ICOT Technical Memorandum: TM-0868

TM-0868

オブジェクト間の制約式の伝播による問題解決

横山　孝典

March. 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

標　　題

論文

オブジェクト間の制約式の伝播による問題解決

Problem Solving Based on Constraint Propagation among Objects

著者名・所属

横山　孝典　　正会員 8416995

Takanori Yokoyama

（財）新世代コンピュータ技術開発機構

Institute for New Generation Computer Technology

著者連絡先

〒108　東京都港区三田１丁目４番２８号

三田国際ビル２１階

（財）新世代コンピュータ技術開発機構

研究所第五研究室

TEL 03-456-3192

梗　概

最近様々な分野でオブジェクト指向に制約を導入したシステムが開発されている。しかし、オブジェクト指向に制約を扱う機能を単純に付加しただけでは、両者の概念がうまく融合せず、統一的なプログラミングは不可能である。本研究の目的は従来手続き的プログラミングの観点で扱われてきたオブジェクト指向を制約プログラミングの観点でとらえ直すことにより、宣言的なオブジェクト指向プログラミングの実現を図るとともに、それを知識工学へ適用することにある。本論文ではオブジェクトをデータと制約のまとまりと見なし、制約充足機能を有するオブジェクト間で制約式を伝播しあうことにより問題解決を行う計算モデルを提案する。ここで各オブジェクトは制約論理プログラミングの考え方に従った制約評価を行い、それによりオブジェクトの属性値を算出したり、他の関連あるオブジェクトへ制約式を伝播する。このモデルによれば、従来のオブジェクト指向プログラミングにおけるメソッド記述は制約式の定義として、メッセージ・パッシングは制約式の伝播として扱うことができる。そして本方式を知識表現システムに適用し、その有効性を確認する。

1. はじめに

オブジェクト指向プログラミングは自然でモジュール性のよい記述が可能なため、最近では様々な分野において広く用いられている。

しかし、従来のオブジェクト指向の枠組のみでは、オブジェクト間やオブジェクトの属性間の関係を宣言的に表現するすることができず、記述のしやすさという点で問題があった。そこで最近、知識工学、ユーザインタフェース、ＣＡＤ等の分野で、オブジェクト指向に制約を導入することによりこの問題を解決しようという試みがなされるようになった。

しかし、従来手続き的プログラミングの観点で扱われてきたオブジェクト指向言語に、宣言的プログラミングの考え方に立つ制約概念を単純に導入にしただけでは両者の概念が融合せず、効率的なプログラミングが難しくなる。この問題を解決し、統一的なプログラミングを実現するには、単に制約を扱う機能を付加するのではなく、むしろオブジェクト指向を制約の概念でとらえ直し、改めて宣言的なオブジェクト指向プログラミングの実現を目指すべきであると考える。これは特に知識表現など、宣言的記述が重視される分野において大きな意義がある。

本研究は知識工学への応用を目的に，制約の導入により，宣言的プログラミングとしてのオブジェクト指向を実現することを目指している。本論文では，オブ

ジェクトをデータと制約のまとまりと見なし，制約充足機能を有するオブジェクト間で制約式を伝播することにより問題解決を行う計算モデルを提案する。そして、このモデルを知識表現システムの問題解決機構に適用する。

以下ではまず，オブジェクト指向への制約の導入の背景と意義について述べるとともに，従来システムの問題点について論じる。次に制約概念を導入し、オブジェクト間で制約式を伝播することにより問題解決を行う計算モデルを提案し、これにより従来手続き的プログラミングの立場でとらえられてきたオブジェクト指向を宣言的プログラミングの観点でとらえ直すことができることを示す。そして，オブジェクト間の制約式の伝播による問題解決アルゴリズムについて説明した後，実際にこれを適用した知識表現システムを紹介する。最後に今後の課題について述べる。

2. オブジェクト指向への制約の導入

2.1 オブジェクト指向計算モデル

オブジェクト指向プログラミングにおけるオブジェクトはひとまとまりのデータとそれに関する手続きをまとめたものと見なされる。[1)2)] データはスロット（インスタンス変数、クラス変数）で表現されるオブジェク

トの属性や構成要素などの情報である。手続きはメソッドで表現され、メソッドはメッセージを受信することにより起動される。そしてオブジェクト間でメッセージを送信しあうことにより，問題解決が進む。

データと手続きを統一的に扱うという考え方はデータ抽象に基づくもので、モジュール性がよく、わかりやすい記述が可能になる[3]。データ抽象の重要な考え方に情報隠蔽，あるいはデータのカプセル化と呼ばれるものがある。情報隠蔽は対象物のデータ構造の詳細を隠蔽し、それに対する手続きによって対象物の外部的な挙動を特徴付けている。オブジェクト指向もこの考え方を踏襲している。

オブジェクト指向が採用しているメッセージ・パッシングに基づく計算モデルの基礎となる理論にアクタ理論[3][4]がある。アクタ理論によれば従来の手続き的なものとデータ的なものをアクタとして統一的に扱うことができる。またメッセージの流れはデータの流れと制御の流れを統一化したものと見ることができる。このようなアクタ理論を背景としたオブジェクト指向プログラミングは優れた統一的計算モデルである。

知識表現の分野では古くからフレーム[5]による表現が広く用いられており、最近ではオブジェクト指向と融合、統合化されてきている[6]。オブジェクト指向表現を用いれば、対象とする問題における「もの」をオブジェクトで表現し、属性をスロットで、動作をメソッドで表わすことができる。また、クラスの継承機能を利

用して概念間の抽象・具体関係を表現することができる。このような特徴から、現在オブジェクト指向は知識表現の標準的な手法のひとつになっている。

2.2 制約の導入

前述のようにオブジェクト指向は知識表現に適しているが、従来のオブジェクト指向言語ではオブジェクト間の関係や属性間の関係をうまく表現する手段は提供されておらず、メソッドにより手続き的に表現する必要があった。このため、これらの関係を理解しやすい宣言的表現で記述できる機能が要求されている。

