ソフトウェア工学 54－1
（1987. 6. 24）

COOLによる制約プログラミング

中島 震
日本電気（株）C&Cシステム研究所

制約伝播概念に基づく推論システムによる応用プログラムが成功を納めているが，データ指向の上にデーモン手続きの複雑な組合せにより実現したものが多い．しかし，制約伝播概念は新しいプログラミング・パラダイムを与え，かつ広い適用領域を持つため，言語／システム側で標準機構を提供すべきであると云われている．COOLはオブジェクト間の関係評価機構として制約伝播機構を内蔵したオブジェクト指向言語であり，本稿で，制約デバイス・モデルによる制約プログラミングをCOOLの制約伝播機構とオブジェクト／デーモン定義により実現する方式について報告する．

Constraint-based Programming in COOL

Shin Nakajima
C&C Systems Research Laboratories, NEC Corporation
1-1, Miyazaki 4-Chome, Miyamae-Ku
Kawasaki, Kanagawa 213 JAPAN

Constraint-based problem solving approach has been applied to a wide variety of applications. Additionally most systems are implemented by using procedural combination of daemons on top of data-oriented programming paradigm. However, the constraint-based paradigm is so useful that the underlying programming tool should offer a standard mechanism for it. COOL is an object-oriented programming language which incorporates a constraint propagator as the evaluation mechanism for relation descriptions among objects. This report discusses how the constraint-based programming, especially the constraint-device model programming, is realized by using COOL's constraint propagator and object/daemon definitions.

COOL … Constraint and Object-Oriented Language

１．はじめに

知識工学の進展と共に，種々の応用プログラムで用いられてきたプログラミング技法を，簡便に使用可能にするＡＩツールの開発が盛んになってきた．オブジェクト指向，データ指向，ルール指向，等のプログラミング・パラダイムを提供するツールである[1][2]．また，論理回路設計時の動作検証や故障箇所の抽出等を行うために，知識工学的なアプローチを採用する記号シミュレータを，このようなツール上で作成する例もあり，特に制約伝播概念を基礎とするシステムが成功している[2][3]．しかし，制約伝播の実現方式をみると，データ指向の機構を基本に，デーモン手続きの組合せによるものが多く，システムが複雑になっている[2][4]．

一方，制約伝播概念は問題解決に関する一般的な考え方のひとつである．データ指向機構の上に利用者が制約伝播機構を構築するのではなく，オブジェクト指向等のように言語／システム側で標準機構として提供すべきものであろう．このような観点から，制約伝播概念のみに基づく制約言語の研究もなされている[5][6]．また，我々も制約伝播機構を内蔵したオブジェクト指向言語ＣＯＯＬ(Constraint and Object-Oriented Language) を開発し[7]，イベント駆動論理シミュレータのように多数のオブジェクトが互いに結び付いている状況の記述が容易になることを示した[8]．

ＣＯＯＬの特徴は任意オブジェクト間の関係を記述するために，任意オブジェクトを値として伝播することが可能な制約伝播機構を内蔵することである．また，問題領域に依存した処理をオブジェクト定義で吸収するので単純な機構になっているが，制約言語を実現するためには上位の機構が必要となる．本稿では，ＣＯＯＬの応用のひとつとして，G.Steeleが定式化した制約プログラミング[6]を，ＣＯＯＬが提供する基本機構を用いて実現する方式について報告する．２節で制約プログラミングの特徴ならびにＣＯＯＬと他制約言語／システムとの関連について述べる．３節でＣＯＯＬの概要を説明した後，４節でＣＯＯＬによる制約プログラミングの実現方式を報告する．

２．制約プログラミング

制約プログラミング・パラダイム

制約プログラミングとは，G.Steele Jr. が，種々の応用プログラムで個別に用いられていた制約処理に共通な概念を定式化したもので，次に示す三つの特徴を持つ[6][9]．

① 宣言的な記述（declarative description）

入出力関係が予め決まっていない複数の変数間に何等かの関係（たとえば代数的な関係）を宣言的に与える．

② データ駆動計算（data-driven computation）

値が定まった変数を入力とみなして，与えられた関係式にしたがって出力変数の値を計算する．入力／出力は局所的な関係に基づいて決まる．

③ 順序に対する非依存性（order independency）

宣言的な記述が与えれらた順序に依存しないで，常に同一の解を得ることができる．計算を進行させるのに十分な情報が無い場合でも，段階的に関係記述を追加することにより，解を得るようにすることが可能である．

