(12)特許協力条約に基づいて公開された国際出願

(19) 世界知的所有権機関
国際事務局

(43) 国際公開日
2005 年8 月4 日 (04.08.2005)

PCT

(10) 国際公開番号
WO 2005/071609 A1

(51) 国際特許分類[7]: G06N 5/00

(21) 国際出願番号: PCT/JP2005/000752

(22) 国際出願日: 2005 年1 月21 日 (21.01.2005)

(25) 国際出願の言語: 日本語

(26) 国際公開の言語: 日本語

(30) 優先権データ:
特願2004-013629 2004 年1 月21 日 (21.01.2004) JP

(71) 出願人 (米国を除く全ての指定国について): 有限会社メタロジック (METALOGIC, INC.) [JP/JP]; 〒1760014 東京都練馬区豊玉南３丁目２６番１４号 Tokyo (JP).

(72) 発明者; および

(75) 発明者/出願人 (米国についてのみ): 福井 利夫 (FUKUI, Toshio) [JP/JP]; 〒1760014 東京都練馬区豊玉南３丁目２６番１４号 Tokyo (JP).

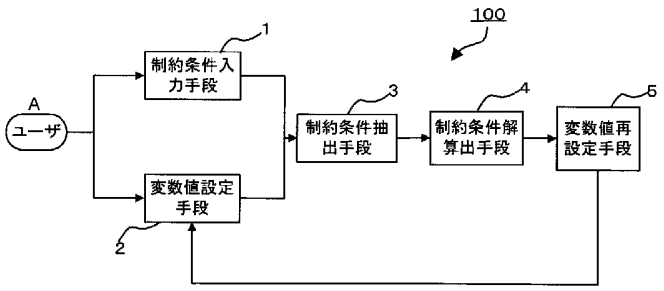
(74) 代理人: 熊倉 禎男, 外(KUMAKURA, Yoshio et al.); 〒1008355 東京都千代田区丸の内３丁目３番１号 新東京ビル 中村合同特許法律事務所 Tokyo (JP).

(81) 指定国 (表示のない限り、全ての種類の国内保護が可能): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE,

[続葉有]

(54) Title: CONSTRAINT CONDITION SOLVING METHOD, CONSTRAINT CONDITION SOLVING DEVICE, AND CONSTRAINT CONDITION SOLVING SYSTEM

(54) 発明の名称: 制約条件解決方法、制約条件解決装置、及び制約条件解決システム



A.. USER
1.. CONSTRAINT CONDITION INPUT MEANS
2.. VARIABLE VALUE SETTING MEANS
3..CONSTRAINT CONDITION EXTRACTION MEANS
4.. CONSTRAINT CONDITION SOLUTION CALCULATION MEANS
5.. VARIABLE VALUE RESETTING MEANS

(57) Abstract: It is possible to surely solve an object problem with a small calculation time without describing an algorithm as a data processing procedure on a computer. The solving means is as follows. When a constraint condition associated with the object problem and a initial value of a variable or a fluctuation value used for the constraint condition are set by variable value setting means (2), constraint condition extraction means (3) extracts a constraint condition associated with the variable; constraint condition solution calculation means (4) successively calculates a solution for the respective constraint conditions; and by using the calculated solution as a new variable, variable value resetting means (5) repeats calculation to obtain a solution of each constraint condition. Thus, the process for calculating the solution is clear and what is necessary is to solve each constraint condition significantly simplified. Accordingly, it is possible to surely reach the final solution and significantly increase the operation speed.

(57) 要約: 計算機上でのデータ処理手順であるアルゴリズムを記述することなく、対象問題を少ない計算時間で確実に解決することを目的とする。解決手段としては、対象問題に関する制約条件及び当該制約条件に用いられる変数の初期値または変動値が変数値設定手段２により設定されると、制約条件抽出手段３はその変数に関連した制約条件を抽出

[続葉有]

WO 2005/071609 A1

SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) 指定国 (表示のない限り、全ての種類の広域保護が可能): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), ユーラシア (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), ヨーロッパ (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

添付公開書類:
— 国際調査報告書

２文字コード及び他の略語については、定期発行される各*PCT*ガゼットの巻頭に掲載されている「コードと略語のガイダンスノート」を参照。

して、制約条件解算出手段４は各制約条件ごとの解を順次求め、変数値再設定手段５はこの求めた解を新たな変数値として各制約条件の解を求めることを繰り返すように構成したので、解を求める道筋が明確になるとともに、格段に単純化された各制約条件を解けばよいことから最終的な解まで確実に到達することが可能となり、これにより演算処理を格段に高速化させることができるようになる。

明 細 書

制約条件解決方法、制約条件解決装置、及び制約条件解決システム

技術分野

[0001] 本発明は、任意の制約条件から構成される問題に対する効率的な解決に関し、特に、制約伝播の処理手順が考慮された制約条件及び変数値の設定のみで実行可能な制約条件解決方法、制約条件解決装置、及び制約条件解決システムに関するものである。

背景技術

[0002] 様々なアプリケーションプログラムに対する現在の一般的なプログラミング手法は、例えば、C言語、JAVA等のようなプログラミング言語を使って、データ構造を手続き的に宣言した上でそのデータの処理手順(アルゴリズム)をユーザが逐次記述する手続き型プログラミング手法が主流である。コンパイラの最適化機能により最適化が図られること等を除けば、この手続き型プログラミング手法によるデータの処理手順は、計算機が実際に実行する処理手順と一致しているのが原則である。したがって、手続き型プログラミング手法は、計算機(CPU)の動きを詳細に記述することによって各種の問題に柔軟に対応できる解決方法である。

[0003] このようなCPUの動きを記述する手続き型プログラミング手法に対するものとして、宣言的プログラミング手法がある。これは、対象問題に対応した制約条件がユーザにより定義されるとその後の実際のデータ処理手順については、CPUにすべて任せてしまう手法であり、特定の問題領域には好都合な問題解決手法であるとして従来より用いられている(例えば、特許文献1または特許文献2)。

前記宣言的プログラミングにおいては、ユーザは、制約条件を記述するのみで解が出力されるのを待つだけで良い半面、CPUがデータ処理をどのような手順で行うかについては干渉できないという特徴を有している。現在の計算機環境では、前述したような大きく2つに分けられるプログラミング手法のうち、手続き型プログラミング手法が主に使用されている。

[0004] この手続き型プログラミング手法によってプログラム開発を行う場合、ユーザがデー

タ構造の定義を行った時点で、そのデータ構造によって記述すべきアルゴリズムも連動して事実上決定されてしまうのが殆どである。しかもそのアルゴリズムは類似した対象問題では同様な手順で表現され、パターン化することが可能である。

[0005] このため、各アプリケーションで同じようなプログラミング作業が重複して行われることが多く、プログラム開発の非効率化が従来より問題となっていた。この問題に対処するため、現在では汎用的に使用されるプログラミング部分のライブラリ化やオブジェクト指向に基づくソフトウェア開発によってソフトウェアの部品化を積極的に導入し、既存のプログラムにない部分に集中したプログラム作業の高効率化が推進されている。

[0006] 特許文献1:特開平5−3548号公報

特許文献2:特開平9−81387号公報

発明の開示

発明が解決しようとする課題

[0007] しかしながら、現実的にはライブラリ化されるソフトウェア部品は膨大であってしかも年々その数は増加している。各プログラマーが、この膨大な部品群から対象問題に最適なものを抽出する際には選択ミスが少なからずあり、再度選択し直すことで開発作業をやり直さなければならない危険性があることを考慮すると、ソフトウェア部品を利用せずに始めから各プログラマーがコーディング作業を行うケースも多々あるという問題があった。

また、オブジェクト指向によるソフトウェア開発は一定のスキルが要求されることから、どのプログラマーでもオブジェクト指向のプログラミング作業ができるとはかぎらず、全体的にみれば作業効率の向上がさほど上がらないという問題があった。

このように、プログラム開発作業には、個人技量に依存した閉鎖的な面や非効率的な面が依然として解消していないという問題があるのが現状である。

[0008] 一方、前述したように、宣言的プログラミング手法では、少なくとも制約条件の記述と、当該制約条件で用いるデータ値とが必要であって、実際の問題に適用した場合、その制約条件及びデータの数は非常に多くなる。ここで、実際の問題とは、例えば、

$f_k(X_1, X_2, \cdots, X_n) \leqq b_k$ (k＝1, 2, …, m)の連立式を満たし、且つ、評価関数$f_0(X_1, X_2, \cdots, X_n)$を最小値にするような組合せ最適化問題である。つまり、図17に示すように、m個の連立式及び評価関数f_0で表した制約条件を計算機側に与えると、計算機(図17の制約解消系)は全ての制約条件を同時に満たすデータを一括して探索することが行われる。

[0009] この組合せ最適化問題は、変数X及び連立式の数が増加するほど解空間における組合せの可能性が指数関数的に増加することから、計算機上での処理が複雑かつ膨大となる。しかも、特に関数f_kが非線形関数の場合は、分散した複数の解空間が存在することもあり、解が存在しない解空間をひたすら探索してしまって無限ループに陥ることのないよう効率的に解を求める工夫等が必要になる。なお、組合せ最適化問題の他にも、組合せ充足問題、整数計画法などについても同様の問題点がある。

[0010] さらに、制約条件の条件式または入力データの一部を変更する必要が生じたときは、先に計算した同様の制約条件問題をまた最初から計算し直さなければならず、リソースの無駄使いになるとともに非常に非効率的であるという問題があった。

[0011] そこで、本発明は前述した問題点に鑑み、計算機上でのデータ処理手順であるアルゴリズムを記述することなく、対象問題を少ない計算時間で確実に解決することを目的としている。

課題を解決するための手段

[0012] 本発明の制約条件解決方法は、ユーザにより対象問題に関する制約伝播の処理手順が考慮された制約条件を入力する制約条件入力処理と、前記制約条件入力処理により入力された制約条件に用いられる変数の初期値または変動値を設定する変数値設定処理と、前記制約条件入力処理により入力された制約条件の中から、前記変数値設定処理により設定された値に関係する制約条件を抽出する制約条件抽出処理と、前記制約条件抽出処理により抽出された制約条件全てに対して、前記ユーザにより考慮された制約伝播の処理手順に基づいて前記変数の初期値または変動値を代入して当該制約条件の解を算出する制約条件解算出処理と、前記制約条件解算出処理により算出された解を前記変数値設定処理での新たな変動値として設定する変数値再設定処理とを含み、前記変数値再設定処理は、再設定される新たな

変動値が存在する間は、前記制約条件抽出処理及び前記制約条件解算出処理を繰り返し行うことを特徴とする。

[0013] 本発明の制約条件解決方法は、制約条件を入力する制約条件入力処理と、前記制約条件入力処理により入力された制約条件に用いられる変数の値を設定する変数値設定処理と、前記制約条件入力処理により入力された制約条件の中から、前記変数値設定処理により設定された値に関係する制約条件を抽出する制約条件抽出処理と、前記制約条件抽出処理により抽出された制約条件の解を算出する制約条件解算出処理とを含む制約条件解決方法において、前記制約条件は、現在の配列要素数を含んでいること、前記変数に複数個の値を持たせること、構造体配列の要素値による検索を含んでいること、及びデフォルト規則による文字列数値変換の少なくとも何れかを特徴として構成される。

[0014] 本発明の制約条件解決装置は、ユーザにより対象問題に関する制約伝播の処理手順が考慮された制約条件を入力する制約条件入力手段と、前記制約条件入力手段により入力された制約条件に用いられる変数の初期値または変動値を設定する変数値設定手段と、前記制約条件入力手段により入力された制約条件の中から、前記変数値設定手段により設定された値に関係する制約条件を抽出する制約条件抽出手段と、前記制約条件抽出手段により抽出された制約条件全てに対して、前記ユーザにより考慮された制約伝播の処理手順に基づいて前記変数の初期値または変動値を代入して当該制約条件の解を算出する制約条件解算出手段と、前記制約条件解算出手段により算出された解を前記変数値設定手段での新たな変動値として設定する変数値再設定手段とを含み、前記変数値再設定手段は、再設定される新たな変動値が存在する間は、前記制約条件抽出手段及び前記制約条件解算出手段が繰り返し行うことを特徴とする。

[0015] 本発明の制約条件解決装置は、制約条件を入力する制約条件入力手段と、前記制約条件入力手段により入力された制約条件に用いられる変数の値を設定する変数値設定手段と、前記制約条件入力手段により入力された制約条件の中から、前記変数値設定手段により設定された値に関係する制約条件を抽出する制約条件抽出手段と、前記制約条件抽出手段により抽出された制約条件の解を算出する制約条

件解算出手段とを含む制約条件解決方法において、前記制約条件は、現在の配列要素数を含んでいること、前記変数に複数個の値を持たせること、構造体配列の要素値による検索を含んでいること、及びデフォルト規則による文字列数値変換の少なくとも何れかを特徴として構成される。

[0016] 本発明の制約条件解決システムは、複数の機器が互いに通信可能に接続されている制約条件解決システムであって、上記機器のうち少なくとも1つの機器は、前記何れかに記載の制約条件解決装置の機能を有することを特徴とする。

発明の効果

[0017] 本発明によれば、対象問題に対する制約伝播の処理手順が考慮された制約条件、及び制約条件に用いられる変数の値(初期値)を入力すると、当該変数の値に関連する制約条件を抽出した上でその制約条件のそれぞれを満たす解を導くとともに、導かれた解を変数の新たな値(変動値)として代入して前記各制約条件の解を求める処理を繰り返すように構成したので、他の制約条件とのAND関係をもつことなく各制約条件単独での解を求めることが可能となり、計算処理を高速に行いながら最終解を得ることができる。

発明を実施するための最良の形態

[0018] 以下、本発明の好適な実施の形態について図面を参照しながら詳細に説明する。

図18は、従来のプログラミング言語環境と、本発明の制約条件の解を求める制約解消系との位置付けを概念的に示した図である。図18に示すように、従来のプログラミング言語環境には、プログラム言語処理系と、プログラムコードと、データとがあって、プログラム言語処理系がプログラムコード及びデータを制御する形態になっている。本発明では従来のプログラミング言語環境におけるデータの他に、制約式(制約条件)及び制約解消系を備え、それぞれが相互に情報のやり取りを行うことができるように接続される構成である。

また、図19も従来のプログラミング言語環境であるが、前記従来の制約解消系をライブラリの形で構成したときの構成図である。

[0019] (第1の実施形態)

まず、本発明の第1の実施形態における制約条件解決装置の基本構成及びその

基本動作について説明する。

＜制約条件解決装置100の基本構成＞

図1は制約条件解決装置100の基本的な構成要素を示す全体構成図である。

図1に示すように制約条件解決装置100は、制約条件入力手段1と、変数値設定手段2と、制約条件抽出手段3と、制約条件解算出手段4と、変数値再設定手段5とを含んでいる。

[0020] 制約条件入力手段1は、ユーザが計算機（CPU）による制約伝播の処理手順を頭に思い浮かべて決定した制約条件を入力するためのものである。

変数値設定手段2は、制約条件入力手段1により入力された制約条件を構成する各変数にユーザが指定した具体的な初期値を設定するためのものである。また、詳細は後述するが、各変数に対して初期値以外の値（以下、変動値と称する）を設定することも行う。

また、制約条件入力手段1及び変数値設定手段2は、いわゆるデータ入力用のU／Iであり特に限定するものではなく、入力を容易に行える様々なツールを含むものであり、ユーザが入力した値を確認することができる表示（モニタ）画面を備えることができる。なお、制約条件解決装置100により計算された値をユーザに提示する出力表示手段（不図示）を含む構成にしてもよい。この出力表示手段も出力値を確認することができる表示（モニタ）画面を備えることができる。さらに、制約条件入力手段1（変数値設定手段2を含む）及び出力表示手段の一つにして、データを入出力が行える入出力手段であってもよい。

[0021] 制約条件抽出手段3は、入力された制約条件の中から、初期値または変動値が設定された各変数を構成要素として有する制約条件を抽出する。本実施形態では、制約条件抽出手段3は、該当する制約条件をまとめて抽出するように構成したが、後述する制約条件解算出手段4によってある1つの制約条件の解を求めた後、次の制約条件を抽出することを順次繰り返すように、一つずつ抽出するようにしても構わない。

[0022] 制約条件解算出手段4は、初期値または変動値を代入した各制約条件を解くが、本発明の特徴として、制約条件すべてを満足するようないわゆる連立解として一括して解くのではなく、各制約条件の一つずつを対象にして解くようにしている。このため

、ある変数の値が設定または変更されると、制約条件の1回目の解がシーケンス的に所定の順序に従って順次求まることになる。ここで、前記所定の順序とは、ユーザ(プログラマー)が制約伝播の処理手順を考慮して入力した制約条件で示す順序である。

[0023] ここで、制約条件の具体例を以下に示す。

(A)制約条件の例1

宣言的プログラミング手法を容易に理解するために説明として使用される典型的な例題としては、以下の会計計算問題が挙げられる。

いま、4つの物品の値段と個数をそれぞれ、(1)100円，2個、(2)200円，3個、(3)300円，2個、(4)50円，3個としたとき、値段、個数、及び小計の配列変数を用いて、本発明のいう制約条件をあらわせば、

値段[1]＝100、値段[2]＝200、値段[3]＝300、値段[4]＝50 …(式1)

個数[1]＝2、個数[2]＝3、個数[3]＝2、個数[4]＝3 …(式2)

小計[i]＝値段[i]×個数[i](i＝1ー4) …(式3)

合計＝Σ小計[i](i＝1ー4) …(式4)

となる。(式1)及び(式2)は、制約条件のうちの特に変数データの設定である。

[0024] 前記(式1)ー(式4)の記述から明らかなように、手続き型プログラミングの場合に必要な0→合計(→は代入を表す、以下同じ)、合計＋小計[1]→合計、合計＋小計[2]→合計、合計＋小計[3]→合計、合計＋小計[4]→合計という繰り返しループの記述が、宣言的プログラミングの場合には不要である。

[0025] (B)制約条件の例2

string name, message, outputs, output; …(式5)

name＝”特許太郎” …(式6)

message＝”こんにちは。この文章はHTML文書の本です。” …(式7)

outputs＝”名前”＆ name ＆”メッセージ”＆ message; …(式8)

output＝”＜HTML＞＜BODY＞” ＆ outputs ＆ ＜／BODY＞＜／HTML＞ …(式9)

前記(式5)が変数データの定義、(式6)ー(式7)が変数への値代入、及び前記(式

8)ー(式9)が具体的な実行式である。また、&演算子は文字列連結演算子である。前記(式6)ー(式7)のように、文字列name 、messageを任意に書き換えてoutputで出力されるHTML用テキストが生成される。

[0026] (C)制約条件の例3

文字列strに別の文字列addを次々連結していく動作を記述した制約条件を次に示す。

string str; …(式10)

string add; …(式11)

str = str & add …(式12)

前記(式10)及び(式11)は変数データの定義であり、前記(式12)が具体的な実行式である。本例での文字列strの値変更による制約伝播は、文字列addの値が変更されるたびに、文字列strへ文字列addの値が追加されることが繰り返される。

[0027] ここで、制約条件は、前記(A)ー(C)で示すような等号または不等号で表される式である必要はないことに留意されたい。そして、制約式の評価は最終的に真か偽を答えとし、C言語等と同様に例えばf(x)だけが制約条件として記述されていた場合にはこのf(x)を実行し、結果が真または偽かによって判断され処理される。

[0028] 変数値再設定手段5は、制約条件抽出手段3及び制約条件解算出手段4により2回目以降の解を求めるために、制約条件解算出手段4で求めた1回目の解を各変数の新たな値、すなわち変動値として設定する。そして、一連の制約条件の解を求める処理は、制約条件解算出手段4で変更される解が存在しなくなって最終解に到達した後、制約条件解決装置100が別途終了指令を受信したときに終了する。

[0029] 前述したように、本発明の特徴の1つは、抽出した各制約条件の変数に値を代入して制約条件1つずつを順次解いていくことを繰り返すことである。このためには制約伝播の処理手順を考慮して制約条件にあらかじめユーザが設定しておくことが前提となる。

[0030] なお、制約条件解算出手段4による処理中に、解を求めることができない解不能、または解を特定することができない解不定の状態に陥った場合は、計算機に予め定義した適切な例外処理を実行するようにしておくことで、プログラムエラーとして出力

されるようにするが、これはユーザが解けない制約条件を定義したことを意味する。本実施形態の制約条件解決装置100において、ユーザは、本来解ける筈の処理手順になるように、計算機の処理手順を考慮した制約条件を設定しなければならない負担を負うが、これは従来の手続き型プログラミング作業に較べはるかに楽な作業である。

[0031] ＜制約条件解決装置100の全体動作＞

次に、前述した構成の制約条件解決装置100の全体動作について説明する。

図2は、制約条件解決装置100の動作手順を示すフローチャートである。

まず、ステップS20で、制約条件に用いられる変数に初期値が設定されているか否かを判断する。前記判断の結果、制約条件入力手段1により入力された制約条件を構成する各変数に対してまだ初期値が設定されていない場合、ステップS21で、変数値設定手段2は、ユーザが指定した初期値を設定する。

[0032] 次に、ステップS22で、制約条件抽出手段3は、各変数の値が変更されていないかを判断する。制約条件として変数に初期値を設定したときは、最初の変更であるとして次のステップS22へ進む。一度、初期値を設定後、後述するステップの処理が終了すると、制約条件解決装置100は、このステップS22で新たな変数の値変更があるまで待ちの状態が維持されている。なお、本フローチャートには「終了」を示していないが、動作終了指示命令が外部から入力されたとき、制約条件解決装置100は、所定の動作処理（不図示）を実行して一連の動作を終了するように制御されている。

