制約型プログラミングによるオフィス処理の実現

石井 恵[†,☆] 金田重郎[†,☆☆]

事務処理システムは，従来，手続き型言語，テーブルドリブン記述，トリガといった，状態遷移を表現する言語により記述されてきた．しかし，この種の状態遷移記述では，(1) 事務処理に必要な手続きをすべて網羅的に書き切ることが難しい，(2) 事務規則と整合性のとれた処理結果が生成されることをプログラム記述上で確認することが難しい，等の問題を有していた．上記問題を解決するため，本論文では，事務処理を，「事務処理で扱うデータを事務処理規則と整合する状態に維持する整合性管理処理」とモデル化する．そして，それを具現化する，制約プログラミングによる整合性管理システムを提案する．具体的には，事務規則を制約として記述し，制約充足エンジンによって，事務規則と整合するデータ状態を生成する．本手法を実用事務処理システムのタスクに試用した結果，事務規則を容易に制約記述でき，プログラム記述量を半減できた．また，手続き型記述の実用システムでは十分には対応できなかったレアケースに対しても対処可能となった．

A Constraint Programming Approach for Office Systems

MEGUMI ISHII[†,☆] and SHIGEO KANEDA[†,☆☆]

This paper presents a constraint satisfaction mechanism for office systems. Regulations for business can be regarded as constraints in terms of the data in business databases and application forms. Clerical work can be regarded as consistency maintenance to keep the data consistent with the constraints. We rewrote an office system using constraints. Experimental results show that all the regulations can be easily described in constraints, the description size is reduced by 50% of the original system and the constraints cover rare cases exhaustively.

1. はじめに

帳票処理を中心とした事務処理は企業・行政機関等に広く存在し，社会生活に不可欠である．そして，その正確さの向上，処理の効率化を目指し，種々の事務処理システムの開発が行われている．

これら事務処理システムの構築にあたっては，従来，手続き型言語，テーブルドリブン記述，トリガといった，状態遷移を表現する言語が記述言語として利用されてきた[1)~5)]．これらのアプローチは，procedure-automation アプローチと呼ばれ，情報の流れに注目しており，実際の業務担当者が行っている手続きをそのままコンピュータ上にインプリメントできる特長がある．

反面，このアプローチにより作成されたシステムは推論機能を持たないため，業務に必要な具体的な処理をすべて記述しなければならない．その結果，以下の問題を生じ，事務処理システムの開発・維持管理を困難とする一因となっていた．

- 事務処理で行われている手続きを網羅的に記述することは難しく，その結果，処理洩れが往々にして起こる．
- 記述された手続きから，最終的に得られる処理結果を読み取ることは困難であり，その結果，生成された結果が業務規則に合致しているか否かの確認が難しい．

上記の問題点は，現実に行われている表面的な処理手順を中心に業務をとらえている点に起因する，と我々は考える．そこで，我々は，従来と異なった見方で事務処理をとらえ直す．

通常，事務処理には，その背後に事務規則が存在する．そして，事務処理はその事務規則に従って行われている．すなわち，事務処理を実現するのに必要な手続きは事務規則から生成されており，事務処理の本質は事務規則にある，と考える．

† NTT コミュニケーション科学研究所
NTT Communication Science Laboratories
☆ 現在，NTT 情報通信研究所
Presently with NTT Information and Communication Systems Laboratories
☆☆ 現在，同志社大学大学院総合政策科学研究科・同志社大学工学部
Presently with Graduate School of Policy and Management/Faculty of Engineering, Doshisha University

そこで，本論文では，事務処理の本質を規定する事務規則に注目し，事務規則を「事務処理が扱うデータ間の関係を規定する制約」として解釈し，事務処理を，「事務規則と事務処理が扱うデータ間をつねに整合状態に維持する整合性管理処理」としてモデル化する．そして，制約充足エンジンによる制約充足により，事務規則を制約として記述するのみで事務処理を実現できるプログラミング手法を提案する．

本手法は，整合性管理処理の実現に有効な制約プログラミング手法[6]の一種であるが，制約プログラミング手法の新たな適用領域として，事務処理分野に着目し，事務処理に適した処理モデルと言語仕様を提示するものである．事務処理への制約プログラミング手法の導入により，以下の利点が得られる．

- 制約表現により事務規則を記述するため，事務規則の宣言性が保存され，事務規則の本質を自然に，明確に記述することができる．その結果，処理洩れを抑制でき，従来忘れがちなレアケースに対する洩れのない対応が可能となる．
- 制約充足エンジンによる制約の充足により，与えられた制約と整合した処理結果の生成を保証する．すなわち，事務規則に整合した処理結果が保証される．
- 処理結果を制約として表現しているので，最終的に生成される処理結果（データの状態）を明示的に示すことができ，プログラミングが容易である．

本手法は，前記 procedure-automation アプローチに対して，業務担当者の専門知識がシステムに与えられ，業務担当者が行っているタスクを，システムが解くべき問題として明確に与える，problem-solving アプローチと呼ばれるアプローチである．このアプローチでは，システムは与えられた知識と推論機能を用いて，与えられた問題を解く．推論機能により，大局的な視点でシステムは問題を解くことができ，問題を解くための具体的な処理の記述は不要である．

本手法のほかに，problem-solving アプローチに基づく非定型的な事務処理を支援するシステムとして，Odyssey[7]，また，非定形型的な事務処理を支援するシステムを記述する言語として OMEGA[8]がある．本手法とこれら両手法との大きな違いは，事務規則を制約として解釈し，事務処理を整合性管理処理ととらえ，事務処理支援システムの構築手法の指針を与え得る点である．

以下，2 章では事務規則の制約の解釈方法を示し，事務処理を整合性管理としてモデル化する．3 章では，制約プログラミングを事務処理へ適用する際の課題を述べた後，事務処理を実現する整合性管理システムの構成，および，制約シンタックスおよび制約充足エンジンを提案する．4 章では実用アプリケーションである総務業務エキスパートシステム KOA を，実験的に制約記述した結果を示す．5 章では関連研究について述べ，6 章ではまとめを示す．