本パラダイムを直感的に説明するモデルとして，制約デバイス・モデルがある．制約デバイスとは，複数の入出力ポートを持ち，ポートが変数を参照することによって変数の間に特定の関係を与えるものである．第１図に制約デバイスを図式化した簡単な例を示す．(a) は加算デバイスを示し，三つのポートａ，ｂ，ｃが参照する変数の間に，

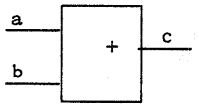
$$a + b = c$$

という代数的な関係を与える．手続き的には，『ｂとｃの値が確定していれば，ａ＝ｃ－ｂ　により，ａの値を計算する』等をあらわす．

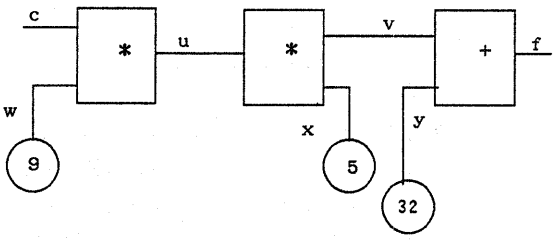
第１図(b) に，このような加算デバイスおよび乗算デバイスを組み合わせた例を示す．本例は，温度に関する二つの尺度，摂氏（Ｃ）と華氏（Ｆ）との変換公式をあらわす．Ｃの値が確定した場合，各制約デバイスが規定する局所的な関係から値が順次確定し，その結果Ｆの値を求めることができる．一方，Ｆの値が確定した場合には，逆向きに伝播してＣの

値を決定する．変換公式は次の通り．

$$9C = 5(F - 32)$$



(a) 加算デバイス



(b) 制約ネットワーク

第1図　制約デバイス

他の制約言語／システムとの比較

制約言語としてよく知られている G.Steele Jr.の制約言語[6]，A.Borning の ThingLab [10] とＣＯＯＬの比較を下表に示した．

G.Steele は整数に関する代数的な制約に限定し，最小限のプリミティブ制約デバイスを定義した．複合制約デバイスをプリミティブ制約デバイスをノードとする制約ネットワークに展開するコンパイラを持つこと，依存型バックトラック機能を持つこと，相矛盾する値の集合（NoGoodセット）をリゾルーションにより管理することが特徴である．また，ThingLab は制約保持メソッドをプランニングするアルゴリズムと，制約伝播方向を指定するプリファレンス記述および Smalltalk をベースにしたグラフィックス・インタフェースが特徴である．一方，ＣＯＯＬは，任意オブジェクト間の関係を記述するために，任意オブジェクトを値として伝播することが可能な制約伝播機構を内蔵する．G.Steele の方式と同様に変数の依存関係を用いた局所伝播に基づいているが，問題領域に依存した処理をオブジェクト定義で吸収するようにしている．

制約言語／システムの比較表

システム名	制約の管理者	制約伝播		その他
		局所的伝播	大域的解析	
ThingLab	全体オブジェクトが自分を構成する部分オブジェクトに関する制約を管理． 制約ルールと制約保持メソッドを記述．	自由度の伝播／定数の伝播により，制約保持メソッドのプランニング（複雑なアルゴリズム）を行う	連続量数値に関する制約の場合，緩和法により解決する． 組合せ的な制約は扱えない．	プリファレンスにより制約伝播の方向を制御することができる． プランしたSmalltalkメソッドをコンパイル後に実行する．
G.Steele の制約言語	システムが提供するプリミティブ制約デバイス． 整数に関する代数的な制約に限定．	多値論理に拡張したＢＣＰ（整数値を伝播する）．	依存型バックトラック． リゾルーションによりNoGoodセットを管理する．	複合制約デバイスをプリミティブ制約デバイスをノードとするネットワークに展開するコンパイラを持つ．
ＣＯＯＬ	オブジェクトと独立な概念である”関係”が管理． 制約デバイスの機能を持つ”関係”を作成．	拡張ＢＣＰとデーモンの起動（任意のオブジェクトを伝播する）．	依存型バックトラック． 制約デバイスでNoGoodセットを管理する必要がある．	利用者が任意のオブジェクトに関する任意の制約関係を定義することが可能．

