特徴的である 規則のランダムな適用は ,リミット・サイクルのようなのぞましくない状態をさけるやくわりをはたしている [Kan 93d]．また，評価関数を使用することにより，4.2 節でのべるように，環境が変化すると平衡点をめざしてふたたび動作するというような，ひらかれた環境のもとでの柔軟な計算が実現しやすいとかんがえられる [Kan 93d, Kan 94d]．

しかもこの評価関数には，ゲームのプログラムなどでつかわれる評価関数や，おなじく評価関数の一種とかんがえられるニューラル・ネットのエネルギー関数や遺伝的アルゴリズムにおける適応度関数などとはちがって，局所的な情報だけで計算されるという特徴がある．すなわち，CCM は創発的計算がみたすべき “局所的な情報による計算” という性質をみたしている．ただし，この性質をみたしているだけでは創発的計算とよべないのはもちろんである．

3. CCM にもとづく言語 SOOC-94

この章では，魔方陣の問題を例として，CCM にもとづく計算言語 SOOC-94 (Self-Organization-Oriented Computing 94) についてかんたんにのべる．

3.1 魔方陣の問題とその解法の概略

魔方陣は $N \times N$ のますに 1 から N^2 までの整数値をいれて各行各列および対角線上の数の和がすべて一定になるようにした方陣のことをいう．3 次 ($N = 3$) の魔方陣の例を図 2 にしめす．

2	9	4
7	5	3
6	1	8

図 2　魔方陣の例

この報告では魔方陣をもとめる制約充足問題を例題としてとりあげる．この問題をつぎのようにしてとくことにする．まず用意した $N \times N$ の方陣に 1 から N^2 までの整数を適当に配置する．整数は昇順あるいは降順にならべてもよいし，ランダムにならべてもよい．この状態から出発して，2 個の整数を交換する操作をくりかえすことによって解をもとめる．解法の詳細は次節以降でしめす．

3.2 データ型の定義

SOOC-94 は Common Lisp 上に実装されていて，構文も Lisp にちかいし，たいていのばあい Lisp の表現をまぜてつかうことができる．

魔方陣においては各ますを基本データすなわち原子としてあつかう．すなわち，マクロ defelement をつかって型 column をつぎのように定義する．

```
(defelement column
  value                 ; 整数値
  (* summation)         ; 非排他要素
        ; このますが属する行，列，対角線の和に
        ; 関する 0 個以上のデータ．
  x  y)                 ; 座標 (印刷用)
```

マクロ defelement は Lisp の defstruct にちかいが拡張されている．上記の定義においては column 型の原子が value, summation, x, y という 4 つのなまえの要素をもつ構造体であることが定義されている．このうち summation をのぞく要素は通常の構造体要素である．value という要素が整数値を保持し，x と y は印刷時にだけつかう，ますの座標を保持する．一方，summation という名称の要素は 0 個以上の任意個存在することがゆるされる．同名の他の要素を排斥しないという意味において，このような要素を非排他要素とよぶ．summation の具体的な意味については次節で説明する．

ひとつの魔方陣全体をあらわすデータ構造は図 3 のようになる．この図では整数値をますに昇順にならべてある．このようなデータ構造のつくりかた，すなわち初期設定については 3.5 節で説明する．

3.3 局所秩序度の定義

CCM においては，各原子の局所秩序度が解の状態においてよりたかい値をとるように局所秩序度を定義する．魔方陣のばあいは，各ますについて局所秩序度を定義すればよい．ます C は，それが属する行，列，対角線に関する制約をみたさなければならない．金田 [Kan 93c] による制約充足問題の解法においては，各制約についてそれがみたされていれば 1，みたされていなければ 0 となるように局所秩序度を定義している．しかし，ここでは，つぎのようにして 2 値ではない局所秩序度 $o(C)$ を定義する．