2. 事務処理と整合性管理処理

本章では，事務規則の制約としての解釈方法，および，事務処理の整合性管理処理として見なすモデルを示す．

2.1 制約としての事務規則

我々のアプローチはまず，事務処理の本質は事務規則にあると考え，事務規則に注目することから始まる．ここでは，事務規則が表す意味について考える．

たとえば，事務規則「年収 130 万円未満の家族がある社員は，扶養申請書を提出し，3000 円/月の扶養手当が支給される.」を例として考える．この規則は本質的には，**図 1** に示す 3 つの状態を意味すると考えられる．各状態は，申請書，家族の存在，扶養手当，家族の収入の関係を表している．図 1 の記述は NTT 社内における実際の給与規定の表現方法に準じている（ただし，簡単化してあり金額等も変えてある）．我々は，これらの業務知識が，状態を変化させるための手続きとしてではなく，処理順序を規定しない宣言的な文章で記述されていることに着目する．

この規則に従う事務処理では，社員のデータ状態は図 1 に示す 3 つの状態のどれかに当てはまらなければならない．なぜなら，上記 3 つにあてはまらない社員のデータ状態（例：家族の収入が 130 万円以上で，扶養手当を貰っている状態）は事務規則に反し，現実に存在してはならないからである．よって，事務規則は，その規則が許すデータ状態を規定する制約として解釈することができる．すべての事務規則が正しく適

扶養手当規則

状態(a)：家族の年収が１３０万円以下の社員には，
扶養手当が月々３０００円支給され，
扶養手当支給申請書が提出されている．

状態(b)：家族の年収が１３０万円を超える社員には，
扶養手当は支給されず，扶養手当支給申請
書は提出されていない．

状態(c)：家族がいない社員には，扶養手当は支給さ
れず，扶養手当支給申請書は提出されてい
ない．

図 1 事務規則の例
Fig. 1 An example of a regulation for business.

用されている状態が，制約充足であり，このとき，各制約中では，宣言されているデータ状態のどれか1つのデータ状態と，処理対象としているデータ状態が整合している．

もちろん，事務規則を手続きとしてプログラム化することは可能である．しかし，このことは，1章で述べた問題のほかに，事務規則の適用方向の制限という，もう1つの問題を発生させる．事務規則を手続きとして表した場合，規則の適用方向ごとに手続きを用意しなければならない．その結果，各事務規則に対して多くの手続きを記述する必要がある．

たとえば，図1の状態(a)は，(1)家族の年収が130万以下で，扶養手当3000円/月の支給を受けていれば，扶養手当の申請書が提出されている，(2)扶養手当の申請書が提出されており，家族の収入が130万以下であれば，扶養手当3000円/月を支給する，(3)扶養手当の申請書が提出され，扶養手当3000円/月が支給されていれば，収入は130万以下とする，という3つの方向へ解釈できる．よって，状態(a)に対しては，各解釈方向に対する3つの手続きが必要となる．同様に状態(b)，状態(c)各々に対しても手続きを用意しなければならない．一方，事務規則を制約として表現した場合，1章で述べた利点以外に，解釈の方向の柔軟性を事務規則に与えることができる．よって，我々は事務規則を制約として表現するアプローチをとる．

2.2 整合性管理処理としての事務処理

事務処理が行っている本質的なタスクについて考える．たとえば，家族（配偶者）の年収が200万円から70万円に減ったので扶養手当を申請する場合について考える．申請前の社員は，家族の収入が多いので，扶養手当は支給されていない．この状態は，図1の状態(b)に整合しており，事務規則が許す状態である．その結果，以前の家族年収においては，社員のデータ状態は事務規則と整合している．

以下の手続きは，社員は，社員の名前，家族の名前，家族の収入が記入された扶養手当申請書を作成し，業務担当者に提出した場合，業務担当者が行う事務処理の例である．

(1)社員情報（データベース）の家族（配偶者）の年収を200万円から，申請書に記入されている，家族の年収の金額70万円に変更する．

(2)扶養手当のデータを3000円に変更し，扶養手当の申請処理を完了する．

(1)，(2)の手続きの後，社員のデータ状態は図1の状態(a)と整合した状態となり，再度，事務規則と整合した状態となる．すなわち，業務担当者は，社員のデータ状態を，処理要求を反映し，かつ，事務規則と整合した状態にするため，上記の手続きを行ったのである．よって，事務処理は「事務処理で扱うデータを，処理要求に応じた手続きを行い，つねに事務処理規則と整合した状態に維持管理する整合性管理処理」と見なすことができる．

3. 整合性管理システム

本章では，前述の整合性管理モデルに基づく，整合性管理システムを提案する．本手法では，制約プログラミングの適用により，整合性管理システムを実現する．この際，(1)制約を表す制約シンタックス，(2)制約充足解の定義が必要となる．以下，本整合性管理システムの構成を述べた後，事務処理と親和性の高い制約シンタックス，制約充足解の定義，および前記解を求める制約充足エンジンを提案する．

3.1 システム構成

本システムでは，整合性管理の対象となる事務処理上のデータの構造をフレーム構造と仮定し，フレーム名とスロット名の対により，各データを識別する☆．以後，説明を簡単にするために，前記フレーム名とスロット名の対により識別されるデータ項目を**ユニット**と呼ぶ．ユニット間でつねに成り立っていなければならない関係を**制約**と呼ぶ．**図2**は，(1)ユーザの要求☆☆を反映し，かつ，(2)制約を充足するように，ユニットの状態を管理する整合性管理システムの構成を示す．本提案の整合性管理システムは，以下のブロックから構成される．

ユーザインタフェース：ユーザとのインタラクションを行うブロックである．具体的には，ユーザの処理要求の受信，ユーザへの質問の出力，および，質問に対するユーザからの応答の受信を行う．

ワーキングメモリ：後述するシンタックスで表現された現在のユニットの状態を格納する．

制約ベース：後述のシンタックスで表現されたユニット相互の制約関係を格納する．

対話的制約充足エンジン：必要なデータをユーザから取得しながら，充足状態を生成する．ワーキングメモリ上のユニットの状態が制約ベース中の制約を充足しているか否かをチェックする**非整合検出部**，ユニットの状態をユーザの要求および制約を

☆ データ構造をリレーショナルデータベースとすれば，整合性管理の対象となるデータは，リレーション名とそのリレーションを構成する属性名により識別される．

☆☆ 具体的には，前述の家族年収の収入のように，一部のデータ項目の値を変更することに相当する．

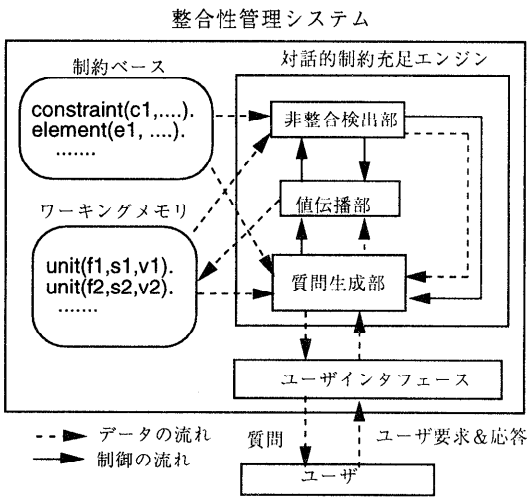


図2 整合性管理システム構成
Fig. 2 System block diagram.

