コンピュータと教育 15－5
(1991. 4. 26)

学習者の問題解決を支援する知識ベースシステムの試作

宮本 健　　伊丹 誠　　伊藤 紘二

東京理科大学基礎工学部

学習者の自由な問題解決手順をサポートするような学習支援システムを考えるに当たり，問題解決のステップ毎のプロセスの状態をスタックを用いて保存する事によって，学習者が少し戻って違った方略を使ってやり直したり，以前行なった方法を再度検討したりすることを可能にするシステムの構築を試みた．システムは，Prolog上にオブジェクト指向のスタイルを取り入れ，その上で開発を行った．支援は，学習者に問題のタイプに合った枠組みで問題記述を行わせ，問題解決のための知識検索，知識の適用の支援を行う．今回は，システムの具体的な内部処理や，動作を報告する．

A Computer Assisted konwledge explpration environment for learning by problem solving

Takeshi MIYAMOTO　　Makoto ITAMI　　Kohji ITOH

Department of Applied Electronics, Science University of Tokyo
2641, Yamasaki, Nodashi, Chiba 278

This paper concerns with a knowledge-based assistant named CAFEKS (Computer-Assisted Free Exploration of Knowledge Structure)that helps students to discover such knowledge as may be applicable to solving the problem he attacks. In order to realize a system enabling students to go back and forth solving paths, we employ a stack data structure to push down and pop up the status of the problem solving process in terms of the strategy descriptions. The system is realezed in an object-oriented style of programming in Prolog. The system assists the student understanding the problem, searching for knowledges, applying them and controlling the problem solving process. The paper reports some details of the system structure and its performance.

1．問題解決支援システム －ＣＡＦＥＫＳ－

我々は学習者に問題解決の方法を”教える”のではなく，”気付かせる”という目的で支援を行うシステムを提案してきた．このシステムは，学習者が試行錯誤しながら，システムの持つ知識の中を自由に歩き回るという意味で，ＣＡＦＥＫＳ(Computer Assisted Exploration of knowledge Structure)と名付けている．

ＣＡＦＥＫＳの教授形態は，一般に複数の学習者が同時に１つのコンピュータを使い，人間の教師も交えて，共に考え議論するという方法を想定している．

2．CAFEKS の構想

ＣＡＦＥＫＳでは，教える側が，知識の追加，変更，削除を行い易いシステムを構築してゆく為に，モジュール性の高さを重視してオブジェクト指向の考え方を採用する．また，以下の理由により，プログラミングには Prolog を用いる．

（１）支援システムの基礎となる知識ベースの構築に適している．
（２）プログラミングを宣言的に行えるので，クラス定義によってプログラミングを行うオブジェクト指向と相性がよい．
（３）オブジェクト指向を実現するために必要なさまざまな仕組みを作るのに適している．（メソッドの継承のためのグラフ探索が比較的容易）
（４）自然言語の処理に適している．

2.1 Prolog 上でのオブジェクト指向スタイル

オブジェクト指向にもとづく支援システムを試作するための第一段階として Prolog 上でオブジェクト指向のスタイルを実現する方法について考える．これは，文献[1]や[3]に詳しい．

2.2 クラスとインスタンスの定義

まず，オブジェクト指向において第一に考えなければならないのが，クラスの定義とインスタンスの定義である．Prolog上でのクラスとインスタンスの定義を

```
class(<クラス名>,superClass,<親クラス>).
inst(<インスタンス名>,class,<クラス名>).
```

とする．

2.3 インスタンス変数と値の定義

第二に，インスタンス変数とその値の定義であるが，これは，

```
class(<クラス名>, instVar,
      <インスタンス変数名>, <上位概念>).
inst(<インスタンス名>, var,
      <インスタンス変数名>, <値>).
```

という形で定義する．

2.4 メソッドの定義

第三にメソッドの定義は，受け手がクラスの場合，

```
class(<クラス名>, method,
  [<メソッド名>, <必要な引数>],
    class(メッセージの受け手クラス),
      <応答>):- <メソッドの記述>
```

として，受け手がインスタンスの場合，

```
class(<クラス名>, method,
  [<メソッド名>, <必要な引数>],
    inst(受け手インスタンス),
      <応答>):- <メソッドの記述>
```

