制約伝播機構を内蔵するオブジェクト指向言語：COOL†

中島 震††

オブジェクト指向プログラミングと異なる観点を持つプログラミングパラダイムとして，制約指向や関係指向概念が知られている．前者によると，対象の構造的な側面の記述を自然に行えるのに対して，後者は対象間の関係記述を行うのに適している．本論文で提案する COOL は，オブジェクト指向および制約関係指向がそれぞれ持つ長所を生かすために，両者をひとつの言語システム内に融合させる試みである．構造を持つオブジェクトというモデルに加えて，オブジェクト間の制約関係をクローズで記述するためにオブジェクトを命題とみなすという考え方を導入した．オブジェクトとは独立な概念により制約関係を記述するため，ひとつの複合オブジェクト内に閉じた制約関係と複数のオブジェクトに跨る制約関係とを，ひとつの枠組みで取り扱うことができる．また，局所的な制約関係を大域的に伝播するために，関係記述の評価機構として，TMS に基づく制約伝播機構を持つ点が特徴である．最後に，COOL の記述例として，論理シミュレータを取り上げ，ゲートやフリップフロップをオブジェクト指向概念によって，接続関係を制約関係概念によってモデル化する手法を紹介し，COOL によるモデル化技法の有効性について述べる．なお，Frantz Lisp 上で実現したシステムが作動している

1. はじめに

オブジェクト指向概念は，複雑で大規模なシステムをモデル化する時に有効なプログラミングパラダイムである．オブジェクトは，データを中心として関連する手続きをまとめたモジュールであり，対象を属性の集まりとして表現するので，ものごとの構造的な側面の記述に適している．また，オブジェクト間の情報交換手段をメッセージ送受で実現することにより，オブジェクトの外部から見えるプロトコルと内部の実現方式を分離することができる．その結果，独立性の高いオブジェクトをうまく定義することで，プログラム部品として再利用することも可能となる．特に，実世界に現れる登場人物をそのままオブジェクトとしてモデル化することができるため，シミュレーション等の分野で広く使われる言語である[1]．

ところが，実世界のような複雑な系では，多数のオブジェクトがなんらかの相関関係によって結び付いていることが多い．あるオブジェクト内で起こった状態の変化は他のオブジェクトに伝播しなければならない．オブジェクト指向プログラミングは，ものごとの構造的な側面の記述に適した表現技法であるため，逆にオブジェクト間の関係を簡明に記述することが難しいという問題点がある．関係を記述する手段として，メッセージ送受以外の表現方法が必要と考えている．

一方，ものとものの間にある関係を基本概念に採用し，関係の積み重ねによってモデル化を行う関係指向や制約指向の立場もある[2]．しかし，この種の言語は，ものの構造的な側面の記述能力に欠けているため，オブジェクト指向言語が扱うような問題領域への応用が難しい．

本論文で提案する COOL (Constraint and Object-Oriented Language) は，オブジェクト指向と制約関係指向の長所を生かすために，両者をひとつの言語システム上で提供するものである．構造を持つオブジェクトという従来のモデルに加えて，オブジェクトを素命題とみなすという考え方を導入して，オブジェクト間の制約関係を命題のクローズにより記述できるようにした．制約関係を，オブジェクトとは独立な概念により記述するため，ひとつの複合オブジェクト内に閉じた制約関係と，複数のオブジェクト間に跨る制約関係をひとつの枠組みで取り扱うことができる．また，局所的な制約関係を大域的に伝播するために，関係記述の評価機構として，TMSに基づく制約伝播機構を持つ点が特徴である．本論文では，従来のアプローチの問題点を指摘した後，COOLの概要と制約伝播方式について説明し，最後に論理シミュレータの例を用いて，COOL によるモデル化技法の有効性について示す．

2. 従来のアプローチ

複数のオブジェクトが相関を持つ簡単な例として，三つの信号（青，黄，赤）からなる交通信号を考える．オブジェクト指向概念によれば，信号オブジェク

† COOL: Constraint and Object-Oriented Language by SHIN NAKAJIMA (Application System Research Laboratory, C&C Systems Research Laboratories, NEC Corporation).