制約 c_i はます C が属するある行，列，または対角線に関する和 S_i が S (3 次のときは $S = 15$) でなければならないという制約だとする．このとき，制約 c_i に対応する秩序度 o_i をつぎのように定義する．

$$o_i = -(S - S_i)^2 \qquad (3.1)$$

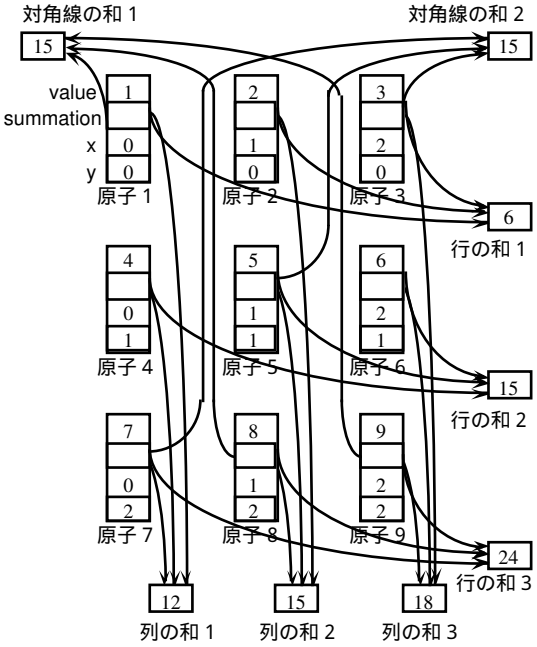

図 3　魔方陣をもとめるためのデータ構造

ます C に関する全制約を c_i $(i = 1, 2, \ldots, m)$ とするとき，ます C の局所秩序度をつぎのように定義する．

$$o(C) = \sum_{i=1}^{m} o_i \qquad (3.2)$$

すなわち，すべての和が S であるときは 0 とし，それからはずれるにつれて減少するように定義する．このように定義するのは，金田 [Kan 93c] の方法では解をもとめるのに膨大な時間がかかるからである．

SOOC-94 によって記述するとつぎのようになる．

```
(deforder ((c column))
  (let ((sum 0))
    (do-* (column c :summation summation)
      (let ((diff (– *expected-summation*
                    (summation-value summation))))
        (decf sum  (* diff diff))))
    sum))
```

deforder は局所秩序度を定義するマクロであり，構文は CLOS の defmethod にちかい．最初の行にあらわれる column は前節で定義した型名である．

この例では局所秩序度が単独の原子に対して，すなわち自己秩序度 [Kan 93c] として定義される．しかし，それが 2 個の原子のあいだに，すなわち相互秩序度 [Kan 93c] として定義されるときにはつぎのかたちで定義することができる (ただし未実装)[注1]．

```
(deforder ((atom1 type1) (atom2 type2)) … )
```

データ型 column の要素 summation は前記の和 S_i の値を保持するためのデータであり，その構造はつぎのように (Lisp によって) 定義する[注2]．

```
(defstruct summation
   value)                                  ; 和の値
```

summation は 1 要素からなる構造体である．この構造体は同一の和をもつますのあいだで共有される (共有可能にするために構造体として定義している)．この構造体の値はデータの初期化の際に計算され，2 つのますの整数値が交換されるたびに更新される．

上記の局所秩序度定義にあらわれる do-* は，同名の非排他要素全部に対して演算する，SOOC に用意されたマクロである．(do-* (column c :summation summation) …) という表現は column 型の原子 c の summation (表現上は :summation) というなまえの各要素についてその値を summation という変数に束縛して "…" をくりかえし評価するという意味である．上記の定義では，式 (3.2) における総和の計算のために do-* をつかっている．なお，ここで定数 *expected-summation* は S とおなじ値をもつ．

3.4 反応規則の定義

反応規則の主要な機能は 2 つのますがふくむ整数値を交換することである．しかし，前節でものべたように，交換にともなって行，列，対角線の和の値も更新しなければならない．これらをおこなう反応規則はつぎのようになる．

```
(defrule magic-square                  ; 反応規則名
  (var V1 V2 C1 C2)                      ; 変数宣言
  (exist column C1 :value V1)  ; 左辺のパタン 1
  (exist column C2 :value V2)  ; 左辺のパタン 2
  —>
  (exist column C1 :value V2)  ; 右辺のパタン 1
  (exist column C2 :value V1)  ; 右辺のパタン 2
  (action  (update-summations C1 (– V2 V1))
           (update-summations C1 (– V1 V2)))
                                        ; 動作定義 1
  (action  (update-summations C2 (– V1 V2))
           (update-summations C2 (– V2 V1))))
                                        ; 動作定義 2
```