充足した状態にするための質問を生成する**質問生成部**，ワーキングメモリに制約ベース中の制約を充足するユニットの値を書き込む**値伝播部**から構成される．

3.2 制約シンタックス

本節では事務規則を表現する制約シンタックスを提案する．本シンタックスでは事務規則との親和性を高くするために，制約素の概念と，制約素を表現するための制約変数を導入している．以下にユニット，制約，制約素のシンタックスを述べる．

(1) ユニット

`unit`(フレーム名, スロット名, 値).

ユニットは整合性管理の対象となる事務処理上のデータ項目を示す．各ユニットは，フレーム名とスロット名の対により識別される．ユニットはユニット相互でユニークでなければならない．上記式はユニットへの値の割付けを定義する．ユニットに割り付けられる値は，定数，または，定数が割り付けられていないことを示すシンボル「_」である．値が「_」となっているユニットに対しては，制約とのユニフィケーションにより，制約で宣言されている定数の割付けが可能である．

(2) 制約素

```
element(制約素名,
        (pattern(フレーム名, スロット名, 値),
         ···,
         test(Boolean 関数),
         ···)).
```

制約素は，後述する制約を構成する要素であり，事務処理で扱うデータ項目に関して，事務規則が許す値の組合せ方を人間が整理しやすい単位で表現したものである．たとえば，2章で述べた扶養手当申請規則を制約と仮定すると，制約素は図1の(a)，(b)，(c)の各状態に対応する．

制約素名は制約素相互でユニークでなければならない．`pattern`は**パターン**と呼ばれ，ユニットに割り付けられるべき値を宣言する．パターン中の値は定数または制約変数をとる．制約変数は，Prologにおける変数と同様，大文字アルファベットから始まる．制約変数には，次節で提案する対話的制約充足エンジンが，制約が充足されているかを判定するために，ユニットとパターンのユニフィケーションを行うつど，ユニットの値が動的に割り付けられる．制約変数は各制約素内でのみ有効なローカル変数であり，制約変数に定数が割り付けられているとき，その制約変数は**バインド状態**にあるといい，同一制約素内ではすでに割り付けられている定数と異なる定数を割り付けることはできない．「_」が割り付けられている制約変数に対しては，定数の割付けが可能である．制約変数により，ユニットの個々の具体的な値を記述せずとも，まとめて複数項目間の関係が記述できる．

`test`は**テスト節**と呼ばれ，制約変数を介してユニットの値を制限するBoolean関数を表す☆．Boolcan関数は，Boolean関数中の制約変数がすべてバインド状態になった時点で評価される．制約素は，パターンとテスト節の積として解釈される．

上記，制約素，パターン，テスト節は**表1**に示す真偽値を持つ．これら真偽値は，次節で提案する対話的制約充足エンジンにより決定される．

(3) 制 約

```
constraint(制約名, (制約素名1,
                    制約素名2, ···)).
```

制約はユニット間でつねに成り立っていなければならない関係を表す．制約名は制約相互でユニークでなければならない．制約は，1つ以上の制約素から構成され，同一制約内の制約素相互は以下の条件を満足しなければならない．これは，事務規則を1つの制約と解釈した場合，制約記述者に対して，網羅的なデータ列挙を意識させ，処理抜けを抑制させるためである．

【制約素の排他条件】

制約中の制約素の1つがユニットの状態に整合しているなら，その制約中の他の制約素はそのユニットの状態と非整合となる． □

☆ 現状のエンジンは，SICStus prologにより実装されており，テスト節の中に記述できるBoolean関数は，prologの述語により定義できる関数である．

表1 真偽値
Table 1 Truth value table.

要素	ユニフィケーション結果／関数評価結果	真偽値
パタン	ユニフィケーション成功。	真
	ユニフィケーション失敗。	偽
テスト節	関数返却値が真。	真
	関数返却値が偽。	偽
	関数内に定数とユニフィケーションしていない制約変数が存在。	未定
制約素	制約素中の全てのパタン、テスト節が真であり、制約素中のユニットの値は全て変数ではない。	真
	制約素中の少なくとも1つのパタンまたはテスト節が偽。	偽
	その他。	未定
制約	制約中のただ1つの制約素のみ真であり、他の制約素は全て偽。	真
	制約中の全ての制約素が偽	偽
	その他。	未定

ここで，制約素がユニットの状態に整合するとは，制約素の真偽値が与えられたユニットの状態に対して「真」となるとき，すなわち，ユニットに割り付けられている値が制約素で宣言されているとおりとなっていることであり，それ以外の真偽値をとるとき，制約素はそのユニットの状態と非整合状態にある．

制約は，次節で提案する対話的制約充足エンジンにより決定される表1に示す真偽値を持つ．制約を構成する制約素のうち，ただ1つの制約素の真偽値が「真」であり，その制約を構成する残りの制約素の真偽値がすべて「偽」となるとき，制約の真偽値は「真」となり，このとき，制約は充足されている状態となる．すべての制約の真偽値が「真」のとき，**制約充足状態**である．このときのユニットの状態を充足解と呼ぶ．