３．ＣＯＯＬの概要

ＣＯＯＬは，オブジェクト間の関係を記述するために，Smalltalk 等が採用している『構造を持つオブジェクト』というモデルに加えて，『オブジェクトを命題と見なす』という考え方を新しく導入し，オブジェクト間の関係を命題論理式に似た形式で簡明に記述できるようにした[7]．以下，ＣＯＯＬの概要について述べる．

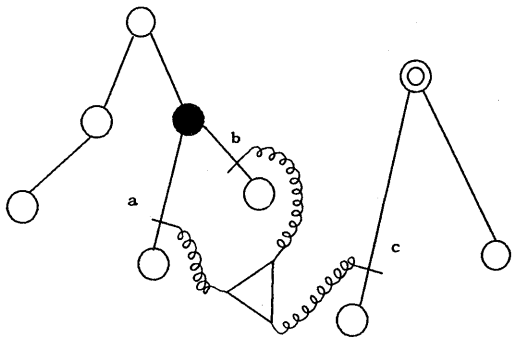
構造の記述

ＣＯＯＬでは，オブジェクトおよびメソッド呼び出しに関して 'classical' な立場をとる．すなわち，オブジェクトとは，内部状態を持つモジュールであり，自分自身の性質を特徴づける属性の集合からなる構造体である．また，オブジェクトが持つメソッドを起動することでのみ，オブジェクトを操作することが可能であり，メソッドは操作対象オブジェクトに処理内容を記したメッセージを送ることで起動できる．

オブジェクトを定義するには，(a) 内部構造（スロット）等を与えるクラス定義（def-classによる）と，(b) メッセージにより起動するメソッド定義（def-methodによる），とを行う．なお，ＣＯＯＬで云うオブジェクトとは，インスタンス・オブジェクトのことで，クラスはオブジェクトではない．また，メタクラスも存在しない．

関係の記述

関係記述の基本的な枠組みとして命題論理式に似た形式を採用した．オブジェクトが満足すべき関係を，そのオブジェクトを値として格納するスロット間の命題論理式として与える．すなわち，オブジェクトを命題とみなして関係記述を行う．第２図はオブジェクトの部分全体階層を木構造で表示したもので，○はオブジェクトを，｜はスロットを，-ოთ- と△とが関係をあらわす．特に，●オブジェクトのスロットａ，ｂ，および◎オブジェクトのスロットｃが値として格納するオブジェクトに関係△を課すことを示している．この例が示すように，構造オブジェクトの部分全体階層を超越して，任意のオブジェクト間の関係を記述することが可能である．



第２図　関係の記述

『関係の評価を行う』とは，『与えられた関係にしたがって，関連するオブジェクトの状態を変更する』ことである．ＣＯＯＬでは，①命題としてのオブジェクト（命題オブジェクト）間に真偽値を伝播させること，②命題の真偽値が変化する毎に，その命題に付与されているデーモンを実行すること，の二段階で行う．したがって，関係を定義するためには，(a) 命題論理式および命題とデーモンとの対応関係を与える関係定義（def-relationによる）と，(b) 命題論理値の伝播にともなって起動されるデーモン定義（def-daemonによる）とを行う．

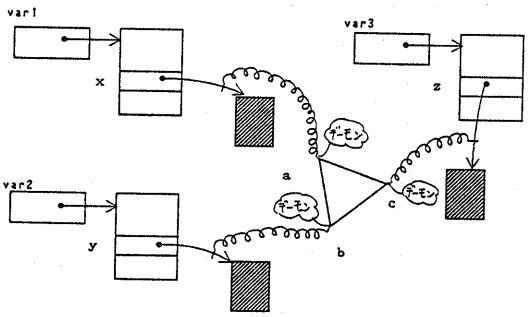
オブジェクト間に関係を課すためには，つぎに示すようにオブジェクトのスロット名を引数として関係名を指定する．本記述を簡明に行うために，パス指定記述を提供し，¯var1/x とは，変数 var1 が参照するオブジェクトのｘスロットのことである．

```
(AddNode ¯var1/x ¯var2/y ¯var3/z)
```

この例は，三つのオブジェクト間に

a ＋ b ＝ c

という宣言的な代数関係を与える．第３図は，本関係を実体化（instantiate）した場合のイメージを示し，各スロットが値として持つオブジェクト（斜線で示した）間に指定の関係を課す．



第３図　関係の実体化

関係の評価機構

ＣＯＯＬが内蔵する制約伝播機構は，ＢＣＰ（Boolean Constraint Propagator)[7][11]を拡張したもので，値に関する矛盾検出をデーモンで行うことが可能になっている．以下，本制約伝播機構とデーモン実行制御について説明する．