とする．メソッド呼び出しにおいて，必ずしも<必要な引数>がなければならないということはなく，第３引数は，[<メソッド名>]とだけ定義するメソッドがあってもよい．簡単のため，<メソッド名>と<必要な引数>を合わせて<メッセージ>と呼ぶ．

2.5 手続き呼び出し型オブジェクト指向

本システムは，手続きをあるインスタンスやクラスへ要求し，その応答を返してもらうという，手続き呼び出し型のオブジェクト指向のスタイルをとる．呼び出しを受けたインスタンスやクラスが，さらに別のインスタンスやクラスへ手続き呼び出しを行うこともでき，自分自身の呼び出しもできる．これは，

```
send(<インスタンス名>, [<メッセージ>],
                              SUCCFAIL).
send(<クラス名>, [<メッセージ>],
                              SUCCFAIL).
```

という send 命令を定義しておくことによって行う．<メッセージ>の中は，上で述べたように，<メソッド名>と<必要な引数>になっている．

2.6 スタックを用いたプロセス状態の保存

学習者が問題を解く過程において，行きつ戻りつしながら解決を進められるようにするためには，各プロセスにおいて，その状態を保存しておかなければならない．そのような目的で，各プロセスインスタンスに，必要なデータを保存するためのスタックを用意しておく方法を考える．（文献[2]参照）

ここでの必要なデータとは，そのプロセス以降で必要となる方略列リストで，そこまでの過程におけるインスタンス化を行ったものであり，processStack という名でインスタンス変数を作っておき，ここに Prolog のリストの形で方略列リストを保存する．こうしておくことによって，各プロセスでの支援手続き，及び支援順序をこの processStack を参照することによってスムーズに行うことが出来る．

processStack の内容としては，problem タイプとして，

[problem,<問題名>,<問題型>,<問題記述>]

というかたちで，新たに副問題が生じたときや，システム起動時にスタックされ，問題型に応じた方略に従って，学習者に問題解決の選択を行わせる目的のものと，exec タイプとして，

[exec,<問題名>,<方略実行支援手続き名>,
<方略実行支援に必要な引数>]

というかたちで，具体的に方略実行支援を行うためにスタックされるものがある．

2.7 知識の表現

本システムは，問題を解いてあげるのではなく，学習者に問題を解くための一連の手順を獲得させることが目的である．さらに，学習者に直接こうしなさいとか，ここが間違っているぞと教えてあげるのではなく，問題解決のための手掛りを示すことによって，学習者に解決法を気付かせるという間接指導の教授形態をとる．従って，学習者への知識の提示は，テキストや画像に大きく依存するものとする．そしてシステムが，支援に必要とする限りにおいて，図1に例示するような階層構造を持った内部表現が置かれる．

各種の問題を解くための支援手順は，問題型名をクラス名として problem クラスの下位クラスに置き，問題記述をさせる際の基本的な枠組み（型）を cognition クラスの下位クラスに定義しておく．これら problemクラス や，cognitionクラス は，システムの持つ知識として，knowledgeBaseクラス の下位クラスとする．問題解決のための分野知識は，constraintクラスの下に置かれる．

3. CAFEKS の実現

本節では，今回試作した問題解決支援システムＣＡＦＥＫＳの構成について，クラスを単位として説明する．クラス構成は，図1のようになっている．図1において矢印の向きはそのクラスの親（superClass）を示している．

3.1 システム動作概要

システムは学習者にある問題を設定させ，その問題を解く過程の各プロセスにおいて，適用できる知識や方略を提示して選ばせながら，解決支援を行う．この一連のやり取りをセッションとして assistSessionクラス でこれを制御する．