[0033] ステップS23では、制約条件抽出手段3は、ステップS22で変更された変数を要素とする集合Sを生成する。ここで集合Sを生成するとは、例えば、計算機上のある記憶領域に、前記変更された変数の名前や、または変数の配列要素を識別できるサフィックスを書き込むこと等を意味する。

さらに、制約条件抽出手段3は、前記集合Sに要素が存在するか否か、すなわち、変数値設定手段2により値を変更した変数があるか否かを判断する（ステップS24）。前記判断の結果、集合Sに要素があるとき（集合Sが空集合でないとき）は、制約条件抽出手段3は、集合Sから要素Smを1つずつ取り出す（ステップS25）。

[0034] 次に、ステップS26で、前記取り出した要素Smに関連した制約条件の集合Qを生

成する。つまり、要素Smを含んで構成される制約条件があれば、制約条件抽出手段3はこれを集合Qに出力する。次に、ステップS27で前記集合Qに要素がある間は、以下のステップS28ーステップS30を繰り返して行う。

[0035] ステップS28で、制約条件解算出手段4は、集合Qから1つの制約条件Qmを取り出す。そして、制約条件解算出手段4は、ステップS28で取り出した制約条件Qmを解いて、制約条件Qmを満たすために変更するべき変数及びその値を求める(ステップS29)。

ここで、ステップS29における処理の詳細な内容は以下に示す通りである。

(1)制約条件Qmが満たされているかどうかをチェックして、満たされているときは本ステップS29を終了する。

(2)制約条件Qmを満たすために変更するべき変数とその値を求める。

(3)前記(2)で求めた変数の値が正しいかどうかを検証する。

前記(2)はステップS29の処理として必須であるが、その他の(1)及び(3)については必ずしもそうとは限らない。対象問題に応じてユーザが必要か否かを判断し、制約条件として明示的または黙示的に指示するように構成してよい。

[0036] さらに、ステップS30で、変数値再設定手段5は、ステップS29で求めた変数に求めた値を再設定して、その後の処理ではその再設定した値を基に制約条件が解かれる。

したがって、ある変数が変更される場合、その変数に関連する複数個の制約条件があれば、1つずつ取り出された制約条件ごとの解が順次求まり、同時に、得られた解によって集合Sの最後部にその変数が追加されるようになる。

[0037] 一方、ステップS28によって制約条件Qmが取り出された結果、集合Qに要素がないとステップS27で判断された場合、一連の処理はステップ24へ戻り、集合Sから次の変数Smが取り出されて同様の処理が繰り返される。

この結果、集合Sに要素がなくなったとき、制約条件解決装置100の処理としては安定状態に入った状況に相当することになり、前述したように、制約条件に用いられる任意の変数に新たな値が設定されるまでステップS22で待ちの状態になる。

[0038] 本実施形態では、ステップS25において集合Sに複数の変数Smを含んでいたとき

の変数Smを取り出す順序や、ステップS28において制約条件Qmを取り出す順序は、制約伝播の処理手順を考慮して、ユーザがその順序(優先順位)を明示的に指定して計算機に行わせる制約条件の設定方法にしている。具体的には例えば、その優先順位は制約条件の記載した順序であるというようにあらかじめ決めておくことで達成される。この具体例を以下に示す。

[0039] 前述した会計計算問題を再び用いる。

制約条件は、

値段[1]＝100、値段[2]＝200、値段[3]＝300、値段[4]＝50…(式1)

個数[1]＝2、個数[2]＝3、個数[3]＝2、個数[4]＝3 …(式2)

小計[i]＝値段[i]×個数[i](i＝1ー4) …(式3)

合計＝Σ小計[i](i＝1ー4) …(式4)

である。

前記(式1)ー(式4)の記載順序には意味がある。例えば、1番目に記載した制約条件(制約式)が最も優先順位が高く、以下順に低くなるものとする。この場合、(式1)及び(式2)により各変数に値を設定することが優先的に行われ、(式3)、(式4)はその後に制約式を計算機内部へ登録することを意味している。

制約式の実行順序は実行時に決まるが、集合S、集合Qから要素を取り出す順番が不確定である。これを決定するためには明示的な方法として、記載順序で判断したり制約式に優先順位を設定する方法がある。

[0040] その他、前述した優先順位を計算機(CPU)が自動的に行うように構成してもよい。この場合、ユーザは、計算機がどのような順序で行っても計算過程で矛盾が生じないような内容及び記載順序からなる制約条件を設定する必要があるが、これはユーザがプログラマーとして通常の知識や経験を備えていれば容易に成し得る作業である。

[0041] 次に、図2のフローチャートに示す動作手順を、前述した会計計算の例を用いて具体的な制約条件の変数値の変化を追いながら説明する。

例えば、値段[2]を、200→250に変更したときを想定する。初期値の設定ではなくデータの変更なので、ステップS20からステップS22へ進む。ステップS23によって、

集合S＝{値段[2]}となる。ステップS24で集合Sには要素があると判断されるので、ステップS25で値段[2]が取り出される。

[0042] 次に、ステップS26で、値段[2]だけに関連する制約条件である、小計[2]＝値段[2]×個数[2]が集合Qの要素となる。つまり、集合Q＝{小計[2]＝値段[2]×個数[2]}が生成される。

次に、ステップS27で集合Qには要素があると判断されるので、ステップS28で、小計[2]＝値段[2]×個数[2]という制約条件が取り出される。ステップS29で、制約条件の解、つまり、小計[2]＝250×3＝750が計算されて、ステップ30では、小計[2]にこの変更値750を設定し、小計[2]を集合Sに追加して、集合S＝{小計[2]}を生成する。

[0043] そしてステップS27に戻るが、先の集合Qから1つしかなかった要素を取り出してしまい集合Qは空集合であるので、この後ステップS24へ進む。

ステップS24では、先ほど追加した集合Sの小計[2]を判断して、ステップS25でこの小計[2]を取り出す。

[0044] 次に、ステップS26で、小計[2]だけに関連する制約条件である、合計＝Σ小計[i](i＝1ー4)を要素とする集合Qを生成する。つまり、

集合Q＝{合計＝Σ小計[i](i＝1ー4)}が生成される。ステップS28で当該制約条件を取り出し、ステップS29で、制約条件の解、つまり、合計＝200＋750＋600＋150＝1700を計算して、ステップS30で、変数の(合計)にこの変更値1700を設定する。同時に変数の(合計)は集合Sへ格納される。続いて、ステップS27、S24、S25、S26が実行され、関連のある制約式が存在しないので、集合Qには追加される要素がない。

この時、計算時間を短縮するために以下のように単純化することが可能である。すなわち、ステップS29で、制約条件の解を求める時に加算をもう一度行うのではなく、変化した値のみを利用して差分解消する。そして、ステップS30では、合計に関連する制約条件は存在しないことがわかっているので、計算時間短縮のために集合Sには要素を追加しないようにする。

[0045] 再びステップS27に戻るが、先の集合Qから1つしかなかった要素を取り出してしま

い集合Qは空集合であるのでこの後ステップS22へ進み、制約条件解決装置100は次に外部からのデータ変更があるまで待ち状態になる。

なお、前述した制約条件解決装置100の動作におけるステップS28の処理では、集合Qより制約条件Qmを1つずつ取り出してその解を求めていく処理を説明したが、必ずしも1つずつ取り出さなければならないことはない。制約式の解を同時に求めることが可能な場合、集合Qから2以上の制約条件を取り出すこととする。これにより、解決可能な対象問題をより広げることができるようになる。

さらに、制約条件Qmの実際の解法は、会計問題のような単純な式変形で解くやり方に限っていない。既存の式に関する解法はすべて使用可能であるものとしており、これによりn次方程式の解法を導入することも認めている。

[0046] また、別の具体的例として、制約条件に用いるデータの件数そのものを追加・削除するときの処理について説明する。

前述した会計計算の例では、値段及び個数の変数をそれぞれ配列で表現してデータ件数を各4個に固定して制約条件を設定したが、データ件数そのものを変数にしたときの制約条件を考える。現在の制約条件に値段[5]、個数[5]を加えて以下のように変更する。

値段[1]＝100、値段[2]＝250、値段[3]＝300、値段[4]＝50、値段[5]＝310…(式13)

個数[1]＝2、個数[2]＝3、個数[3]＝2、個数[4]＝3、個数[5]＝3 …(式14)

項目総数＝5 …(式15)

小計[i]＝値段[i]×個数[i](iは任意の整数) …(式16)

合計＝Σ小計[i](i＝1ー項目総数) …(式17)

[0047] 前述した処理を全く同様に行えば、

小計[5]＝値段[5]×個数[5]＝310×3＝930、

合計＝Σ小計[i]＝200＋750＋600＋150＋930＝2630となり、未解決のままの制約条件を残すことなく解を導くことが可能である。このような変更は変数または制約条件の削除する場合でも同様であり、可変長配列などで表された変数の個数を変更することも可能である。

[0048] これを実際の入出力手段(例えば、制約条件入力手段1)を用いてユーザが制約条件及び値を入力する画面例を図24に示す。なお、図24の画面例は出力機能を有する制約条件入力手段1及び変数値設定手段2の画面例に相当し、UNIX上へ一般的なプログラミング言語と同等の入出力インタフェースを持つ構造として実装した実行例である。図24(a)ー(h)の入出力は一連のものであるが、便宜上、分けて表示している。

図24(a)のように、まず「$」で示されたUNIXプロンプトの後に、ユーザは「ripple」を入力する(図24(b))。これにより本発明の処理が実装されたプログラムがロードされて、画面上に「%」を示してユーザにデータの入力を促す(%以降がユーザの入力データであり、改行されたその後すぐの行に、受信したデータ値や計算出力値を表示している)。

[0049] そこでユーザは、初期値制約式入力する(図24(c)の241)。242の制約式はここでは合計計算を関数 配列全合計()として抽象化しているが、図21で示したような制約式で表現することもできる。また、手続的なループ構造を直接記述することもできる(関数型言語との融合を参照)。そしてステップS29の処理で合計計算の差分解消を実現する時には、制約条件解決装置100側が判断して自動処理され、ユーザは意識していない。

[0050] 次に、ユーザは初期値を設定する(図24(d)の243,244)。ステップS22ーステップS30に示した処理が実行されるが、実行の効率を図るために明らかに必要ない時には制約条件解決装置100側の判断で実行させないことができる。図24(e)の245では、「load」によって残りの入力値をファイルから読み込む。この時点で小計、合計は計算されて設定済となり、246で「合計」を入力することによって、現在の合計値が表示されて実行結果を取得できる。図24(f)の247を入力すると、図2のステップS26が実行される。そして、計算途中の画面出力はないが、ステップS22ーステップS30に示した処理が実行され、248の入力によって実行結果の合計値が表示される。249,250の入力は、制約条件に用いるデータの件数を追加設定している。そして、図24(g)の251の入力よって実行結果に合計値が表示され、図24(h)の252の入力よって、終了指令の発行を行っている。

[0051] また、いくつかのデータ定義と制約式をまとめたものをクラス化概念における1クラスとして定義し、パラメータを与えて実例、すなわちインスタンス化させ、制約式と値をセットにするよう構成してもよい。これにより、設定される制約条件を複数に分割させることができるので、必要最小限の処理のみを実行させる場合等に好都合となる。このクラス化に関する概念を図20に示す。このクラスは継承させることが可能であり、例えば、図20に示す変数1、変数2、及び制約式1を有するクラスAと、変数3を有するクラスBとがあるとき、このクラスA及びクラスBを継承し、さらに制約式2を継承した図20のクラスを生成することができる。

[0052] なお、本実施形態の制約条件解決装置100の構成及び動作から分かるように、本発明では、例えば、すべての制約条件が制約違反をしないという評価式(評価条件)を設定していない。従来の宣言的プログラミングでは、制約条件の解を求める際に当該評価式に基づいて制約条件間で矛盾しないかを判断するように処理することが多い。本発明では、制約条件の設定時に、矛盾が生じないようにユーザが設定することを前提としているためである。したがって、制約条件解決装置100は、最後に必ず解を求めることが可能であり、解が求まらないときはユーザによる制約条件の設定誤りである。

実行時に制約違反の存在に関する評価式は必要ないが、補助機能として用意することによってデバッグ時等の問題発生時の対処やミスの発見に役立つことは言うまでもない。

[0053] 本実施形態の制約条件解決装置100によれば、対象問題に関する制約条件及び当該制約条件に用いられる変数の初期値または変動値が設定されると、その変数に関連した制約条件を抽出して各制約条件ごとの解を順次求めて、この求めた解を新たな変数値として各制約条件の解を求めることを繰り返すように構成したので、任意のデータ構造と、そのデータ構造によるデータ間を連結させる制約条件とによって対象問題を解決することができるようになる。

つまり、設定される制約条件は、従来の手続き型プログラミングのような対象問題を解決するためのアルゴリズムを記述することなくこのアルゴリズムに相当する指令をデータ構造に含ませた上で制約伝播の処理手順に沿って記載し、すべての制約条件

が成立する最適解を求めずに制約条件1つずつの解を求めるように構成したので、解を求める道筋が明確になるとともに、格段に単純化された各制約条件(例えば、1元の方程式)を解けばよいことから最終的な解まで確実に到達することが可能で演算処理を格段に高速化させることができるようになる。

[0054] また、問題の解決方法を制約条件において直接的に記述できるため、ユーザが従来のようなプログラミング言語の特殊な専門知識を必ずしも有している必要はなく、対象問題を容易に解くことができる。

[0055] (第2の実施形態)

前述した第1の実施形態では、制約条件解決装置100において、制約条件入力手段1と、変数値設定手段2と、制約条件抽出手段3と、制約条件解算出手段4と、変数値再設定手段5とを含むように構成したが、本実施形態の制約条件解決装置200では、第1の実施形態の制約条件解決装置100における各構成手段1ー5に加えて、制約条件格納手段6及び変数値格納手段7をさらに備えているのが特徴である。

なお、本実施の形態の制約条件解決装置200の構成と、第1の実施の形態における制約条件解決装置100とで共通する構成部分については同一の符号を付して、詳細な説明を省略する。

[0056] 本実施形態の制約条件解決装置200の全体構成を図3に示す。

制約条件格納手段6は、制約条件入力手段1により入力された制約条件を所定の記憶領域に格納する。また、変数値格納手段7は、変数値設定手段2により設定された変数値(初期値、変動値)を格納する。

また、本実施形態の制約条件抽出手段8は、第1の実施形態における制約条件抽出手段3の機能に加えて、新たに設定された制約条件または変動値を、前記記憶領域に格納された制約条件または値と比較して、両者が一致しない場合のみ制約条件の抽出を行う機能を有している。なお、制約条件抽出手段8は前記比較を常に行う必要はなく、制約条件格納手段6及び変数値格納手段7から制約条件及び変数値を抽出するようにしてもよいことは言うまでもない。

[0057] 図4(a)は、制約条件格納手段6が計算機の記憶領域(例えば、メモリ、磁気ディスクなど)へ制約条件を格納する形態例を示す図である。つまり、変数Xを3つの制約

式(制約式1、制約式2、制約式3)で参照する制約条件があるとし、この変数Xに値246が設定されたときの記憶領域の表し方を示している。前述した会計計算問題の制約条件をこの形式によって表したときの一例を図6及び図5に示す。

[0058] 例えば、図4(b)に示すように、数式(x=y+z)は分割して格納される。ただし、このような表し方に限定されるものではない。制約条件の格納形態として、値と制約式とでメモリ領域を必ずしも分割する必要はなく、例えば制約式を構造として分割せずに文字列などで持ってもよい。

また、図5に示すようなデータ構造に限らず、例えば、以下に示すような記憶領域でも構わない。

・変数の値は種類分けされ、ある値はメモリ上のみに記録されるが、他の値はファイルやデータベースへ記録されることで、記録領域上の値を永久的にしてもよく、または時刻システム等において特別の意味を持たせるようにしてもよい。

・変数名をポインタで示す必要はなく、直接持ってもよい。

・変数の値を必ずしも直接的に保持することはない。

・プログラムのコンパイル時等では変数を識別する必要はないので、変数名は必須ではない。

・制約条件に用いる制約式はリストでなくてもよい。

・制約式リストを持たず、図7に示すように、変数名から制約式を検索する構成であってもよい。ただし、図7に示すようなメモリとポインタ構造に限定するものではない。

その他、上記の変形による無数に存在するデータ型はすべて含むものとする。

[0059] 本実施形態の制約条件解決装置200によれば、対象問題に関する制約条件及び当該制約条件に用いられる変数の初期値または変動値を所定の記録領域に一時的または永久的に保持するように構成したので、本実施形態の制約条件抽出手段8は、制約条件または変数値に変更が生じた場合のみ、変更された変数Smに関連する制約条件Qmを解いてQmを満たすために変更すべき変数及びその値を求める処理を行うことが可能となる。

これにより、制約条件に変更があったとしても、複数の制約条件すべてについて最初から計算し直さなくても済む場合が生じ、計算時間の効率を図ることができるように

なる。

[0060] (第3の実施形態)

前述した第1の実施形態の制約条件解決装置100では、制約条件入力手段1と、変数値設定手段2と、制約条件抽出手段3と、制約条件解算出手段4と、変数値再設定手段5とを含むように構成し、前述した第2の実施形態の制約条件解決装置200では、さらに、制約条件格納手段6及び変数値格納手段7を備えている構成にしたが、本実施形態の制約条件解決装置300では、図8に示すように、関数定義保持手段9及び関数定義実行手段10を備えているのが特徴である。

第1の実施形態の制約条件解決装置100と、第2の実施形態の制約条件解決装置200のどちらであっても、関数定義保持手段9及び関数定義実行手段10をさらに備える構成にすることができるが、説明を簡略化させるために制約条件解決装置100に付加した構成で説明することとする。

なお、本実施の形態の制約条件解決装置300の構成と、第1の実施の形態における制約条件解決装置100とで共通する構成部分については同一の符号を付して、詳細な説明を省略する。

[0061] プログラミング手法からみた本実施形態の制約条件解決装置300の構成が有する役割は、第1の実施形態が、問題解決を完全な宣言的プログラミング手法で行っていたのに対し、第3の実施形態は、従来の手続き型プログラミング手法を融合している点にある。これは、宣言的プログラミングで解決することが困難な一部の問題については、手続き型プログラミング手法を利用するようにして、どんな対象問題であっても柔軟な問題解決を果たすためである。

[0062] 本実施形態の制約条件解決装置300の全体構成を図8に示す。

関数定義保持手段9は、制約条件入力手段1及び変数値設定手段2によりユーザが従来の手続き型プログラミングで記述する手続き、及び変数値を保持する。関数定義保持手段9は、制約条件抽出手段3と相互にデータの入出力を行うことが可能であるように接続されている。

なお、本実施形態の制約条件入力手段1のような、必ずしも制約条件以外に手続き型の記述を許可する構成にする必要はない。例えば、手続き型の記述専用の入力

手段を別途備えるような構成にしてもよい。変数値設定手段2についても同様で、手続き型で記述された手続きの変数専用の変数値設定手段を備えるようにしてもよい。

[0063] 関数定義実行手段10は、関数定義保持手段9で保持された手続きを計算機に実際に実行させるものである。また、関数定義実行手段10は、制約条件解算出手段4と相互にデータの入出力を行うことが可能であるように接続されている。

[0064] 本実施形態の制約条件解決装置300の処理を具体例で説明する。いま、制約条件として記述された制約式の中に、y＝sin(x)、またはy＝逆関数のない関数(x)等が含まれている場合を想定する。

この場合、図2に示すステップS29において、変数xの値を求めるためにx＝$\sin^{-1}$(y)という式変形をして処理しようとしたところ、逆関数$\sin^{-1}$の値領域の関係でx値を求めることが不可能なときがある。また、y＝逆関数のない関数f(x)では、y値からx値を求めることができない。このようなときに、従来の手続き型プログラミングによるユーザ関数を自由に取り入れて対処するようにする。

[0065] 例えば、制約式を解くにあたり、解決方法を複数の中から選択可能に記述することを許可するようにしておく。制約条件をa＝f(b)とすれば、変数bの値を求めるために、{a＝f(b), b＝f_rev(a)}を用意し、どちらかが適用できれば制約式を解くことができるようにしておく。

また、y＝f(l, m)という制約式の変数l, mの値が決まっていない場合、或いはy＞xのような不等式の場合も、当該制約式を単純に解くことができない。不等式y＞xの場合で、xの値がnとなったとき、yはnより大きいという情報のみが得られるだけで、変数yの値は最終的な解が未確定ともいえる。このようなケースでは制約条件の設定エラーとして処理してしまう他に、y＞nという情報をyに関係する制約式にそのまま代入させて処理を続行させるようにしてもよい。1つの方程式だけに注目すると解けないz＝x＋yについても同様で、複数の式と複数の値との間で連立方程式を扱うように許可することで解決することもできる。

[0066] これにより、他の制約式の条件との関連で変数yの値を限定できたり、より絞り込むことが可能になったり、或いはy＞nがそのまま解として成立することもあるためである。このように、変数値として確定した値でなく不等式などをそのまま扱い、別の制約式

で解決可能なときに可能な限り解決するようにする。つまり、このときの制約条件抽出手段3は、図9に示されるような関係にある解決方法を実行する。図9における制約式のx値が変更されると、xへ代入する形式でないすべての式(解決方法1ー解決方法3)を実行していくというように、制約式の解過程に柔軟性をもたせ、手続き型プログラミングを用いている従来のプログラムと同様、またはそれ以上の解決能力を持たせることが可能となる。

[0067] また、逆関数等の問題で制約式として表すことが困難な場合にかぎらず、制約式として通常に表すことができる式についても図9に示すような解決方法の形式に変形して保持するように構成すれば、処理実行時の速度を向上させることも期待できる。