ＢＣＰでは，命題論理の枠内で矛盾を検出し，依存型バックトラックを行うことで，矛盾を解消しようとする．しかし，ＣＯＯＬのように，任意の値（オブジェクト）を制約伝播の対象とする場合，下位の制約伝播機構だけでは，値に関する矛盾／整合の判定を行うことができない．伝播するオブジェクトに関する情報を持つデーモンにより処理を行う必要がある．そこで，以下のようにＢＣＰを変更して用いる．

(1) 命題は unknown/known/transient のいづれかの状態値を取る．通常のＴＭＳで云えば，unknown は OUTに，known は IN に対応する．また，transient は IN であるが，デーモンが起動されていないことを示す過渡的な状態をあらわす．
(2) 関係づけるべき命題を選言クローズ（disjunctive clause）で表現する．クローズが含むすべての命題の状態値が known である時，クローズが満足されると云う．
(3) クローズを満足させるという制約を，そのクローズが含む命題に課すことにより，命題間に状態値を伝播させる［制約伝播］．すなわち，known の命題を基にして，unknown の命題を transient に変更する．同時に，状態が変化した命題に付与されているデーモンを実行キューに格納する．
(4) 制約伝播が終了した時点で，デーモン実行キューから順にデーモンを取り出して，キューした命題の環境で実行する．実行後，その命題の状態値を変更して known にする．

第３図の例について，本制約伝播機構を用いた関係の評価過程を説明する．図中，スロットｘ，ｙ，ｚに格納されるオブジェクトに x + y = z という関係が成立していて，ｘ，ｙ，ｚの間に上記 (3) の制約が与えられている．最初，x = 10 で定数とする．これは，本スロットに対応する命題がアサーション（assertion）であることで実現する．すなわち，(assert x) により，本命題の状態値は known になる．ここで，外部から z = 30 にし，値が確定したことを下位機構に伝えるために，(assume z) を実行する．この結果，本スロットに対応する命題の状態値も known になる．次に，制約伝播機構が作動して，ｙの状態値を unknown → transient にし，ｙの状態値を変更した理由（x and z）を依存関係として格納し，かつ対応するデーモンをキューする．以上の他に関係する命題は無いので，制約伝播が終了し，デーモンを実行する．本デーモンは，減算を計算する手続き，y = z − x を実行して，y = 20 を得る．また，同時に，ｙの状態値を transient → known とする．

つぎに，y = 30 を設定しようとすると，上記でｘおよびｚから導出された値 y = 20 と不一致なため矛盾を導く．この場合，下位機構に対して，削除すべき仮説命題を要求する．ｙを導出した依存関係（x and z）から，assume により状態値を与えられた命題を求め，ｚを得ることができる．次に，ｚを削除して制約伝播機構を用いることにより，ｙに関する情報（導出関係，値）を削除して矛盾状態を解消する．また，複数の命題を削除すべき候補として得た場合，そのうちの一つを選択する機構を利用者に開放している[7]．

４．ＣＯＯＬによる実現手法

３節の制約伝播機構を用いてＣＯＯＬ上で制約デバイス・モデルによるプログラミングを実現する方式について報告する．

簡単な例

第１図（２節）で示した例，『温度に関する二つの尺度の変換』をＣＯＯＬにより実現する．以下のプログラム例において，($foo ...) はその第一引数オブジェクトをレシーバとするメッセージ・センド式で，セレクタが foo であることを示す．

最初に，制約デバイスを結合したり，値の設定／読み出しを行うコネクタ・オブジェクトを定義する．このクラス Connector のスロット i-port には :R オプションを付けて宣言しておき，本スロットが値として持つオブジェクト間に制約関係を課すことができることを指示する．また，start! メソッドはメッセージ引数として受け取った新しい値 new-value を設定し($i-port!)，伝播する(assume)．ここでは，簡単のため，値に関する矛盾検査を行っていない．

```
(def-class (Connector) ((i-port :R)))
(def-method (start! Connector) (new-value)
  ($i-port! self new-value)
  (assume i-port))
```

また，定数は i-port スロットが命題としてアサーション（assertion）であること，すなわち，削除（retract）不能であるとして実現する．

```
(def-method (constant Connector) (new-value)
  ($i-port! self new-value)
  (assert i-port))
```