関係の宣言的記述法として、人工知能の分野では以前から制約表現が用いられてきた[7)8)]。そこで、近年，オブジェクト指向に制約を導入することにより上記の問題を解決する手法が提案されている[9)~16)]。制約表現を用いることによりオブジェクト間や属性間の関係を宣言的に記述できる。

しかし，従来手続き的プログラミングの視点でとらえられてきたオブジェクト指向の枠組みに，単に制約を扱う機能を付加しただけではオブジェクトと制約の概念がうまく融合せず、いくつかの問題が生じる。

まず、オブジェクト指向は全てをオブジェクトで表現できるという利点があったが、オブジェクト以外の制約という概念の導入はこの利点を失い、統一的表現を損なう可能性がある。

また、従来のオブジェクト指向言語ではオブジェクトの動作がメソッドという手続き的表現により記述され，メッセージ・パッシングはメソッドの起動という手続き的な意味を持つ。これに対し、制約表現は宣言的プログラミングの立場に立つものであり、統一的なプログラミングが不可能となる。

さらに、オブジェクト指向に制約を導入し，異なるオブジェクトのデータ間の制約を記述可能とすることは，オブジェクト指向の基本的な考え方である情報隠蔽に反し、モジュール性のよいプログラミングという特徴を損なう可能性がある。

また，制約充足方法についても課題がある。オブジェクト指向に制約を導入した従来のシステムでは，制約単位に制約充足手続きをメソッド（あるいはルールやデモン）の形で表現するか，局所伝播（local propagation）や弛緩法（relaxation method）のような制約充足アルゴリズムを利用しているが，オブジェクトと制約の両概念に一致した，より自然でかつ効率的な制約充足法が望まれる。

以上の問題を解決するため、本論文では、オブジェクト指向を制約プログラミングの立場でとらえ直すことにより、従来のオブジェクト指向を包含する新しい計算モデルを提案する。そしてこれにより宣言的なオブジェクト指向プログラミングの実現を目指す。

3. 制約式の伝播によるオブジェクト指向計算モデル

3.1 オブジェクトと制約

一般に、オブジェクトはひとまとまりのデータとそれに関する手続きをまとめたものと見なされる。これに対し、ここではオブジェクトをデータと制約をまとめたものと考える。そしてオブジェクトは常に制約充足状態を維持する機能を有するものとし，オブジェクトが満足すべき制約を宣言的に記述したり、動的に付加するのみで，オブジェクトが自律的に問題解決を行う。すなわち，

オブジェクト ＝ データ ＋ 制約の記述
＋ 制約充足機構

と表現できる。

制約は数式や述語の形で記述でき、これらを制約式と呼ぶことにする。実際の問題ではそれに適した様々な形の制約式が用いられるが、それらを全て直接解釈することのできる制約充足機構をシステム提供することは困難である。従って、一般には基本的な制約式のみをシステムがサポートし、問題に合わせてユーザが自由に制約式を定義可能とするのがよい。この場合制約充足機構は制約式の定義を参照しながら制約式の評価を行うことになる。

オブジェクト指向に制約を導入した従来システムのいくつかは，制約をオブジェクトとは独立の概念と見なしたり，制約オブジェクトという概念を導入する等

の方式を採用している。しかし、複数のオブジェクト間に制約が発生するのは，それらのオブジェクトを含む上位のオブジェクト，あるいは場（これもオブジェクトで表現できる）が存在するためと考えるのが自然である。従って制約は上位のオブジェクトに記述すればよく，制約をオブジェクト外の独立した概念とする必要はないと考える。

制約記述の例を図１に示す。この図はオブジェクトｏがその構成要素であるオブジェクトａとｂを含んでおり、構成要素の属性に関する制約が上位のオブジェクトｏに記述されている。ここで、“ａ→ｘ”は構成要素ａの属性ｘを表わす。 図1

3.2 情報隠蔽

制約の導入が情報隠蔽に反する可能性があることは既に述べたが、オブジェクト指向における情報隠蔽はもともとデータ抽象の考え方に従ったものである。すなわち、図２（ａ）のように、データ抽象ではデータを公開せず、データへのアクセスは特定の手続きを介して行う必要がある。同様にオブジェクト指向ではスロットのデータは外部からは見えない。しかし、メッセージの形式，すなわち，メソッド名、引数などのインタフェース情報は外部に公開する必要がある。これは、データ抽象が手続き的プログラミングの立場であり、（外部から見える）オブジェクトの性質（機能） 図2

をそれが持つ「手続き」によって特徴づけるという考え方を採用しているためと考えられる。

これに対し，宣言的プログラミングの立場からは，（外部から見える）オブジェクトの性質は，手続きでなく，それが持つ代表的な「属性」データによって特徴づけられるという考え方を採用するのが自然である。従って，図２（ｂ）のように、必ずしも全てのデータを隠蔽する必要はなく，オブジェクトの抽象的性質を表す属性（データの一部）は公開してよいと考える。

外部から見えるオブジェクトの性質を表わすデータを「公開属性」と呼ぶことにする。公開属性は外部から参照でき，複数のオブジェクトの公開属性間に制約を定義できる。もちろん，宣言したオブジェクト内であれば公開されてないデータ間にも制約を定義できる。公開されていないデータを内部属性と呼び、単に属性という場合には両者を含むものとする。公開属性という考え方はデータ抽象の考え方とは異なるが，宣言的プログラミングや知識表現においては自然な抽象化であると考える。

なお，オブジェクトの構造の詳細は本方式でも隠蔽され，制約式中で参照できるのはその制約を記述したオブジェクトの属性と、そのオブジェクトが含む（構成要素）オブジェクトの公開属性のみである。