†† 日本電気(株) C&C システム研究所応用システム研究部

トの性質を定義するために信号クラスを導入するのが自然である．交通信号であるから，たとえば青信号が点灯した時に，他の二つは消灯しなければならないが，この性質は信号クラスに記述することができない．他の信号が点灯したために自分が消灯するという性質は，信号のインスタンスオブジェクトごとに生じる性質であって，信号オブジェクトに共通な性質ではないからである．

Smalltalk-80 では，全体を管理する信号システムのクラスを導入し，さらに従属関係プロトコルを用いて記述する[3]．信号システムは，三つの信号オブジェクトが，互いに相関を持つことを従属関係として登録する．青信号を点灯させようとすると次のような処理を行う．青信号オブジェクトは changed メッセージを実行すると，従属関係に登録されている各信号に update メッセージを送り，青信号の状態が変わることを知らせる．そこで，信号クラスの update メソッドとして，自分自身が何色であるかを調べ，青信号以外ならば消灯するという記述を与えておく．この方法では，信号の属性の定義と信号間の相関の記述を信号クラスに記述することになり，オブジェクト記述の独立性という観点から好ましくない．関係に関する記述を，他の属性から明確に分離すべきである．

一方，制約関係を基本とする制約言語は，オブジェクト指向言語が対象とするシミュレーション等の分野に適用することが難しい．Steele の制約言語は，対象を整数に限定し，加算乗算等の基本的な代数的制約をプリミティブとして提供する．したがって，問題領域に現れるオブジェクトを，応用プログラムが整数にコード化しなければならない．また，新しいデータオブジェクトとその制約関係を導入できないという問題点がある．また，論理型言語に制約概念を導入する研究も活発化している[4]．たとえば，複雑なデータ構造を扱えるように単一化アルゴリズムを拡張する試みもあるが，シミュレーション等の分野で扱うオブジェクトまで適用範囲の広い言語はまだない．

Smalltalk に制約概念を導入すること，すなわち，オブジェクト指向と制約指向とを融合することによって，上記の問題点を解決しようとする研究もあり，ThingLab[5] や Monju[6] では，任意のオブジェクトに対する制約を取り扱うことができる．しかし，制約記述を管理するのもオブジェクトであるため，制約を課す対象のオブジェクトをすべて包含する複合オブジェクトを導入し，その複号オブジェクトに制約を記述する方式をとる．多数の対等なオブジェクトに跨る制約を記述する場合，記述対象モデルには現れない複合オブジェクトを導入せざるを得ない．特に，ThingLab では，複合オブジェクトの定義時に，構成要素オブジェクトを決める必要があるため，段階的にオブジェクトと制約を追加する問題の取り扱いが困難である．

3. COOL におけるオブジェクト

COOL では，Smalltalk 等の『構造を持ちメッセージにより処理起動されるオブジェクト』というモデルに加えて，『オブジェクトを素命題とみなす』という考え方を導入した．その結果，オブジェクト間の関係を命題のクローズにより記述できるようにした．以下，本章では，オブジェクトに関する考え方を中心に COOL の概要について述べ，次章で関係の評価機構について説明する．

3.1 構造としてのオブジェクト

オブジェクトとは，内部状態を持ったモジュールであり，そのオブジェクトの性質を特徴づける属性の集まりである．なんらかの操作を行うためには，オブジェクトに対してメッセージを送ってメソッドを起動し，メッセージはセレクタとメッセージ引数からなる．