[0068] このように、本実施形態の制約条件解決装置300によれば、関数定義保持手段9及び関数定義実行手段10を含むように構成したので、制約条件と従来のプログラミングで定義していた関数等とが融合し、対象問題の性質または特長に応じて制約条件としての解決方法、及び従来のプログラミング手法を用いた解決方法を、ユーザ側が自由に切替えながら設定することが可能となる。これにより、第1の実施形態及び第2の実施形態の制約条件解決装置100, 200では解決できない逆演算処理などを包含する制約条件の記述が格段に容易になる。

[0069] (第4の実施形態)

前述した第1の実施形態ー第3の実施形態の制約条件解決装置100, 200, 300の各構成手段に加え、本実施形態の制約条件解決装置400は、データベースやシステムリソース等からの情報を適宜利用して変数値を設定できることを特徴としている。

図10に本実施形態の制約条件解決装置400の全体構成図を示す。

本実施形態の説明を簡略化させるために、第2の実施形態の制約条件解決装置200に付加した構成の制約条件解決装置400により説明することとする。他の実施形態の制約条件解決装置において、データベースやシステムリソース等からの情報を利用できることは言うまでもない。

なお、本実施の形態の制約条件解決装置400の構成と、第2の実施の形態における制約条件解決装置200とで共通する構成部分については同一の符号を付して、

詳細な説明を省略する。

[0070] 図10に示すように、本実施形態の制約条件解決装置400の構成によれば、制約条件内の変数の一部または全部が、前記データベース11やシステムリソース12内に記録されている値や、時刻等の別の事象と連携されるシステム内の変数と連結することが可能となり、本制約条件解決装置400外の変数と制約条件の変数との整合を図った変数の管理を容易に行うことができるようになる。

[0071] 次に、制約条件の内容としてさらに複雑なデータ構造の変数を扱うことが可能な例をいくつか示す。以下の具体例は、前記第1の実施形態ー第4の実施形態における制約条件解決装置100, 200, 300, 400すべてにおいて実行可能である。

[0072] (変数に複数の値を定義した制約条件)

通常、変数へ値を代入する制約条件は、例えば、x＝10のように記述するが、変数Xに複数個の値を代入することを定義する。具体的には例えば、

x＝{10、11、12};

という記述を認める。この場合、変数同士の加減乗除の演算は、加算を例にすればx＋yは、各変数x, yのそれぞれの要素について＋演算を行うものとする。

[0073] (演算子が定義された配列構造の変数による制約条件)

いま、1次元配列vecを考える。この1次元配列vecを、vec＝{50, 200, 300}とし、i番目(0≦i≦文字列長−1)の要素に対するアクセスをvec[i]で表すこととする。1次元配列vecの各要素は、vec[0]＝50、vec[1]＝200、vec[2]＝300である。

ここで、1次元配列vec1, vec2に対する演算子を次のように定義する。

vec1＋vec2 …各々の1次元配列の要素を全て加算した配列、

vec1−vec2 …各々の1次元配列の要素を全て減算した配列、

vec1＊vec2 …各々の1次元配列の要素を全て乗算した配列、

vec1／vec2 …各々の1次元配列の要素を全て除算した配列、

vec1＝vec2 …各々の1次元配列の要素が全て同一である配列の比較、もしくは代入、

vec1＆vec2 …各々の1次元配列の要素を全て連結した配列。

1次元配列に対する演算子をこのように定義することにより、前記第1の実施形態ー

第4の実施形態における制約式と同様に扱うことが可能になる。

配列構造の制約条件にアクセスするには、限定するものではないが、例えば図21に示すようなオペレータを用意すればよい。このオペレータには任意の関数を使用することが可能となっている。

[0074] また、前記1次元配列vecの要素に対して柔軟性のあるアクセスを実現するために、vec[i]において範囲指定が可能なようにする。具体的には、vec[n. . m, l. . k, i. . j]を、次元配列のnーm番目、lーk番目、及びiーj番目の要素を取り出して並べた配列を表しているとすると、

vec1[0. . 3]＝vec2[0. . 1]&vec2[4. . 5]＋vec3[0. . 3]というような演算が可能となり、1次元配列を制約式(制約条件)として扱うことができる。

[0075] 例えば、前記1次元配列(ここでは、strとする)の各配列要素に文字を格納すればstrは文字列となる。文字列str＝”Hello　World　！”に対して、各str[i]に任意の文字を代入することや、文字列1＝”Hello　”、文字列2＝”World　！”とすれば、文字列str＝文字列1＆文字列2で表される制約式を扱うことが可能となる。さらに、文字列のみの正規表現によるパターンマッチングや文字列置換を可能にするなどによって、制約式をさらに柔軟に記述することが可能となる。

[0076] これを応用して、例えば、変数値の変更に従って出力が変化するWebによるHTML(hypertext markup language)出力を以下のように記述することができる。

日付け＝”2003/01/01”;

名前＝”特許　太郎”;

内容＝”長文文章”;

HTMLデータ＝”＜HTML＞＜BODY＞”＆日付[0. . 3]＆”年”＆日付[5. . 6]＆”月”＆日付[8. . 9]＆”日”＆”名前:”＆名前＆”内容:”＆内容＆”＜／BODY＞＜／HTML＞”;

[0077] (デフォルト変換規則を備えた制約条件)

任意のデータ変換についての変換規則を用意し、特にデフォルトで指定可能とする。

特に数値から文字列へ変換する場合、参照毎に変換関数を毎回用いる方法がある

が、データ変換規則のデフォルトを用意することにより、効率的な記述が可能となる。

一例として、次のような規則を用意する。

・変換規則は必ず右詰めとする

・Sは符号を表し数が負の時のみ-を、それ以外の時は空白とする

・Xnはn桁以下の数を表す

・Ynはn桁固定の数を表す

・Znはn桁固定のゼロサプレスを表す

この時、例えば変数定義時に、

float(″SX9.Y3″) val1;

とすれば、符号付、仮数部9桁以下、少数点、少数部3桁固定で右詰めされた全14文字の文字列としてval1を参照できる。

また、unsigned int(″Z8″) val2;

とすれば、val2は符号無、ゼロサプレス付8桁の全8文字の文字列としてval2を参照できる。

もちろん、値参照時に直接val2(″Z10″)等として記述することでデフォルト規則だけではなく適用時に別の規則を適用できることとする。

このようなデフォルト変換規則を備えた制約条件による変換規則を定義することで、事務処理アプリケーション等における制約式を簡単に記述することができるようになる。

他のデータ構造にも同様に応用できたり、変換規則のフォーマットとして以上のものは1例であり多種のものが存在することは言うまでもない。

[0078] (優先順位付き制約条件)

制約条件に重みを付加する場合の記述例を以下に示す。

{条件式}[:優先順位];

or

{条件式}[:優先順位]→{条件式が不成立時の処理};

ここで、複数の制約式条件が合致した場合、優先順位の値に従って実行すべき制約式が選択されるようにする。また、同じ優先順位の値が複数ある場合はすべて実行

されるようにする。このような優先順位付き制約式を定義することで、優先順位の最も小さいまたは大きい制約式のみを実行させたり、或いは制約式の実行順序を指定させたりする場合に有効になる。

[0079] (構造体による制約条件)

構造体(抽象的なデータ構造)は、複数のデータ型をまとめて扱うことが可能である。前述したWeb出力のデータをこの構造体で表現しようとすると、

```
struct 構造体名{
        date 日付け;
        name 名前;
        string 内容;
}掲示板データ[];
```

のように表すことができる。実際のWebシステムでは、前記構造体のデータはデータベースやファイル上に存在することになり、前記構造体の場合、配列番号5に関する"内容"に対するアクセスは、例えば、"掲示板データ[5].内容"により可能となる。

なお、本構造体のデータのアクセスにおいて、メンバ(日付け、名前、内容、及びHTMLデータのこと)に対する文字列マッチングの演算子などを定義することによって、検索の処理を制約条件として記述することができるようになる。

[0080] (組み込み関数や追加演算子、規則追加による制約条件)

さらに前記の各種データ構造を利用して少し実用的な例題を示した図12(a)及び図12(b)を実現するために、いくつかの規則を追加する。なお、図12(a)及び図12(b)は1つの図で示すところ、便宜上、別図として示している。

組み込み関数や演算子、規則として、例えば以下のものを用意する。

・vsize(arg) …引数argで与えられた配列の大きさを求める組み込み関数。

・inttostr(arg) …整数を文字列へ変換する組み込み関数。

・strtrunc(arg1, arg2) …arg1の文字列を複写するが、arg2で指定された文字列長より長い場合、その分の文字を切り捨て。

・配列範囲指定演算子[＊] …配列すべての要素を意味する。

・{}で囲まれたいくつかの制約式をブロックとしてまとめて処理する。

・[＄i] …今までに登場した制約式のインデックス[i]と同等だが、ブロックでのみまとまった制約条件の中ですべて同じ値を持つ。ここでは通常の変数との識別のためにiではなく＄iとして記述している。

以上によって、たとえば以下のような記述が可能である。

・output_block[＊]. output__html …配列内のすべてのoutput__html(図12(a)参照)を並べた複数要素値列を意味し、文字列との比較、代入演算が行われる時は、すべての要素を連結した一つの文字列として扱われる、

・vsize(output__block)＝list__count …配列数、すなわち配列サイズと変数との関係を示す制約式である。但し、vsize()で得られた値の参照は通常の数値であるが、変更は配列サイズの変更を意味する必要があるため、処理系の内部で値の変更が抽象化されたサイズ変更のための関数呼び出しを意味するように定義付けるか、オブジェクトによるメソッド呼び出し処理等の抽象化した処理が必要である。

[0081] 以上のような各種データ構造の制約条件を含む問題への入力データを与えるためには、該当の変数をユーザが直接変更したり、図12(a)に示した手続き型関数のbbs__additem()を呼び出す等の処理を行うことになる。また図12(a),(b)の例題では出力については前記output__htmlを参照する。以上の処理によって、Webサービスを実現することができるようになる。なお、出力を必ずしもHTMLフォーマットに限定する必要はなく、任意のクライアント／サーバシステムにおけるサーバ側の入出力サービスを同様の方法で実現することも可能である。

[0082] なお、これまで1次元配列vecを示して説明したが、2次元配列等の多次元配列への拡張はもちろんのこと、配列長が異なる配列同士の演算やこのような配列と数値との演算に関する定義が同様に行えることは言うまでもない。

また、前述したように、制約条件の内容としてさらに複雑なデータ構造の変数を扱うことが可能な例を示したが、これらデータ構造に完全に一致する必要はなく、当業者であればこれらから類推できる変形例が含まれることは言うまでもない。

[0083] 以上説明したように、制約条件を単なる数値でなく各種のデータ構造をも扱えるようにしたので、現在格納されたデータとHTML出力の関係が制約条件(制約式)によって記述可能となり、Webアプリケーションにおけるサーバ側のプログラムを制約条

件だけで単純に記述することができるようになる。このWebサーバと組合せてデータ入力、表示を行うシステムの概念図を示したのが図11であり、このときの関数型定義保持手段及び制約条件格納手段に格納する情報や、制約条件の一例を示したのが図12(a),(b)である。

[0084] また、前述したようなWebにおけるHTMLに限らず、図13に示すような一般のウィンドウインタフェースでは各項目を変数にマッピングするように構成することで、データとユーザインタフェースとの関係を制約条件で記述することが可能となり、一般的なユーザインタフェースを記述しやすく、且つ理解することが容易となる。これにより、従来のプログラミング方式と比較して高い生産性を実現することができる。

[0085] <本発明の他の適用例:アプリケーションプログラム構築、トランザクション処理>

次に、WEB上の入力インタフェースを用いて任意のアプリケーションプログラムを構築する問題、または任意のデータが蓄積されたデータベースを検索する際のトランザクション処理問題に本発明が適用可能であることを説明する。

前記第1の実施形態ー第4の実施形態における制約条件解決装置において、制約条件により解決していく処理手順及び処理内容を、履歴情報(更新履歴)として持つことにより、アプリケーション上のユーザインタフェースの概念の1つにあるUNDO機能、或いはデータベースのトランザクションを実現することが可能となる。

[0086] このUNDO機能、或いはデータベースのトランザクション処理を実現できるようにすることで、データの更新から始まってデータ変更に伴う制約解を求める一連の動作が行われなかったものとすることが容易に行える。これにより、制約条件のデータ値を一部変更して入力データに含まれる誤りを発見するという入力データの確認や、そのキャンセル時における元のデータ値に復帰させるための処理を確実かつ迅速に行うことができるようになる。

[0087] 次に、GUI環境における基本的な画面構成要素であるウィンドウを用いたウィンドウシステムまたはマルチウィンドウシステムに本発明が適用可能であることを示す。ここでは、図14に示すようなウィンドウシステムを一例に説明する。なお、図14では3つのウィンドウを示しているが、任意個のウィンドウを扱えることは言うまでもない。

[0088] 図15は、ウィンドウシステムを実現するクライアント/サーバシステムにおけるクライ

アント側の構成を概念的に示す図である。サーバ側のシステム構成については図1、3、8、10に示すものと同様であるので省略する。

図15に示すように、GUIクライアントはGUI処理部とGUI制御部を有し、GUI制御部は図16に示すような各パーツの動作を制御する。

ここでのGUI制御部の中身は、図1、3、8、10と同様の構成を持ち、各パーツの状態変更は、クライアント内の図1、3、8、10の入出力I／Fに対するI／F制御部に対して行われる。制約条件として、各パーツ間の連携規則、クライアントで処理するべき入力データの整合性チェック、入力補助、サーバ側へ渡すべきデータへの加工規則などが記述され、処理される。

GUI制御部で加工されて処理されたデータは、伝達用の変数を介してサーバへと伝えられる。

もちろん、クライアントのGUI処理部分とサーバの制約処理系をまとめて1台で処理するなど、システム構成が柔軟に変形して使用できることは言うまでもない。

なお、本発明の特徴である変数の値変更に応じた制約条件の逐次解法の機能は、クライアント／サーバシステムの全体として有する他、前記GUI制御部においても有している。また、クライアント／サーバ間のデータ転送は、独自形式でもよいが、XMLを使用する等、標準化したデータ形式を用いることで従来手法と互換性のあるオープンなプロトコルを実現することが可能である。

[0089] このように、GUI側では、GUIの動作制御、伝達するデータの整合性の確認、入力補助、データの状態に応じたGUIの状態変更、及びサーバ側への通知処理などを制約条件の記述に基づいて行うことができる。

[0090] 次に、ウィンドウの動作について詳しく説明する。各ウィンドウ内の構築は、前述した各種のデータ構造の扱いを許可する制約条件の設定や、UNDO機能或いはデータベースのトランザクション処理を実現できるユーザインタフェースによって可能であるが、このウィンドウ自体の動作は次のようにして記述する。

[0091] ウィンドウの情報として、例えば次の構造体を定義する。

```
struct{
    位置  x, y;
```

サイズ x, y;

表示中フラグ;

}ウィンドウ情報[];

なお、ウィンドウ情報[0]…アプリケーション1、

ウィンドウ情報[1]…アプリケーション2、

ウィンドウ情報[2]…アプリケーション3を表すものとする。

これらの変数値の参照更新は、図10のシステムリソース12と同様の扱いとし、実際のGUIシステムのウィンドウと連動させるものとする。

[0092] アプリケーションを実行する画面上に表示されているウィンドウの動作(表示／非表示、移動、サイズ変更)を制約条件で記述しようとする場合の一例を次に示す。

(a)ウィンドウnの表示／非表示

画面上にウィンドウを開く(表示)…ウィンドウ情報[n]. 表示中フラグ＝true、

画面上のウィンドウを閉じる(非表示)…ウィンドウ情報[n]. 表示中フラグ＝false、

また、画面上にウィンドウを閉じるための「closeボタン」があるときは、例えば、ウィンドウ情報[n]. 表示中フラグ＝！closeボタンの状態(！はnot演算子)とすればよい。

(b)ウィンドウnの移動

ウィンドウ情報[n]. 位置. x＝移動したいxサイズ;

ウィンドウ情報[n]. 位置. y＝移動したいyサイズ;

(c)ウィンドウのサイズ変更

ウィンドウ情報[n]. サイズ. x＝変更したいxサイズ;

ウィンドウ情報[n]. サイズ. y＝変更したいyサイズ;

[0093] その他、アプリケーション1の右にアプリケーション2を隣接するようにするには、

ウィンドウ情報[1]. 位置. x＝ウィンドウ情報[0]. 位置. x＋ウィンドウ情報[n]. サイズ. x;

ウィンドウ情報[1]. 位置. y＝ウィンドウ情報[0]. 位置. y;とすればよい。

また、ウィンドウのサイズを固定にするには、変数に変更可能／不可能の属性を用意した上で、ウィンドウ情報[n]. サイズ. x、ウィンドウ情報[n]. サイズ. y、を変更不可能にすればよい。

[0094] このように、GUI環境におけるウィンドウシステムまたはマルチウィンドウシステムにおいて、従来のプログラミング手法ではウィンドウの動作及びウィンドウ内のアプリケーション動作をイベントが発生したらそのイベント状況に依存する関連動作を逐一記述しなければならなかったのに対し、本発明による動作制御によれば単純明快な制約条件に基づいてアプリケーションの複雑な動きを簡潔且つ容易に記述することができるようになる。

[0095] なお、ウィンドウシステムにおける動作を制約条件の記述によって制御可能なことをGUI環境で前述したが、データを画面上に出力する場合に用いるだけでなく、ウィンドウを用いて処理した内容を任意の媒体(例えば、紙)に出力する出力システム(例えば、帳票印刷システム)を実現する際にも本発明を適用することができることは本発明の技術分野における当業者であれば容易に想到できることは言うまでもない。

また、前述した説明では、クライアント／サーバシステムにおける具体例を示したが、必ずしもクライアント／サーバシステムである必要はなく、クライアントとサーバとをそれぞれ別個のシステムとして構築し、クライアントのみまたはサーバのみに前述した処理が行えるように構成してもよい。

[0096] ＜本発明の他の適用例:並列化処理と構築された計算機システム＞

また、本発明は、手続型プログラミングで面倒な問題である並列化処理の問題に適用することが可能である。並列処理においては、大きく分けて2種類の動作が考えられる。すなわち、いくつかのCPUがステップS23ーステップS30のアルゴリズムを実行する方法と、データフローマシンのようにアーキテクチャで変数と関係式を構築してしまう方法である。

[0097] 前者の場合、通常の手続型プログラミング手法で並列処理を行おうとすれば、アルゴリズムを構築する段階で、ユーザはCPUの並列動作を意識しながらプログラミングの処理手順を分割して記述しなければならない。これに対して本発明を用いた並列化処理の場合は、図2のアルゴリズム中にあるステップS22による変数値の変更から開始されるステップS23ーステップS30の処理を並列に動作させるだけで、ユーザは直接並列処理を意識することなく並列処理を実現できる。

[0098] 具体的には、前記第1の実施形態ー第4の実施形態における制約条件解決装置

において、ある処理に対するステップS23ーステップS30の実行中にも、別の処理に関してのステップS22による変数値の変更による一連の動作を実行させることを行っている。

実際上、従来の手続き型プログラミング手法における並列アルゴリズムの実装と同様に、この並列処理の実行中においては各変数間で排他制御や同期制御を実現する必要があるが、従来の手続き型プログラミングでの手法をそのまま適用することができる。

[0099] 後者のアーキテクチャで変数と関係式を構築する場合、変数間の関係式は、例えば並列計算機上に物理的に実現した時の通信路となる。この並列計算機上では、並列プロセッサが複雑な通信路によって結合されており、通信コストが経路によって異なる。この通信コストを優先度で表わすことによって、データ経路を自動的に選択することが可能である。優先順位については、制約条件の一例として挙げた(優先順位付き制約条件)を用いることができる。このような並列処理と同様の方法で、一般に構築されたシステム全体に本発明を適用することが可能である。

なお、ここでいうシステムとは、複数の特性の異なる計算機や用途別専用機と、それらを結合するネットワークから構成される複合物を意味しており、いわゆる一般の企業システムや社会システムを指す。

[0100] つまり、本発明をシステム全体へ適用する場合、概念的には、前記データフローマシンでの並列処理実現方法と非常に似たものとなり、本発明によって構成される計算機は、計算機間のインタフェースとなる変数を介して別の計算機のインタフェースである変数と結ばれる。変数間の関係式が通信路であり通信コストが優先度であり、したがって本発明によって前記システム全体を設計し構築することができる。

[0101] ＜本発明の他の適用例:既存プログラミング言語との融合＞

これまで説明してきたように、本発明は任意の制約条件から構成される問題を、外部(ユーザ等)からの指示(イベント)を契機に変数の値を変更して制約伝搬によって解決するイベント駆動型の計算実行原理を扱っている。前述したように関数定義保持手段や関数定義実行手段において既存の手続型プログラミング言語を利用する等から明らかなように、図18にも示したが既存のプログラミング言語との組み合わせ

可能なことを前提としている。そこで、本発明を既存プログラミング言語へ融合することが可能であることをさらに詳述する。

[0102] なお、ここまで説明のために、プログラミング言語を分類した場合に、1つは手続型プログラミング、別の1つは宣言的プログラミングであるとして大別してきたが、実際にはもっと細かなプログラミング言語の分類が存在する。すなわち、宣言的プログラミング言語に対する非宣言的プログラミング言語には、手続型プログラミング言語と並ぶプログラミング言語としての関数型プログラミング言語が含まれている。以下、本発明と関数型プログラミング言語との融合について扱う。

[0103] (A)関数型プログラミング言語との融合

ここでは、関数型プログラミング言語として最も歴史があり代表と言えるLISP言語(Common LISP)との自然な融合方法について述べる。いま、2つの変数a, bの初期値、及び制約条件として、

a={11, 12, 13}

b={1, 2, 3}

a[$i] = b[$i] + 10

を定義するとする。a, bの値は共に3つの要素を持つ配列である。また、本発明の実施例において制約式を定義する関数defconstraintを新たに定義し、変数への代入をシンボルへのバインドに対応させるとすれば、LISP言語では次のように記述できるだろう。

```
(set 'a #(11 12 13))
(set 'b #(1 2 3))
(defconstraint '(= (aref a $i) (+ (aref b $i) 10))    …(式18)
```

