データベースシステム 109－20
（1996．7．25）

ナビゲーション戦略を獲得するデータエントリシステムの実現

石井 恵　　金田 重郎
NTT コミュニケーション科学研究所

あらまし： データベースの更新を行うデータエントリシステムでは，必要なデータを洩れなく正しく，かつ，効率的に，またユーザにとって自然な順序で入力できるようユーザをナビゲートしなければならない．従来では，自然な順序での入力を実現するため，専門家から獲得した入力の順序をナビゲーション戦略として，プログラム化しなければならなかった．しかし，予め全てのナビゲーション戦略を記述することは困難である．そこで，ユーザのオペレーション履歴からシナリオを自動獲得することにより，自然な順序での入力が可能なシステム構築手法を提案する．本手法の特徴は，与えられたオペレーション履歴を，データ項目間の制約関係にもとづき汎化して獲得する点にある．本手法により，システムはシステムの運用過程でナビゲーション戦略を獲得できる．そのため，予め全てのナビゲーション戦略を用意せずとも，ユーザにとって自然な順序でユーザをナビゲートできるデータエントリシステムが実現できる．

Data-entry systems with navigation strategy learning

Megumi ISHII and Shigeo KANEDA

NTT Communication Science Laboratories

Abstract: In this paper, we propose a method to realize user friendly data-entry systems. The feature is that our method learns navigation strategies by generalizing user's operation logs based on constraints among data items. The proposed method enables data-entry systems to acquire navigation strategies through its operation stage. As a result, data-entry systems which show users a candidate data item to be input next in user friendly order can be realized without preparing navigation strategies completely as programs.

1　はじめに

ユーザがデータベースの更新を行なうためのデータエントリシステムにおいては，(1) 洩れなく，正しく (結果の整合性の保証)，(2) ユーザにとって使い易く (ユーザフレンドリ)，ユーザが必要なデータを入力できなければならない．通常，結果の整合性を保証するためには，チェックプログラムを用意し，矛盾が存在する時は，その箇所をユーザに通知し，ユーザに正しい値を入力させる．代表的なチェックの種類としては，データの型チェック，値域チェック等，各データ毎に閉じた単項チェックがある．しかし，現実のアプリケーションでは，単項チェックのみでなく，複数のデータ項目間の関係の整合性をチェックする必要がある．この際，洩れのないチェックを実現するために，どのように整合性を定義するかが問題となる．

一方，ユーザフレンドリなシステムであるためには，ユーザ主導とシステムによるユーザのナビゲーションが調和よく実現されていなければならない．具体的には，(a) ユーザによるシステムへの自由な値の通知，すなわち，システムに制約を受けず任意の順序でのデータ投入を許容する入力機能と，(b) 処理の流れに対するユーザの戸惑いを防ぐために，ユーザに自然なナビゲーション戦略 (データの入力順序) をもつナビゲート機能が実現されていなければならない．更に，ユーザの負担を軽減するため，できるだけ少ないデータ投入数での処理が実現されることが望まれる．

従来，ユーザフレンドリであるシステムの構築にあたっては，手続き，トリガといった，状態遷移を表現する記述言語を利用して試みられてきた．なぜなら，これらのアプローチでは，実際の業務担当者が行な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からである．反面，業務に必要な具体的な処理，すなわち，どのような順序でユーザをナビゲーションするか，また，どのような時にどのデータをどのような値に変更するかという手続きを全て記述しなければならない．現実には，ユーザにとって自然なナビゲーション戦略や，データの変更手続きを，予め全て用意することは難しい．その結果，ユーザによるシステムへの自由な値の通知を規制したインタ

フェースとなり，ユーザフレンドリなシステムの構築が困難であった．

ここでデータベースの更新処理をみてみる．更新された結果は，常に与えられたチェック関数を満足した状態でなければならない．よって，データベースの更新処理は，データ状態を与えられたチェック関数，すなわち，与えられた制約関係と常に整合した状態に保つ整合性管理処理とみなすことができる．このことから逆に，与えられた制約関係を満足させることにより，データベース更新処理を実現できるはずである．そこで，我々は与えられた制約関係を積極的に利用する制約充足のアプローチにより，整合性を保証し，かつ，ユーザフレンドリなデータエントリシステムを実現する手法を提案する．