新しい種類のオブジェクトを導入するには，クラスとメソッドを定義する．クラス定義は def-class フォームにより，メソッド定義は def-method フォームにより行う．ここで，クラス定義には，定義するクラスの名前のほかに，直接のスーパクラスのリストと新たに追加するスロットを与え，メソッド定義には，セレクタ名，そのメソッドを持つクラス名，それにメッ

```
(def-class (OneOutput) (out-pin-0))
(def-method (propagate OneOutput) (value)
  (when (neq value ($out-pin-0 self))
        (retract out-pin-0)
        ($out-pin-0! self value)
        (assume out-pin-0)))

(def-class (TwoGate (TwoInput LogicSymbol OneOutput)) ())

(def-class (TwoAnd TwoGate) ())
(def-method (eval TwoAnd) ()
  ($propagate self ($and ($in-pin-0 self) ($in-pin-1 self))))
```

図 1 クラス定義の例
Fig. 1 Example class definitions.

セージ引数のリストとメソッド本体を記述する．

(def-class (クラス名 スーパクラスのリスト)
スロット定義リスト)
(def-method (セレクタ名 クラス名)
(メッセージ引数のならび)
メソッド記述本体)

図1に，5章で述べる論理シミュレータの一部である三つのクラス OneOutput, TwoGate, TwoAnd を定義した例を示した．ここで，(\$foo bar…) はセレクタが foo でレシーバオブジェクトが bar のメッセージ式を表す．クラス OneOutput は，回路モジュールの出力ピンを定義するオブジェクトで，スロット out-pin-0 を持つ．このクラスのように，スーパクラスを省略した場合，新しいクラスはクラス階層のルートにあるクラス Object の直下に作られる．クラス TwoGate は2入力ゲートに共通な性質を定義し，スーパクラスのリストにある三つのクラスから性質を継承する．また，クラス TwoAnd は，2入力の And ゲートを定義するクラスで，クラス TwoGate のサブクラスである．出力値を計算するために eval メソッドを定義した．

3.2 命題としてのオブジェクト

『関係の評価を行う』とは，『あるオブジェクトの状態が変化した時，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』ことと考える．関連するオブジェクトの状態を変更するには，①変化の発生を伝える制御の流れを発生させること，②変更に必要な値オブジェクトを伝えること，のふたつを行う．COOL では，Steele の制約と異なり，制約伝播機構が伝播するオブジェクトの種類を限定しているわけではない．任意のオブジェクトを伝播させる必要があるため，システムが内蔵する制約伝播機構のみで関係評価を行うことができない．したがって，関係の評価処理を2段階に分けて行うことにした．すなわち，①命題としてのオブジェクト間に状態値を伝播し，②命題の状態値が変化するごとに，その命題に与えられたデーモンを実行して値を伝播する．

関係記述の意味を明確にするため，記述の枠組みとして命題論理を採用した．一連のオブジェクトが満たすべき関係を，そのオブジェクトを値として取りうるスロット間の関係として与える．値オブジェクトを素命題とみなすことに相当し，制約言語でいうセルをオブジェクトのスロットで実現するものである．

図2に関係記述の例を模式的に示した．ここで，○はオブジェクトを，＼はスロットを，△は関係を図式化したもので，三つのスロット A, B, C が値として格納するオブジェクトの間に，指定された関係△を課すことを表現する．オブジェクトの部分全体階層と独立に関係を与えることができるので，制約関係を記述するためだけに，新しいオブジェクトを導入する必要はない．

関係を定義するためには，def-relation フォームによる関係定義と，def-daemon フォームによるデーモン定義とを行う．関係定義は，関係を課すスロットを参照するためのポート名宣言と本体部からなる．後者には，命題とデーモンの対応関係を決める記述やクローズ定義等を与える．デーモン定義は，そのデーモンが対応する関係定義により宣言したポート名を引数として持つほかは Lisp 関数と同様である．

(def-relation 関係名 (ポート名のならび)
関係記述本体)
(def-daemon デーモン名 (ポート名のならび)
デーモン記述本体)