3.3 制約伝播

一般のオブジェクト指向ではオブジェクト間のメッセージ・パッシングにより処理が進むのに対し，ここでは、図３のように制約充足機能を有するオブジェクト間の制約伝播により処理が進むという計算モデルを提案する。このモデルでは、各オブジェクトは伝播されてきた制約を評価し、制約充足状態を保つため、必要によりその状態を変化させたり、他のオブジェクトへ制約を伝播する。 図3

従来のシステムにおける制約伝播では、それによって伝達される情報は属性値であることが多い。これに対し、本方式で伝達されるものは述語や数式のような制約式そのものである。制約式で表現されるものは属性値の範囲，複数の属性間の関係，オブジェクト間の関係，オブジェクトがとりうる型（クラス）等である。制約式については標準的な形式についてはシステム提供するものとし、その他，それぞれの問題に適した制約の形式をユーザが定義することも可能とする。

本方式では、静的に宣言される制約のみでなく、動的にオブジェクトに付加される制約も扱う。前者はクラスに記述され，変更したり，削除したりすることはできない。これを「恒久的制約」と呼ぶことにする。オブジェクトは恒久的制約を満足した状態で生成される。後者は処理の実行中に動的に付加され，不必要になれば削除することができる。これを「動的制約」と呼ぶ。ただし動的制約は付加された後，明示的に削除されない限りオブジェクトは常にそれを満足する必要

がある。

さらにここでは，付加時に一時的に作用するものの，一旦制約充足が終了した後は無効となる制約を導入する。これを「一時的制約」と呼ぶ。すなわち、恒久的制約及び動的制約はオブジェクトに記憶されるが，一時的制約は記憶されない。

制約が与えられたときのオブジェクトの基本動作を図4により説明する。図4はオブジェクトの概念的な構造を表わしたものである。まず制約式が伝播されてきたならば，制約評価機構は制約式の定義を参照しながら，与えられた制約式と記憶されている制約式を同時に評価する。制約評価の基本は制約論理プログラミング[17]の考え方に従っており，評価とは制約式を簡約化すること（標準形への変換）である。簡約化により，属性値を算出したり，他の特定のオブジェクトのみに関する制約式を抽出することができる。

評価結果はオブジェクトの状態に反映される。すなわち，得られた属性値はデータ記憶に格納され，伝播されてきたのが一時的制約でなければ，簡約化された制約式は制約式記憶に記憶される。また、他のオブジェクトに関する制約式が得られれば，制約伝播機構により、その制約式を伝播先のオブジェクトに合った形に変換した後（例えば制約式 a→x + a→y = 3 を構成要素 a に伝播する場合 x + y = 3 と変換する）、そのオブジェクトに伝播する。与えられた制約が通常の動的制約であれば伝播する制約も動的制約，一時的制

約であれば伝播する制約も一時的制約として扱う。

このようにオブジェクトは伝播されてきた制約式を評価し、その評価結果に従って状態を変化させ、他のオブジェクトへの制約伝播を行うことにより、全体として計算が進む。制約の評価と伝播の詳細については4節で詳述する。

なお，制約充足に基づく問題解決では，制約を大域的にとらえ，全ての制約をまとめて解く手法も考えられる。しかし，オブジェクトを動的に生成、消去したり、オブジェクトの構造が動的に変化するような問題では，制約を大域的に扱うよりも、本方式のようにオブジェクト単位に扱う方がよい。また，一般に制約式の数が増えるとそれを解くのに有するコストは指数関数的に増加する可能性があり，できる限り部分問題に分割して解くことが重要である。一般に知識工学が対象とするのはこのような問題であり，ここで提案した計算モデルが有効になる。

3.4 統一的プログラミング

以上述べた計算モデルでは，伝播される制約は伝播先のオブジェクトの動作を規定するものであり、一般のオブジェクト指向におけるメッセージと見なすことができる。この考え方に従えばメッセージ・パッシングと本方式の制約伝播を統合することができる。ただし，通常のメッセージパッシングは副作用的にオブジ

ェクトの状態を変更するが，その後も作用し続けることはないから，一時的制約の伝播と見なすべきである。

また、伝播されてきた制約は制約式の定義に従って評価される。従って制約式の定義とメッセージ受信時のオブジェクトの動作を定義するメソッドを統合してとらえることができる。この考え方に従えば、一般のオブジェクト指向言語におけるメソッド記述は制約式（メッセージ）の定義、すなわち解釈法を手続き的に記述したものと見なせる。

以上のように本方式の計算モデルにおいては、メッセージパッシングを制約伝播の一種、メソッド記述を制約式の定義を手続き的に記述したものと見なすことができる。これによりそれらを統合化でき、統一的なプログラミングが可能となる。また、従来手続き的プログラミングの視点で扱われてきたオブジェクト指向を宣言的プログラミングの立場から再構築することができる。

なお、オブジェクト指向の特徴のひとつであるクラスの継承機能はそのまま本方式に適用でき，オブジェクトが満足すべき制約や制約式の定義を継承させることができる。

4. 制約充足方式

4.1 制約式の定義

オブジェクト指向に制約を導入した従来のシステの多くは数式など、システムが直接解釈できる形式の制約しか扱えなかった。これに対し、本方式では数式のほか、述語表現された制約も扱うことができ、ユーザは問題に適した制約式を定義して使うことができる。

制約式の定義は

```
dif_1(X, Y) :- X - Y = 1;
```

のようなホーン節の形式で記述できる。これは変数 X と Y に関する制約 dif_1 が X と Y の差が 1 であるということを表現している。また、制約式をそのクラスに属するオブジェクトの属性に依存した形で、例えば

```
dif_p_q(X) :- p - q = X;
```

のように定義することもできる。ここで p、q はオブジェクトの属性名であり、制約 dif_p_q(X) は属性 p と q の値の差が X であることを表現している。