本手法では，制約充足の枠組により，整合性の保証，ユーザ主導の入力，少ないデータ投入数での処理を実現する．そして，ユーザの操作履歴からの学習により，予め作成が困難であるユーザにとって自然なナビゲーション戦略を自動的に獲得する．その結果，プログラミングを行なわずとも，自然なナビゲーション戦略でユーザをナビゲートする機能を実現できる．

以下，第2章では制約充足によるデータエントリ処理の実現手法について説明し，第3章で履歴からのナビゲーション戦略獲得によるナビゲート機能実現手法について提案する．第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のタスクへの適用結果を示す．第5章はまとめである．

2 制約充足を用いたデータエントリシステムの実現

2.1 整合性の定義

データエントリシステムを構築する際は，複数データ項目間の関係をいかに記述するかが問題となる．それには，アプリケーションの整合性を定義する記述表現がなければならない．今回，我々は既に提案している事務処理のための制約表現を用いる [1]．なぜなら，事務処理はデータベースの更新処理により処理が実現されるので，データベースの更新処理を行なうデータエントリシステムにおける整合性の記述表現として利用できるからである．以下に具体的な例をあげて簡単に記述表現法を説明する．

例えば，扶養の申請に関するデータベースの更新処理について考える．複数データ項目間の関係として，扶養の規則「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．」が与えられているとする．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す．この規則を満足するようにデータベースの更新が行なわれた場合，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更新処理（例：家族の収入が 13 0 万円以上で，扶養手当を貰っている状態）は規則に反し，正しい処理結果ではないからである．

よって，項目間の関係を規定する規則は，その規則が許すデータ状態を列挙することにより制約として表現できる．この際，データ状態相互は排他的に記述する．これは，各データ状態相互を包含関係，および，重複関係なく記述することにより，洩れなく条件を列挙できるようにプログラマに意識させるためである．規則を制約として記述した結果，データベースの更新処理では，全ての規則が満足されている状態が制約充足である．この時，各制約中では，宣言されているデータ状態のどれか1つと，更新処理後のデータ状態が整合する．

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: 規則の制約としての解釈の例

2.2 制約充足とデータベースの更新処理

ここで，家族の年収が 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のデータベースの更新処理について考える．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，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，業務担当者が行なうデータベース更新処理の例である．

(1) 社員情報（データベース）の家族の年収を，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，規則と整合した状態となる．

我々は，上記更新処理を，新たに入力されたデータの値 (扶養手当申請書に記述されている各データの値) と整合するデータ状態を，与えられた規則が許すデータ状態の宣言の中から決定することにより，制約充足の枠組で実現する手法を提案した [1]．

2.3 対話的制約充足システム

以下に，[1] で提案した制約表現，制約充足エンジンについて簡単に述べる．

表 1: 真偽値表

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

2.3.1 制約シンタックス

データ項目の関係，すなわち，アプリケーションの処理結果の整合性を規定する規則を制約と呼ぶ．アプリケーションは複数の制約により記述される．各制約は，その規則が許す1つ以上のデータ状態の宣言から構成される．各データ状態を制約素と呼ぶ．制約及び制約素は，制約充足エンジンにより計算される，充足の状態を表す真偽値をもつ．表1に真偽値を示す．全ての制約の真偽値が**真**であるとき，制約は充足状態となる．

1．ユニット

```
unit(ユニット名,　値).
```

ユニット名は，データベース中のデータ項目名に対応する．ユニット名はユニークであり，値は定数，または，変数を書く．値が変数の場合，値が未割り当てであることを示す．

2．制約

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

制約名はユニークであり，制約素名は次の element で定義される制約素の名前である．制約条件の洩れを防ぐため，制約素相互に以下の関係を設ける．

[制約素の排他性]
同一制約内の制約素は，1つの制約素が真となる時は，残りの制約素は偽となる．　□

3．制約素

```
element(制約素名, (pattern(ユニット名, 値), ···,
                   test(Boolean 関数), ···)).
```