図3 は図1に対応する扶養手当の規則を提案シンタックスで記述した例である．本シンタックスでは，値の決定の順序に関する記述はいっさいない．

3.3 対話的制約充足エンジン

事務処理結果を表す制約充足解の定義は，制約充足により事務処理を実現する際の課題となる．本論では，以下のように事務処理における制約充足解を定義する．

【事務処理における制約充足解の定義】

ユーザの処理要求を反映し，かつ，ユーザの処理要求が影響していないユニットに関しては，初期の充足状態（充足解）の値を保持した充足解を制約充足解とする☆． □

これは，事務処理を開始する際，過去のデータが存在し，処理を行う際，ユーザの処理要求が影響を与えない部分は過去のデータをそのまま流用する現実の事務処理を反映したものである☆☆．ここで，ユーザの処理要求を反映した充足解とは，ユーザの処理要求として変更されたユニットの値を解に含むものをいう．また，ユーザの処理要求が影響していないユニットとは，制約とユーザの処理要求により値が一意に決定されないユニットである．

```
constraint(扶養判断制約, (a1, a2, a3)).

element(a1, (pattern(家族, 存在, 有),
             pattern(家族, 収入, X),
             test(X =< 1300000),
             pattern(扶養申請書, 存在, 有),
             pattern(社員, 扶養手当, 3000))).

element(a2, (pattern(家族, 存在, 有),
             pattern(家族, 収入, X),
             test(X > 1300000),
             pattern(扶養申請書, 存在, 無),
             pattern(社員, 扶養手当, 0))).

element(a3, (pattern(家族, 存在, 無),
             pattern(扶養申請書, 存在, 無),
             pattern(社員, 扶養手当, 0))).
```

図3 制約の例
Fig. 3 An example of constraint description.

本エンジンは，ユーザの処理要求（すなわち，一部ユニットのデータ値の変更）により生じた制約非充足の状態を検知し，すべての制約を再充足することにより，最終的にユニットを制約と整合したデータ状態に変化させ，事務処理を実現する．この際，本エンジンは，非充足となっている制約を1個ずつ選択しては，この選択された制約を充足する．この逐次的な制約の充足は，すべての制約が充足されるまで繰り返され，本エンジンの大きな特徴である．

ユーザの処理要求が影響していないユニットに関して，初期の充足状態の値を変更しないと，ユーザの処理要求を反映した解が求まらない場合がある．これは，ユーザの要求が不完全であることを意味し，実際の事務処理では多々起こり得る．たとえば，ユニットが，図1の状態 (b) であったのに，ユーザの要求として，年収をゼロに下げられ，非充足状態が生じたとする．ここで，家族がいないとして，状態 (c) にして整合性をたもつべきか，あるいは，扶養手当と申請書を出して状態 (a) に遷移すべきかは，この年収の変更のみで

☆ ユーザの処理要求は制約に矛盾しないと仮定する．

☆☆ たとえば，住所変更申請書作成処理と改氏名届け作成処理は独立である．このような場合，住所変更申請書作成処理では，特にユーザが名前の変更を行わない限り，改氏名届け作成処理は行われない．

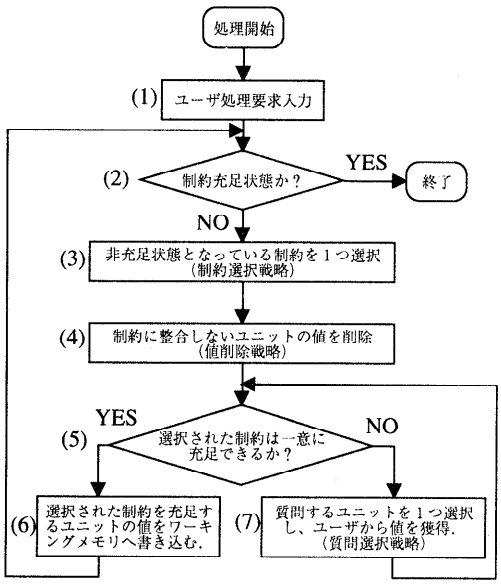


図4 対話的制約充足エンジン動作
Fig. 4 Algorithm of interactive constraint satisfaction engine.

は確定できない．本エンジンでは，ユーザの処理要求が影響していないユニットの値をユーザに質問することにより，前記ユーザの処理要求の不完全性を補う．

以下，本エンジンの動作を説明する．項目番号は，図4中の番号に相当する．

【対話的制約充足エンジン動作】

(1) ワーキングメモリ上のユニットが，制約ベース上の全制約を充足している状態において，エンジンは，ユーザからの処理要求を受け付ける．ユーザからの処理要求が，ユーザインタフェースから入力されると，ワーキングメモリ上の現在のユニット状態を，ユーザインタフェースから入力されたユーザの要求（ユニット値）に変更する．その結果，制約充足状態とは限らなくなる．

(2) 非整合検出部が全制約の真偽値が「真」となっているか否かを調べる．全制約の真偽値が「真」であるなら，制約充足状態であり，整合性管理処理は終了する．

(3) 真偽値が「偽」，または「未定」の制約を1つ選択する．真偽値が「偽」，または「未定」の制約は，ユーザの処理要求を反映した解を求めるために，ユーザの処理要求が影響していないユニットに関して，初期の充足状態の値を変更しなければならない可能性があることを示す．選択された制約をSCとする．

(4) エンジンはSCの非整合性を解消するために，SCに対して非整合となりうるユニットの値を削除する（「_」を割り付ける）．

(5) SCと整合するユニットの状態を一意に決定できるかを判断する．具体的には，SCの制約素のうち，ただ1つの制約素の真偽値が「未定」，SCの残りの制約素の真偽値が「偽」であり，真偽値が「未定」の制約素中の制約変数のうち，前記「未定」の制約素とユニフィケーションした結果，すべての制約変数がバインド状態であれば，ユニットの状態を一意に決定できる．この状態を決定できれば，(6)へいく．そうでなければ，(7)へいく．

(6) 値伝播部により，ワーキングメモリ上のユニットの値を，SCと整合するユニットの値へ変更する．具体的には，SC中の真偽値が「未定」となっている制約素とワーキングメモリ上のユニットの状態のユニフィケーションにより決定された値をユニットへ割り付ける．(2)へいく．

(7) 質問生成部により，値が「_」であるユニットをユーザに質問する．ユーザ入力された値は，ワーキングメモリ上のユニットに割り付けられる．(5)へいく． □

3.4 質問数を抑える戦略

本エンジンでは，前節で述べたように，ユーザの処理要求の不完全性を補うため，ユーザにユーザの処理要求が影響していないユニットの値を質問する．この際，ユーザへの負荷を抑えるため，エンジンは極力少ない質問数を生成することが望まれる．

このため，本エンジンでは以下の2つの質問戦略を導入している．1つは，ユーザの処理要求を反映しているユニットから値を決定できるユニットへは，質問を行わないことである．もう1つは，整合状態の決定に有効でないユニットの質問を行わないことである．