このような制約式の定義はそれが宣言されたクラス（およびその下位クラス）に属するオブジェクトのみで有効である。

本方式では、オブジェクト間で伝播される制約式は基本的にはそのシステムの制約充足機構が直接解釈できる形式であるが，ユーザ定義の制約式の伝播も可能とする。

例えば上位オブジェクトに制約式 dif_1(p, a→r) が存在し，p = 2 が与えられ、この制約式の定義がそのオブジェクトのクラスで定義されていなければ，

dif_1(2, a→r) は構成要素 a のみに関する制約式なので，そのオブジェクトに伝播される。そして、そのオブジェクトで上記のように制約式 dif_1(X, Y) が定義されていれば、そのオブジェクトの属性 r の値は 1 と決定される。

ところで，制約式の定義に従って一般的手法により制約式を解くことはコストが大きいので，制約式の定義（解き方）をオブジェクト（クラス）毎にドメインに依存したヒューリスティクスを含んだ手続き的な形で記述することが考えられる。この定義はそのオブジェクト（クラス）内でのみ有効なものとなる。

さらに，あらかじめ制約式を伝播すべきオブジェクトがわかっている場合は，制約式の定義を例えば

```
dif_p_ar(X) :- Y = p - X,
                    propagate(a, dif_r(Y));
```

のように記述し，制約 dif_r(Y) を構成要素 a に伝播することを明示的に指定することも考えられる。

以上のように本方式では伝播されてきた制約式はオブジェクト毎に制約式の定義に従って評価される。また、制約式の定義をヒューリスティクスを含んだ手続き的な形で記述したり、制約式の伝播先を指定することもできる。従って本方式における制約式の定義と制約伝播はオブジェクト指向のメソッド記述とメッセージ・パッシングを包含するものと考えられる。

4.2 制約式の簡約化アルゴリズム

各オブジェクトにおける制約評価は制約式の簡約化である。制約式の簡約化は次の手順により行う。

(i) 記号を持つ制約式が存在すればその記号の消去を試みる

(ii) 構成要素のオブジェクトの属性を含む制約式があれば，ひとつの構成要素の属性のみを含む制約式に局所化する

(iii) 制約式を標準形に簡約化する

(i) は 4.3 節で述べるようにオブジェクトの共有により制約式の参照関係のループが生じる時の制約充足に必要な処理である。(ii) は構成要素のオブジェクトに関する制約を抽出し、制約伝播を実行するのに必要な処理、(iii) は制約式を解き、属性値を算出するのに必要な処理である。

(iii) は制約解消の中心となる処理であり、制約式の定義にホーン節を用いれば，その処理は制約論理プログラミングの処理系と同様の動作である。すなわち、ひとつのオブジェクト内における制約は制約論理プログラミングの考え方に従って扱われる。

4.3 制約式の伝播アルゴリズム

各オブジェクトは制約式の評価結果に従って，必要により以下の手順で制約式の伝播処理を実行する。

(i) ひとつの構成要素のオブジェクトのみの属性

を含む制約式はそのオブジェクトに伝播する

(ii) 上位オブジェクトから参照されている属性値が得られたら，それを上位オブジェクトに伝播する

(iii) 複数の上位オブジェクトを持つオブジェクトの場合，複数の親から参照されている属性が存在したり，異なる親から参照されている属性間に制約が存在する時には，属性の値を記号化して表現し，上位オブジェクトに伝播する

(i)、(ii) は本計算モデルの基本動作である。(iii) は構成要素の共有により制約式の参照関係にループが存在する場合に必要な処理である。

制約伝播の例を図5に示す。ここでオブジェクトa、b は上位オブジェクトoの構成要素である。ここでオブジェクトbの属性 z の値に関する制約が与えられた（属性値設定）とすると(1)、オブジェクトbで制約評価を実行し属性 y の値が決まり(2)、それが親オブジェクトoに伝播される(3)。oでの制約評価により構成要素 a の属性のみに関する制約式が得られるので(4)、これをオブジェクトaに伝播し(5)、ここでの制約評価によりオブジェクトaの属性値が決定される(6)。 図5

複数のオブジェクト間に構成要素の共有があり、制約式における参照関係のループを生じる場合には (iii) の処理により、共有しているオブジェクトの持

つ属性のうち複数の親から参照されているものの値を記号化して表現し，親のオブジェクトに伝播する。記号化された属性を受け取ったオブジェクトはその記号を含む制約式をその親に伝播していく。そして上位の共通の親のところで，記号を優先的に消去することにより解くことができる。

図6は，オブジェクトaの構成要素pとオブジェクトbの構成要素qが同一のオブジェクトrを共有し、制約式の参照関係にループが生じる例である。この場合には(1)から(5)のように、オブジェクトrの属性uの値を例えば記号αで表現してその親であるオブジェクトaとbに伝播し，さらにそれを含む制約式をオブジェクトoに伝播し，4.2節（i）の処理により、αを消去して制約式を解くことができる。そしてこの後、通常の制約伝播により全ての属性値が決定される。 図6

5. 知識表現システムへの適用

5.1 知識表現システム FREEDOM

以上提案したアルゴリズムを知識表現システムFREEDOM[16]の問題解決機構に適用した。FREEDOMは設計対象モデルの表現を目的に開発したもので、オブジェクト指向を基本に制約の宣言的記述を可能とし、制約充足に基づく問題解決を行うシステムである。

FREEDOM におけるオブジェクトは制約充足状態を維持する機能を有し、複数のオブジェクトがが互いに制約伝播を行いながら問題解決を行う。そしてオブジェクトは制約充足のため、その属性値や、構造、属するクラスなどを変更することができる。

しかし、FREEDOM はオブジェクト指向と制約を有機的に統合化することを目指したものの、最初のバージョンでは統合化が不十分で、制約充足とメッセージ・パッシングを併用する必要があった。そこで、本論文で提案した手法を適用することにより，制約充足による統一的な問題解決の実現を試みた。