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．パタン中の値は定数値，及び，各制約素内でのみ有効な変数である制約変数をとる．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．制約素は，パタンとテスト節の積として解釈され，パタン及びテスト節の評価の順序は記述順である．そして Boolean 関数は評価される時点で，Boolean 関数中の制約変数は全て**バインド状態**[1]になければならない．

図 2は図 1に対応する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，複数データ項目間の関係を，各関係毎にその関係が許すデータ状態を排他的に列挙させることにより，アプリケーションの整合性を定義する．

```
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 2: 制約の例

2.4 対話的制約充足エンジン

本エンジンはユーザから入力された値と制約を用いて，他のユニットの値を決定し，ユニットを制約と整合した状態とする．ただし，各制約を充足する際，制約を充足するユニットの状態を一意に決定できない場合がある．その場合，本エンジンでは，ユーザに質問を行なう．これは，ユーザの処理要求を反映した処理結果とするためである．エンジンの基本的な動きを図3に示す．本エンジンでは，ユーザフレンドリなデータエントリシステムを実現するために，アプリケーション個々の充足手続きを含まない汎用的な制約充足動作を行ない，処理結果の整合性の保証と，ユーザによる任意の順序でのデータ投入を許容する入力機能を実現した．

図 4は，本エンジンを用いたデータエントリシステムのＧＵＩである．アイコンは帳票を表し，アイコンを左右の領域に移動することにより帳票を作成，削除する．アイコンを開くことにより，帳票に記入されているデータ項目の値を把握でき，また，自由にその値の変更ができる．画面左上に次に入力すべきデータ項目名が表示される．ユーザはそのメッセージに従ってもよいし，他のデータ項目を帳票画面上で自由に入力してもよい．また，ユーザが入力した値と制約から他

[1] 変数に定数値が束縛されている．

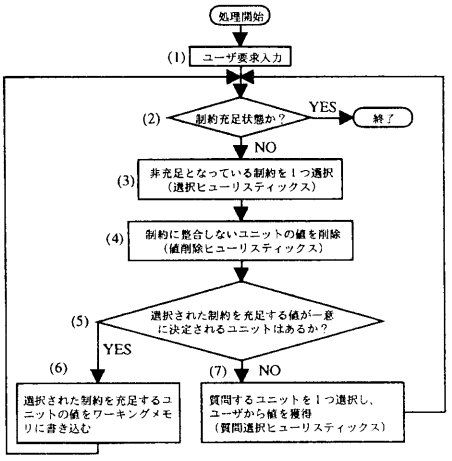

図 3: 制約充足エンジンフロー

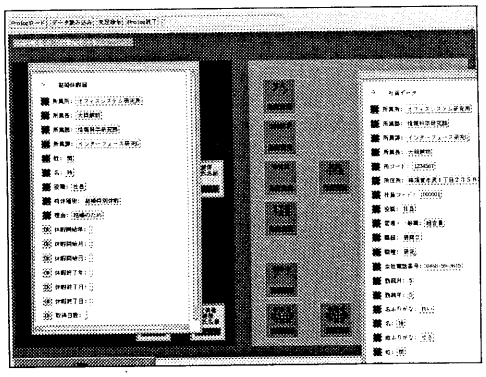

図 4: システム画面

のユニットへの値の伝播と，質問数を削減するためのヒューリスティック [2] により，データ投入数を抑える．

3 履歴からのナビゲーション戦略獲得

3.1 自然なナビゲーションへのアプローチ

ユーザフレンドリなデータエントリを実現するためには，ユーザに自然なナビゲーション戦略をもつナビゲート機能，すなわち，ユーザにとって自然な順序での質問生成が課題となる．例えば，家族のデータを入力する際は，姓→名→住所→生年月日といった順序で，入力を促されれば，ユーザは自然に感じるはずである．

前記エンジンのように制約構造を用いた動的な質問生成では，ユーザに自然なナビゲーション戦略でのデータ入力は困難である．なぜなら，全ての処理に対して，人間にとって自然となる入力順序を生成する関数を予め定義することは困難であるからである．また，陽に入力順序のシーケンスを予め記述するアプローチも，同様に困難である．

我々は，ユーザにとって自然なシーケンスは，ユーザが実際に利用するシステムのユーザインタフェース上でしか獲得できないと考える．なぜなら，実際のシステム上でデータを入れる状況下にユーザを置き，実際にユーザに利用させることにより，ユーザにとっての自然さを獲得できると考えるからである．