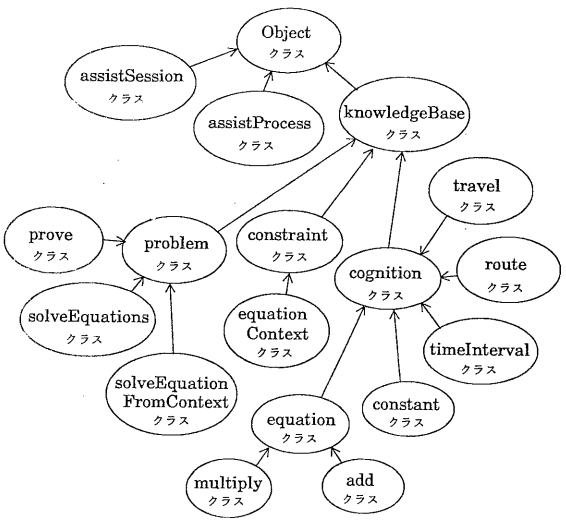

図1　　クラスの階層構造

ここでいうプロセスとは，問題解決過程の事であり，ある証明を行わせるといったマクロなものから，１つの式を解かせるといったミクロなものまで，様々である．これらのプロセスの処理を assistProcessクラス に制御させ，この assistProcessクラス のインスタンスとして，各々のプロセス（プロセスインスタンスと呼ぶ）を置く．また，以前に戻ってやり直すことを可能にするためにプロセスインスタンスの属性として，スタック（プロセススタックと呼ぶ）を持たせ，そのプロセス以降で必要な処理や手順を保存する．

3.2 objectクラス

objectクラスはシステム内の全クラスに対し，最上位にあるクラスである．それ故，全オブジェクトは objectクラス に定義されたメソッドを継承できる．

3.3 assistSession クラス

assistSessionクラス でのセッション制御の流れ図を図２に示す．

assistSessionクラス はメソッドとして explore と goSession を持ち，インスタンス変数として processHistory と currentProcess を持つ．メソッドexploreはシステムの起動時に呼び出され，次に示すように支援を行う．

```
assistSession
  explore
      問題型を選ばせる．
      問題インスタンスを作り，問題記述を入力させる．
      最初のプロセスを作る．（親プロセスを nil，子プロセスを []．）
      問題型と問題記述を processStack に積む．
      processStack 先頭データのテキスト表現をプロセスインスタンスの processText に置く．
      カレントプロセスをこのプロセスとし，goSession を呼ぶ．
  goSession
      カレントプロセスのスタックの先頭を見て，扱っている問題名の問題記述を「problemTextウインドウ」に表示．
      プロセスを選ばせる．（プロセスの内容として processText を「processText」に表示．）
      case
        ├ parent : カレントプロセスを parent プロセスにして goSession を呼ぶ．
        ├ child  :        〃          child     〃        goSession を呼ぶ．
        └ currnt : goProcess を呼ぶ．（行われた選択と新スタックデータが戻る．）
                   case
                     ├ プロセスが failure : "failure" を表示して，続けるがどうか選ばせる．
                     │    case
                     │      ├ 続ける : goSession を呼ぶ．
                     │      └ 終了   : "failure" を戻して終了．
                     └ プロセスが success : スタックが空かどうか見る．
                          case
                            ├ スタックが空 : "solved" を表示して，続けるかどうか選ばせる．
                            │     case
                            │       ├ 続ける : goSession を呼ぶ．
                            │       └ 終了   : "solved" を戻して終了．
                            └ スタックが空でない : 行われた選択が既実行のものかどうか見る．
                                  case
                                    ├ 既実行 : カレントプロセスを既実行プロセスとして goSession を呼ぶ．
                                    └ 新実行 : 新スタックデータを processStack とする新プロセスを作る．
                                               その processText に processStack 先頭のテキスト表現を置く．
                                               旧プロセスの子プロセスに新プロセスと選択を追加．
                                               旧プロセスを新プロセスの親プロセスにして goSession を呼ぶ．
```

図２　セッション制御の流れ図

（１）最初に学習者に問題型を選ばせる．
（２）この問題型のクラスに問題インスタンスを作り，問題型にあった問題記述を学習者に要求する．ここで行われた問題記述は，問題インスタンスの属性 problemDescription に保存される．
（３）assistProcessクラスに最初のプロセスインスタンスを作り，goSession を呼び出す．最初のプロセスインスタンスは親プロセスを nil，子プロセスを[]としてprocessStack に問題型と問題記述を積む．

goSessionでは，学習者に親プロセス，子プロセスへの移行か，現行プロセスの実行かを選ばせる．親プロセスや，子プロセスへの移行を選択すると，カレントプロセスを親プロセスもしくは子プロセスに移して，goSession を再帰呼び出しする．現行プロセスの実行を選択すると，assistProcessクラス のメソッド goProcess を呼び出してプロセス制御を行う．呼び出し終了後，goSession は今行ったプロセスの実行が失敗か成功かチェックし，失敗であればその旨を学習者に伝え，成功であれば新しいプロセスインスタンスを作り，新しいスタックデータをprocessStack に積む．スタックデータが空であれば，問題は解けたと見なすが，失敗にせよ成功にせよ，再び戻って見直すことができる．