関係定義とデーモン定義の例として，ゲート間の接続関係を表す Connect の定義を図3に示した．Connect は二つのポート from, to を持ち，from が参照するスロットの値が変化した時に to で参照するスロットに変化を伝播する．そのために，add-clause フォームと add-true-daemon フォームによって，クローズとデーモンを登録する．このクローズは，(¬from)∨to を表し，from→to，すなわち，from に生じた変化を to に伝えることを示している．また

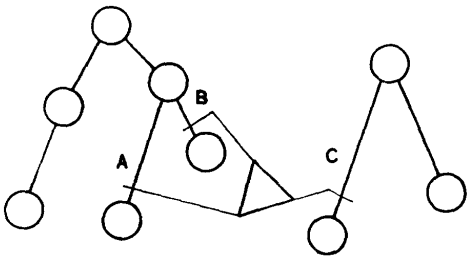


図2 オブジェクト間の関係記述
Fig. 2 Relation description among objects.

```
(def-relation Connect (from to)
  (add-true-daemons to 'transfer-daemon nil)
  (add-clause (list (false-term from) (true-term to))))

(def-daemon transfer-daemon (from to)
  (setf to from)
  ($eval self))
```

図3 関係定義の例
Fig. 3 Example relation definition.

transfer-daemon は値オブジェクトを伝播する手続きデーモンである.

オブジェクトと関係評価機構のインタフェースとなる組込みフォームとしては,クローズを登録する add-clause のほか,オブジェクトの状態が変化したことを伝える assume, assert, retract, forget-me 等がある.これらのフォームに関しては第4章で述べる.

3.3 擬変数 self

オブジェクト指向言語では,実行中のオブジェクトをメソッドから参照したい場合が多い.そのため,Smalltalk 等は擬変数 self を導入して,実行中のレシーバオブジェクトを参照できるようにしている.COOL でも同様の考え方から擬変数 self を導入した.また,デーモン本体からも self を使えるようにしてある.

メソッド本体で self を用いた場合は Smalltalk と同様に実行中のレシーバオブジェクトを参照する.一方デーモン本体で用いた場合は,制約を課したスロットを持つオブジェクトを参照する.スロットの値オブジェクトではない.

4. 関係評価機構

COOL の関係評価機構は,任意のオブジェクトを伝播するため,オブジェクトの状態が変化したことを伝播する命題間の制約伝播と,状態を変更するデーモン実行の2段階を行う.本章で,関係評価機構について述べる.

4.1 命題間の制約伝播

関係定義の本体部に与えるクローズ記述は,ポートが参照するスロットの間の局所的な制約関係を決めるのみなので,下位の関係評価機構は,この局所的な関係を基にして,大域的に状態変化を伝播しなければならない.また,関係定義を実体化してクローズを追加したり,assume フォーム等を実行して命題の状態値を変更するたびに,制約伝播を行う必要がある.すなわち,制約記述の評価機構は次の処理を行う.①評価可能な制約記述を評価してその結果を保持すること,②段階的に追加した制約記述について①を行うこと,③新たな情報の追加により①の保持結果を無効にする可能性があること,である.

TMS (Truth Maintenance System)[7] は,上に述べた三つの特徴を持つので,制約伝播機構として用いることができる.特に,COOL は,McAllester の制約伝播機構[8]を基にしている.本機構は,命題の真偽値を伝播し,命題論理の枠内で矛盾を検出し,原因を解消する.検出した矛盾を再度引き起こさないために,矛盾状態を禁止するクローズを追加する.一方,COOL のように,任意のオブジェクトを対象とする場合,制約伝播機構は,伝播するオブジェクトの性質をあらかじめ知ることができない.メソッドやデーモンにより,矛盾検出方法やその解消方法を与える必要がある.そこで,命題の状態値を五つに拡張し,過渡的な状態を導入することにより,矛盾を検出するデーモンを実行できるようにした.4.2 節で制約伝播機構について説明し,4.3 節でデーモンの実行制御について述べる.