反応規則中で使用するすべての変数は，変数宣言によって宣言する．これは，SOOC-94 においては OPS5 や Prolog などとはちがって変数のために特別の構文を用意していないため，宣言がないと変数と定数とがくべつできないことがあるからである．

この反応規則においては —> のまえすなわち左

注1 3 個以上の原子のあいだに局所秩序度が定義されるばあいをかんがえても，構文を変更することなく対応できる．

注2 ここで defelement を使用しないのは，summation が SOOC によって直接は参照されない，すなわち 3.4 節の規則にはこれを参照するパタンが存在しないからである．

辺に 2 個，—> のあとすなわち右辺に 2 個，(exist …) という表現がある．これらは原子にマッチするパタン (存在条件) である[注3]．(action …) という表現を動作定義とよぶ．上記の反応規則における動作定義は和の値の更新のためにある．整数値の交換だけをおこなうのなら，これらの動作定義は必要ない．

反応規則は左辺のパタンにマッチする原子があるときに適用され，それらを右辺のパタンにマッチするようにかきかえる．パタンの第 2 要素はマッチすべき原子の型名であり，第 3 要素はその原子を束縛すべき変数名である．第 4 要素以下が原子の値に関する条件をあらわす．上記の反応規則中のパタンについていえば，型名は column，変数名は C1 または C2 である．変数名 C1 をもつパタンが両辺に存在するが，これらは同一の原子に関するパタンである．すなわち，反応前には原子 C1 の value というなまえの要素の値は V1 だが，反応後にはこれが V2 になる．変数 C2 についても同様である．4 つのパタンをあわせると，2 つのますがふくむ値の交換を意味する．変数 C1, C2, V1, V2 の値は左辺のマッチング時にきまり，右辺ではその値がつかわれる．

左辺の各パタンには，かならずことなる原子がマッチする．これは通常のプロダクション・システムにはない CCM / SOOC-94 の特徴であり，排他マッチングの原則という．この原則があるのは，たいていのばあいそのほうが記述に便利だからだが，これにより化学反応式とのアナロジーもつよまっている．

動作定義は (action *action reversed-action*) というかたちをしている．*action* は反応の際に実行されるべき動作である．また，*reversed-action* はその動作を無効にしてもとの状態にもどす，すなわちバックトラックするための動作である．このようにバックトラック動作を記述する必要があるのは，つぎのような理由による．反応をおこすかどうかをきめるには，反応前と反応後の局所秩序度の値の和をもとめなければならない．すなわち反応をおこすかどうかがきまらないうちに反応後の状態を知る必要がある．したがって，SOOC-94 の処理系においてはまず反応後の状態を計算し，もし反応させないことにきまれば，それをもとの状態にもどす (4.1，5.2 節参照)．反応による変化がすべてパタンとして記述されていればそれを参照するだけでバックトラックを実現することができるが，それ以外のかたちで記述するときにはバックトラックの方法を明示してやる必要があるわけである．なお，ここで update-summations はつぎのようなユーザ定義の関数である．

```
(defun update-summations (column diff)
  (do-* (column column :summation summation)
        (incf (summation-value summation) diff)))
```

3.5 初期設定と起動

SOOC-94 システムは Lisp から起動する．現在のところ，ユーザが Lisp をつかって初期設定をしてから起動する．これは，原子が存在しない状態から SOOC によって，すなわち反応規則をつかって初期設定をするのがむずかしいという理由による．将来は SOOC によって記述できる範囲を拡大したいとかんがえている．ただし，現在でも初期設定の際に make，modify，exist などの SOOC-94 のコマンドをつかうことができる．例として，魔方陣のシステムを起動するための Lisp プログラムをしめす．

```
(defun magic-square (n)                    ; n は次数
  (setq *N* n)
  (init)                ; SOOC-94 システムの初期設定
  (make-square n)       ; データ構造の初期設定
          ; (make-square はユーザ定義の関数．
        ;  そのなかで make などをつかっている)
  (run              ; SOOC-94 システムを起動する．
      :global-strategy random
           ; 大域的なスケジュリング戦略の指定
      :rules (magic-square)
                    ; 使用する規則の集合の指定
      :max-reaction-times (* 100 n n)))
        ; 計算うちきり条件 (テスト回数) の指定
```