インスタンス変数 processHistory は，これまでにたどられた，プロセスインスタンスのリストを格納する変数で，インスタンス変数 currentProcess は，現在ユーザがとりかかっているプロセスのプロセスインスタンス名を格納する変数である．

3.4 assistProcess クラス

assistProcessクラス でのプロセスの制御の流れを図３に示す．

assistProcessクラス はメソッドとして goProcess を持ち，インスタンス変数として processStack と processText を持つ．メソッド goProcess は，与えられたプロセスインスタンスの processStack の先頭を見て，それが problem であればその後ろに記述されている問題型の方略を提示して選ばせ，processStack 先頭には，problem 記述のかわりにこの方略を積む．方略を取って来る際，問題インスタンス（もし未定義ならこれを定義する）を返して，その属する問題型にアクセスすることにより問題記述で部分的にインスタンス化された方略を得ることが出来る．一方，プロセススタックデータの先頭が exec であれば，その後ろに記述されている支援手続きを実行し，その結果生じる新しいスタックデータを返す．支援手続きは，その支援が対象としている問題インスタンスの属するクラスのメソッドを起動することによって行われる．

```
assistProcess
  goProcess
      プロセススタックを取ってきて先頭を見る．
      case
        ├ 先頭が problem : 副問題であれば＜問題型＞の問題インスタンスを作る．
        │                  問題記述の修正追加を行わせる．  ･･･ assistProblemdescription
        │                  方略を選択させ，その内容で問題記述を置き換えた新スタックデータを作る．
        └ 先頭が exec    : 子プロセスデータから既選択データを取り，これを引数に加えて支援手続きを実行する．
                           スタックの残りを取って来る．その先頭について，
                           case ･･･ （#）
                             ├ ＜condスクリプト＞ : 条件の合う＜スクリプト＞部について（#）へとぶ．
                             ├ failure           : failure を返す．
                             └ どちらでもない    : 取ってきた残りのスタックの２番目以降のスックを
                                                   新しいスタックとする．
```

図３　プロセス制御の流れ図

4．実行例

CAFEKS は，UNIX のウインドウ上で，移動の概念を用いた簡単な問題の解決支援が行える．

学習者は，最初に問題型を選び，図5に示すように問題記述に移る．ここでは，solveEquationFromContext クラスの型にしたがって，未知数，方程式，未知数と方程式の意味（コンテキスト）を入力させるが，学習者は，どれから入力してもいいし，入力しない項目があってもよい．"Problem description" ウインドウには，逐次，入力項目を加えた問題記述が表示され，学習者はこのウインドウを見ながら問題記述や，問題解決を行う．

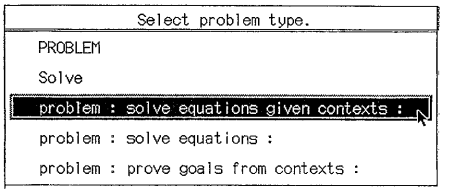

図4　　問題型選択

図6と図7では，コンテキスト入力の例を示す．コンテキストや方程式の入力においては，図6のようにキーワードを入力すると，システムが持っている知識データの枠組みにしたがって，問題記述ができる．キーワード入力が終わると図7の（1）－（5）のようにキーワードから検索された型に従って問題入力が進む．図7の（6）は，方程式の記述と，コンテキストの記述の一部が入力し終わり，未知数の記述に移ろうとしているところである．

問題記述が終わると，図8に示すように，"Process text" ウインドウに選択できるプロセスの処理が示され，学習者にプロセスを選択させる．プロセス選択が終わると，選択されたプロセスの処理が実行される．この繰り返しによって支援が進む．"Process text" ウインドウや，"Problem description" ウインドウは，常に表示されていて，学習者はそれを見ながら選択や，記述を行う．