本方式を適用した FREEDOM のクラス定義の文法を図７に示す。一般にオブジェクト指向言語ではクラスの継承機能を提供しているが、FREEDOM ではこのうち、意味的な上位概念（抽象・具体関係）を表わすものと、単なる機能の包含関係を表わすものとを明示的に区別して表現する。前者は“is_a”関係、後者は“includes”関係で表現する。また、“consists_of”でそのオブジェクトの構成要素（全体・部分関係）を表現する。オブジェクトの属性は“attributes”のところに、オブジェクトが満たすべき制約は“constraints”のところに記述する。制約式の定義は“def(制約式) :- 定義;”の形式で記述する。 図7

従来はメッセージ送信やメソッド定義等、手続き的な記述が必要であったが、本方式の採用により、制約の記述、制約式の定義のような宣言的な記述が可能に

なった。ただし、現在はシステムが能動的な制約評価、すなわち制約式の簡約化を行うのは方程式に限っている。その他の制約については、その解法を制約式の定義において記述する必要があり、Prolog 的な述語呼び出しがなされる。

FREEDOM はＥＳＰ[18]で記述され、その制約評価機構は Buchberger アルゴリズムを採用した制約論理プログラミング言語ＣＡＬ[19]の制約評価系を利用している。

5.2 例題

FREEDOM を簡単な設計問題に適用した例を紹介する。一般に設計は一定の制約条件のもとで要求仕様を満足する解を生成する問題で、要求仕様を設計解に関する制約とみなすことにより、制約充足問題として扱うことができる[20]。ただし、通常の問題では設計対象および要求仕様から導かれる制約のみで解を一意に決定することはできず、何らかのヒューリスティクスを必要とする。しかし、このヒューリスティクスも制約表現可能な場合には設計全体を制約充足問題として統一的に扱うことができる。

このような問題に本方式を適用すると、本来設計対象が持っている制約は恒久的制約、与えられる要求仕様は動的制約として扱うことができる。またヒューリスティクスとしての制約は必ずしも成り立つとは限らないが解の導出に役立つものであるから、一時的制約

として扱うのが都合がよい。もちろん実際の設計問題は非常に複雑で、必ずしも制約充足問題として定式化することは容易ではないが、設計対象が構造の良い（well-structured）問題であればこの枠組で対応できる。このような問題の例として、ここでは電子回路の回路定数を求める問題を取り上げる。

図８（ａ）は設計対象の増幅器のクラス定義の例である。電子回路の諸定数間の制約が等式で表現されている。FREEDOMでは構成要素ａの属性ｘを参照する場合、“ａ:->ｘ”と記述する。

設計手順の例を図８（ｂ）に示す。まず、２段増幅器のオブジェクト（インスタンス）を生成する(1)。次に要求仕様が電子回路の属性に関する制約として与えられるので、これをオブジェクトに付加する(2)。しかし、この状態では制約が不足しているため、全ての属性値を一意に決定するには至っていない。そこでさらに、通常よく用いられる設計式や経験上多用される属性値を一時的制約として付加する(3)。これはこの種の回路設計におけるヒューリスティクスである。これにより、全ての属性値が決定される。実際、(4)でオブジェクトの属性値を読み出すと、得られた値を知ることができる。なお、(3)で与えた属性値はあらかじめデフォルト値として与えておいてもよい。

以上のようにオブジェクトに制約表現された要求仕様やヒューリスティクスを付加するのみで、オブジェクトの持つ制約充足機能により自動的に属性値を決定

することができる。なお、ここでは属性値の決定のみを行っているが、FREEDOM は制約充足によりオブジェクトの種類や構造をも決定する機能を有している。

6. 今後の課題

以上オブジェクト間の制約式の伝播による問題解決方式を提案し、知識表現システム FREEDOM への適用を図った。ただし現在の FREEDOM はプロトタイプであり、実用化を目指し、システムが能動的に評価できる制約の範囲の拡張と処理効率の向上を図る必要がある。

前述のように FREEDOM が能動的に制約評価を行うのは方程式のみであり、その他の制約式の評価は制約式の定義に従って Prolog 的な呼び出しを行うのみである。従ってそれらの制約の評価アルゴリズムはユーザが記述する必要がある。しかし、実際の問題では等式以外にも多くの種類の制約が用いられており、よく使用される制約についてはその評価系をシステム提供することが重要である。制約論理プログラミングにおいて述語表現された組み合わせ的な制約を扱うのに有効な手法として展開／畳込みプログラム変換が知られており[21]現在その導入を検討中である。

問題解決効率の向上については、まず、制約評価の効率化が必要である。現在制約式の簡約化はＣＡＬの制約評価系に本方式のための機能を付加して使用して

いるが、効率化を図るため、専用の制約評価系を開発することを考えている。

また、大規模な問題を扱うには、制約評価のみでなく処理全体を高速化する必要がある。そこで現在、本方式の並列処理への展開を検討中である。オブジェクト指向は並列性を自然に記述できるものとして注目され、並列処理を前提としたオブジェクト指向言語も多数提案されており、本方式も並列化が可能と考えられる。ただしそのためには、複数のオブジェクトにまたがる制約を並列に効率良く解く手法を実現する必要があり、大きな課題である。

7. おわりに

以上オブジェクト指向と制約を融合する手法を提案した。オブジェクト指向言語への制約の導入はこれまでいくつかの試みがあるが、情報隠蔽の考え方に反する可能性があることや、手続き的プログラミングと宣言的プログラミングの手法が混在していること、統一的な計算モデルとなっていないことなど、いくつかの問題があった。

本論文では、公開属性による抽象化という考え方を導入するとともに，各オブジェクトが制約論理プログラミングの考え方に従った制約評価機構を有し、制約式の簡約化処理を実行し、それによりオブジェクトの属性値の算出や他の関連あるオブジェクトへの制約式

の伝播を実行するという計算モデルを提案した。

さらに、従来のオブジェクト指向言語におけるメッセージ・パッシングは（伝播先を指定した一時的）制約の伝播、メソッドの記述は制約式の（手続き的な）定義として扱えることを示し、オブジェクト指向と制約を融合した統一的なプログラミングを実現できることを明らかにした。これにより、宣言的オブジェクト指向プログラミングを実現できる。そして本方式を宣言的記述が重要視されている知識表現に適用し、その有効性を確認した。