前者は制約の選択時に，ユーザの処理要求を反映したユニットの値のみから制約を充足するユニットの状態を一意に決定できる制約を優先的に選択することにより実現されている．後者は，ユニットの値の削除，ユニットの値の質問をする際，以下の処理により実現されている．ユニットの値を削除する際，選択された制約の中で，ユーザの処理要求と矛盾しない制約素の少なくとも1つから値を束縛されるユニットの値を削除する．これは，値の束縛がないユニットは，どんな値が割り付けられていても，その値を変更することなく選択された制約を充足することができるからである．ユニット質問時では，「真」となる制約素の決定に有効なユニットほど優先して質問を行う．これにより，「真」とならない制約素のみに関係するユニットに対する無駄な質問が削除される．

前記戦略の詳細を記述したエンジン動作アルゴリズムと，その動作例を付録に示す．

4. 評　　価

本手法の有効性を実証するため，NTT 社内で実用されている総務業務エキスパートシステム KOA[9]と同等の機能を制約を使って記述し，記述面，および，エンジンの動作を評価した．

4.1 総務業務エキスパートシステム KOA

KOA は，社員自身の身の上に生じた，「結婚しました」，「子供が生まれました」等の出来事を契機として，社員自身が予備知識・専門知識なしに，必要帳票を容易に作成可能とするシステムである．このため，システムは，総務担当者が行っていた，必要帳票の選択，記入項目の誘導とチェック，および，帳票内容のデータベースへの反映等の機能を持っている．

現在運用されている KOA は，総務担当者からのヒアリングに基づく手続き的なフローにより構築されており，社員とシステム間のインタラクションにより帳票が自動的に作成される．現在のところ，出力可能な帳票数は約 30 種，データ項目数は約 900 項目である．

4.2 評価結果

KOA の主要機能である，ユーザの要求に応じた帳票の選択，作成を行う機能を制約で記述した．ユーザインタフェースおよびデータベースアクセス機能は除いている．評価対象に用いた処理は，独身者の転入処理，既婚者の転入処理，結婚処理，出産処理の 4 処理である．可能な処理はこれ以外に多数あるが，利用頻度から見ると，上記 4 処理で約半分を占める．

以下に記述面，および，エンジンの動作結果を示す．

【開発およびシステム維持管理面での利点】

(1) 記述可能性： KOA が対象とする事務規則は，83 個の制約によってすべて記述できた．制約素の数は 274 個である．事務規則を表す制約の初期バージョンは，わずか 2〜3 日で作成できた．このことは，事務規則を制約で記述することは，宣言性を保存し，我々の制約シンタックスが制約と規則の間に高い親和性を有することを示す．つまり，我々の表現は従来の状態遷移表現よりも事務規則との表現ギャップが小さい．

(2) 記述量： 従来の KOA では，社員の身の上に生じた出来事ごとに，必要な帳票作成する手続きを C 言語で記述していた．具体的には，データ項目の値の更新のされ方に応じて，他のどのデータ項目をどのように更新するかという手続きが，各出来事を前提して記述されている．そのため，同じ帳票の作成に対して，出来事ごとに微妙に異なる手続きとなっていた．これに対し，本手法によるプログラムの記述量は従来の KOA に対して約 50%にまで削減された．これは，従来の KOA では，各事務規則を各出来事ごとに解釈した結果生成されていた手続きが，1 つの制約素として表現されたためであると考えられる．

(3) モジュラリティ： **図 5** は制約を構成する制約素の数を表す，制約密度の分布を示す．90 %以上の制約が多くとも 4 個の制約素から構成されている．よってプログラマはたかだか 4 個の制約素相互が，排他的な関係（制約が充足されているときは，ただ 1 つの制約素のみがユニットの状態と一致する関係）にあるかどうかをチェックすればよい．**図 6** は，ユニットを参照する制約の数を表すユニット参照数の分布を示す．80 %以上のユニットが多くとも 3 つの制約から参照されている．このことは，事務処理をモジュラリティ高く記述できたことを示す．約半数以上の制約から参照されるユニットも 1 つあったが，このユニットは規則の変更によって影響を受けないものであった．

(4) バグの発見： 同一制約内の制約素相互の排他的な

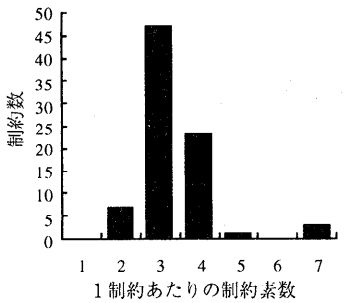


図 5 制約密度（制約を構成する制約素数）の分布
Fig. 5 constraint density.

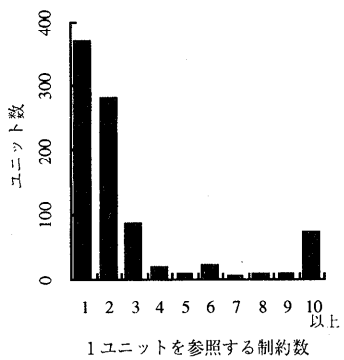


図 6 ユニット参照数（ユニットを参照する制約の数）の分布
Fig. 6 unit reference frequency.

関係のチェックは，ソフトウェア工学における境界値チェックに相当し，事務処理システムの動作確認時おいて，制約内のバグの発見に有効である．

(5) 網羅性：従来システムでは対応できなかった2つのレアケースに対しても，制約記述したシステムは対応できた．このことは，本手法が事務処理を記述するために高い能力を持つことを示す．

以上の分析から，制約による事務処理の記述は，事務処理システムの開発・維持管理をより容易にすると考えられる．

【処理効率】

(1) 質問による負荷：実用KOAにおいても，ユーザからの処理要求が不完全な場合，不完全性を補うためにユーザへ質問を行っている．実用KOAは専門家へのインタビューによって獲得された質問数を抑える質問戦略を有する．**表2**には，本手法による整合性管理過程で値が変化したユニットの数を示す．整合性管理過程で変化したユニット数は処理ごとに異なる．一方，**表3**は従来のKOAと制約によるKOAによって行われた質問数を示す．ユーザに質問されたユニットの数は，どの処理でも従来システムとほぼ同等である．このことは，本手法で用いている質問を削減するための戦略により，処理に必要な本質的なユニットに対してのみ質問が行われていることを示す．すなわち，質問数の増加をすることなく，制約を用いて事務処理システムを実現できることを示す．

(2) 処理速度：各制約を充足する間に行われる質問にかかる時間はSun Sparc Station 20で1秒以下であり，ユーザにとって十分速く感じられる速度であった．一方，すべての制約の真偽値を計算する際にかかる時間（図4(2)）は約10秒であった．

表2 変更ユニット数
Table 2 Number of units whose values are changed.