4.2 制約伝播機構

(1) 命題の状態値は, unknown/true/false/to-true/to-false の五つの値のいずれかである.unknown は状態が確定していないことを,true は状態が確定し値が真であることを示す.また,to-true は true であるが,その命題に対してデーモンが起動されていない過渡的な状態であることを示す.

(2) 関係を命題の選言クローズ (disjunctive clause) として表現し,クローズが真である時,そのクローズが満足されていると呼ぶ.たとえば,((¬a)∨b∨c) 等により,三つの命題に関する関係を表現し,a が偽, b と c が真である時,このクローズは満足されている.

(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,命題間で状態値を伝播する.制約伝播アルゴリズムを以下に示す.

(a) assume の処理

未確定の命題を確定状態にし,クローズに対する制約を保持しながら,関連する命題の状態値を変更する.

ステップ1 : assume (P) により命題 P の状態値を設定する (unknown→true).

ステップ2 : P を含むクローズの集合 $\{C_i\}$ を求め,次のような C_i を選択する.すなわち,C_i は P と異なる命題 Q_i^j を含み,Q_i^j の状態を決定することにより,C_i が満足する.このような C_i が存在しなければ処理終了.

ステップ3 : C_i の形によって,Q_i^j の状態値を変更し (unknown→to-trun/to-false), Q_i^j に対してデーモンが定義されていれば,デーモンをキューする.同時に,Q_i^j の導出理由として C_i を設定する.

ステップ4：Q_i を基にして，ステップ2から繰り返し，大域的な状態伝播を行う．

（b） retract の処理

確定している命題を未確定状態にし，クローズに対する制約を保持するように，関連する命題の状態値を変更する．

ステップ1：retract (P) により命題 P の状態値を削除する (→unknown)．

ステップ2：P を含み，かつステップ1を行う以前には満足していたクローズの集合 $\{C_i\}$ を求める．このような C_i が存在しなければ処理を終了する．

ステップ3：C_i を導出理由として持つ命題 Q_i を求め，Q_i の状態値と導出理由を削除する．デーモンが定義されていればキューする．

ステップ4：Q_i を基にして，ステップ2から繰り返し，大域的な状態伝播を行う．

（c） add-clause の処理

新たにクローズを追加し，クローズに対する制約を保持するように，関連する命題の状態値を変更する．

ステップ1：add-clause (C) により，クローズ C を追加する．

ステップ2：C が含む命題の状態値を調べて，unknown の命題が唯一存在すれば，この命題を P とし，P に対して（a）のステップ3の処理を行う．このような命題が存在しなければ，処理を終了する．

ステップ3：（a）のステップ4と同様の処理により，大域的な状態伝播を行う．

4.3 デーモンの制御

関係評価を行うために，命題状態値の伝播だけではなく，関係定義の手続き処理を行うデーモンも実行する．COOL では，無駄なデーモンの実行を避けるために，キューベースのデーモン管理方式を採用した．

制約伝播過程で，命題の状態値が変更し，かつ該当の命題にデーモンが付与されている場合，デーモンとその起動条件の組を FIFO キューに蓄える．ここで，起動条件とは，デーモンをキューした条件，すなわち，命題とその状態値の組である．制約伝播が終了した後，キューから順次要素を取り出し，その時点で起動条件を満足しているデーモンだけを実行する．デーモン本体で，assume 等を実行し，制約伝播機構を起動することもあるため，制約伝播とデーモン実行とは，コルーチン的に互いを呼び合う形で実現する．特に，あるデーモンの実行後，すでにキューされている他デーモンの起動条件が壊われ，そのデーモンが無効になることもある．

デーモンを登録するには，add-true-daemon 等を用いる．**図3**の Connect の定義で示したように，デーモン登録フォームは，ポート名，to-true デーモン名，true デーモン名の3引数をとる．