[0104] ここで、関数defconstraintは、図9のデータ構造を実現するために引数からシンボル(変数)aとbに関する解決式を作成し、

```
(defconstraint2 '(= (aref a $i) (+ (aref b $i) 10))
        '(setf (aref a $i) (+ (aref b $i) 10))
        '(setf (aref b $i) (- (aref a $i) 10)))
```

を実行して、関数defconstraint2が図9のデータ構造を維持する。式のデータ構造とし

ては図6等のデータ構造でよいのだが、LISPではLISP自体の機能のみで構造を自然に記述することができるので、ここではこのLISPの特徴を生かして上記のような式を記述している。

[0105] C言語のポインタ等についても同様なことが言えるのだが、LISP言語においては各データが共有されている可能性があり、変数とデータとの関係が1対1であるとは限らない。このため、あるデータを変更する時にそれが特定のシンボルに関係しているかどうかは変数へ値を代入する時にはわからないのが通常である。したがって、このような変数とデータとの関係構造は、本発明の制約解消を実現する際には問題となるので、これを説明する。

[0106] いま、シンボルbと同じデータを指すシンボルcを加えて、シンボルa,b,cを中心とした通常のLISP言語の内部データ構造を極単純化した一例を図22に示す。LISP言語では、すべてのシンボルはパッケージに記録され、名前空間別に保存されているのでここではパッケージから記述している。また、標準の名前空間名は「user」であるため、シンボルa,b,cは「user」に登録されるが、この例ではシンボルbは別のパッケージにも登録されている。

[0107] LISP言語で、b[1]=12を実行する場合、

(setf (aref b 1) 12)

を実行するが、値の代入を実行する関数setfは上記の関数defconstraint2で定義された制約式とマッチングさせる必要がある。一方、関数setfに渡されるデータは、通常のLISPでは図22中の(＊)で示すポインタであり、マッチングさせるべき情報は存在しない。そこでこのような単純な例の場合、関数評価時に評価結果だけではなく、評価した式の情報を関数setfの実行時まで持ちまわる方法が考えられる。しかし、実際には非常に複雑な式になるため持ちまわるデータが巨大になることや、(setf (aref c 1) 12)でb[1]が変更された場合に対応が難しくなる。

[0108] これを解決するためには、LISP言語の基本データ構造をすべて双方向に辿れるように構成すればよい。そのデータ構造を一例として図に示すと、図23のようになる。図23中の(＊)で示した関数arefによる配列参照は、逆方向へ辿れるように配列内のポインタだけではなく、配列そのものの情報も持つ。配列情報を含むすべてのデータ

は、バインドされているシンボルの情報を追えるように各種情報を保存する。

[0109] また、実際にはシンボルまでのみならず登録されたパッケージ情報まで辿らなければならず、したがって制約式のシンボルはパッケージを考慮する必要がある。(*)からb[1](パッケージを含めてuser:b[1])を辿る時の道順を図23では太線の矢印で示している。

[0110] まお、Common LISPのシンボルにはもともとホームパッケージの情報を持つが、1つのシンボルが複数のパッケージに保存されることがある。このため登録されたすべてのパッケージを追うことができる新たなデータ構造の追加が必要となるが、このようにすれば、setfの実行時に制約式のマッチングを検査することができるようになる。以上説明したように、本発明をLISP言語を代表とする関数型プログラミング言語と完全に融合させて自然に扱うことは可能である。

[0111] 前記(A)の関数型プログラミング言語との融合によって、LISP言語上で本発明を適用することができるようになり、関数型プログラミング言語上への実装としては理想的になるが、例えばsetf等の代入操作が実行される度に制約式のマッチングが必要となることから原理上、実行効率が悪くなる。特に、代入時の逆方向の制約式とのマッチング(逆方向のマッチング)に際しては、かなり上の方まで遡らなければ、該当する制約式の定義が存在するかどうかわからない。例えばポインタの時点でわかるのは[1]等のインデックス情報だけであるが、単純な制約式でさえa[x]、b[x]のような構造をしているため、インデックス情報だけではマッチさせるべき制約条件が存在するかどうかを判断できるのは限定された特殊条件の時だけとなる。

[0112] これに対して、シンボル名(変数名)から辿るマッチングの場合、多くの場合で制約条件が存在するかどうかが最初からわかっており、変数(シンボル)そのものにそれらの情報を付加させることによって処理の高速化が可能である。

また、実際の制約式はa[x]等の単純な例が登場することは少なく、逆方向とのマッチングは常に相当に複雑な式をかなり上方、おそらくは完全に辿る必要がある。これを実際のCommon LISPの複雑なパッケージ機構上に実装することを考えると、実装は実に複雑となり、実行時間については様々な方法によって多少の改善は期待できることを考慮しても、変数から開始する制約式のマッチングに比べて明らかに実行効

率は非現実的なまでに悪くなってしまう。

[0113] そこで、現実的解決方法として、完全な融合をあきらめて、制約条件を使用する場合は、専用の式処理機構を用意してその中では必ず変数(シンボル)からしか値を参照できない制限を加えれば前述した問題を防ぐことができる。例えば、文字列データを「″」で括って表わしているように、式データをデータ型の一つとして抽象化して例えば「？」で括って表現し、専用の式処理機構を実行する関数pevalを定義し、代入式を関数peval内で実行するならば、前記式(18)は、

(peval '?a={11, 12, 13}?)

(peval '?b={1, 2, 3}?)

(defconstraint '?a[$i]=b[$i]+10?)

と記述できる。そして、b[1]=12を実行する場合は、(peval '?b[1]=12?)を実行すればよい。

[0114] この場合、必ず式処理の中では変数(記号)からの参照しか行えないことにするため、逆方向のマッチングは必要ないので、前述した(A)で述べたLISP言語上での実装の問題は発生せず、さらにLISP言語の他の機能にはまったく影響を与えないため実行効率の問題が解決する。

[0115] ただし、プログラミング言語としては、言うまでもなくとても不自然である。そこで、プログラミング言語の文法そのものを新しく定義して、代入操作時に専用の式処理機構を自然に使わせるように解釈するパーサー(構文解析器)を用意すればよい。これは、入力時に文法を構文解析してLISPである内部表現に置き換えるので、本来のLISP言語とは違うものであるが、LISP言語の良いところと本方式を組み合わせた実行効率が著しく低下しない能力を持つプログラミング言語の実現が可能となる。

[0116] なお、ここでは、式評価は常にpeval関数の中で行うことを基本とした、最もメジャーなC言語風の文法を持つプログラミング言語の基本要素を示すことによって、peval関数を自然に使わせる方法について述べる。変換後を見ればわかるように、C言語風の文法を持ちC言語同様の記述ができるとはいえ、本質的にはLISPであり、LISPの柔軟な記述能力を持っているという優位性がある。さらに、純粋なLISPプログラムとも完全に混在させて実行することができる。

[0117] 本システムの基本単位と言える関数を含む式の解釈実行は次のように翻訳する。なお、ここでは型宣言に関しては本質的に関係ないため省略している。

入力:

```
a[1] = func1(100);
```

実行関数:

```
(peval '?a[1]=func1(100)?)
```

[0118] C言語の基本構造である局所変数を持つブロック構造は次のように解釈する。

入力:

```
{
    int x, y;
    x＝10;
    y＝20;
}
```

実行関数:

```
(let (x y)
    (peval '?x=10?)
    (peval '?y=10?))
```

[0119] C言語の大まかな基本要素をすべて含む、局所変数とループ構造を持つ関数は次のように解釈する。

入力:

```
int f( int v )
{
    int i, n = v;
    for (i = 0; i < 10; i++)
        n++;
    return n;
}
```

実行関数:

```
(defun f (v)
    (let (i, n)
        (peval '?n=v?)
        (prog ()
            (peval '?i=0?)
            loop
            (if (not (peval '?i<10?))
                    (return nil))
            (peval '?n++?)
            (peval '?i++?)
            (go loop)
        (return-from f n)))
```

[0120] 式の解釈と実行方法は様々なものが考えられ、これらの変換方法はほんの一例でしかないが、変換方法についてはすでに確立された様々なコンパイラ技術が適応できることに留意されたい。このように、制約式の関係する式実行を自然に関数pevalで置き換えて実行する、本発明とLISPの特徴を組合せた極めて強力なプログラミング言語を設計することは可能である。これを用いて、関数型言語と融合させた強力なプログラミング言語で図12(a),(b)を記述することで、本発明の図3,図8,図10に示した制約条件解決装置が現存する手続型、関数型プログラミングの機能を網羅しながら実際の実行効率を落とさずに実行することが可能となる。この際、図21に示すオペレータは、関数PEVAL内の式処理で実現されることとなる。

[0121] ＜本発明の他の適用例:設計手法との組み合わせ＞

また、本発明は、設計手法に組合せて適用することが可能である。開発設計手法には、実に様々なものがあるが、どのような方法であれ、まず実現したい要求を分析し、次に要求を実現するために開発するターゲットの仕様を決定し、そしてその仕様を満たすプログラムを開発することが共通的な概念としてある。しかしながら、従来の手続き型プログラミング手法の場合、どの開発手法であっても最終的にはCPUの動きをプログラムコードとして記述しなければならなかった。一方、宣言的プログラミング

手法はこの問題を解決する可能性を秘めているが、冒頭で述べたように計算機の実行効率が良くないという問題が大きい。そのため、現在の一般的プログラミング作業といえば、手続き型プログラミング手法のことを指すことが多い。

[0122] さて、システム設計手法のひとつとして、データ中心指向がある。データ中心指向は業務システムをうまく設計できる方法として知られているが、これによって要求を実現するシステム設計を行ったとしても、最終的にはコーディング作業が必要であり、プログラムコードとしてCPUの動きを記述しなければならない。CPUの動きの記述については、ソフトウェア工学が発達した現在でも本質的にはプログラマーの経験と勘に頼っており、未だ人間の創造的作業の領域内に存在する開発設計作業が大変であるのが現状である。開発設計作業を低減させる道具としてCASEツール等がある。しかし、これらは開発作業を軽減することはできても、人間がCPUの動きを記述することを不要にするという本質的な問題については依然として解決できていなかった。

[0123] そこで従来の手法では、前記本質的問題の解決には目をつぶる代わりにプログラムコードを再利用することに重点を置いている。例えばソフトウェアのライブラリ化やオブジェクト指向プログラミング言語は、この再利用を目的として概念化され生み出されてきたのである。

[0124] これに対して本発明は、計算機の動きを考慮する必要があるものの、従来のような複雑な動きを考慮する必要が全くない。つまり、プログラマーは各種設計手法に共通する問題であるCPUの動きを直接記述する作業から解放され、コーディング時での創造的作業を行う必要がない。したがって、本発明によれば、CASEツール又はCASEツールに類似したツール及びその他ツールの発展形として、仕様からプログラムコードを自動生成させることが可能となり、自動プログラミングを実現することができる。

[0125] もし計算機単体ではなくシステム全体を本発明によって設計開発するならば、広大なシステム全体を自動生成させることも可能である。この場合非常に多くの変数と関係式が使用されるが、従来の設計手法と同様に、トップダウン的設計手法又はボトムアップ的設計手法が実現できたり、さらには機能による領域分割を行うことや、分割された領域を再利用することも可能であることは言うまでもない。また、本発明がどのよ

うな開発設計手法についても共通して適用できることは言うまでもない。

[0126] ＜本発明の他の適用例:グラフィックデータやロボットへの応用＞

本発明の更なる適用例として、コンピュータグラフィックのモデルやロボットの関節等の可動部への応用について説明する。ここではロボット制御の精度といった本発明と直接関係ない複雑な問題は省略して、連結された関節の扱いについてのみ説明する。

いま、結ばれた点a,eの間に4つの関節A, B, C, Dがあるとして、関節AB間の位置をb、関節BC間の位置をc、関節CD間の位置をdとすれば、aーe間の関係はこれら5つの点で表現することができる。各点が(x, y, z)の3つの座標を持つベクトルで表わされた3次元の場合、各関節の回転の動きは3×3の回転行列で示される。なお、この回転行列にさらに並行移動成分を入れて4×4行列としてもよい。

[0127] 本問題を単純化するために、各点間の長さ等を省略した単なる回転のみを制約式で表すとするならば、

b＝a＊A …(a)

c＝b＊B …(b)

d＝c＊C …(c)

e＝d＊D …(d)

となる。ここで、A, B, C, Dは対応する関節の回転行列を示す。

[0128] 通常、関節の移動範囲には制限があるため各回転行列AーDの値にも制限があり、ユーザはこの制限を制約式で記述することになる。もし、aが固定されていて関節Bが変化した場合、前記(b)→(c)→(d)の順の制約伝搬が実行されることが必要である。また、もし、固定されたaに対して動かしたい位置eがある場合、この制約伝搬によって、各回転行列D, C, B, Aの必要な回転成分だけが修正されて自動的に定まる。ここでaが固定されていることは、制約条件として固定されていることを示す変数を追加するか、システムの変数や関係式へ属性として持たせること等で実現できる。そこで、回転行列AーDをそれぞれ関節を制御するモーター等の角度とするとこれはロボット制御に適用することができる。また、3次元座標aーeを透視変換してディスプレイデバイス等へ描画させれば、これはコンピュータグラフィックへ適用する場合とまっ

たく同様にある。

[0129] このようなロボットの関節等の可動部へ本発明を適用する場合、現実的にはもっと複雑になるが、それでも手続型プログラミング手法での実現よりは格段に単純で自然な形で実現することができる。なお、ロボットの関節等の可動部への適用例はあくまでも一例に過ぎず、関節に類似した任意の問題をうまく解決できることは言うまでもない。

[0130] ＜本発明の他の適用例:化学分野、物理分野、シミュレーション分野への応用＞

また、本発明の更なる適用例として、化学分野への応用について説明する。

化学式を扱う場合には様々なモデル化が考えられるが、ここでは説明のために簡単化して本発明に直接関連する内容のみを説明する。

[0131] 化学式における各元素をオブジェクトとして扱うために、元素の種類別にクラスとして定義し、使用したい元素を必要に応じてインスタンス化する。各元素オブジェクトは、制約条件として他元素との結合条件や元素の性質を持つことにする。また、元素の組合せは化合物オブジェクトで管理し、元素オブジェクトを化合物オブジェクトへ登録することによって表すようにする。化合物オブジェクト内では、組合せた時の性質や組合せ条件を制約式で持つことにする。このように構成することによって、ユーザはオブジェクトの組合せを自由に変更できるが、一方で制約条件によって不可能な組合せは禁止され、そして元素の変更が成功した時にはこれに応じて化合物の性質も更新されたことを知ることができる。また、この時、例えば元素オブジェクト内に電子の状態を変数(属性)として持っているならば、電子変数に関する制約伝搬が発生して、物理現象である電子の挙動を表すことができる。

[0132] 以上説明したように、本発明は化学現象や物理現象等を効率的に表すことができるが、これはある状況のモデル化と、事象の発生に対する現象のシミュレーションであり、変数値の変更に対する制約解消それ自体が特定の現象そのものとなっている。なお、化学現象や物理現象等に限らず多くのシミュレーション分野で本方式が効果的に使用できることは言うまでもない。

[0133] ＜本発明の他の適用例:線形計画法(シンプレックス法等)への応用＞

また、これまでの説明から、本発明が、典型的にはシンプレックス法等の線形計画

法、整数計画法、輸送問題などに記載される制約条件において、対象問題に関する制約伝播の処理手順が考慮された制約条件をユーザが入力することができる場合であれば適用可能であることは当業者であれば明白である。

[0134] ＜本発明の他の適用例:ニューラルネットワークへの応用＞

ニューラルネットワークにおけるニューロンモデルは、閾値を変数として制約式で記述することができる。この観点で見るならば、ニューラルネットワークのニューロンモデルの結合は、本方式で扱う制約式関係の特殊形であり、サブセットとみなすことができる。

ニューラルネットワークでは、求めたい結果を得るためにネットワーク内の各ニューロンモデルの閾値を決定することを学習と呼ぶが、この学習では所望の結果を得るための閾値をどのようにして設定させるかが課題であった。そしてその学習方法といえば、本方式に対する従来の制約プログラミングに概念的に近いものであった。しかし、ニューロンモデルの閾値の設定は変数の値を設定する作業であり、本発明で述べてきた方式によるプログラミングと見なすことができる。従って、本発明をニューラルネットワークへ応用することができる。すなわち、本発明によって、値を自動調整させるだけではなく、動きを考慮して一部あるいは全部の閾値を設定することによる学習方法を実現することができる。

[0135] なお、前述した内容はあくまでも本発明の技術思想を説明するための一例であってこれらに限定されることはなく、本発明の技術思想を実現するにあたり、本発明の技術分野における通常の知識を有する者が、制約条件解決装置の各構成や制約条件の定義の方法を追加・変更しても本発明の根本的な技術思想は不変であることには変わりはないことは明らかである。

[0136] また、本発明の目的は、前述した実施形態の制約条件解決装置100, 200, 300, 400の機能を実現するソフトウェアのプログラムコードを記憶した記憶媒体を、システム或いは装置に供給し、そのシステム或いは装置のコンピュータ(またはCPUやMPU)が記憶媒体に格納されたプログラムコードを読みだして実行することによっても、達成されることは言うまでもない。

[0137] この場合、記憶媒体から読み出されたプログラムコード自体が本実施の形態の機

能を実現することとなり、そのプログラムコードを記憶した記憶媒体及び当該プログラムコードは本発明を構成することとなる。プログラムコードを供給するための記憶媒体としては、ROM、フレキシブルディスク、ハードディスク、光ディスク、光磁気ディスク、CD—ROM、CD—R、磁気テープ、不揮発性のメモリカード等を用いることができる。

[0138] また、コンピュータが読みだしたプログラムコードを実行することにより、上記本実施の形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼動しているOS等が実際の処理の一部または全部を行い、その処理によって本実施の形態の機能が実現される場合も含まれることは言うまでもない。

図面の簡単な説明

[0139] [図1]本発明の第1の実施形態における制約条件解決装置の基本的な構成要素を示す全体構成図である。

[図2]本発明の第1の実施形態における制約条件解決装置の動作手順を示すフローチャートである。

[図3]本発明の第2の実施形態における制約条件解決装置の全体構成図である。

[図4]第1の実施形態における制約条件解決装置の制約条件格納手段が計算機の記憶領域へ制約条件を格納する形態例を示す図である。

[図5]会計計算問題に関するデータ構造の一例を示す図である。

[図6]会計計算問題に関する制約条件の格納形態の一例を示す図である。

[図7]データ構造の一例を示す図である。

[図8]本発明の第3の実施形態における制約条件解決装置の全体構成図である。

[図9]制約条件抽出手段による解決方法の実行手順を示す図である。

[図10]本発明の第4の実施形態における制約条件解決装置の全体構成図である。

[図11]Webサーバと組合せてデータ表示を行うシステムの概念図である。

[図12(a)-(b)]図11に示すシステムにおいて、関数型定義保持手段及び制約条件格納手段に格納する情報や制約条件の一例を示した図である。

[図13]一般のウィンドウインタフェースにおいて、データとユーザインタフェースとの関係を制約条件で記述する一例を示した図である。

[図14]マルチウィンドウシステムの一例を示す図である。

[図15]ウィンドウシステムを実現するクライアント／サーバシステムにおけるクライアント側の構成を概念的に示す図である。

[図16]GUIクライアントにおいて、GUI制御部が各パーツの動作を制御することを説明するための図である。

[図17]従来の宣言的プログラミング手法における解法手順を示す図である。

[図18]従来のプログラミング言語環境と、本発明の制約条件の解を求める制約解消系との位置付けを概念的に示した図である。

[図19]従来のプログラミング言語環境の構成を概念的に示した図である。

[図20]データ定義と制約式をまとめたクラス化に関する概念を示じた図である。

[図21]配列構造の制約条件にアクセス際のオペレータ例を示す図である。

[図22]通常のLISP言語の内部データ構造を極単純化した一例を示す図である。

[図23]LISP言語の基本データ構造を双方向に辿ることができるようにした一例を示す図である。

[図24(a)-(h)]ユーザが制約条件及び値を入力し、計算結果などを出力する画面例を示す図である。

符号の説明

[0140] 1 制約条件入力手段

2 変数値設定手段

3 制約条件抽出手段

4 制約条件解算出手段

5 変数値再設定手段

6 制約条件格納手段

7 変数値格納手段

8 制約条件抽出手段

9 関数定義保持手段

10 関数定義実行手段

11 データベース

12 システムリソース

100 制約条件解決装置

200 制約条件解決装置

300 制約条件解決装置

400 制約条件解決装置

請求の範囲

[1] ユーザにより対象問題に関する制約伝播の処理手順が考慮された制約条件を入力する制約条件入力処理と、

前記制約条件入力処理により入力された制約条件に用いられる変数の初期値または変動値を設定する変数値設定処理と、

前記制約条件入力処理により入力された制約条件の中から、前記変数値設定処理により設定された値に関係する制約条件を抽出する制約条件抽出処理と、

前記制約条件抽出処理により抽出された制約条件全てに対して、前記ユーザにより考慮された制約伝播の処理手順に基づいて前記変数の初期値または変動値を代入して当該制約条件の解を算出する制約条件解算出処理と、