つぎに，加算デバイスを関係定義により記述する．三つのポートａ，ｂ，ｃ各々が加算制約関係を課す変数（スロット）を参照する．関係定義 AddNode の本体部では，デーモンと命題との対応関係の記述(add-known-daemon)，制約を課すクローズの記述(add-clause)，を行う．

```
(def-relation AddNode (a b c)
  (add-known-daemon a 'c-b)
  (add-known-daemon b 'c-a)
  (add-known-daemon c 'a+b)
  (add-clause (list a b c)))
(def-daemon c-b (a b c)
  ($i-port! self (- (value c) (value b))))
(def-daemon c-a (a b c)
  ($i-port! self (- (value c) (value a))))
(def-daemon a+b (a b c)
  ($i-port! self (+ (value a) (value b))))
```

以下，例に示したプログラムを作成する．コネクタ・オブジェクトを生成し，複数の制約デバイスのポートが同一の i-port スロットを共有することでネットワーク構造を作る．

```
(setq c (new Connector))
(setq w (new Connector))
  ...

(MultNode ~c/i-port ~w/i-port ~u/i-port)
(MultNode ~v/i-port ~x/i-port ~u/i-port)
(AddNode  ~v/i-port ~y/i-port ~f/i-port)
```

また，定数を設定するためにコネクタ・オブジェクトに対して，constant メッセージを送る．

```
($constant w 9)
($constant x 5)
($constant y 32)
```

ここで，($start! c 25) により，Ｃの値を25に設定すると，start! メソッド本体の (assume i-port) により，本命題の状態値が known になり，制約伝播機構により，順次計算が進行する．その結果，Ｆの値を読み出すと($i-port f)，77を得る．

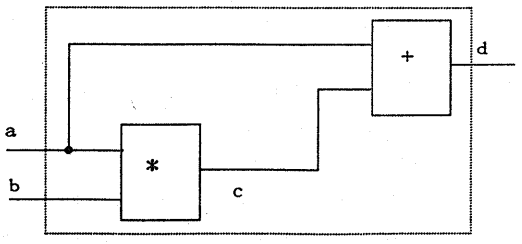
局所伝播で計算できない例

上の例では，局所伝播によって値が確定して計算がうまく進行した．しかし，一般的にはうまくいかない．例えば，第４図のように制約ネットワークがループ状になる場合がある．ここで，(a b) または

(a d) の組が確定している場合，上記手法で各々 d および b を求めることができる．しかし，たとえば b = 2, d = 3 の場合，a = 1 であるにもかかわらず，局所伝播が起こらないために，a 値を求めることができない．

a ← b and c by-using ＊-node
c ← d and a by-using ＋-node

からわかるように，a に関してループを成しているからである．



第４図　局所伝播で失敗する例

この制約ネットワークが表現する代数関係式は，中間変数 c を省略して，

a＊b ＋ a ＝ d　　（1式）

であるから，制約ネットワークに展開する前に，

a＊（b ＋ 1） ＝ d　（2式）

のように変形しておけば必ず計算できる．ところが，対話的な環境で，ボトムアップに制約ネットワークを作成する場合，このような前処理を利用者が行えるとは限らない．

制約ネットワークの大域的な解析

この問題に対処するために，二つの手法が提案されている[5][6]．

① 変数伝播
a を変数として伝播させて，数式処理により値を決定する．
② 冗長ネットワーク
制約ネットワークの意味を保ちつつ，a を計算するネットワークを付加する．

ところが，対象が代数的な関係式である場合，両者は同等のことを行っている．すなわち，制約ネットワークから対応する代数式を抽出し，局所伝播で計算が進行する形に変形する．具体的には（2式）を得ることになる．①の場合，得た数式を数値的に解く．一方，②の場合，得た数式を冗長ネットワークとして付加する．後者では，再度同様な計算を要求された場合，局所伝播のみで値を求めることができる．

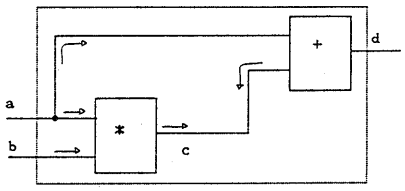
第４図の例で，制約ネットワークを大域的に解析する方式について説明する．以下の二つを行う．
① 求めたい変数を計算するのに十分な情報が確定しているかどうか，
② 局所的な伝播のみで計算不能な場合，どのような冗長ネットワークが必要か．
今，b = 2 の状況で，a を求めたいとする．a に対応する命題を含むクローズ（＊，＋関係定義により与えられている）が作るネットワークを辿ると，含む命題がすべて unknown のクローズ（＋関係に対応）が存在する．本クローズは，端点の命題 d を含むので，外部より d の値が与えられないと a の値も決まらない．そこで，利用者が d = 3 を設定する．