以下，実行するデーモンを選択するアルゴリズムを示す．

ステップ1：キューから要素 (D_i, Q_i, V_i) を取り出す．ここで，D_i はデーモン，Q_i は命題，V_i はデーモンをキューする条件となった Q_i の状態値 (true, false, unknown) を示す．

ステップ2：Q_i の現在の状態値 V_c と V_i を比較し，一致する場合，デーモン D_i を実行する．ただし，V_i が true で V_c が to-true の場合，to-true デーモンを実行した後，V_c の値を V_i の値に変更する．

標準的には，to-true デーモンが値の伝播設定処理，true デーモンが値の矛盾検査処理を行うように作成する．

4.4 矛盾解消機構

メソッドやデーモンが計算した値が，既にスロットに格納されている値と矛盾する場合，組込みフォーム forget-me によって矛盾原因を解消することができる．forget-me は，引数として与えられた命題 P の導出理由をたどることにより，P を導出する原因となった命題を削除することで，P の状態値を未確定にする．以下，アルゴリズムを示す．

ステップ1：forget-me (P) により，命題 P に関する矛盾の原因を解消しようとする．

ステップ2：命題 P の導出理由であるクローズ C_0 から，P 以外の命題をひとつ任意に求めて，これを Q とする．

ステップ3：Q が assume フォームにより，状態値を設定された命題であれば，矛盾原因候補集合 L に追加する．assert フォームにより値設定された命題であれば，削除不能なので L に追加しない．また，Q が C_0 と異なる導出理由クローズ C を持つ場合はステップ4へ，それ以外はステップ5へ．

ステップ4：クローズ C から Q 以外の命題について，ステップ3を繰り返す．

ステップ5：以上のステップにより，P が矛盾する原因となった命題の集合を L として得ることができる．$L=\{R_i\}$ の中から，命題 R_i をひとつ選択して，その状態値を削除すると，retract の処理を行って，P の状態を未確定，したがって矛盾状態を解消すること

ができる．

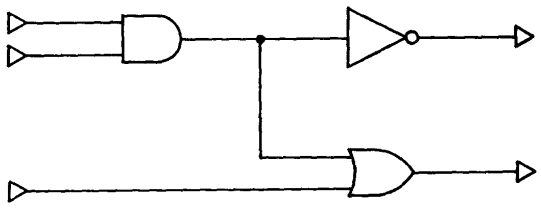
ところで，ステップ5では，R_i を選択する戦略を与えていない．どの命題を削除するかという問題は，命題に対応するオブジェクトの性質に依存するもので，下位機構は決めることができない．COOL では，プログラマが指定したクラスのオブジェクトに対して，L を引数とするメッセージ select を送る．したがって，矛盾解消機能を定義した select メソッドを持つクラスを，あらかじめ組込みフォーム add-backtracker により登録しておく必要がある．

5. イベント駆動シミュレータ

COOL のプログラム例として，論理回路の簡単なイベント駆動シミュレータを取り上げる．ゲートやフリップフロップといった回路要素と，回路要素間の接続情報によって構成される電子回路の論理シミュレーションを行うものである．

5.1 組合せ回路

組合せ回路の例を図4(a)に示した．図1のプログラム例のように，ゲートをオブジェクトとしてモデル化する．一方，接続関係は，ゲートの入出力ピンに，次のような制約関係を与えて考える．すなわち，『ソース側ゲートの出力ピンの値と，ロード側ゲートの入力ピンの値とが等しい』という制約を課す．図3に示した関係定義 Connect とデーモン定義 transfer-daemon により実現した．