処理例	変更ユニット数
独身者の転入	166
既婚者の転入	432
結婚	243
出産	77

表3 質問数
Table 3 Number of queries.

処理例	KOA	本手法
独身者の転入	59	57
既婚者の転入	122	130
結婚	46	40
出産	27	26

しかし，この処理時間は，差分計算やCなどのPrologよりも効率の良いプログラミング言語でエンジンを記述することにより，削減可能となると予測される．

5. 関連研究

事務処理以外の分野で，整合性管理処理を実現する手法として*DeltaBlue algorithm*[10]がある．本手法も，DeltaBlue algorithmも，各制約の局所的な充足を繰り返すことにより，制約充足解を求める*propagation method*[11]により，整合状態を求める．各制約を充足するために，DeltaBlue algorithmではその制約に割り付けられているメソッド中の1つのメソッドを実行する．一方，本手法ではユニフィケーションとユーザとのインタラクションを用いて各制約を充足する．

また，ユニフィケーションに基づく制約充足機構を有し，制約階層[12]を扱うことができる階層型制約論理プログラミング言語（HCLP）[13]を用いても，整合性管理処理の実現は可能である．この際，整合性管理の仕方はタスクによって異なるので，タスクに適した解を求める戦略が必要となる．

6. まとめ

本論文では事務処理を，事務処理で扱うデータを事務規則と整合する状態に維持する整合性管理処理と見なし，制約プログラミングによる事務処理システムの構築手法を提案した．本手法では，事務規則を制約として記述し，制約充足エンジンによって，事務規則と整合するデータ状態を生成する．

本手法を，既存の事務処理システムのタスクに適用した結果，事務規則を制約として容易に記述することができ，システム記述量は50％に削減された．また，レアケースへも網羅的に対処でき，処理速度もユーザにとって十分短いものであった．

今後，本手法を実際運用されている事務処理システムへ導入するため，整合性管理システムとユーザとのインタラクションを容易にするユーザインタフェースの枠組みの研究を行う予定である．

謝辞 本研究を進めるにあたり，特にPrologプログラミング手法について種々のアドバイスをいただいた，NTTコミュニケーション科学研究所，佐々木裕研究主任に深謝いたします．

参考文献

1) Baumann, L.S. and Coop, R.D.: Automated Workflow Control: A Key to Office Produc-

tivity, *Proc. ACM AFIPS National Computer Conference*, Vol.49, pp.549–554 (1980).
2) Hogg, J., Nierstrasz, O.M. and Tsichritzis, D.: Office Procedures, *Office Automation*, Tsichritzis, D. (Ed.), pp.137–165, Springer-Verlag (1985).
3) de Jong, S.P.: The System For Business Automation (SBA): A Unified Application Development System, *Proc. IFIP Congress 80*, pp.469–474 (1980).
4) Lum, V.Y., Choy, D.M. and Shu, N.C.: OPAS: An Office Procedure Automation System, *IBM Systems Journal*, Vol.21, No.3, pp.327–350 (1982).
5) Shu, N.C., Lum, V., Tung, F.C. and Chang, C.L.: Specification of Forms Processing and Business Procedures for Office Automation, *IEEE Trans. Softw. Eng.*, Vol.8, No.5, pp.499–512 (1982).
6) Podelski, A. (Ed.): *Constraint Programming: Basics and Trends*, Springer-Verlag (1995).
7) Fikes, R.E.: Odyssey: A Knowledge-based Assistant, *Artificial Intelligence*, Vol.16, pp.331–361 (1981).
8) Barbar, G.: Supporting Organization Problem Solving with a WorkStation, *ACM Trans. Office Information Systems*, Vol.1, No.1, pp.45–67 (1983).
9) 中野，小島，金田：容易に知識修正ができるオフィス業務エキスパートシステム，電子情報通信学会人工知能と知識処理研究会 AI91-80, pp.49–56 (1992).
10) Freeman-Benson, B.N., Maloney, J. and Borning, A.: An Incremental Constraint Solver, *Comm. ACM*, Vol.33, No.1, pp.54–63 (1990).
11) Sussman, G.J. and Steele, G.L.: CONSTRAINT – A Language for Expressing Almost-hierarchical Descriptions, *Artificial Intelligence*, Vol.14, pp.1–39 (1980).
12) Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A. and Woolf, M.: Constraint Hierarchies, *Proc. OOPSLA '87 Conference*, pp.48–60 (1987).
13) Borning, A., Maher, M., Martindale, A. and Wilson, M.: Constraint Hierarchies and Logic Programming, *Proc. 6th Int. Conf. on Logic Programming*, pp.149–164 (1989).

付録 対話的制約充足エンジンの動作詳細

本付録では，3章で述べた対話的制約充足エンジンの動作を質問戦略を含め詳細に説明する．さらに，本エンジンによる事務処理の実現例を示す．

A.1 対話的制約充足エンジンの動作

本エンジンでは，質問数が少なく，ユーザ処理要求が不完全な場合でも，制約充足解を求めるため，各ユニット対してそのユニットが質問候補となり得るか否かを示す強度を付与する．強度は，「decided」と「default」の2値をとる．ユーザによって値が入力されたユニット，および，前記ユニットと制約から一意に値が決まるユニットの強度は「decided」となる．これにより，強度が「decided」のユニットはユーザの処理要求を反映していることを示す．

質問は，強度が「default」のユニットに関してのみ行われる．処理が開始される初期状態においては，すべてのユニットの強度は「default」に設定される．以後，強度が「default」のユニットを default ユニットと呼び，強度が「decided」のユニットを decided ユニットと呼ぶ．

以下に，詳細なエンジンの動作を述べる．3章で述べた質問戦略は，上記強度を用いた3つの戦略「制約選択」，「値削除」，「質問選択」戦略から構成される．項目番号は，図4中の番号に相当する．