謝辞

本研究の機会を与えていただくとともに御指導いただいたＩＣＯＴの淵一博所長、古川康一次長、生駒憲治第五研究室長、ＣＡＬの処理系を提供していただいたＩＣＯＴ第一研究室の坂井公氏、相場亮氏、インプリメントに御協力いただいたＮＴＴデータの佐塚秀人氏、有益な御討論をいただいたＩＣＯＴ第五研究室の皆さんに感謝する。

参考文献

1) Goldberg, A. and Robson, D. “Smalltalk-80 : The Language and Its Implementation”, Addison-Wesley (1983)

2) 米澤明憲“オブジェクト指向プログラミングについて”, コンピュータソフトウェア, vol.1, no.1, pp.29-41 (1984)

3) 木村泉, 米澤明憲“算法表現論”, 岩波書店 (1982)

4) Hewitt, C. “Viewing Control Structures as Patterns of Passing Messages”, Artificial Intelligence, vol.8, pp.323-364 (1977)

5) Minsky, M. “A Framework for Representing Knowledge”, in Winston,P.H.(ed.) “The Psycology of Computer Vision”, McGraw-Hill (1975)

6) 溝口文雄“オブジェクト指向概念による知識表現言語”, 情報処理, vol.29, no.4, pp.382-389 (1988)

7) Stallman, R.M. and Sussman, G.J. “Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis”, Artificial Intelligence, vol.9, pp.135-196 (1977)

8) Sussman, G.J. and Steel, G.L.Jr. “Constraints- A Language for Expressing

Almost-Hierarchical Descriptions", Artificial Intelligence, vol.14, pp.1-39 (1980)

9) Borning, A. "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", ACM Trans. on Prog. Lang. and Syst. vol.3, no.4, pp.353-387 (1981)

10) Borning, A. and Duisberg, R. "Constraint-Based Tools for Building User Interfaces", ACM Trans. on Graphics, vol.5, no.4, pp.345-374 (1986)

11) Harris, D.R. "A Hybrid Structured Object and Constraint Representation Language", Proc. of AAAI-86, pp.986-990 (1986)

12) Buchmann, A.P. et al. "A Generalized Constraint and Exception Handler for an an Object-Oriented CAD-DBMS", International Workshop on Object-Oriented Database Systems, pp.38-49 (1986)

13) Struss, P. "Multiple Representation of Structure and Function", in Gero, J.(ed.) "Expert Systems in Computer-Aided Design", North-Holland (1987)

14) Szekely, P.A. and Myers, B.A. "A User Interface Toolkit Based on Graphical Objects and Constraints", OOPSLA '88 Proceedings, pp.36-45 (1988)

15) 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”，情報処理学会論文誌，vol.30，no.1，pp101-108，(1989)

16) 横山孝典，佐塚秀人 “制約に基づくオブジェクト指向知識表現システム”，情報処理学会論文誌，vol.31，no.1，pp.68-75 (1990)

17) Jaffar，J. and Lassez，J.-L. “Constraint Logic Programming”，Proc. 14thw ACM Symposium on Principles of Programming Language，pp.111-119 (1987)

18) Chikayama，T. “ESP Reference Manual”，ICOT Technical Report，TR-044 (1984)

19) Aiba，A. et al. “Constraint Logic Programming Language CAL”，Proc. of International Conference of Fifth Generation Computer Systems 1988，pp.263-276 (1988)

20) Tong，C. “AI in Engeneering Design”，Artificial Intelligence in Engineering，vil.2，no.3，pp.130-132 (1987)

21) 橋田浩一 “情報の部分性と制約プログラミング” 淵一博（監修）溝口文雄，古川康一，Lassez，J-L.（編）“制約論理プログラミング” 第7章，共立出版 (1989)