前記制約条件解算出処理により算出された解を前記変数値設定処理での新たな変動値として設定する変数値再設定処理とを含み、

前記変数値再設定処理は、再設定される新たな変動値が存在する間は、前記制約条件抽出処理及び前記制約条件解算出処理を繰り返し行うことを特徴とする制約条件解決方法。

[2] 前記制約条件入力処理により入力された制約条件を格納する制約条件格納処理と、

前記変数値設定処理により入力された変数の値を格納する変数値格納処理とをさらに含み、

前記ユーザにより前記制約条件または前記変数値の追加、変更、削除が行われると、前記制約条件抽出処理は、前記制約条件格納処理または前記変数値格納処理により格納されている制約条件または変数値と比較して、相違が生じた制約条件のみを抽出することを特徴とする請求項1に記載の制約条件解決方法。

[3] 手続き型または関数型のアルゴリズムを設定可能な関数型定義保持処理と、

前記関数型定義保持処理により設定されたアルゴリズムに基づいて、前記対象問題を手続き型または関数型で解決することを可能にする関数型定義実行処理とをさらに含むことを特徴とする請求項1または2に記載の制約条件解決方法。

[4] 前記変数値設定処理及び前記変数値再設定処理により設定される変数は、数値

のみならず、文字、文字列、抽象的なデータ構造、組み込み関数を包含するデータ、及びオブジェクト型データ構造の少なくとも何れかを含むことを特徴とする請求項1ー3の何れか1項に記載の制約条件解決方法。

[5] 前記対象問題が、WEB上の入力インタフェースを用いて任意のアプリケーションプログラムを構築する問題、または任意のデータが蓄積されたデータベースを検索する際のトランザクション処理問題であって、

前記対象問題を前記制約条件により解決していく処理手順及び処理内容を、履歴情報として保持する履歴保持処理を含むことを特徴とする請求項1ー4の何れか1項に記載の制約条件解決方法。

[6] GUI環境における基本的な画面構成要素であるウィンドウを用いたウィンドウシステムまたはマルチウィンドウシステム、または前記GUI環境におけるウィンドウを用いて処理した内容を出力する出力システムにおいて、前記アプリケーションプログラムの構築処理を行うことを特徴とする請求項5に記載の制約条件解決方法。

[7] 制約条件を入力する制約条件入力処理と、

前記制約条件入力処理により入力された制約条件に用いられる変数の値を設定する変数値設定処理と、

前記制約条件入力処理により入力された制約条件の中から、前記変数値設定処理により設定された値に関係する制約条件を抽出する制約条件抽出処理と、

前記制約条件抽出処理により抽出された制約条件の解を算出する制約条件解算出処理とを含む制約条件解決方法において、

前記制約条件は、現在の配列要素数を含んでいること、前記変数に複数個の値を持たせること、構造体配列の要素値による検索を含んでいること、及びデフォルト規則による文字列数値変換の少なくとも何れかを特徴として構成される制約条件解決方法。

[8] ユーザにより対象問題に関する制約伝播の処理手順が考慮された制約条件を入力する制約条件入力手段と、

前記制約条件入力手段により入力された制約条件に用いられる変数の初期値または変動値を設定する変数値設定手段と、

前記制約条件入力手段により入力された制約条件の中から、前記変数値設定手段により設定された値に関係する制約条件を抽出する制約条件抽出手段と、

前記制約条件抽出手段により抽出された制約条件全てに対して、前記ユーザにより考慮された制約伝播の処理手順に基づいて前記変数の初期値または変動値を代入して当該制約条件の解を算出する制約条件解算出手段と、

前記制約条件解算出手段により算出された解を前記変数値設定手段での新たな変動値として設定する変数値再設定手段とを含み、

前記変数値再設定手段は、再設定される新たな変動値が存在する間は、前記制約条件抽出手段及び前記制約条件解算出手段が繰り返し行うことを特徴とする制約条件解決装置。

[9] 前記制約条件入力手段により入力された制約条件を格納する制約条件格納手段と、

前記変数値設定手段により入力された変数の値を格納する変数値格納手段とをさらに含み、

前記ユーザにより前記制約条件または前記変数値の追加、変更、削除が行われると、前記制約条件抽出手段は、前記制約条件格納手段または前記変数値格納手段により格納されている制約条件または変数値と比較して、相違が生じた制約条件のみを抽出することを特徴とする請求項8に記載の制約条件解決装置。

[10] 手続き型または関数型のアルゴリズムを設定可能な関数型定義保持手段と、

前記関数型定義保持手段により設定されたアルゴリズムに基づいて、前記対象問題を手続き型または関数型で解決することを可能にする関数型定義実行手段とをさらに含むことを特徴とする請求項8または9に記載の制約条件解決装置。

[11] 前記変数値設定手段及び前記変数値再設定手段により設定される変数は、数値のみならず、文字、文字列、抽象的なデータ構造、組み込み関数を包含するデータ、及びオブジェクト型データ構造の少なくとも何れかを含むことを特徴とする請求項8ー10の何れか1項に記載の制約条件解決装置。

[12] 前記対象問題が、WEB上の入力インタフェースを用いて任意のアプリケーションプログラムを構築する問題、または任意のデータが蓄積されたデータベースを検索する

際のトランザクション処理問題であって、

前記対象問題を前記制約条件により解決していく処理手順及び処理内容を、履歴情報として保持する履歴保持手段を含むことを特徴とする請求項8ー11の何れか1項に記載の制約条件解決装置。

[13] GUI環境における基本的な画面構成要素であるウィンドウを用いたウィンドウシステムまたはマルチウィンドウシステム、または前記GUI環境におけるウィンドウを用いて処理した内容を出力する出力システムにおいて、前記アプリケーションプログラムの構築処理を行うことを特徴とする請求項12に記載の制約条件解決装置。

[14] 制約条件を入力する制約条件入力手段と、

前記制約条件入力手段により入力された制約条件に用いられる変数の値を設定する変数値設定手段と、

前記制約条件入力手段により入力された制約条件の中から、前記変数値設定手段により設定された値に関係する制約条件を抽出する制約条件抽出手段と、

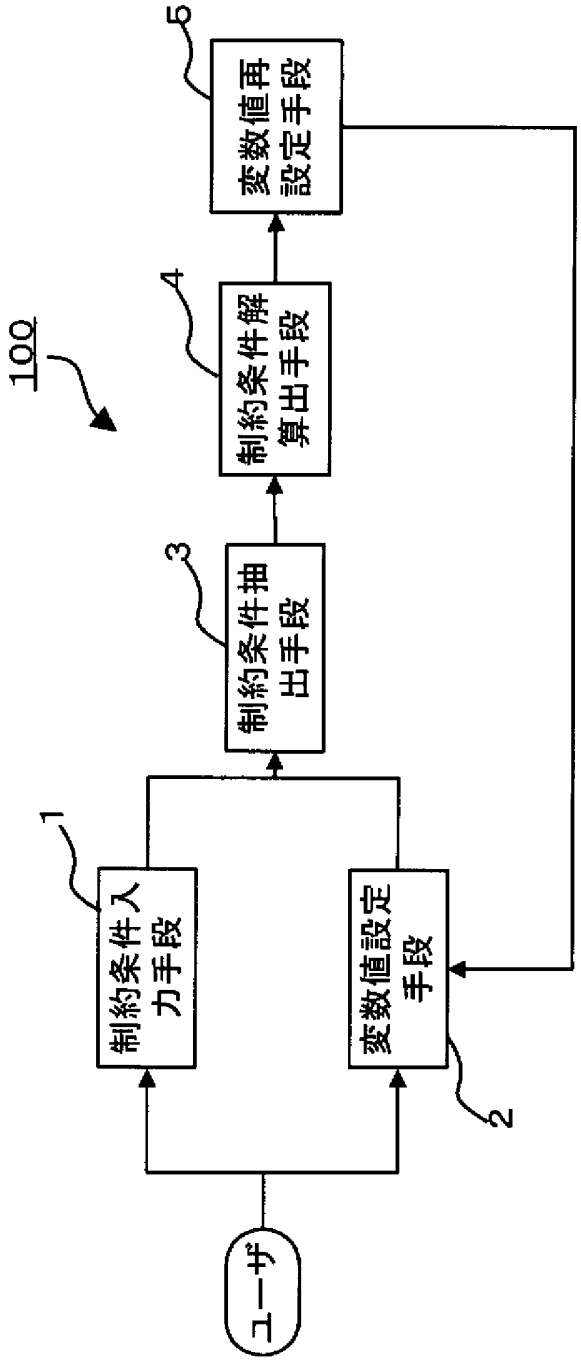
前記制約条件抽出手段により抽出された制約条件の解を算出する制約条件解算出手段とを含む制約条件解決方法において、

前記制約条件は、現在の配列要素数を含んでいること、前記変数に複数個の値を持たせること、構造体配列の要素値による検索を含んでいること、及びデフォルト規則による文字列数値変換の少なくとも何れかを特徴として構成される制約条件解決装置。

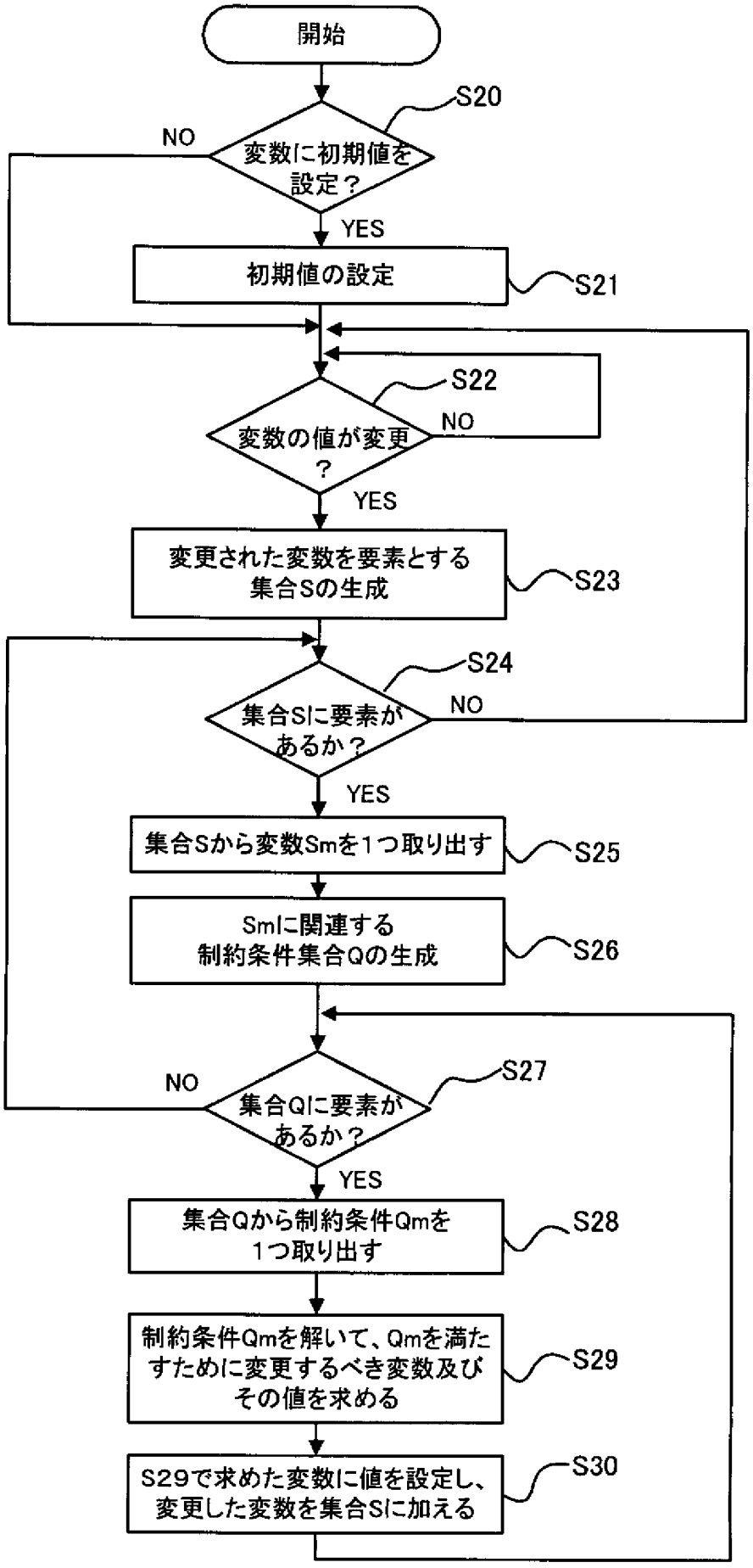
[15] 複数の機器が互いに通信可能に接続されている制約条件解決システムであって、

上記機器のうち少なくとも1つの機器は、請求項8ー14の何れかに記載の制約条件解決装置の機能を有することを特徴とする制約条件解決システム。

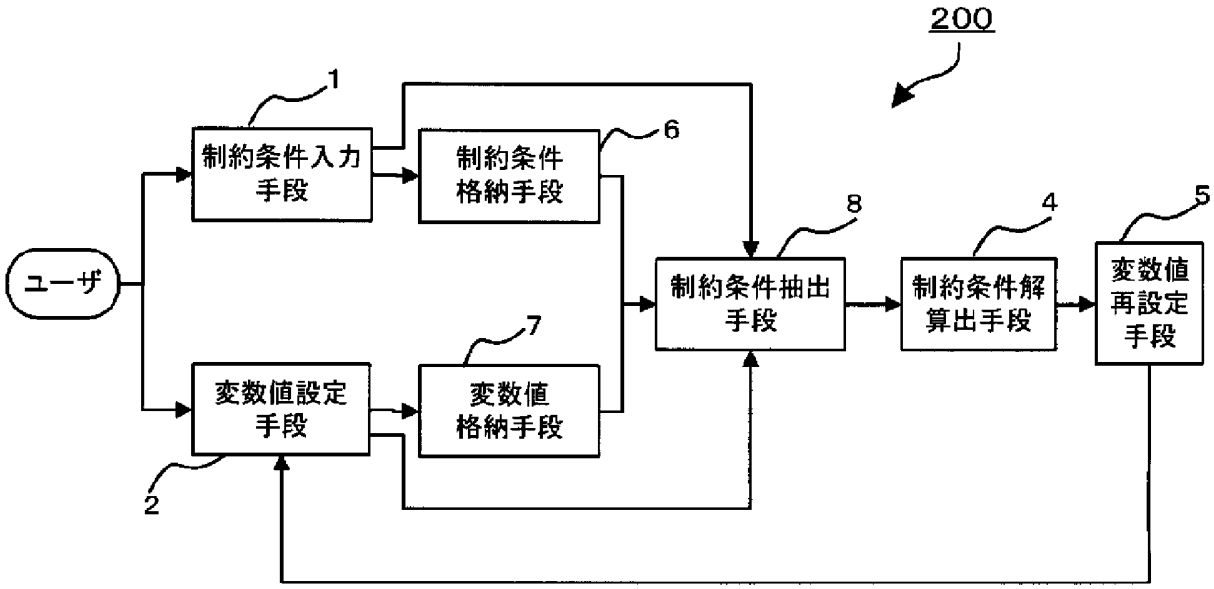
[図1]


100
ユーザ
制約条件入力手段
1
変数値設定手段
2
制約条件抽出手段
3
制約条件解算出手段
4
変数値再設定手段
5

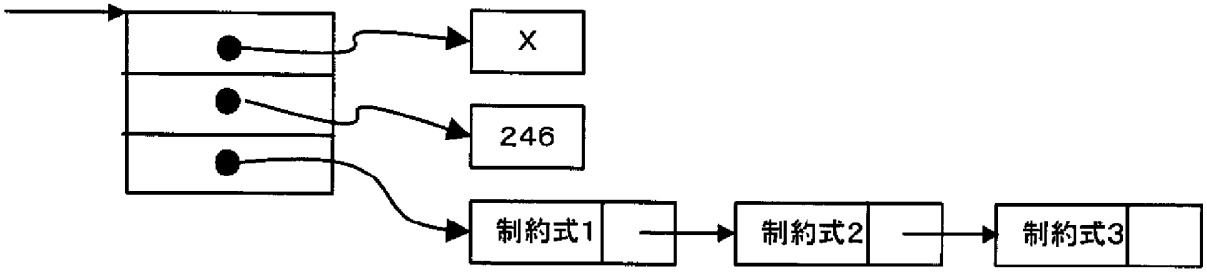
[図2]


開始
S20
NO
変数に初期値を設定？
YES
初期値の設定
S21
S22
変数の値が変更？
NO
YES
変更された変数を要素とする集合Sの生成
S23
S24
集合Sに要素があるか？
NO
YES
集合Sから変数Smを1つ取り出す
S25
Smに関連する制約条件集合Qの生成
S26
S27
NO
集合Qに要素があるか？
YES
集合Qから制約条件Qmを1つ取り出す
S28
制約条件Qmを解いて、Qmを満たすために変更するべき変数及びその値を求める
S29
S29で求めた変数に値を設定し、変更した変数を集合Sに加える
S30

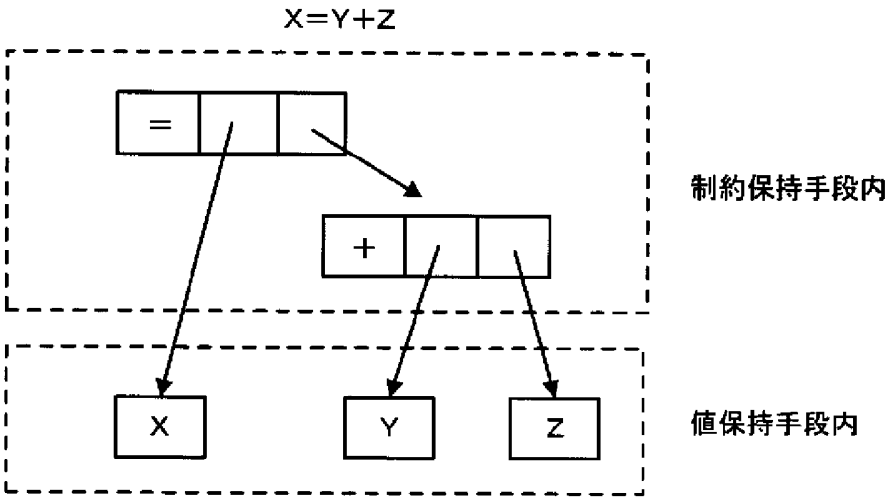
[図3]


200
1
制約条件入力手段
6
制約条件格納手段
8
4
5
ユーザ
制約条件抽出手段
制約条件解算出手段
変数値再設定手段
7
変数値設定手段
変数値格納手段
2

[図4]

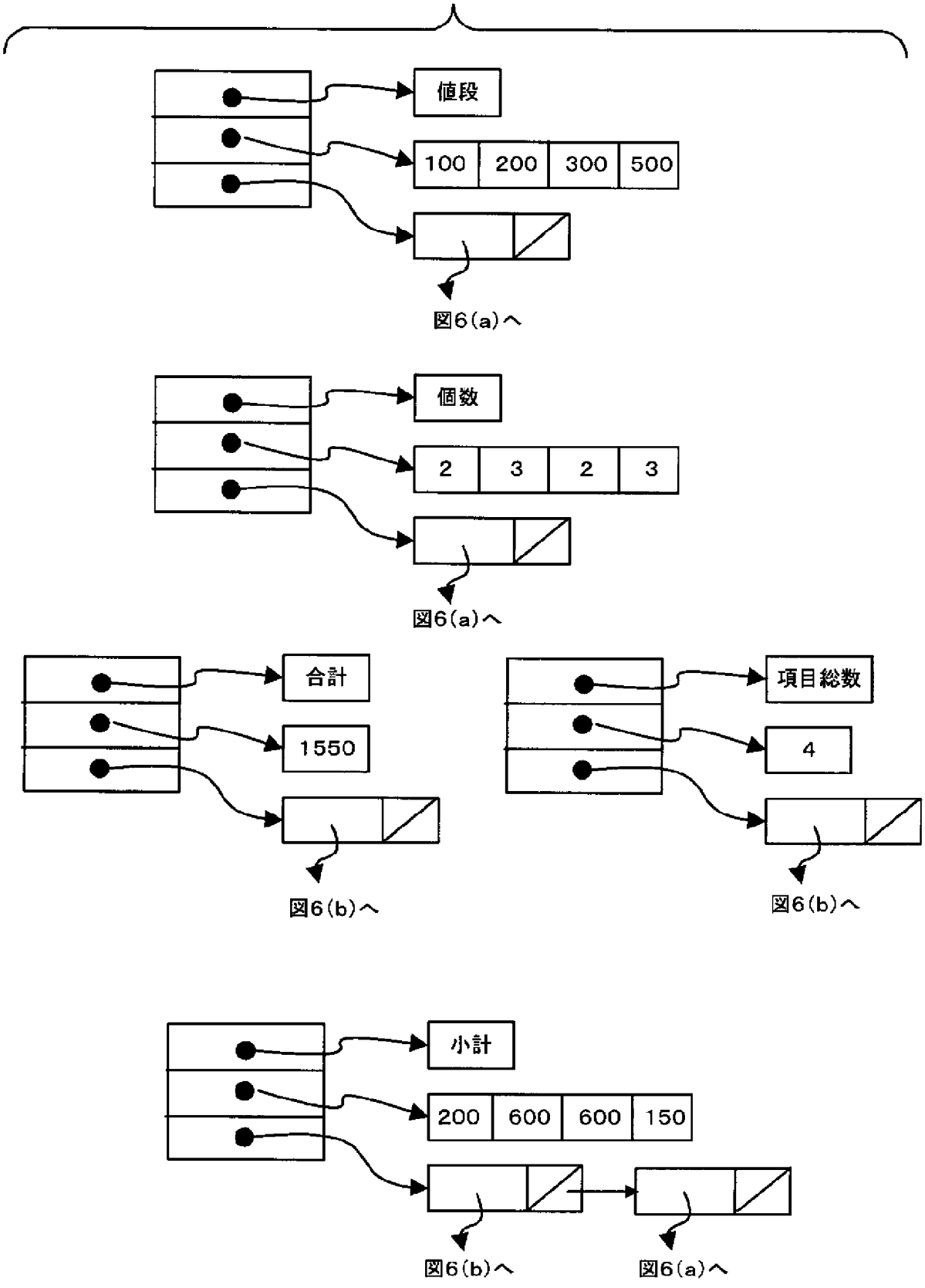

X
246
制約式1
制約式2
制約式3

(a)