4. SOOC-94 の特徴

この章では計算言語 SOOC-94 のおもな特徴，すなわち秩序度の自動計算とそれにともなうバックトラック，反応規則の両辺の構文の統一，2 種類のスケジューリング戦略，非排他要素について説明する．

4.1 秩序度の自動計算とバックトラック

秩序度の自動計算とそれにともなうバックトラックが自動的におこなわれることが SOOC-94 の特徴のひとつである．すなわち，まず反応をおこすかどうかをきめるために秩序度が自動的計算される．そして，3.4 節でのべたようにこの計算のためには一般には実際に反応をおこしてみる必要があるが，反応をおこさないときめたときにはその効果はバックトラックによって無効にされる．生成されるオブジェクト・コードの例は 5.2 節にしめす．

このような局所的なバックトラックが必要になるという点は並列論理型言語に似ている．SOOC の並

[注3] パタンの構文は金田 [Kan 93c] とは一部変更されている．

列処理をかんがえるときには，並列論理型言語の並列処理で問題になったのとおなじように外部からみえてはならない計算結果が外部に輸出されるという実装上の問題 [Ued 85] がおこりうる．

4.2 反応規則の両辺の構文の統一

プロダクション・システムにおいては，通常，左辺すなわち条件部と右辺すなわち動作部の構文はことなっている．SOOC-94 においてはできるだけそれらを統一しようとしたことが特徴のひとつである．

SOOC-94 においては，原子に対するマッチングと動作は，同一の構文にしたがうパタンによって記述される．たとえば 3.4 節の反応規則 magic-square の右辺の最初のパタンは，通常のプロダクション・システムにおいてはたとえばつぎのように左辺とはことなる構文にしたがって記述されるはずである[注4]．

```
(modify C1 :value V2)
```

動作定義 ((action …)) は上記の対称性をやぶっているが，おおくの反応規則は動作定義なしに記述されるので，対称なかたちをしている[注5]．

両辺の構文を統一しようとする理由は，通常のプロダクション・システムとはちがって CCM においては本来，反応規則が可逆だからである [Kan 93d]．すなわち，本来は反応規則の右辺を条件部，左辺を動作部としてもつかうことができる．通常のプロダクション・システムにおいては実行が評価関数のようなもので制御されていないから規則を逆むきにつかうことをゆるすと停止しなくなる．これに対して CCM においては実行が化学反応系のように一種の評価関数によって制御されているので，停止しなくなることはない．むしろ環境の変化によって平衡点がかわるような柔軟な計算の実現のためには反応規則を可逆なものとしたほうがよいとかんがえられる．

構文の統一によって同一の表現からことなるオブジェクト・コードをつくりだす必要が生じているが，その実現法については 5.1 節でふれる．

4.3 スケジューリング戦略とその選択

CCM においては，反応の順序をきめることをスケジューリングとよぶ [Kan 92]．SOOC-94 においては複数のスケジューリング戦略のなかから適当なものを選択することができるが，どの戦略を選択するかによってまったくことなる制御構造をもつオブジェクト・プログラムが生成される (5.2 節の例を参照)．これが SOOC-94 の特徴のひとつである．

反応に使用する規則と対象データとはランダムに選択するのが CCM においては標準的であり，このようなスケジューリング戦略をランダム戦略とよんでいる．すなわち，ランダム戦略においては反応規則の集合から一様乱数によって反応規則を選択し，その左辺のパタンにマッチするデータ型をもつ原子の集合から一様乱数によって対象データを選択する．第 2 章でのべたように CCM の動作はミクロにみれば決定的だが，ランダム戦略をつかうときにはマクロな動作はランダムになる．ランダム戦略は反応規則の定義においてつぎのように指定する．

```
(defrule (rule-name :strategy random) … )
```

一方，ばあいによっては系統的な戦略が必要であるか，またはそのほうがよりよい．系統的な戦略とは，可能なすべての原子の順列を順に生成し，それにもとづいて原子を選択するスケジューリング戦略のことである．現在 SOOC-94 の処理系で実現されている方法としてはふかさ優先戦略がある．この戦略では反応規則ごとにその左辺のパタンの数だけのネストしたループをつかう ([Kan 92] の脚注にこの方法をプログラムとして記述した)．ふかさ優先戦略は反応規則の定義においてつぎのように指定する．

```
(defrule (rule-name :strategy depth-first) … )
```