エンジン動作

(1) ワーキングメモリ上のすべてのユニットの強度を default に設定した後，ユーザインタフェースを通じて，ユーザ処理要求を受け付ける．ユーザにより値が変更されたユニットの強度を decided に変更する．その結果，ユニットの状態は与えられた制約に対して，非整合状態となり得る．

(2) 非整合検出部は，すべての制約の真偽値が「真」となっているかを調べる．すべての制約が「真」の場合，制約充足状態となり，整合性管理処理は終了する．

(3) 質問数を抑制するため，以下の戦略に従い，真偽値が「偽」または「未定」の制約を1つ選択し，SC とする．

> **制約選択戦略**：すべての default ユニットの値を「_」と仮定した状態を SA とする．(i) SA に関してただ1つの制約素が「未定」となり，その制約中の残りの他の制約素がすべて「偽」であり，(ii) 前記真偽値が「未定」の制約素が，SA とユニフィケーションした結果，ground☆となる制約を1つ選択する．前記条件を満たす制約が存在しない場合は，真偽値が「未定」または「偽」の制約を任意に1つ選択する．

☆ 制約素中の制約変数にすべて定数が割り付けられている状態．

以下のステップでは，SC で参照されるユニットの値は，decided ユニットとユーザが入力したユニットの値に基づき，SC を満足するように局所的に決定される．

(4) ここでは，できるだけ少ない質問数で SC を充足するように質問をしなければならない．そこで，以下の戦略により質問候補集合 VU を作成し，それら候補のユニットの値を「_」とする．

> **値削除戦略：** SC 中の制約素で参照される default ユニットの集合を DU とする．DU 中のユニットの値を「_」としたユニットの状態を SA_{DU} とする．状態 SA_{DU} に対する，SC 中のすべての制約素の真偽値を計算する．前記計算の結果，真偽値が「未定」となる制約素の集合を SC_u とする．まず，SC_u 中の制約素のテスト節で参照される各ユニットを VU に入れる．次に，前記真偽値計算のユニフィケーションで，SC_u 中の少なくとも1つの制約素により値が定数となるニットを VU に追加する．次に，SC_u 中の同じ制約素中で他のユニットと制約変数を共有するユニットを VU に追加する．

(5) SC でただ1つの制約素の真偽値のみ「未定」となり，それ以外の SC 中の他のすべての制約素が「偽」，かつ，SC 中の真偽値が「未定」の制約素が ground となる場合，(6) へいく．それ以外は (7) へ．

(6) SC 中の真偽値が「未定」の制約素とのユニフィケーションによって割り付けられる値を，ユニットに割り付ける．この際，値が変化したユニットの強度を「decided」とする．(2) へ．

(7) 現時点でのユニットの状態では，SC を充足できないので，SC を充足するための値をユーザから獲得する．どのユニットを質問するかは，以下の戦略により，決定する．

> **質問選択戦略：** 値が「_」のユニットのうち，SC の真偽値が「未定」の制約素で参照されるユニットを1つ選択する．選択は，以下の優先度で行われる．(i) テスト節で参照されるユニット，(ii) ユニフィケーションにより複数の制約素から，異なる値を割り付けられるユニット，(iii) その他のユニット．

ユーザから獲得された値は，ワーキングメモリに書き込み，そのユニットの強度を decided とする．(5) へ．

A.2 動 作 例

本エンジンにより事務処理がどのように実現されるか，扶養手当の申請の例を用いて説明する．**図7** に，エンジンが管理する制約とデータを示す．エンジンが管理する制約は，以下の2つである．制約 A は，扶養の条件を規定し，制約 B は，扶養申請書を規定する制約である．

```
constraint(A,(A-1,A-2,A-3)).
element(A-1,(pattern(家族, 存在, 有),
             pattern(家族, 収入, X),
             test(X =< 1300000),
             pattern(扶養申請書, 存在, 有),
             pattern(社員, 扶養手当, 3000))).
element(A-2,(pattern(家族, 存在, 有),
             pattern(家族, 収入, X),
             test(X > 1300000),
             pattern(扶養申請書, 存在, 無),
             pattern(社員, 扶養手当, 0))).
element(A-3,(pattern(家族, 存在, 無),
             pattern(扶養申請書, 存在, 無),
             pattern(社員, 扶養手当, 0))).
constraint(B, (B-1,B-2)).
element(B-1,(pattern(扶養申請書, 存在, 無))).
element(B-2,(pattern(扶養申請書, 存在, 有),
             pattern(社員, 名前, X),
             pattern(扶養申請書, 社員名, X),
             pattern(家族, 名前, Y),
             pattern(扶養申請書, 家族名, Y),
             pattern(家族, 収入, Z),
             pattern(扶養申請書, 家族収入,
                     Z))).
```

家族の収入が 2,000,000 円の社員が家族の収入が 700,000 円になったので，扶養の申請を行う場合を考える．申請前のデータ状態を図7の (S1) に示す．この状態における真偽値の値は，制約素 (A-2)，制約素 (B-1) のみが真であり，他の制約素は偽となるので，制約 A，制約 B は真である．よって，制約充足状態であり，正しく事務処理が行われている．すべてのユニットの強度はエンジンにより，default とされている．

ここで，ユーザの要求として申請書が提出され，システムは，扶養申請書の存在を有，扶養申請書の社員名を関玲，扶養申請書の家族名を関麗子，扶養申請書の家族収入を 700000，に変更する（付録：エンジン動作 (1)）．この際，上記ユニットはユーザ処理要求なので，前記ユニットの強度は decided となる．データ状態は図7の (S2) となる．エンジンは，データ状態