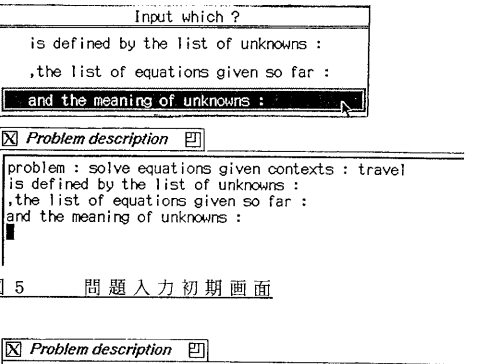

図5　　問題入力初期画面

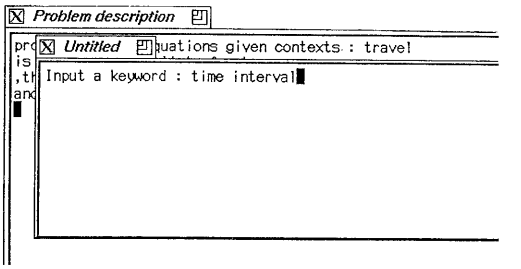

図6　　キーワード入力

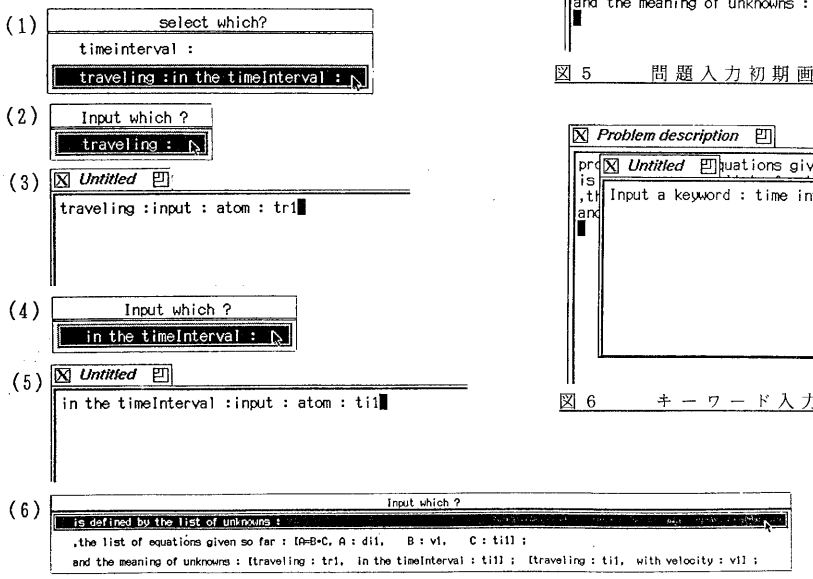

図7　　問題入力

問題解決の実行中に，副問題が生じた場合は，図９に示すように "Problem text" ウインドウに副問題の概要を示し，学習者に問題名を付けさせる．問題解決が成功すると図１０に示すように，"Solved!" と表示して，さらに続けるかどうか学習者に問い合わせる．これは，戻って見直せるようにしたものである．戻る場合は，図１１（１）に示すように一つ前のプロセス（親プロセス）を選択すればよい．このように行きつ戻りつの解決手順をとれるが，一度行った選択には，図１１（２）に示すように"Done"マークが付き，学習者が再実行であることを認識できるようにしている．