そこで，我々はユーザの操作履歴から，データ入力のナビゲーション戦略を獲得するアプローチをとる．この際，以前獲得した履歴と同一，すなわち，ユニットの値とデータ項目の入力順序が全てが過去の履歴と同一であれば，過去の履歴をナビゲーション戦略としてそのまま適用することが考えられる．しかし，ユニットの値とデータ項目の入力順序が全て同一の処理に対してのみ，獲得ナビゲーション戦略を適用したのでは，ユーザフレンドリなシステムとなるまで時間がかかり，実用的でない．そのため，できるだけ少ない履歴数で処理をカバーするナビゲーション戦略が獲得されることが望ましい．

そこで，我々は与えられた制約の充足状況を汎化基準として利用する．なぜなら，同じ各制約において，制約素を充足する制約充足状態同士は，局面的には，意味的に同じ状態とみなせるからである．ただし，制約の充足状況のみでは制約素の真偽値状態の変化を伴わない場合，同一の状況とみなしてしまう．そこで，制約素の真偽値状態の変化を伴わない入力シーケンスを１つのナビゲーション戦略として獲得し，制約の充足状態が変化しない場合は，その状態に対応するナビゲーション戦略にそってナビゲーションを行なう．

制約の充足状況への汎化は，具体的にはユニットの状態を制約素の真偽値の状態へ汎化することにより行なう．例えば，図 2において，ユニットの状態が「家族・存在＝有」，「家族・収入＝700000」，「扶養申請書・存在＝有」，「社員・扶養手当＝3000」，とした場合，ユニット状態(家族・存在，家族・収入，扶養申請書・存在，社員・扶養手当)=(有，700000，有，3000)は，制約素真偽値状態(a1, a2, a3)=(真，偽，偽)に汎化される．

また，ナビゲーション戦略が獲得されていない状況においても，システムは正しくユーザが処理できるようにユーザを支援しなければならない．そこで，本手法ではナビゲーション戦略が獲得されていない状況に直面した場合，システムは質問数を抑える既存の質問戦略を用いてユーザをナビゲートする．その結果，ナビゲーション戦略が獲得されていない状況下においては，効率の良いデータ項目の入力の提示によるナビゲートを行なうことにより，ユーザの支援が行なわれる．本アプローチの大きな利点は，予めユーザにとって自然なナビゲーション戦略を用意せずとも，ユーザのシステム利用時に，ナビゲーション戦略をプログラミングレスでシステムに組み込むことができる点である．

3.2 ナビゲーション戦略獲得機能付き制約充足

以下に，ナビゲーション戦略獲得機能付き制約充足エンジンの動作を説明する．図5は，エンジン動作を示す．

(準備)

U: ユニット状態.

C: 制約素集合.

$check(C,U)$: ユニット状態 U に対する C に含まれる制約素の真偽値状態を返却.

S: ナビゲーション戦略集合.

$remove(a, list)$: $list$ の要素のうち，a を要素に含む要素から a を削除した要素から構成されるリストを返却.
$remove(a,\ ((a\ x_1\ x_2)\ (b\ y_1)\ (a\ z_1)))$
$\rightarrow ((x_1\ x_2)\ (z_1))$

$assoc(a, list)$: リスト $list$ 中の要素のうち，先頭要素が a である要素の a 以外の要素から構成されるリストを返却．先頭要素が a である要素が存在しない場合は，() を返却.
$assoc(x_1,((x_1\ (a_1\ a_2\ a_3)\ (b_1\ b_2))\ (x_2\ (c_1\ c_2)))$
$\rightarrow\ ((a_1\ a_2\ a_3)\ (b_1\ b_2))$.

$add(a, list1, list2)$: リスト $list2$ 中の要素のうち，先頭要素が a である要素を，その要素の最後尾に $list1$ を付加した要素と置換した結果を返却．先頭要素が a である要素が存在しない場合は，要素 $(a,\ list1)$ を $list2$ へ追加した結果を返却.
$add(x_2,(d_1\ d_2),((x_1\ (a_1)\ (b_1\ b_2))\ (x_2\ (c_1))))$