系統的な戦略が必要なのは，たとえばつぎのようなばあいである．制約充足問題をあたえて停止したあとで，制約 1 個がみたされているかどうかをチェックする反応規則をつかって解の正当性をたしかめたいとする．系統的な戦略をつかえばこれを実現できるが，ランダム戦略をつかうとどれだけ時間をかけてもチェックされない制約がのこる可能性がある．

また，系統的な戦略が不可欠とはいえないがそのほうがよりよいのは，たとえばつぎのようなばあいである．系統的な戦略においては計算がリミット・サイクルにおちいる可能性がある．しかし，その確率を十分ちいさくおさえることができ，かつひくいコストでリミット・サイクルの検出ができるばあいがある．たとえば，N クウィーン問題をとくシステム [Kan 92] や，この報告の魔方陣のシステムもその例である．系統的な戦略はランダム戦略より高速なので，このようなばあいは前者をつかえばよい．

[注4] 実際 SOOC-94 においてもこれにちかい構文で記述できる：(modify column C1 :value V2)．型名を記述するのは構文を統一するのとコンパイルを容易にするためである．

[注5] 動作定義も両辺にふりわけて記述することによって反応規則の両辺をより対称にちかづけることができる．しかし，これには問題点もあるので現在はそうしていない．

4.4 原子の非排他要素

原子が 3.2 節でのべた非排他要素をもちうることが SOOC-94 の特徴のひとつである．非排他要素を SOOC-94 にとりいれたのは，それによって任意個のリンクをもつ原子をあつかう反応規則が非常に容易に，また美的に記述できるからである．

3.4 節の魔方陣の反応規則においては summation という名のすべての非排他要素を一度につかって計算する．このばあいは非排他要素の利点があまり明確でない．しかし，非排他要素のなかの 1 個だけをつかうような反応規則においては利点がおおきい (たとえば金田 [Kan 93d] の彩色問題のとき)．

5. 逐次計算機のための実装

この章では，CCM に関する実験をおこなうために開発した処理系 SOOC-94 の基本構造，コンパイル例，そして作業記憶の視覚化機能についてのべる．

5.1 処理系の基本構造

SOOC-94 の構造は図 4 のとおりである．SOOC-94 は逐次計算機の Common Lisp 上に実装されている．反応規則と局所秩序度 (LOD) はともに SOOC-94 コンパイラによってコンパイルされ，Lisp 関数が出力される．Lisp のコンパイラがこれをコンパイルして機械語のオブジェクト・コードが出力される．このように反応規則は機械語のかたちで実行されるが，最適化は十分ではなく，またインタプリタが頻繁に介入するため，非常に高速だとはいえない．

4.2 節でのべたように反応規則の両辺の構文を統一しようとしたために，同一のパタンを左辺ではマッチング条件，右辺では動作 (原子のかきかえ) にコンパイルする必要がある．SOOC-94 のコンパイラは実際には左辺ではパタンをそのまま出力し，右辺ではマクロ名を exist から modify にかきかえる[注6]．右辺に第 3 要素として未束縛の変数をもつパタンがあらわれると原子の生成をおこなうが，これも modify によって実現されるので，コンパイラは変数が束縛されているかどうかを考慮する必要がない．

インタプリタは，ランダム戦略のばあい反応規則を乱数をつかって選択するが，反応規則はそれにつごうがよいように拡張可能なベクトルに格納している．パタンにマッチする原子はコンパイルされた反応規則のなかで選択されるが，反応規則とおなじ理由によって原子も型ごとに拡張可能なベクトルに格納している．また，非排他要素も反応規則中で乱数によって選択されるばあいがあるので，原子からさされる拡張可能なベクトルに格納する．

インタプリタは本来は反応すべき反応規則と原子の組がなくなったときに停止するべきであり，ふかさ優先戦略のばあいは実際そのように動作する．しかしランダム戦略のばあいは，実装を容易にするため，ユーザが指定した回数だけ反応規則の左辺をテストしても反応がおこらないときに停止させている．

これまで実験に使用してきた大半のシステムにおいては (同時に動作させる) 反応規則は 1 個だけであり，複数個の反応規則があるときでもその個数は数個だった．このため SOOC-94 の処理系には多数の規則のマッチングの効率をたかめる RETE [For 82] のような機構は存在しない．このような最適化技法をつかわなわなかった他の理由は，これらがランダム戦略のもとではうまくつかえないことである[注7]．