(a) 組合せ回路の例
(a) Diagram of example combinatorial circuits.

```
(setq gate-1 (new TwoAnd   :print-name 'gate-1))
(setq gate-2 (new Inverter :print-name 'gate-2))
(setq gate-3 (new TwoOr    :print-name 'gate-3))

(setq in-1  (new InputTerminal  :print-name 'in-1))
(setq in-2  (new InputTerminal  :print-name 'in-2))
(setq in-3  (new InputTerminal  :print-name 'in-3))
(setq out-1 (new OutputTerminal :print-name 'out-1))
(setq out-2 (new OutputTerminal :print-name 'out-2))

(Connect ~in-1/output-pin-0   ~gate-1/input-pin-0)
(Connect ~in-2/output-pin-0   ~gate-1/input-pin-1)
(Connect ~in-3/output-pin-0   ~gate-3/input-pin-1)
(Connect ~gate-1/output-pin-0 ~gate-3/input-pin-0)
(Connect ~gate-1/output-pin-0 ~gate-2/input-pin-0)
(Connect ~gate-2/output-pin-0 ~out-1/input-pin-0)
(Connect ~gate-3/output-pin-0 ~out-2/input-pin-0)
```

(b) 回路例を生成するプログラム
(b) Program to instantiate example circuits.

図4 組合せ回路例
Fig. 4 Example combinatorial circuits.

図に示した組合せ回路を生成するためには，図4(b)のコードを実行する．ここで，new はクラス名と初期パラメータを引数として受け取り，指定したクラスのオブジェクトを返す関数であり，関係名と同名の関数（この場合は Connect）を実行して関係を実体化する．また，~foo/bar は，変数 foo が参照するオブジェクトのスロット bar に関係を課すことを示す．

入力端オブジェクトにメッセージを送って，論理値を設定することで，シミュレーションを開始する．たとえば，2番目の入力端に論理値1を設定する場合，関係 Connect により And ゲートの入力ピンに変化が伝播し，以下の過程でシミュレーションが進行する．

(1) And ゲートの入力ピンに対して，デーモン transfer-daemon が起動される．

(2) 本デーモンは，And ゲートオブジェクト (self) に eval メッセージを送り，出力イベントの生成を要求する．この結果，クラス TwoAnd が定義する eval メソッドを起動する．

(3) eval メソッドは出力論理値を計算し，自分 (self) に propagate メッセージを送る．継承解釈を行って，クラス OneOutput が定義する同名のメソッドを起動する．

(4) 新しい出力値が旧値と異なる場合，retract を実行して，出力ピンの値を変更することを伝えると，関係 Connect により本ピンに接続されている入力ピンの命題の状態値が未確定になる．

(5) 新しい出力値を設定した後，assume により，命題の状態値が再び確定したことを伝える．命題の状態値が伝播し，Inverter ゲートと Or ゲートの入力ピンの命題を to-true に設定し，これらの命題に対するデーモンをキューする．

(6) 制約伝播終了後，デーモンを起動して，Inverter ゲートと Or ゲートの入力ピンに，論理値を伝える．

5.2 フリップフロップ

前節で扱った例は組合せ回路であるため，シミュレーション過程を制約伝播過程に置き換えることが簡単にできた．つぎに，フリップフロップとクロック生

```
(def-class (FlipFlop (TwoInput LogicSymbol TwoOutput)) (state))
(def-method (eval FlipFlop) () nil)
(def-method (eval-clock FlipFlop) ()
  (cond ((eq ($in-pin-1 self) logic-1)
         ($propagate self ($in-pin-0 self)))
        ((eq ($in-pin-1 self) logic-0)
         ($state! self ($in-pin-0 self)))))
```

図 5 フリップフロップのクラス定義（部分）
Fig. 5 Class FlipFlop definition (part).