X=Y+Z
=
+
制約保持手段内
X
Y
Z
値保持手段内

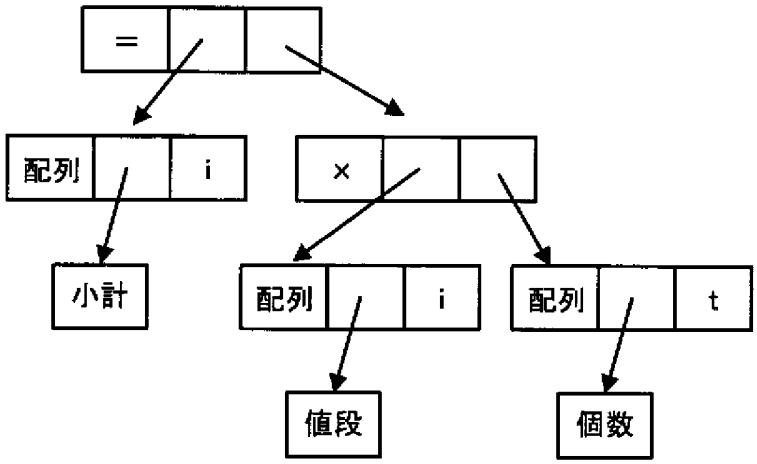
(b)

[図5]


値段
100
200
300
500
図6(a)へ
個数
2
3
2
3
図6(a)へ
合計
1550
図6(b)へ
項目総数
4
図6(b)へ
小計
200
600
600
150
図6(b)へ
図6(a)へ

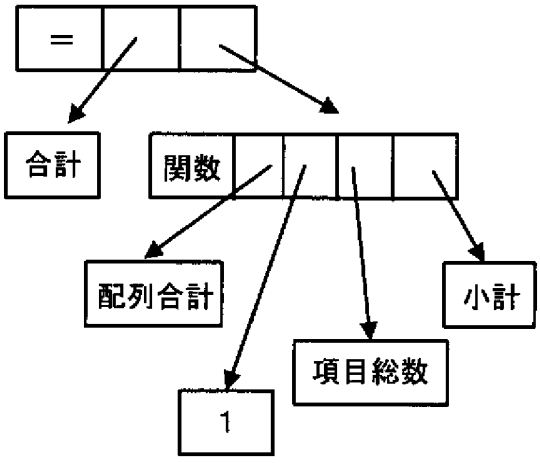
[図6]

小計[i]＝値段[i]×個数[i]



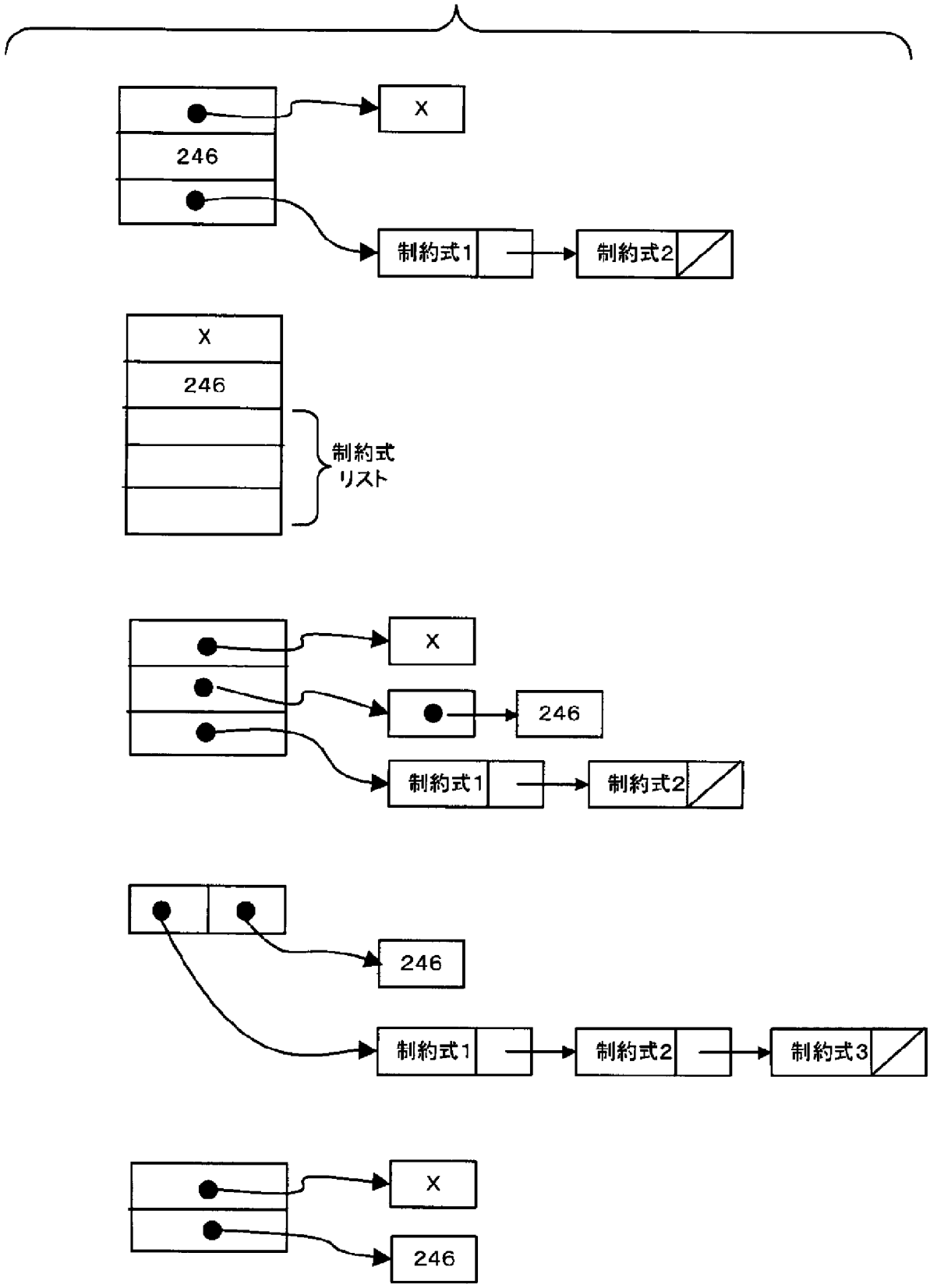
(a)

合計[i]＝配列合計(1、項目総数、小計)

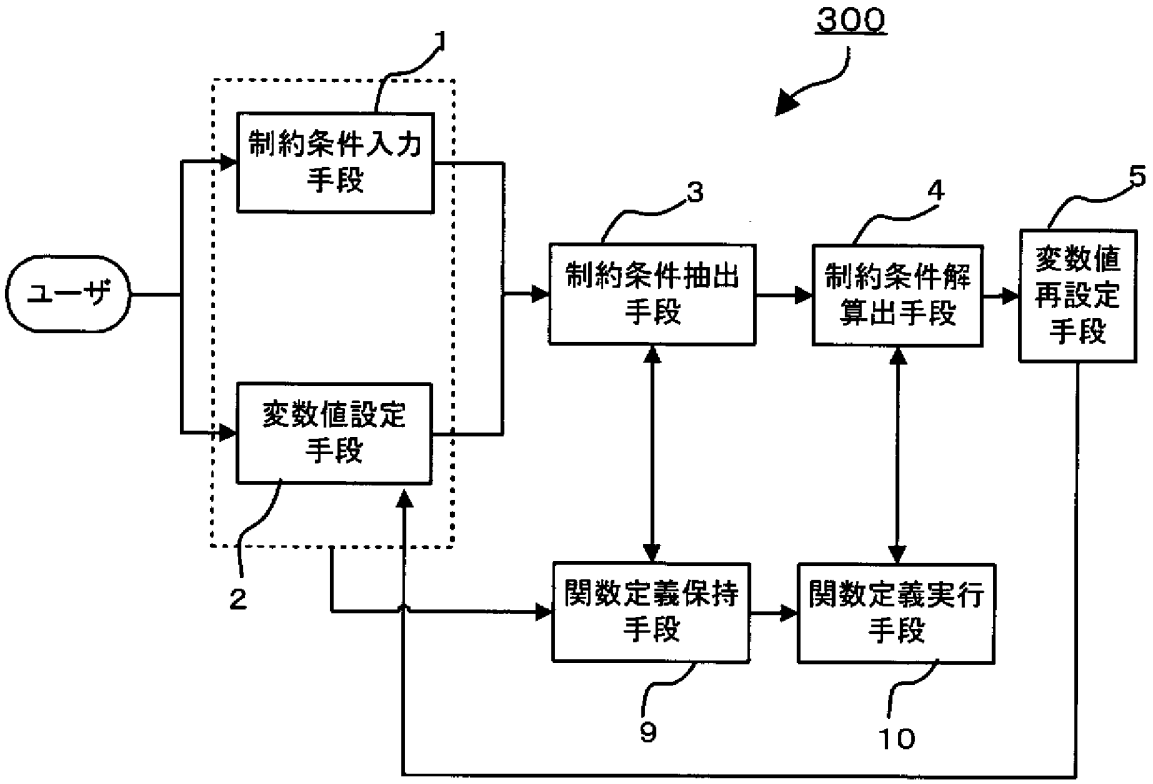


(b)

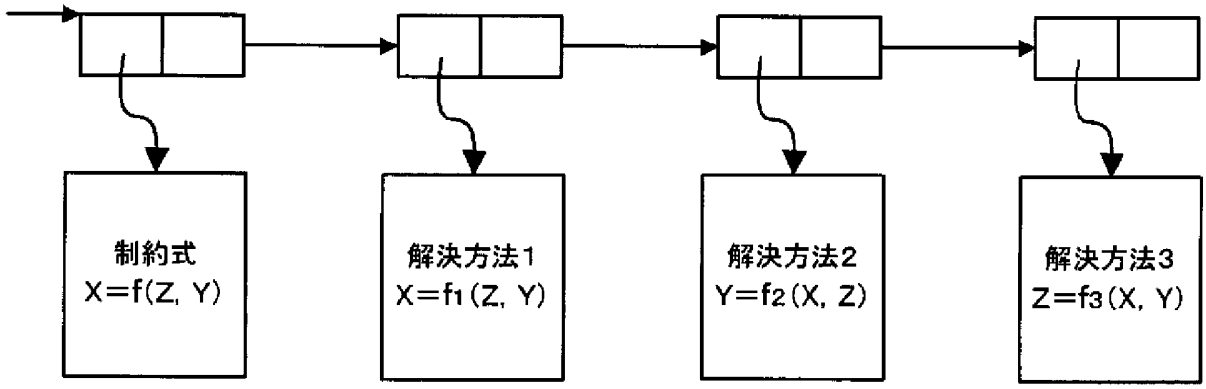
[図7]


X
246
制約式1
制約式2
X
246
制約式
リスト
X
246
制約式1
制約式2
246
制約式1
制約式2
制約式3
X
246

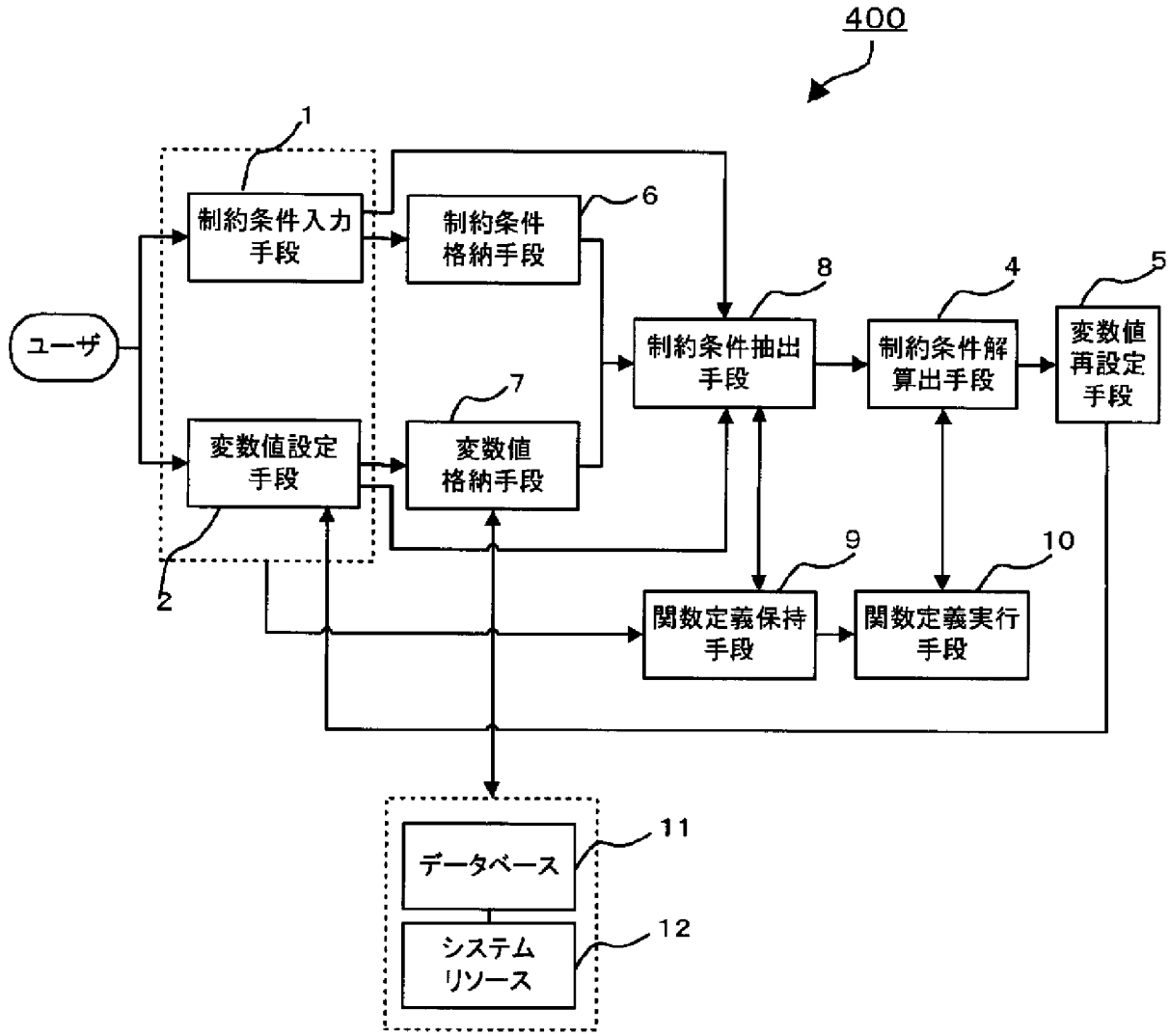
[図8]


300
1
制約条件入力手段
ユーザ
変数値設定手段
2
3
制約条件抽出手段
4
制約条件解算出手段
5
変数値再設定手段
関数定義保持手段
9
関数定義実行手段
10

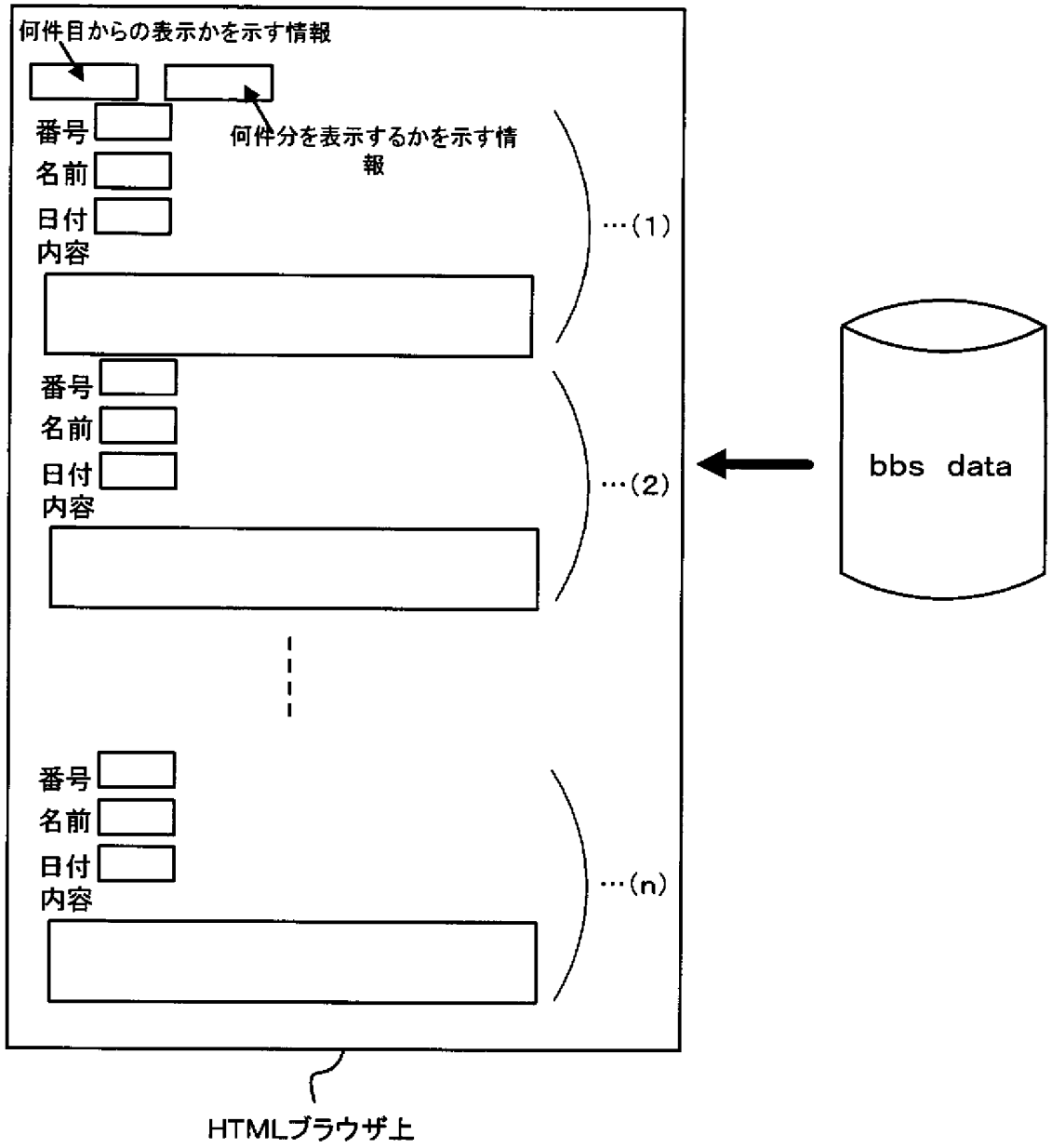
[図9]


制約式
X=f(Z, Y)
解決方法1
X=f1(Z, Y)
解決方法2
Y=f2(X, Z)
解決方法3
Z=f3(X, Y)

[図10]


400
1
制約条件入力手段
制約条件格納手段
6
8
4
5
ユーザ
制約条件抽出手段
制約条件解算出手段
変数値再設定手段
7
変数値設定手段
変数値格納手段
2
9
10
関数定義保持手段
関数定義実行手段
11
データベース
12
システムリソース

[図11]


何件目からの表示かを示す情報
何件分を表示するかを示す情報
番号
名前
日付
内容
…(1)
番号
名前
日付
内容
…(2)
番号
名前
日付
内容
…(n)
bbs data
HTMLブラウザ上

[図12(a)]

```
struct {
        date wdate;
        string name;
        string message;
} bbsdata[];

int start_count = 0;
int list_count = 0;

struct {
        string wdate;
        string name;
        string html;
} output_block[];

string output_html;
```

(以上、値保持手段への定義と格納データ)

```
void bbs_additem( string input_data, string input_name, string input_message )
{
        int t = vsize(bbsdata);

        bbsdata[t].date     = input_date[0..3] & "/" &
                              input_date[5..6] & "/" & input_date[8..9];
        bbsdata[t].name     = input_name;
        bbsdata[t].message = input_message;
        return;
}
```

(以上、関数保持手段へ格納する関数定義)