この段階で上記の解析を行うと，ループの存在により計算が進行しないことがわかる．そこで，第５図(a) に示すように，a に変数を仮定して x を伝播する．x は二つの経路により c の値を導出し，デーモンが二つの結果を等しいとすることで，x に関する代数式を得ることができる．すなわち，

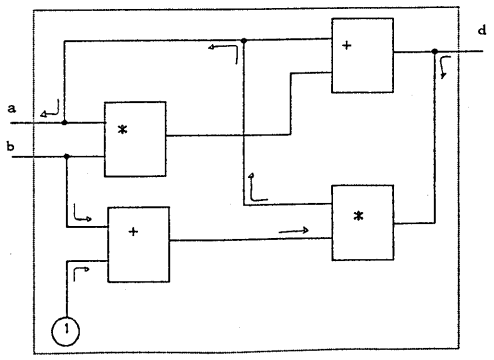
x＊B ＝ D － x．

ここで，b，d は定数であることを示すために大文字であらわした．以下，本式に対して数式処理を施すことにより（2式）を得ることができる．変数 x

の伝播に関する記憶を消去した後，この式を冗長ネットワークとして追加すると第5図(b) を得る．また追加したと同時に局所伝播が起こるので，a = 1 を導出することができる．



(a) 変数の伝播



(b) 冗長ネットワーク

第5図　制約ネットワークの大域的な解析

5．おわりに

本稿で，G.Steele Jr. が定式化した制約プログラミングをＣＯＯＬ上で実現する方式について述べた．ＣＯＯＬが内蔵する制約伝播機構を利用し，オブジェクト／デーモン定義を与えることによって容易に実現することができた．データ指向の機構を基に参照デーモンの複雑な手続き的組合せによる実現手法に比べて，はるかに明解である．

4節で述べたように，整数値に関する加算乗算という単純な制約であっても，制約ネットワークの作り方によっては，大域的な知識が必要になる．そのため，制約概念のみでプログラミングを行うことは現実的ではない．しかし，制約プログラミングの持つ三つの特徴；

(1) declarative description,
(2) data-driven computation,
(3) order independence,

は人間の直感にあったモデルであり，視覚情報を活用した対話的な環境におけるプログラミングのモデルとして有効と思われる．

最後に，本研究の機会を与えて下さった当研究所山本昌弘部長ならびに小池誠彦課長，および本稿に有益なコメントを下さった当研究所福井慎吾氏にそれぞれ感謝いたします．

参考文献

[1] M.Stefik & D.Bobrow / Object-Oriented Programming: themes and variations, The AI Magazine, (Winter 1985).
[2] H.Brown, C.Tong & G.Foyster / PALLADIO: An Exploratory Environment for Circuit Design, IEEE Computer, (Dec 1983).
[3] R.Davis & H.Shrobe / Representing Structure and Behavior of Digital Hardware, IEEE Computer, (Oct 1983).
[4] 渡辺，出口，岩本 / ＣＬにおける制約処理，日本ソフトウェア科学会，知識プログラミング研究会，KP-85-2, (Oct 1985).
[5] G.Sussman & G.Steele Jr. / CONSTRAINTS — A Language for Expressing Almost-Hierarchical Descriptions, AI, 14(1), (1980).
[6] G.Steele Jr. / The Definition and Implementation of a Computer Programming Language based on Constraints, MIT-AI-TR-595, (Aug 1980).
[7] 中島震 / COOL: オブジェクト指向と制約伝播機構，(情処)記号処理研究会，40-1, (Jan 1987).
[8] 中島震 / 制約伝播機構を内蔵するオブジェクト指向言語COOLとその応用例，WOOC87, (Mar 1987).
[9] H.Abelson & G.Sussman / Structure and Interpretation of Computer Programs, MIT Press, (1985).
[10] A.Borning / The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, ACN TOPLAS 3(4), (Oct 1981).
[11] D.McAllester / A Widely Used Truth Maintenance System, MIT AI-Lab, (1985).