成回路のモデル化を考える．話を簡単にするため，クロックの立ち上がりで出力値を更新するエッジトリガ型フリップフロップに限定して考える．

フリップフロップは，入力データの値が変更してもクロックが入らなければ出力は変わらず，クロックの変化（立ち上がり）により出力イベントを作成する．そのため，クラス FlipFlop のイベント評価メソッドとクロック接続関係定義とを次のように変更した．①入力データピンのデーモンが起動するメソッド eval を noop 扱いにする．②クロックピンへの接続関係定義 ConnectClock を新たに導入し，クロックピンのデーモンが出力イベントを作成するメッセージ eval-clock を FlipFlop オブジェクトに送る．一方，クロック生成回路は容易に実現できて，論理値1と0とを交互に繰り返す出力イベントを作成するだけでよい．本プログラムの一部を**図5**に示した．

5.3 COOL によるモデル化技法

論理シミュレータのプログラム例から，以下の特徴を指摘することができる．

（1） ゲートやフリップフロップをオブジェクト指向概念でモデル化する

- オブジェクトとしてモデル化するのが直感的でわかりやすい
- クラス間の継承関係を用いてコンパクトな記述を行うことができる

（2） 接続関係を制約関係概念でモデル化する

- 対象回路の変更とともに，ゲートオブジェクト，接続関係等を容易に追加することができる．すなわち，段階的な制約記述の追加に対処することができる．
- 内蔵する制約伝播機構を接続関係の評価機構として用いることにより，明示的にアジェンダを作成する必要がない．

6. あとがき

本論文で，TMS に基づいた制約伝播機構を内蔵するオブジェクト指向言語 COOL を提案し，言語の概要と制約伝播機構について述べた．また，簡単な例として，論理回路のシミュレーション・プログラムを取り上げ，オブジェクト指向概念と制約関係概念を使い分けることにより記述対象モデルとプログラム・コードのギャップを小さくできることを示した．その結果，制約概念はオブジェクト指向概念と相性が良いこと，オブジェクト指向言語の代表的適用分野であるシミュレーション・システムの構築が容易になることがわかった．現在，COOL は UNIX 4.3 bsd の Franz Lisp 上で動作している．今後，標準的な矛盾解消機構を持つ組込みクラスの整備，制約概念およびオブジェクト指向概念を直感的に理解できるような視覚的プログラミング環境との結合等を行いたいと考えている．

謝辞 本研究の機会を与えて下さった日本電気(株) C&C 情報研究所山本昌弘所長代理および C&C システム研究所コンピュータシステム研究部小池誠彦課長に感謝いたします．

参考文献

1) 鈴木則久(編)：オブジェクト指向，p. 269，共立出版，東京 (1985).
2) Steele, G.: The Definition and Implementation of a Computer Programming Language Based on Constraints, AI Lab., TR-595, Cambridge MIT (1980).
3) Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and Its Implementation*, p. 714, Addison-Wesley, Massachusetts (1983).
4) Dincbas, M.: Constraints, Logic Programming and Deductive Databases, France-Japan AI and CS Symp., p. 27 (1986).
5) Borning, A.: The Programming Language Aspects of ThingLab: A Constraint-Oriented Simulation Laboratory, *ACM Trans. Prog. Lang. Syst.*, Vol. 3, No. 4, pp. 353-387 (1981).
6) Nakajima, S., Ohmori, K. and Horita, H.: Monju: Constraint-Keeping Object-Oriented Language, *Proc. Compsac '84*, pp. 232-239, Chicago (1984).
7) Doyle, J.: A Truth Maintenance System, *Artif. Intell.*, Vol. 12, No. 3, pp. 231-272 (1979).
8) McAllester, D.: An Outlook on Truth Maintenance, AI Lab., Memo-551, Cambridge MIT (1980).

(昭和62年10月28日受付)
(昭和63年11月14日採録)

中島　震（正会員）

昭和30年生．昭和54年東京大学理学部物理学科卒業．昭和56年同大学院修士課程修了．同年日本電気(株)入社．現在，同社C＆Cシステム研究所応用システム研究部主任．オブジェクト指向計算，プログラミング環境などの研究に従事．この間，昭和63年より1年間，オレゴン大学コンピュータサイエンス学科にて，知識ベースアプローチによる仕様獲得の研究に従事．日本物理学会会員．