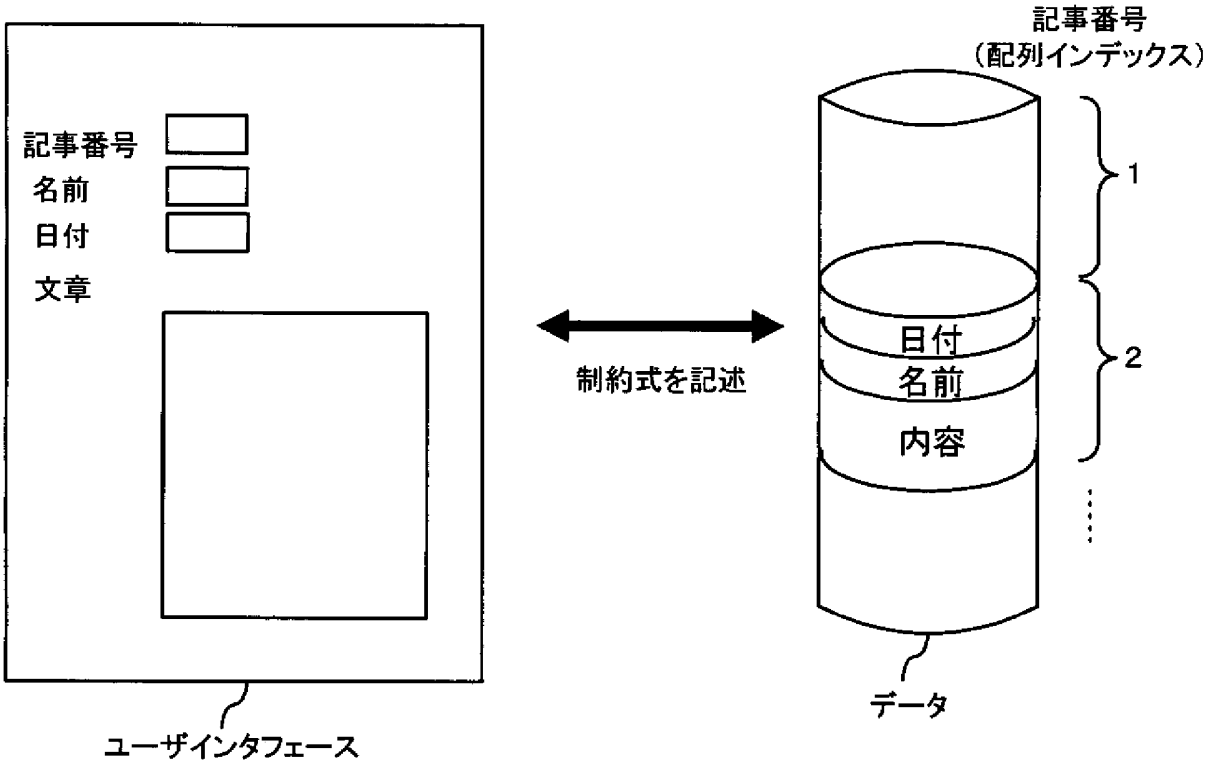
[図12(b)]

```
{ vsize(output_block) = list_count; };

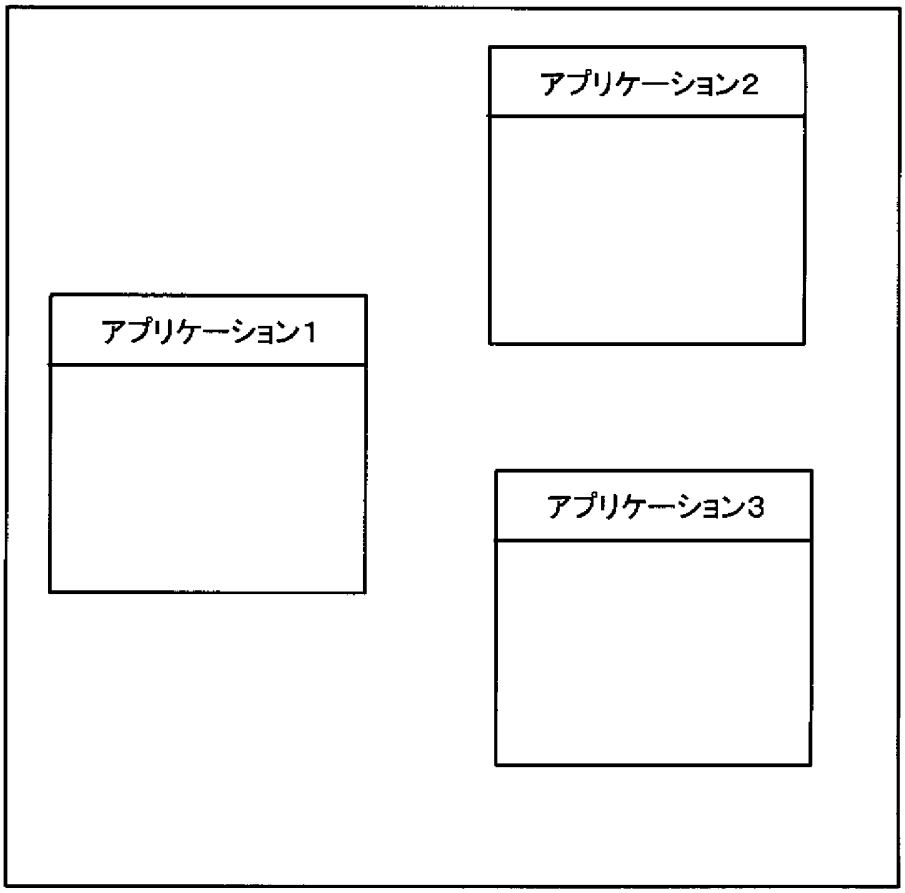
{ output_block[$i].wdate[0..3] = bbsdata[$i+start_count].wdate[0..3];
  output_block[$i].wdate[4..5] = "年";
  output_block[$i].wdate[6..7] = bbsdata[$i+start_count].wdate[5..6];
  output_block[$i].wdate[8..9] = "月";
  output_block[$i].wdate[10..11] = bbsdata[$i+start_count].wdate[8..9];
  output_block[$i].wdate[12..13] = "日"; };
{ output_block[$i].name = strtrunc( bbsdata[$i+start_count].name, 20); };
{ output_block[$i].html = "記事番号" & inttostring($i+start_count) &
        "日付" & output_block[$i].wdate & "<BR>¥n" &
        "名前" & output_block[$i].name & "<BR>¥n" &
        "メッセージ<BR>¥n" & bbsdata[$i+start_count].message & "<BR><BR><BR>¥n"; };
{ output_html = "<HTML><BODY>¥n" &
        "  " & inttostring(start_count) & "  " &
        inttostring(list_count) & "<BR><BR>¥n" &
        output_block[*].output_html & "</BODY></HTML>"; };
```

(以上、制約保持手段へ格納する制約式)

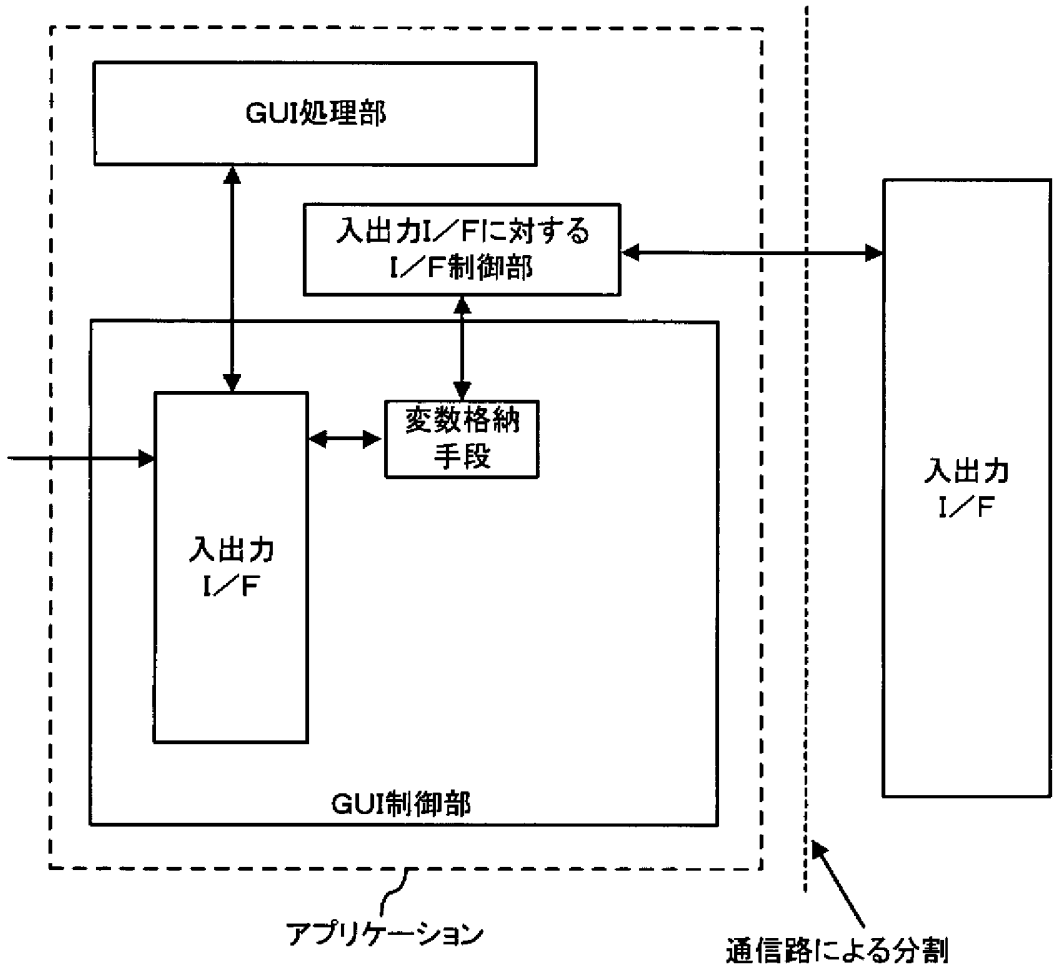
[図13]


記事番号
名前
日付
文章
ユーザインタフェース
制約式を記述
記事番号
（配列インデックス）
1
2
日付
名前
内容
データ

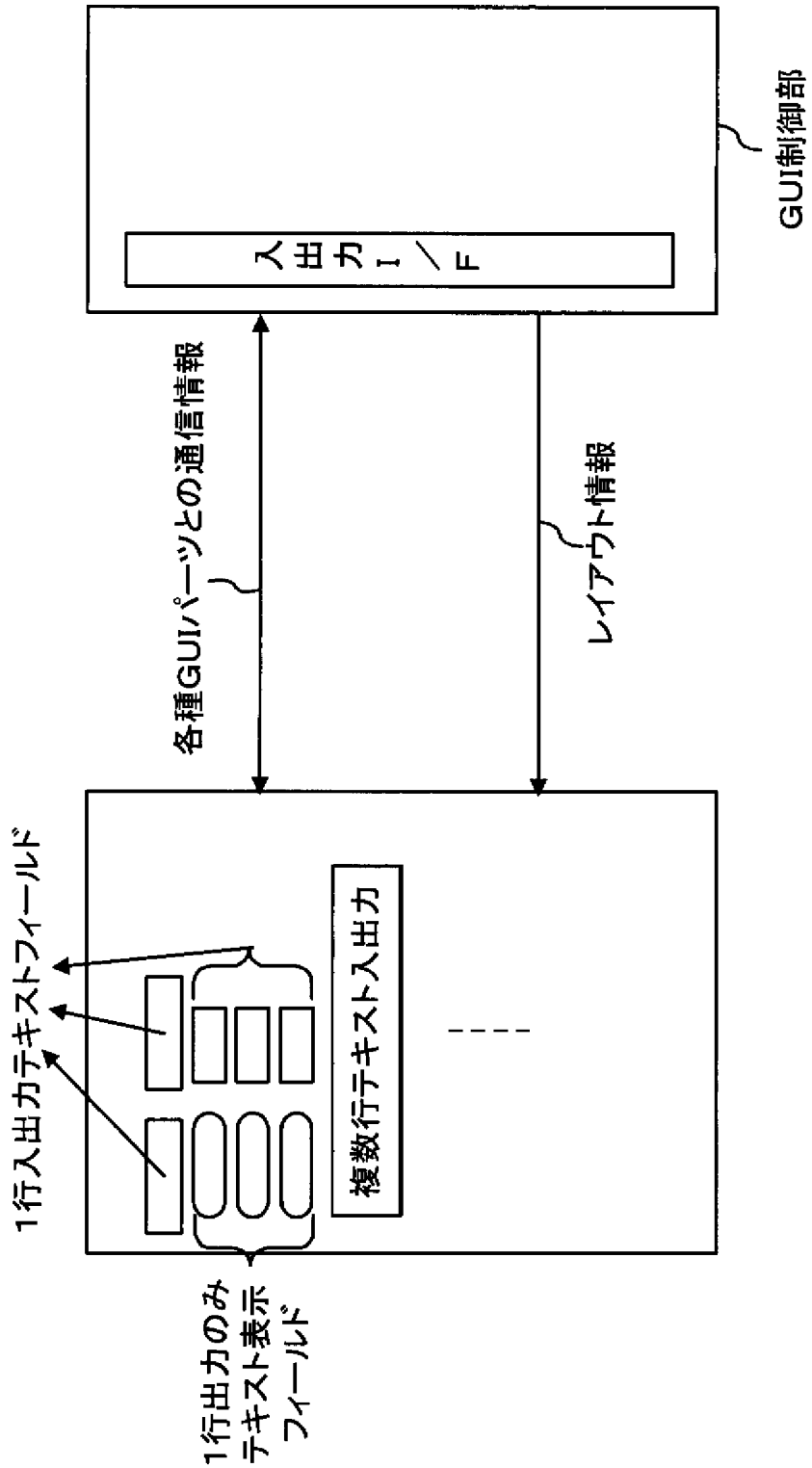
[図14]


アプリケーション2
アプリケーション1
アプリケーション3

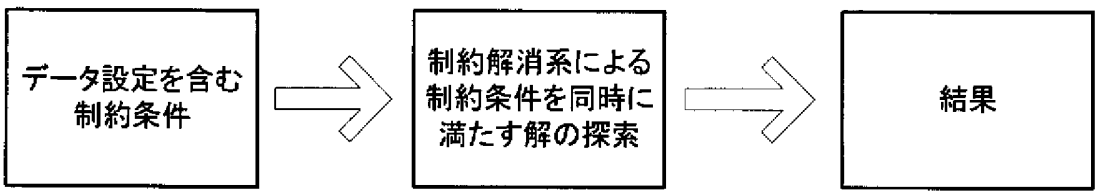
[図15]


GUI処理部
入出力I／Fに対する
I／F制御部
変数格納
手段
入出力
I／F
入出力
I／F
GUI制御部
アプリケーション
通信路による分割

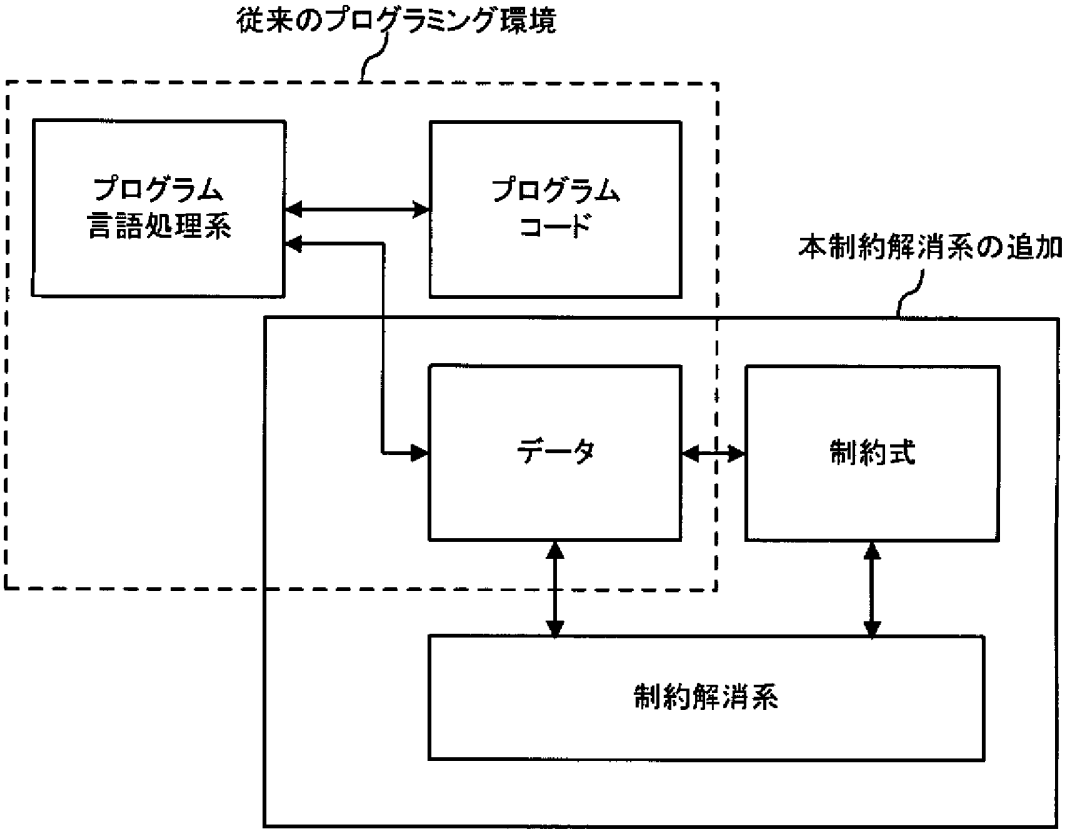
[図16]


GUI制御部
入出力I／F
各種GUIパーツとの通信情報
レイアウト情報
1行入出力テキストフィールド
複数行テキスト入出力
1行出力のみテキスト表示フィールド

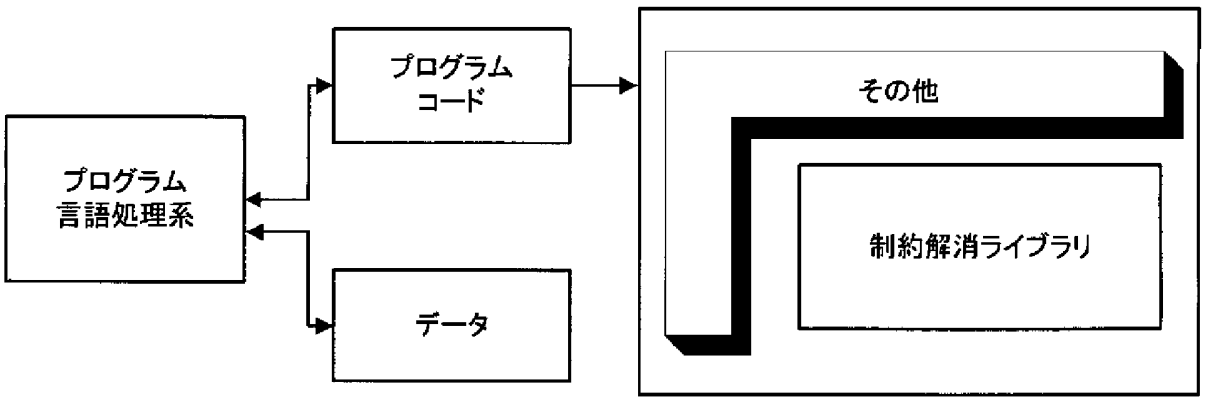
[図17]


データ設定を含む制約条件
制約解消系による制約条件を同時に満たす解の探索
結果

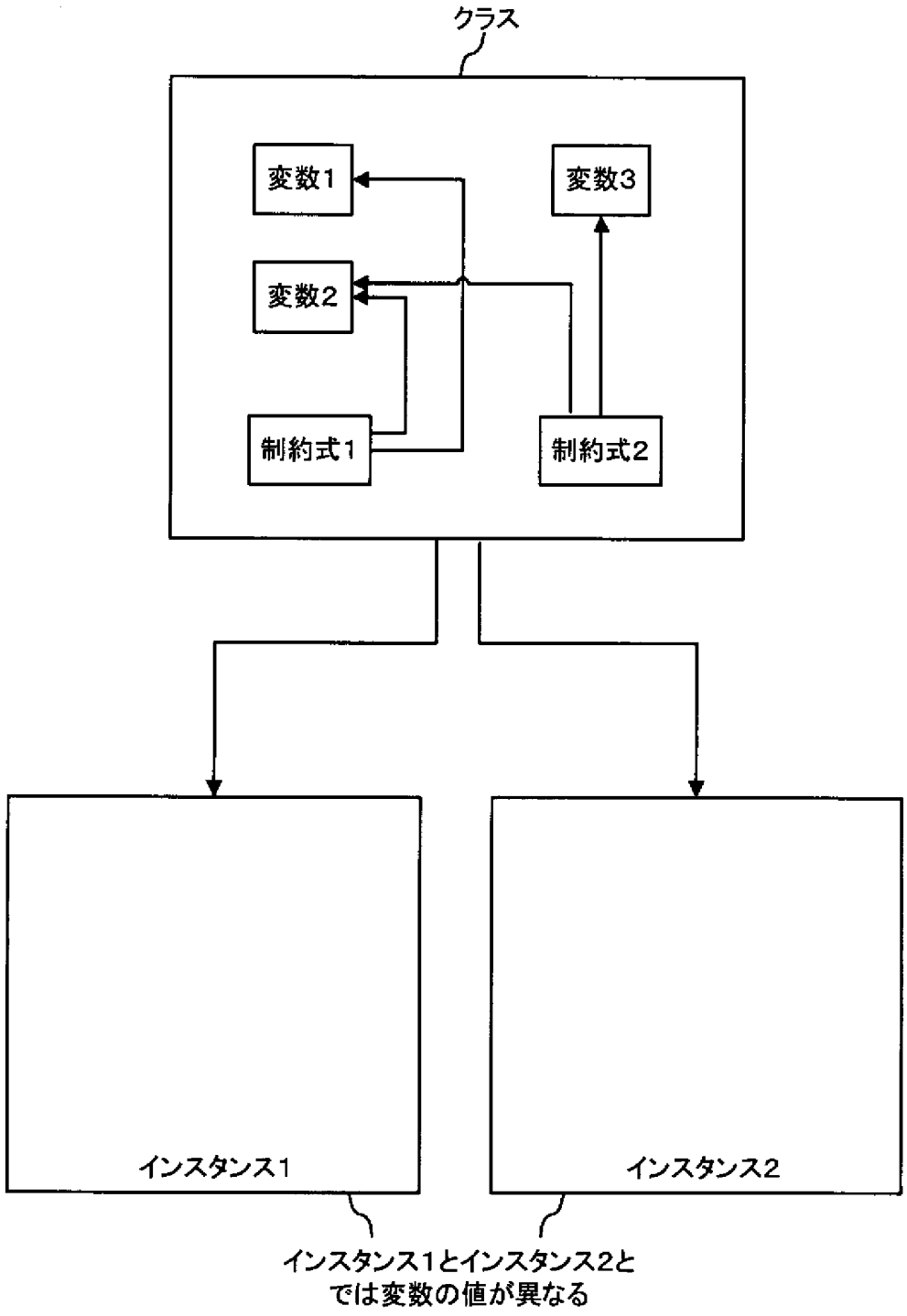
[図18]