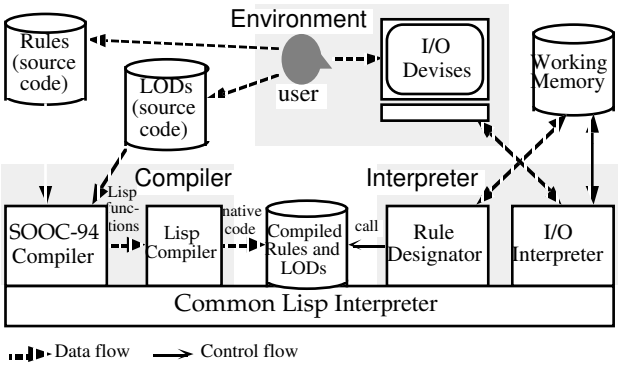

図 4　SOOC-94 の処理系の構造

5.2 コンパイル例

第 3 章の魔方陣のデータ型，局所秩序度および反応規則をコンパイルしてえられた Lisp プログラムを例として図 5 にしめす (重要部分を太字でしめす)．

5.3 作業記憶の視覚化機能

実験において，またデバッグのために，システムの走行中に作業記憶の内容を参照したいことがおおい．そのため，SOOC-94 においては make，modify などのコマンドが実行されるごとに，その対象となる原子をグラフィック表示する機構をもうけている．

ユーザはそのウィンドウの初期設定パラメタと，各データ型ごとの表示ルーティン (魔方陣のばあい

[注6] このほか，右辺では原子の不在をあらわすパタン (absent …) を (remove …) にかきかえるなどしている．

[注7] 将来 CCM を現実の問題解決につかうとすれば，多数の規則が存在するときにも効率よく計算できるように，あらたな最適化技法を開発する必要があるだろう．

には，あわせて 30 行程度) を記述しておけば，システムはそれらを必要なときに使用して表示をおこなう．この視覚化機能は，計算結果を表示するのにもつかうことができる．3.5 節のプログラムにおいて結果の印刷をおこなっていないのは，そのためである．紙面がないので，ここでは詳細は省略する．

```
;;; データ型の定義
(progn
 (fmakunbound 'column-ord)   ; 局所秩序度のふるい定義を削除．
 (defstruct column
   value    ; 整数値
   (summation (sooc94::make-empty-array))
            ; 空の拡張可能配列を要素 summation の初期値として
            ; 設定している (非排他要素は配列として実装)．
   x  y)    ; 印刷用の座標．
 (set-pprint-dispatch 'column (formatter "~/sooc::pprint-atom/")))
;;; 局所秩序度の定義
(defun column-ord (c)
  (let ((sum 0))
    (let ((t24 (column-summation c))) ; この行以下は do-* の展開形．
              ; t24 は拡張可能な配列 (非排他要素は配列として実装)．
      (dotimes (t25 (length t24) t)     ; 非排他要素を順にアクセス
        (let ((summation (aref t24 t25)))
          (let ((diff (- *expected-summation*
                         (summation-value summation))))
            (decf sum (* diff diff))))))
    sum))
;;; 反応規則の定義
(defun run-magic-square ()
  (let ((sooc94::*rule-name* 'magic-square))
    (incf sooc94::*ntests*)
    (block sooc94::body
      (let ((c2 'sooc94::<<unbound>>) ; 以下3行 : 変数値の初期設定
            (c1 'sooc94::<<unbound>>)
            (v2 'sooc94::<<unbound>>) (v1 'sooc94::<<unbound>>))
        (unless (and (exist column c1 :value v1)
                     ; 左辺のパタン 1 に由来するマッチング・パタン．
                     ; c1, v1 に値が代入される．
                     (exist column c2 :value v2)
                     ; 左辺のパタン 2 に由来するマッチング・パタン．
                     ; c2, v2 に値が代入される．
                     (not (eq c2 c1)))  ; c1, c2 にことなる原子が
                                        ; マッチすることを保証．
          (return-from sooc94::body nil))
        (without-interrupts
          ; 動作のただしさを保証するためにわりこみを禁止．
          (let ((sooc94::_order (+ (column-ord c2) (column-ord c1)))
                (t26 (column-value c1))  (t27 (column-value c2)))
            (modify-basic column c1 :value v2)
                                        ; 右辺のパタン 1 に由来する動作．
            (modify-basic column c2 :value v1)
                                        ; 右辺のパタン 2 に由来する動作．
            (update-summations c1 (- v2 v1))
                                        ; 右辺の動作定義 1 に由来する動作．
            (update-summations c2 (- v1 v2))
                                        ; 右辺の動作定義 2 に由来する動作．
            (unless (<= sooc94::_order
                        (+ (column-ord c2) (column-ord c1)))
                        ; 秩序度に関する条件が不成立ならば
                        ;  (バックトラックする)
              (update-summations c2 (- v2 v1))
                        ; 右辺の動作定義 2 に由来する逆動作．
              (update-summations c1 (- v1 v2))
                        ; 右辺の動作定義 1 に由来する逆動作．
              (modify-basic column c2 :value t27)
                        ; 右辺のパタン 2 に由来する逆動作．
              (modify-basic column c1 :value t26)
                        ; 右辺のパタン 1 に由来する逆動作．
              (return-from sooc94::body nil)))
          (setq *reaction-times* 0)  ; 左辺テストのカウンタリセット
          (incf sooc94::*nactions*))))))
```