$\rightarrow ((x_1\ (a_1)\ (b_1\ b_2))\ (x_2\ (c_1)\ (d_1\ d_2)))$.
$add(x_2,(d_1\ d_2),((x_1\ (a_1\ a_2\ a_3))))$
$\rightarrow ((x_1\ (a_1\ a_2\ a_3))\ (x_2\ (d_1\ d_2)))$.

$append(list1, list2)$: $list1$ と $list2$ の要素から構成されるリストを返却．返却リストの要素は $list1$, $list2$ の順でならぶ.
$append((a\ b\ c),(d\ e\ a)) \rightarrow (a\ b\ c\ d\ e\ a)$.

【ナビゲーション戦略獲得エンジン動作】

(1) $x_{pre} \leftarrow ()$, $x \leftarrow ()$, $S_c \leftarrow ()$, $S_h \leftarrow ()$. S には獲得されたナビゲーション戦略が格納されている．ユニットが全ての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．ユニットの状態を入力されたユーザ要求にもとづき変更する.

(2) 全ての制約の真偽値が**真**となっているか否かを調べる．全ての制約の真偽値が**真**なら (3) へ．それ以外は (5) へ.

(3) $S_h = ()$ ならば処理終了.

(4) $S \leftarrow add(x_{pre}, S_h, S)$. 処理終了.

(5) 真偽値が**偽**，または**未定**の制約を 1 つ選択する．選択された制約を，SC とする.

(6) 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 (変数化) する.

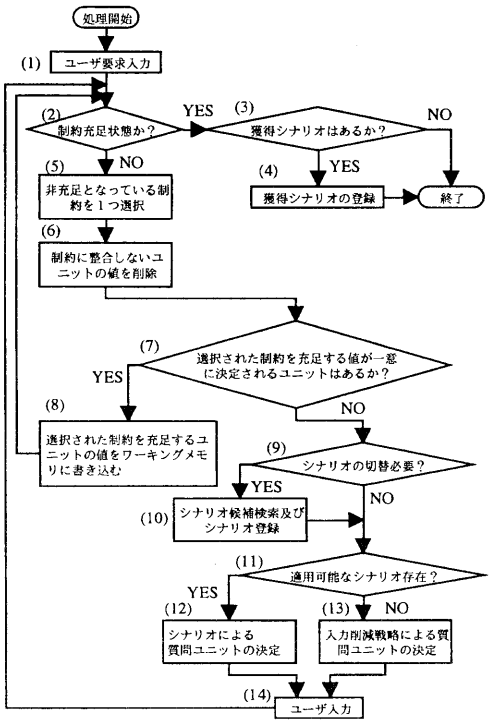

図 5: ナビゲーション戦略獲得エンジンフロー

(7) SC と整合する値を一意に決定できるユニットが存在するか判定する．具体的には，SC の制約素のうち，唯一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，ユニフィケーションの前ではバインド状態でなく，ユニフィケーション後，バインド状態となる制約変数が存在すれば，値を一意に決定できるユニットが存在する．値を決定できるユニットが存在なら，(8) のステップの処理をする．それ以外は (9) のステップへ.

(8) 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状態と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.

(9) $x \leftarrow check(U, C)$. $x = x_{pre}$ なら (11) へ.

(10) $S \leftarrow add(x_{pre}, S_h, S)$. $x_{pre} \leftarrow x$. $S_h \leftarrow ()$. $S_c \leftarrow assoc(x, S)$.

(11) $S_c = ()$ または，S_c に先頭要素をユーザの処理要求を反映していないユニットとする要素が 1 つもなければ (13) へ.

(12) S_c の各要素の先頭要素で出現頻度の最も高い要素で指定されるユニットを u とする.
$S_c \leftarrow remove(u, S_c)$. (14) へ.

(13) 質問数を抑えるヒューリスティックス [2] にもとづき質問ユニット u を決定する.