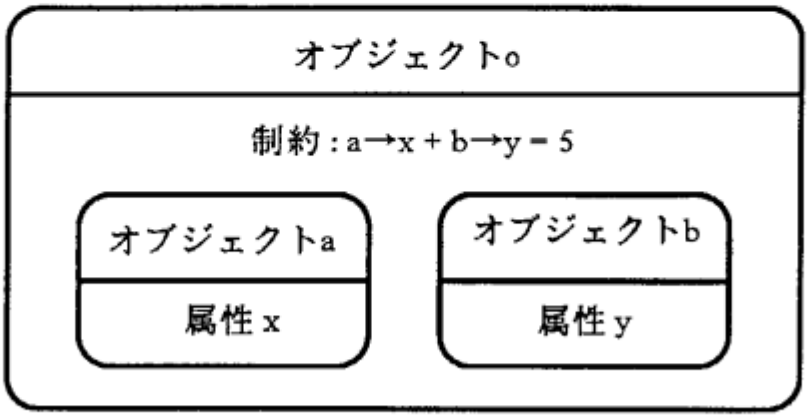


図 1 制約記述の例

Fig. 1 Example of constraint

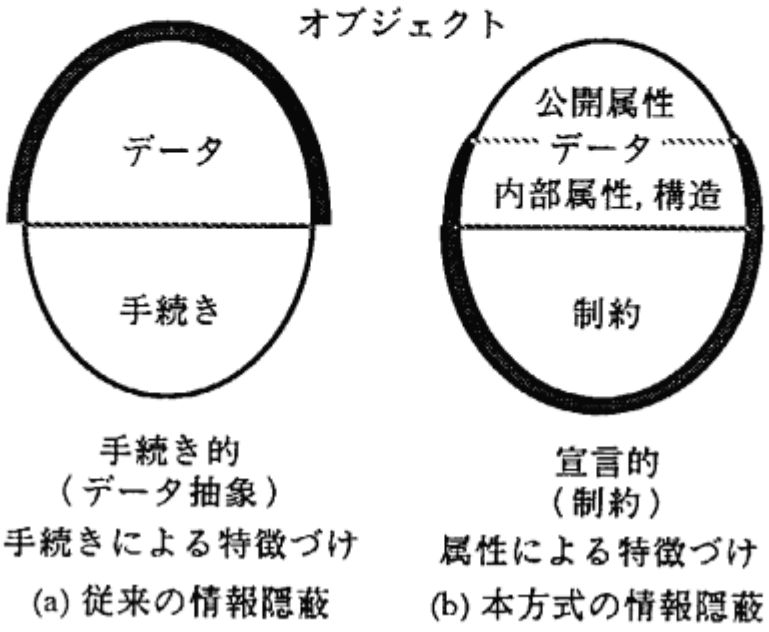


図２ 情報隠蔽の考え方

Fig.2 Encapsulation

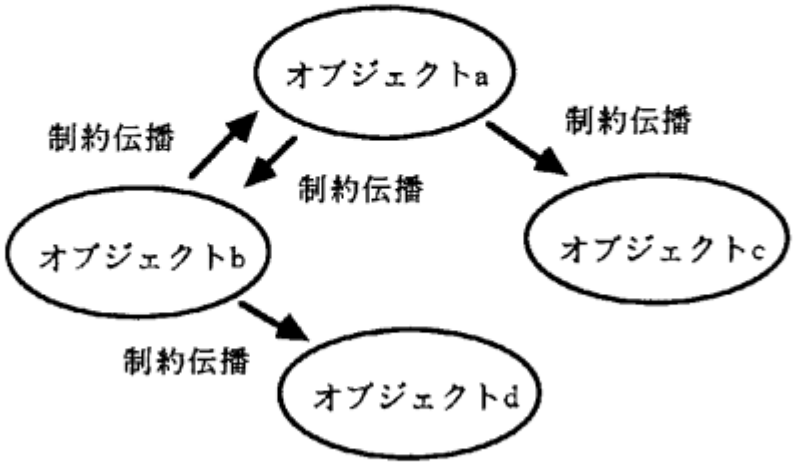


図３ 制約伝播に基づく計算モデル

Fig.3 Computation model based on constraint propagation

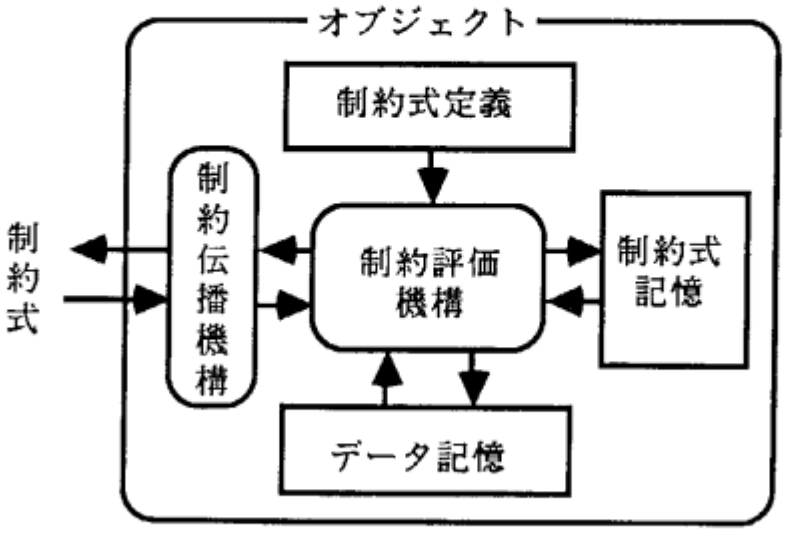


図4 オブジェクトの構造

Fig.4 Structure of Object

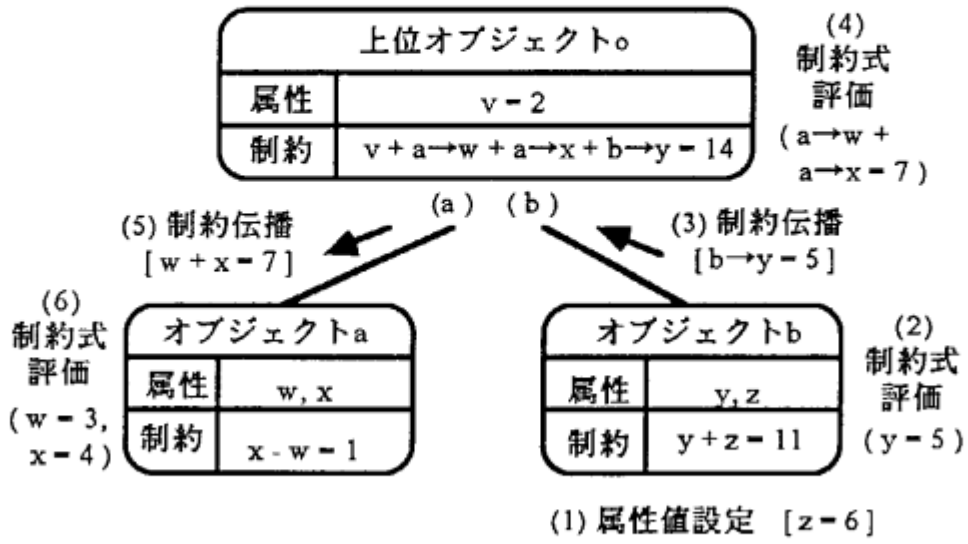


図5 制約伝播の例

Fig.5 Example of constraint propagation

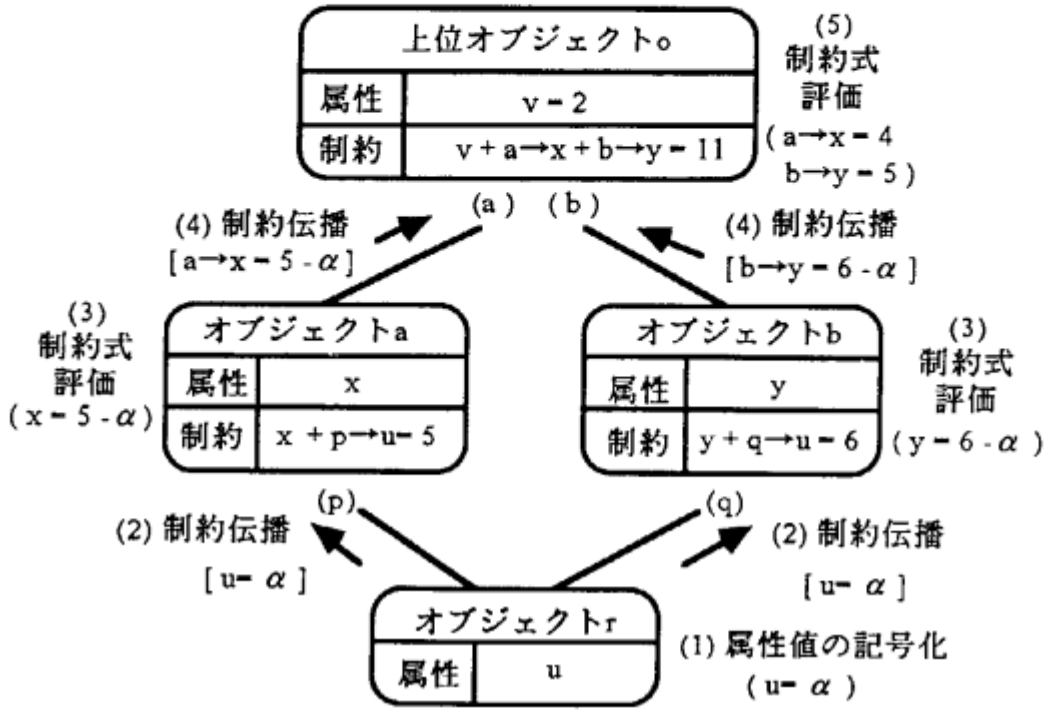


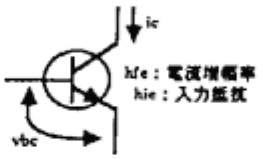
図6 参照関係にループが存在するときの制約伝播の例

Fig.6 Example of constraint propagation with cyclic dependency

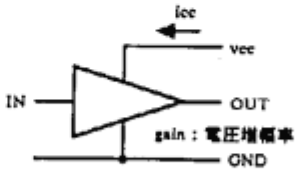
```
fr_class  クラス名  has
    is_a  上位クラス名 ;
    includes
        [ キー名 :: ] 包含クラス名 ,
                  ・・・          ;
    consists_of
        構成要素名 [ ::= デフォルトクラス名 ],
                   ・・・                  ;
    attributes
        属性名 [ ::= デフォルト値 ],
                  ・・・          ;
    constraints
        制約式 ,
        ・・・ ;
    def( 制約式 ) :-  制約式の定義 ;
                 ・・・          ;
fr_end.
```

図7 クラス定義形式
Fig.7 Syntax of class definition

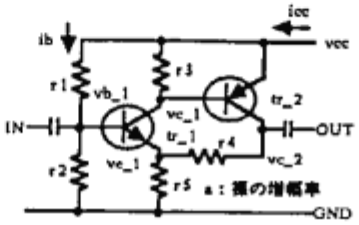
```
%%%  トランジスタ  %%%
fr_class  transistor  has
    attributes
        hfe,  hie,  vbe,  ic;
    constraints
        vbe = 0.6,
        hie * ic = 0.026 * hfe;
fr_end.
```




```
%%%  電圧増幅器  %%%
fr_class  voltage_amp  has
    attributes