図 5　魔方陣のキャスタのコンパイル結果

6. 結論

この報告では CCM に関する実験をおこなうための計算言語 SOOC-94 について，その特徴と実装を中心として説明した．第 4 章でのべた特徴はいずれも SOOC-94 に特有のものであって，すぐには他の言語に応用できない．しかし，これらは今後さらに発展させるに値するとかんがえられる．現在のところ SOOC は大規模応用システムの記述には適さないが，今後改良して，より大規模な実験をおこないたい．とくに当面の課題として，並列計算機への実装，計算や探索の局所性を制御する反応規則の自動合成などの手法 [Kan 94d] の実装などがあげられる．

謝辞

CCM をつかって魔方陣をとくことをすすめていただいた農工大の 小谷 善行 氏に感謝する．

参考文献

[For 82]　Forgy, C. L.: RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificiall Intelligence*, Vol. 19, 1982.

[For 91]　Forrest, S., ed.: *Emergent Computation*, MIT Press, 1991.

[Kan 92]　金田　泰 : コンピュータによる自己組織系のモデルをめざして, *第 33 回プログラミング・シンポジウム報告集*, 1992.

[Kan 93a]　金田　泰 : プロダクション規則と局所評価関数による最適化, *SICE 第 11 回システム工学部会研究会*, 27–34, 1993.

[Kan 93b]　金田　泰 : 確率過程としての計算 — 計算過程のマクロ・モデルの必要性とその例 —, *信学会研究会報告*, COMP92-93, SS92-40, 1–10, 1993.

[Kan 93c]　金田　泰, 廣川　真男 : プロダクション規則と局所評価関数による制約充足問題の解法, *情報処理学会研究会報告*, 93-SYM-68-2, 9–16, 1993.

[Kan 93d]　金田　泰 : プロダクション規則と局所評価関数にもとづく計算モデル CCM による問題解決法の特徴, *SWoPP '93*, 93-AI-89-2, 11–20, 1993.

[Kan 94a]　Kanada, Y., and Hirokawa, M.: Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, *27th Hawaii International Conference on System Sciences*, 82–91, 1994.

[Kan 94b]　金田　泰 : プロダクション規則の合成による記号的ランダム・トンネリング — 計算モデル CCM* による制約充足と最適化 —, *SICE 第 14 回システム工学分科会研究会*, 45–52, 1994.

[Kan 94c]　金田　泰 : 創発的計算のためのモデル CCM による動的なグラフ彩色, *人工知能学会並列人工知能研究会資料* SIG-PPAI-9401, 7–12, 1994.

[Kan 94d]　金田　泰 : 創発的計算のためのモデル CCM による問題解決法における局所性の制御法, *SWoPP '94*, 94-AI-95-4, 29–38, 1994.

[Ued 85]　Ueda, K.: Concurrent Prolog Re-Examined, *ICOT Tech. Report*, TR-102, Institute for New Generatin Computer Technology, 1985.