(14) ユーザにユニット u の値を質問する. ユーザによる値の入力. 値が入力されたユニットを u' とする. $S_h \leftarrow append(u', S_h)$. 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. (2) へ. □

4 評価

本提案手法を, 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 [3] のタスクに適用した. KOAは社員の結婚, 子供の誕生に伴い必要となる帳票の作成を支援するシステムである. ユーザはKOAの質問に答えることにより帳票を作成する.

4.1 自然なナビゲーション戦略の獲得

図6に, 本方式よって獲得されたナビゲーション戦略によって行なわれた処理履歴 (図6(A)) と, 2.4で述べた文献 [1] のエンジンによって行なわれた処理履歴 (図6(B)) を示す. 後者では, 配偶者に関するユニットを聞いた後, 社員に関するユニットを尋ね再度, 配偶者に関するユニットの質問に戻っており, 自然な流れとなっていない (図6(B)). それに対し, 本方式では, ユニットをあるまとまりをもって質問しており, より自然なナビゲーションが行なわれている (図6(A)).

何をもって人間にとって自然なナビゲーションと感じるかはアプリケーションによって異なる. しかし, 本方式では, 自然さを定義しなくても, 思いのままにユーザに利用させることにより, 自然なナビゲーションが実現できる.

4.2 汎化レベルと獲得されるナビゲーション戦略

汎化レベルが低いと汎用性のないナビゲーション戦略となり, また, 汎化レベルが高いとある状況において適用可能なナビゲーション戦略が増え, 適切なナビゲーション戦略が選択されなくなる. 今回用いた汎化レベルは, ある局面における入力ユニットの順序と, 制約の充足状態によるものである.

KOAのタスクに適用した際, 制約の充足状態が同じでも, 異なる順序でナビゲートされた方がよい場合があった. これは, ある状態において, 過去どの程度ユニットの情報を取得したという点が反映されていないためにおこった. KOAに関していえば, 取得されたユニットの情報を考慮に入れた汎化レベルが必要であると考えられる. 上記問題は, ある時点において, 値が入力されたユニットが同じで, 制約充足状況が同じユニットの状態相互は, 同じ状況にあるという思想にもとづく汎化により解決できる. この思想は, ある状態において, それ以前に獲得された特徴的な情報が同一なら, 同じ状況とみなすことを意味し, 自然である. このように, 汎化に制約を用いる本手法は, 汎化に意味をもたせることが可能であり, アプリケーションに適した汎化基準を与えることができる.

図 6: ナビゲーション履歴

5 まとめ

本稿では, ユーザフレンドリなデータエントリシステムの構築手法について提案した. 本手法の特徴は, ユーザの操作履歴を制約の充足状態にもとづき汎化し, ナビゲーション戦略として利用することにある. その結果, 陽に記述することが困難であった, ユーザにとって自然に感じられるナビゲーションを実現できる. また, 学習機能によりそのシステムが使われるに適応したナビゲーションが可能となる.

参考文献

[1] M.ISHII, Y.SASAKI and S.KANEDA: Interactive Constraint Satisfaction for Office Systems, Proc. of the Tenth Conference on Artificial Intelligence for Applications, pp. 116-124 (1994).

[2] 石井, 金田: 制約プログラミングによるオフィス処理の実現, NTT-R&D, Vol.44, NO.5, PP.107-114,(1995).

[3] 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).