        gain,  vcc,  icc;
fr_end.
```




```
%%%  2段増幅器  %%%
fr_class  two_stage_amp  has
    is_a
        voltage_amp;
    consists_of
        tr_1 ::= transistor;
        tr_2 ::= transistor;
    attributes
        vb_1,  ve_1,  vc_1,  vc_2,
        ib,  a,
        r1,  r2,  r3,  r4,  r5;
    constraints
        vb_1 * (r1 + r2) = vcc * r2,
        ib * (r1 + r2) = vcc,
        vb_1 = ve_1 + tr_1 :-> vbe,
        vc_1 = vcc - tr_2 :-> vbe,
        vc_2 = tr_1 :-> ic * r5 + tr_2 :-> ic * (r4 + r5),
        ve_1 = (tr_1 :-> ic + tr_2 :-> ic) * r5,
        a * r5 * (r3 + tr_2 :-> hie)= r3 * tr_2 :-> hfe * (r4 + r5),
        gain * (r4 + r5 + a * r5) = a * (r4 + r5),
        tr_1 :-> ic * r3 = tr_2 :-> vbe,
        icc = ib + tr_1 :-> ic + tr_2 :-> ic;
fr_end.
```



(a) クラス定義

```
?- :create(#two_stage_amp, A).                                  ・・・(1)
 A = $fr_object
?- :add_const_list(A, [ gain = 10, vcc = 20,
            tr_1 :-> hfe = 500, tr_2 :-> hfe = 500 ]).          ・・・(2)
?- :add_const_list(A, [ vc_2 * 2 = vcc,
                        r2 = 50, r3 = 5, r4 = 10]).             ・・・(3)
?- :get_all_attr(A, L).
 L = [ gain = 10, vcc = 20, icc = 1.0431, vb_1 = 1.7113,
       ve_1 = 1.1113, vc_1 = 19.4, vc_2 = 10,
       ib = 0.034225, a = 1283.9, r1 = 534.36,
       r2 = 50, r3 = 5, r4 = 10, r5 = 1.1015 ]                  ・・・(4)
```

(b) 設計手順

図8 例題

Fig.8 Example of electronic circuit design