従来のプログラミング環境
プログラム
言語処理系
プログラム
コード
本制約解消系の追加
データ
制約式
制約解消系

[図19]


プログラム
コード
プログラム
言語処理系
データ
その他
制約解消ライブラリ

[図20]


クラス
変数1
変数3
変数2
制約式1
制約式2
インスタンス1
インスタンス2
インスタンス1とインスタンス2と
では変数の値が異なる

[図21]

配列要素範囲選択演算子の例

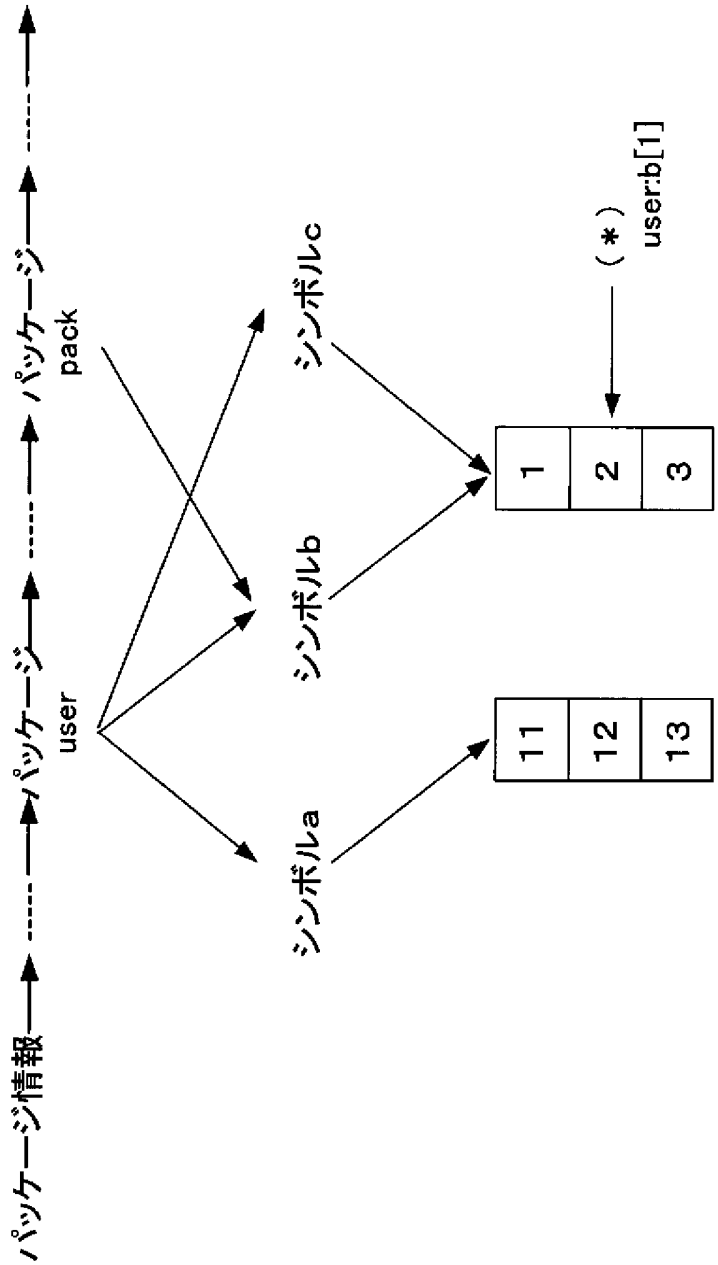
[n..m]	... 配列内の n～m 番目の要素を選択
[*]	... 配列内のすべての要素を選択
[$]	... 配列要素の最後
[#]	... 可変長配列要素の最後 + 1
[!<式>]	... 配列内の指定した値以外の要素を選択
[<式> && <式>]	... 指定要素の and 条件
[<式> || <式>]	... 指定要素の or 条件
[.=y]	... 配列内の要素が y である要素を検索して選択
[x=y]	... 構造体配列のメンバ x の値が y である要素を前から検索して選択
[x(n)=y]	... 構造体配列のメンバ x の値が y である要素を前から検索して n 番目を選択
[x(-n)=y]	... 構造体配列のメンバ x の値が y である要素を後から検索して n 番目を選択
[x(*)=y]	... 構造体配列のメンバ x の値が y である要素を検索してすべて選択
["正規表現"]	... 文字列配列の要素を正規表現で検索して選択
[x="正規表現"]	... 構造体配列のメンバ x の文字列要素を正規表現で検索して配列要素を選択

配列操作の例

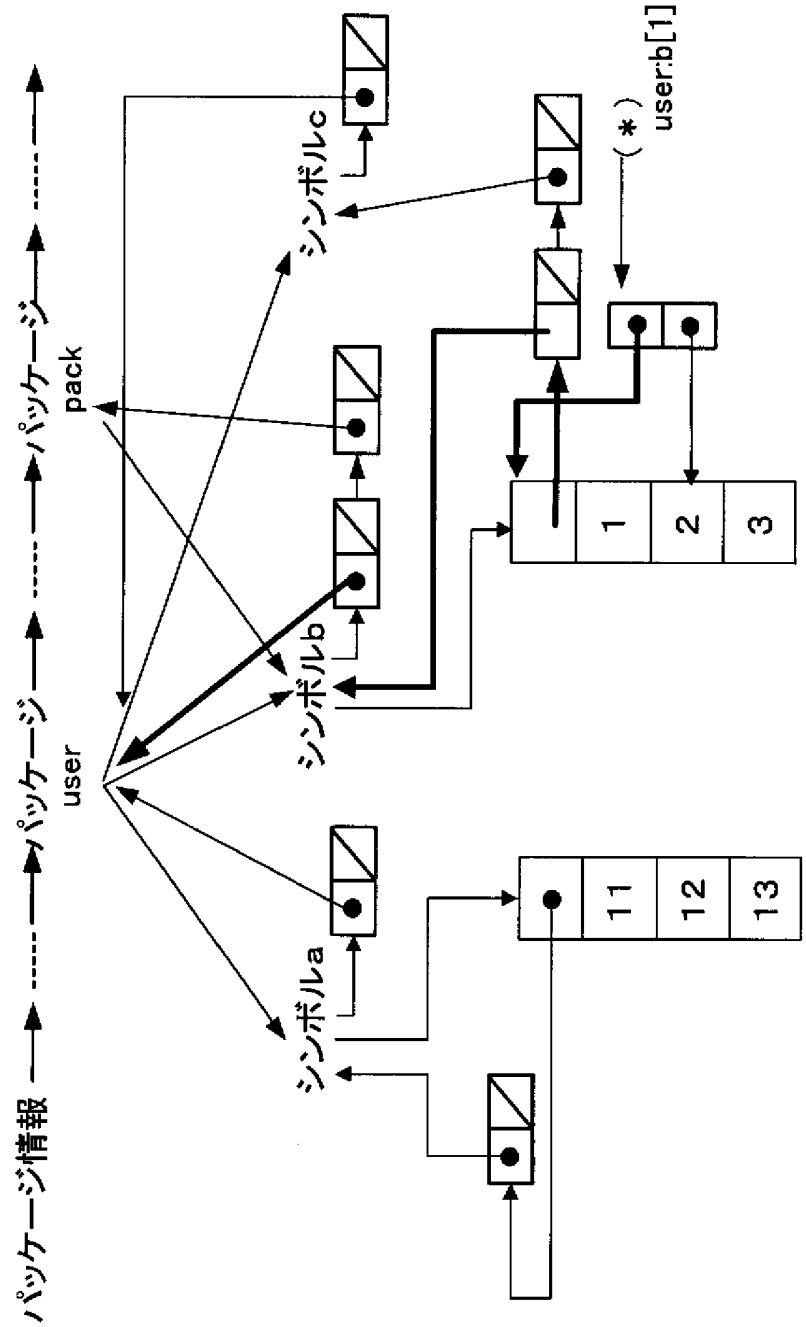
v[5]	... 配列 v 内の要素の 5 番目を表す
v[!n]	... 配列 v 内の n 番目以外の要素を持つ配列を表す
v[0..3 || 5..$]	... 配列 v 内の 4 番目以外の要素を持つ配列を表す
s["[ab]ab*" || "[cd]cd*"]	... 文字列 s 内の正規表現がマッチする内容を表す
v[append(0,1,2,5..$)]	... 配列 v 内の要素 0,1,2,5..$をすべて合わせたものを表すが、 演算方法として一般関数 append()を使用する。 append 関数はプログラミング言語 LISP で定義されるものと同様の意味をもつ

* 上記の演算子は一例です。
* 上記の演算子は一般関数を組み合わせて自由に混在できます。

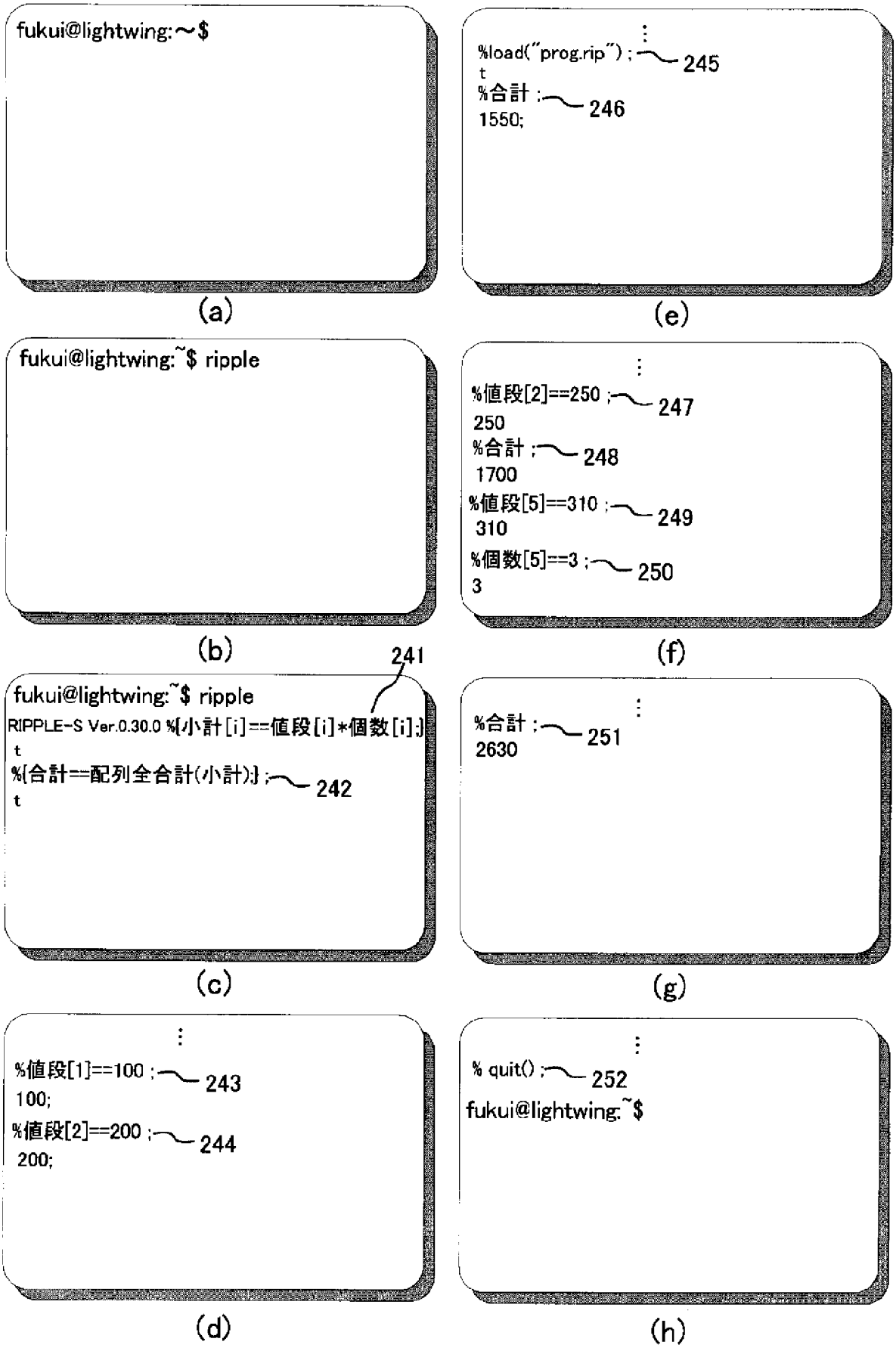
[図22]


パッケージ情報
パッケージ
パッケージ
パッケージ
pack
user
シンボルa
シンボルb
シンボルc
11
12
13
1
2
3
(*)
user:b[1]

[図23]


パッケージ情報
パッケージ
パッケージ
パッケージ
user
pack
シンボルa
シンボルb
シンボルc
user:b[1]
(*)
1
2
3
11
12
13

[図24(a)-(h)]


fukui@lightwing:~$
(a)
fukui@lightwing:~$ ripple
(b)
241
fukui@lightwing:~$ ripple
RIPPLE-S Ver.0.30.0 %{小計[i]==値段[i]*個数[i];}
t
%{合計==配列全合計(小計);} ;
242
t
(c)
%値段[1]==100 ;
243
100;
%値段[2]==200 ;
244
200;
(d)
%load("prog.rip") ;
245
t
%合計 ;
246
1550;
(e)
%値段[2]==250 ;
247
250
%合計 ;
248
1700
%値段[5]==310 ;
249
310
%個数[5]==3 ;
250
3
(f)
%合計 ;
251
2630
(g)
% quit() ;
252
fukui@lightwing:~$
(h)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2005/000752

A. CLASSIFICATION OF SUBJECT MATTER
Int.Cl[7] G06N5/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
Int.Cl[7] G06N5/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Jitsuyo Shinan Koho 1922-1996 Jitsuyo Shinan Toroku Koho 1996-2005
Kokai Jitsuyo Shinan Koho 1971-2005 Toroku Jitsuyo Shinan Koho 1994-2005

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	JP 5-165900 A (Hitachi, Ltd.), 02 July, 1993 (02.07.93), Par. Nos. [0032] to [0042]; Fig. 6 (Family: none)	1,2,8,9,15 3-6,10-13
Y	JP 2003-149326 A (Hitachi, Ltd.), 21 May, 2003 (21.05.03), Par. Nos. [0042] to [0046] (Family: none)	3,10
Y	JP 2001-243056 A (NEC Corp.), 07 September, 2001 (07.09.01), Par. Nos. [0031] to [0037]; Figs. 7 to 8 (Family: none)	4-6,11-13

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 April, 2005 (21.04.05)	17 May, 2005 (17.05.05)
Name and mailing address of the ISA/ Japanese Patent Office Facsimile No.	Authorized officer Telephone No.

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2005/000752

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	JP 2002-196927 A (Kabushiki Kaisha Aizakku), 12 July, 2002 (12.07.02), Par. Nos. [0020] to [0021]; Fig. 1 (Family: none)	5,12
X Y	JP 7-175844 A (Chugoku Nihon Denki Software Kabushiki Kaish), 14 July, 1995 (14.07.95), Par. Nos. [0002] to [0012]; Fig. 1 (Family: none)	1,8,15 2-6,9-13

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2005/000752

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

The search has revealed that the matter common to claims 1-15 is not novel. Moreover, claims 1-6, 8-13, and 15 include variable resetting means (process) but claims 7 and 14 do not include this matter. Accordingly, between claims 1-6, 8-13, and 15 and claims 7 and 14, there exists no common matter which can be considered as a special technical feature within the meaning of PCT Rule 13.2, second sentence. Consequently, no technical relationship within the meaning of PCT Rule 13 between these inventions can be seen. Therefore, claims 1-6, 8-13, and 15 and claims 7 and 14 do not satisfy the requirement of unit of invention.

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.: 1-6, 8-13, 15

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

国際調査報告

国際出願番号　ＰＣＴ／ＪＰ２００５／０００７５２

Ａ．　発明の属する分野の分類（国際特許分類（ＩＰＣ））

Int.Cl.[7]　G06N5/00

Ｂ．　調査を行った分野

調査を行った最小限資料（国際特許分類（ＩＰＣ））

Int.Cl.[7]　G06N5/00

最小限資料以外の資料で調査を行った分野に含まれるもの

日本国実用新案公報	１９２２－１９９６年
日本国公開実用新案公報	１９７１－２００５年
日本国実用新案登録公報	１９９６－２００５年
日本国登録実用新案公報	１９９４－２００５年

国際調査で使用した電子データベース（データベースの名称、調査に使用した用語）

Ｃ．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求の範囲の番号
Ｘ	ＪＰ　５－１６５９００　Ａ　（株式会社日立製作所） １９９３．０７．０２，段落【００３２】－【００４２】，図６ （ファミリーなし）	１，２，８，９，１５
Ｙ		３－６，１０－１３

☑　Ｃ欄の続きにも文献が列挙されている。

☐　パテントファミリーに関する別紙を参照。

＊　引用文献のカテゴリー

「Ａ」特に関連のある文献ではなく、一般的技術水準を示すもの

「Ｅ」国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの

「Ｌ」優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献（理由を付す）

「Ｏ」口頭による開示、使用、展示等に言及する文献

「Ｐ」国際出願日前で、かつ優先権の主張の基礎となる出願の日の後に公表された文献

「Ｔ」国際出願日又は優先日後に公表された文献であって出願と矛盾するものではなく、発明の原理又は理論の理解のために引用するもの

「Ｘ」特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの

「Ｙ」特に関連のある文献であって、当該文献と他の１以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの

「＆」同一パテントファミリー文献

国際調査を完了した日 ２１．０４．２００５	国際調査報告の発送日　17.05.2005
国際調査機関の名称及びあて先 日本国特許庁（ＩＳＡ／ＪＰ） 郵便番号１００－８９１５ 東京都千代田区霞が関三丁目４番３号	特許庁審査官（権限のある職員）　５Ｂ　３１４５ 赤穂　州一郎 電話番号　０３－３５８１－１１０１　内線　３５４６

Ｃ（続き）．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求の範囲の番号
Ｙ	ＪＰ　２００３－１４９３２６　Ａ　（株式会社日立製作所） ２００３．０５．２１，段落【００４２】－【００４６】 （ファミリーなし）	３，１０
Ｙ	ＪＰ　２００１－２４３０５６　Ａ　（日本電気株式会社） ２００１．０９．０７，段落【００３１】－【００３７】，図７－８ （ファミリーなし）	４－６， １１－１３
Ｙ	ＪＰ　２００２－１９６９２７　Ａ　（株式会社アイザック） ２００２．０７．１２，段落【００２０】－【００２１】，図１ （ファミリーなし）	５，１２
Ｘ	ＪＰ　７－１７５８４４　Ａ（中国日本電気ソフトウェア株式会社） １９９５．０７．１４，段落【０００２】－【００１２】，図１ （ファミリーなし）	１，８，１５
Ｙ		２－６， ９－１３

国際調査報告

国際出願番号　ＰＣＴ／ＪＰ２００５／０００７５２

第Ⅱ欄　請求の範囲の一部の調査ができないときの意見（第１ページの２の続き）

法第8条第3項（ＰＣＴ17条(2)(a)）の規定により、この国際調査報告は次の理由により請求の範囲の一部について作成しなかった。

1．☐ 請求の範囲＿＿＿＿＿＿＿＿ は、この国際調査機関が調査をすることを要しない対象に係るものである。
つまり、

2．☐ 請求の範囲＿＿＿＿＿＿＿＿ は、有意義な国際調査をすることができる程度まで所定の要件を満たしていない国際出願の部分に係るものである。つまり、

3．☐ 請求の範囲＿＿＿＿＿＿＿＿ は、従属請求の範囲であってＰＣＴ規則6.4(a)の第２文及び第3文の規定に従って記載されていない。

第Ⅲ欄　発明の単一性が欠如しているときの意見（第１ページの３の続き）

次に述べるようにこの国際出願に二以上の発明があるとこの国際調査機関は認めた。

先行技術調査の結果、請求項１－１５に共通の事項は、新規でないことは明らかとなった。また、請求の範囲１－６、８－１３及び１５は、変数再設定手段（処理）を有しているが、請求の範囲７及び１４は、これらの事項を有しておらず、よって、請求の範囲１－６、８－１３及び１５と請求の範囲７及び１４との間には、ＰＣＴ規則１３．２の第２文の意味において特別な技術的特徴と考えられる共通の事項は存在しないので、これらの発明の間にＰＣＴ規則１３の意味における技術的な関連を見いだすことはできない。よって、請求の範囲１－６、８－１３及び１５と請求の範囲７及び１４とは発明の単一性の要件を満たしていない。

1．☐ 出願人が必要な追加調査手数料をすべて期間内に納付したので、この国際調査報告は、すべての調査可能な請求の範囲について作成した。

2．☐ 追加調査手数料を要求するまでもなく、すべての調査可能な請求の範囲について調査することができたので、追加調査手数料の納付を求めなかった。

3．☐ 出願人が必要な追加調査手数料を一部のみしか期間内に納付しなかったので、この国際調査報告は、手数料の納付のあった次の請求の範囲のみについて作成した。

4．☑ 出願人が必要な追加調査手数料を期間内に納付しなかったので、この国際調査報告は、請求の範囲の最初に記載されている発明に係る次の請求の範囲について作成した。

１－６，８－１３，１５

追加調査手数料の異議の申立てに関する注意

☐ 追加調査手数料の納付と共に出願人から異議申立てがあった。

☐ 追加調査手数料の納付と共に出願人から異議申立てがなかった。