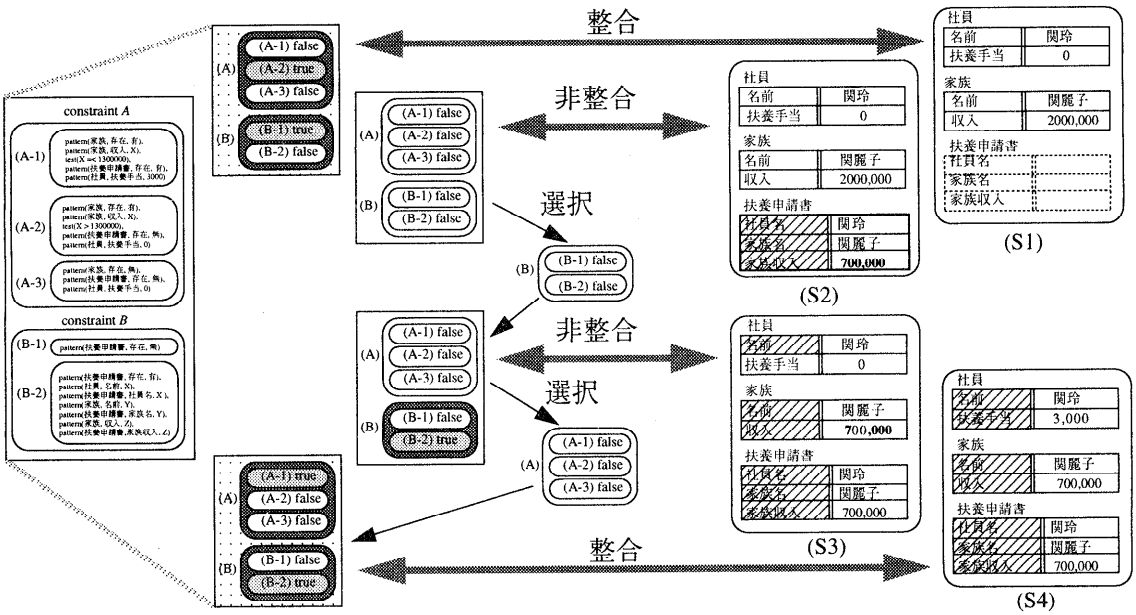


図7 動作例

Fig. 7 A Consistency maintenance example using the proposed engine.

が制約を充足するかチェックする（付録：エンジン動作 (2)）．制約素の真偽値は，すべて偽となるので，非充足である．

非充足となっている制約の中から，質問選択戦略により，値の伝播が可能な制約 B を選択する（付録：エンジン動作 (3)）☆

エンジンは，制約 B に対して値削除を行う（付録：エンジン動作 (4)）．社員の名前，家族の名前，家族の収入の値は削除され，「_」が割り付けられる☆☆．

制約 B が充足されるかどうかチェックする（付録：エンジン動作 (5)）．真偽値は，制約素 (B-2) のみ未であり，他が偽，かつ，真偽値が未となっている制約素 (B-2) は ground となっているので，付録：エンジン動作 (6) へすすむ．真偽値が未となっている制約素 (B-2) とのユニフィケーションによって割り付けられる値，関玲，関麗子，700000 を各々，社員の名前，家族の名前，家族の収入に更新し，それらのユニットの強度を decided とする．その結果，データ状態は，図 7 の (S3) となる．

エンジンは再度，制約充足状態かどうかチェックする（付録：エンジン動作 (2)）．真偽値は，制約素 (B-1) 偽，(B-2) 真なので，制約 B は真，制約 A の制約素は，すべて偽なので，制約 A は偽である．

よってエンジンは非充足の制約 A を選択する（付録：エンジン動作 (3)）．エンジンは，制約 A に対して値削除を行う（付録：エンジン動作 (4)）．家族，扶養手当の値が削除される☆☆☆．

制約 A が充足されるかどうかチェックする（付録：エンジン動作 (5)）．真偽値は，制約素 (A-1) のみ未であり，他が偽，かつ，真偽値が未となっている制約素 (A-1) は ground となっているので，付録：エンジン動作 (6) へすすむ．真偽値が未となっている制約素 (A-1) とのユニフィケーションによって割り付けられる値，有，3000 を各々，家族，扶養手当に更新し，それらのユニットの強度を decided とする．その結果，データ状態は，図 7 の (S4) となる．

☆ 制約 B は，default ユニットの値を「_」として，真偽値計算を行うと，制約素 (B-2) のみ未，他は偽である．ユニフィケーションにより，制約素 (B-2) で参照する社員の名前，家族の名前，家族の収入は各々，関玲，関麗子，700000 がバインドされる．よって，質問戦略で最も優先される条件を満足する．

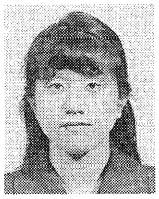
☆☆ 制約 B で参照される default ユニットの値をすべて「_」として真偽値計算を行うと，制約素 (B-2) のみ未となる．(B-2) で参照される default ユニットのうち，社員の名前，家族の名前，家族の収入は，前記真偽値計算の過程で，ユニフィケーションにより，扶養申請書の社員名，扶養申請書の家族名，扶養申請書の家族収入の値が各々バインドされる．よって，それらの値は，削除される．

☆☆☆ 制約 A で参照される default ユニットの値をすべて「_」として真偽値計算を行うと，制約素 (A-1) のみ未となる．(A-1) で参照される default ユニットのうち，家族，扶養手当には，前記真偽値計算の過程で，ユニフィケーションにより，制約で宣言されている値，有，3000 が各々バインドされる．よって，それらの値は削除される．

エンジンは再度，制約充足状態かどうかチェックする（付録：エンジン動作(2)）．真偽値は，制約素(B-1)偽，(B-2)真なので，制約Bは真，制約Aの制約素は，制約素(A-1)真，制約素(A-2)偽，制約素(A-3)偽なので，制約Aも真．よって制約充足状態となり，エンジンは処理を終了する．

この状態は，申請書はファイルされ，手当も支給され，正しく事務処理は行われている．よって整合性管理により事務処理は，実現された．

(平成8年4月26日受付)
(平成9年5月 8 日採録)



石井　恵（正会員）

平成元年，慶応義塾大学理工学部数理科学科卒業．同年，日本電信電話（株）入社．以来，エキスパートシステム，機械学習の研究に従事．平成4，7年，人工知能学会全国大会優秀論文賞受賞．平成5年，情報処理学会秋期全国大会奨励賞受賞．現在，NTT情報通信研究所，研究主任．人工知能学会会員．

金田　重郎（正会員）

昭和49年，京都大学工学部電気第二学科卒業．昭和51年，同大学大学院電子工学専攻修士課程修了．同年，日本電信電話公社武蔵野電気通信研究所入所．以来，誤り訂正符号，フォールトトレラント技術，知識獲得，エキスパートシステム，機械学習等の研究に従事．平成9年4月より，同志社大学大学院総合政策科学研究科・同志社大学工学部，教授．工学博士，技術士（情報処理部門）．人工知能学会，電子情報通信学会，経営情報学会，言語処理学会，IEEE，各会員．