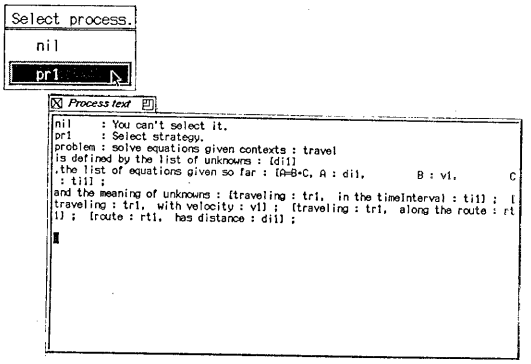

図８　　プロセス選択

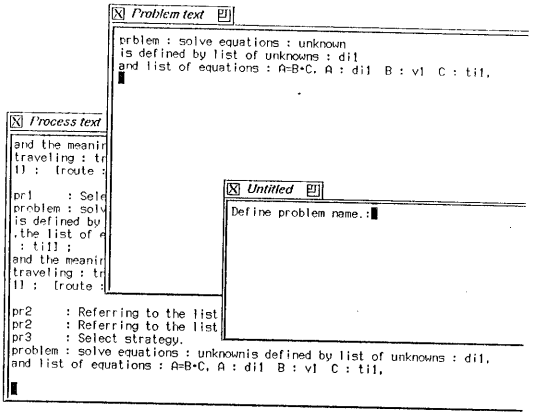

図９　　副問題

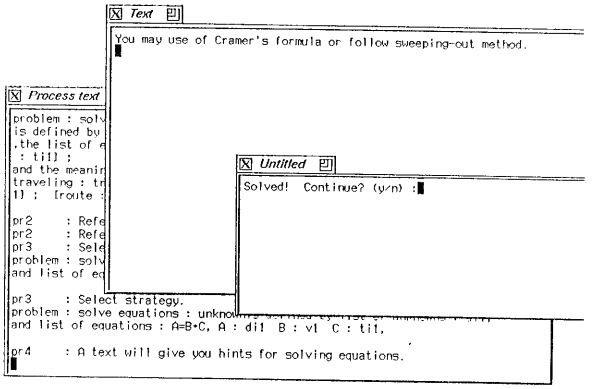

図１０　問題解決成功

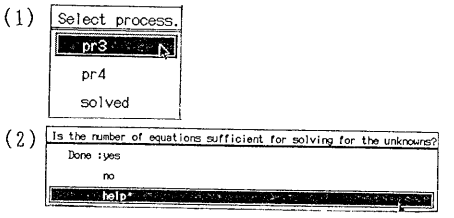

図１１　プロセスの行き来

5．結論

本研究では，論理プログラム言語 Prolog 上にオブジェクト指向のスタイルでプログラミングする方法を考案し，これが学習支援システム構築に有効であるとの考え方にもとづいて移動の概念を用いた文章問題の解決支援を行うシステムの構築を試みた．Prolog を使用することによって知識データの検索が容易に行えた．しかも，プログラムを手続き的に解釈できるために，プログラミングが楽になり，プログラム自体も見やすいものとなっている．さらに，オブジェクト指向のスタイルを採用することによって，モジュール性が高まり，プログラムの追加，変更，削除が容易に行えるとともに，上位クラスへのメソッドの継承によって無駄な記述を省くことができた．

問題解決においては，実際に小学生や中学生にシステムを使用させたわけではないのでなんとも言えないが，各プロセスの状態をスタックを用いて保存することによって，学習者が試行錯誤を行いながら問題を解決していく過程を支援することができると予想する．さらに，問題インスタンスや，プロセスインスタンスに学習者とのやり取りのためのテキスト表現を置いておくことによって，スムーズな支援が行えるとともに，インタフェース部分である assist-Session クラスや，assistProcess クラスのプログラミングがかなり楽になった．

6．今後の課題

最後になるが，今後の課題としては，次のようなものがあげられる．

（１）本研究での試作システムでは，最初に学習者に問題の型を選択させているが，解決支援において，学習者が的確に問題の型を選べるのかという疑問がある．これは，学習者に自然言語で問題を記述させ，システム側で，問題型を決定するのがよいのだが，実現は難しい．当面の策としては，問題型のメニュー表示と同時にいくつかの例題を示してあげることが考えられる．

（２）人間の教師と学習者とのやり取りにおいては，学習するのは学習者だけでなく，教師も学習者のレベルを読み取りながら，最適と思われる教授を行っている．システムにもこのような支援が必要であり，システム自身が学習者の支援を通して有望な解決方略を学習する機構や，学習者のレベルを読み取り，それに合わせた支援を行う機構も必要となると考える．

（３）学習者とのやり取りに置いて，言葉や記号だけでは表現できない，あるいは理解の難しい問題記述や知識に対処する為には視覚的に図形やグラフによるやり取りができるようなインタフェースを作る必要がある．

7．参考文献

[1] 伊藤紘二，"知的ＣＡＩシステム探訪"，情報処理，Vol.29，No.11，pp.1283-1293，1988

[2] 伊藤紘二，伊丹誠，"A COMPUTER-ASSISTED KNOWLEDGE EXPLORATION ENVIRONMENT FOR LEARNING BY PROBLEM SOLVING" International Conference on ARCE，1990

[3] 岡安二郎，"学習者の問題解決を支援する知識検索機構"（東京大学工学部計数工学科卒業論文　1989.3）

[4] 大槻説乎，山本米雄，"知的ＣＡＩのパラダイムと実現環境"，情報処理，Vol.29，No.11，pp1255-1265，1988

[5] 山本米雄，　"スタンドアロン方式によるＣＡＩ"，　電子情報通信学会誌，Vol.71，No.4，pp.379-384，1988

[6] Brad J.Cox，"Object Oriented Programming"，ADDISON-WESLEY PUBLISHING COMPANY，1986

[7] Leon Sterling and Ehud Shapiro，"The art of Prolog"，MIT Press，1986