人　工　知　能 98－9
（1995．1．18）

対話プランニングにおける複数事例適合メカニズムについて

谷口 実　　上原 邦昭　　前川 禎男
神戸大学 工学部 情報知能工学科

古典的プランニングにおける問題点を改善するため，対話のリアクティブ性に重点をおいた事例に基づくプランニングが考案されている．しかしながら，事例に基づくプランニングでは，単一の事例しか利用できないこと，事例の部分的な取り扱いができないことなど非効率的な問題がある．本研究では，事例の再利用性を向上させるために，代替適合，変換適合の２つの枠組みからなる複数事例適合メカニズムを提案する．本手法によって，単一事例の利用が失敗する場合は，ユーザの発話意図を考慮して複数の断片化された発話に詳細化し，それぞれの発話に対して個別にプランニングすることが可能になる．また，詳細化された発話を事例インデックスとみなすことにより，事例の部分的な利用が可能となる．

Multiple-Cases Adaptation for Text Planning

Minoru Taniguchi　　Kuniaki Uehara　　Sadao Maekawa
Department of Computer and Systems Engineering, Kobe University

Our motivation is to improve the efficiency of case-reuse in Case-Based Planning. CBP consists of both retrieving a past similar case from case base and adapting it to a new problem. In current text planning model, adaptation of a new text plan is restricted to retrieve a single analogous case. If multiple-cases or some suitable parts of a case are available in the adaptation process, the planner can make a plan more flexibly. From this viewpoint, we propose a multiple-cases adaptation mechanism to be able to divide user's utterance into some fragments and to retrieve and adapt the appropriate cases separately.

1 はじめに

近年，対話システムの構築において，ユーザの発話に対してシステムが何を話すかを決定する (what-to-say) 問題が盛んに研究されている [1, 2]．これらの研究では，システム内の談話に関する抽象的な知識 (プランオペレータ) を利用して，適切な発話プランを探索するアプローチを採用しているものが多い (これを古典的プランニングと呼ぶ)．

古典的プランニングに基づく対話システムは，予めプランオペレータを用意しておけば，いかなるユーザの発話にも妥当な発話プランを生成できるという仮定に基づいて構築されている．しかしながら，このような仮定には，あらゆる対話に応答できるようなプランオペータを用意することは不可能であるという記述性の問題がある．また，対話の実行には即応性が求められるが，古典的プランニングでは，プランニング時の計算量が増大するという即応性の問題もある．さらに，以前の入力と同じ発話が入力された場合に，再度同じ推論をたどってプランを生成してしまうという効率性の問題もある．

このような，古典的プランニングに基づく対話システムに対し，我々はリアクティブな対話管理メカニズムを提案している．本メカニズムは UNIX コマンドの利用支援システム Assist-R [3, 4] として実現されている．Assist-R は，前述の記述性・即応性・効率性の問題を軽減するために，プラン生成部に事例に基づく推論手法 [5, 6] を用い，過度な推論を回避している．

一般的に，事例に基づく推論手法では，多くの事例が蓄積されていることを前提としているため，事例が少ない場合には有効なプランを生成できないという問題がある．この点に関して，Assist-R は古典的プランニングを補助的に用いて解決しているが，大量の事例ベースを構築することは困難であるため，実際には事例に基づく推論よりも古典的プランニングに頼ることが多くなっている．

本稿では，事例を用いたプランニングの問題を解決し，事例の利用効率を向上させるために，誘導類推の分野で考案されたインデックス詳細化手法 [7] を提案する．また，インデックス詳細化を用いて複数事例の複合的な利用や事例の部分的な利用が可能となることを示す．さらに，発話の詳細化手法と従来のプランニング手法を統合した複数事例適合メカニズムを提案する．

2 Assist-R の枠組み

Assist-R の枠組みを図 1 に示す．ユーザからの発話に対し，Assist-R は事例ベースを検索し，類似した対話事例を獲得する．検索された対話事例は現在の発話に適合するように修正され，新たな対話プランが生成される．さらに，Assist-R はユーザの発話を観測しながら，生成されたプランを動的に変更してユーザとの対話を実現するようになっている．

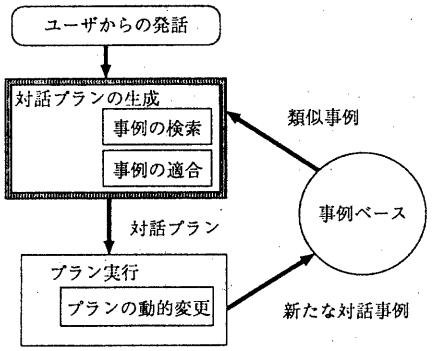

図 1: Assist-R の枠組み

3 従来の Assist-R の問題点

従来の Assist-R に起因するプラン生成時の問題として，次の 2 点が挙げられる．

根拠ノードの問題

4.2 節で詳しく述べるが，従来の Assit-R は根拠ノードを持っている．根拠ノードは，システムが生成した応答に関連する領域知識を示したものであり，事例を検索する際の制約として機能するものである．逆に，このことが検索の柔軟性を妨げる原因となっている．すなわち，内部の記述を僅かに変更すれば利用できる対話の事例があった場合でも，根拠ノードによる制約によってその事例が利用できなくなるという問題がある．

事例利用の効率性の問題

従来の Assist-R では，事例ベース中の過去の対話事例を一つだけ検索し，新たな対話プランを生成するようにしている．このため，対話が複雑になれば，そのまま利用できる対話事例が限定されるという問題がある [8]．また，従来の Assist-R では，ユーザの発話に対して部分的に有効なプランを含む事例が存在しても，その有効部分を認識できず，新たにプランを生成してしまうという問題がある．

これらの問題を解決し，事例の利用範囲を広げるために，本稿では以下の手法を提案する．

まず，根拠ノードの問題に関しては，固定した記述である根拠ノードのかわりに，動的に領域知識を利用しながら柔軟にユーザの発話と対話事例を関連付けるようなメカニズムを提案する．

つぎに，事例利用の効率性の問題については，単一の事例が利用できない場合，ユーザの発話を複数の発話に分解して考え，それぞれの発話に対して別個の事例を利用して，複合的にプランニングする複数事例適合メカニズムを提案する．また，分解された発話をそれぞれ部分的な対話事例のインデックスとすれば，複雑な対話からなる事例をいくつかの基本的な対話事例の組合せとして取り扱えるようにできることを示す．

4 事例表現

4.1 内部表現

本研究では，UNIX コマンドの利用支援システムを対象とし，ユーザがシステムに UNIX コマンドに関する問い合わせを行なう際の対話について検討する．システムに入力されるユーザの発話は日本語で与えられ，システム内部で 述語論理形式のリテラルとして翻訳されるものと仮定する．

システムとの対話において，ユーザの発話には必ず何らかの意図が含まれている．言い換えると，ユーザの発話には，システムがコマンドに関する説明を行なうための助言となるような情報が含まれていると考えられる．このような考え方から，108 個の UNIX コマンドに関して 254 の対話例を収集して，ユーザの発話から得られる 3 種類の情報と，そこで用いられるリテラルを定義している．

3 種類の情報とは，「コマンドの動作に関する情報」，「コマンドの動作とともに使用すべきオプションに関する情報」，「コマンドの実行対象となるオブジェクトに関する情報」である．以下では，記述の簡単化のため，それぞれコマンド情報，オプション情報，オブジェクト情報と呼び，内部では act 述語，opt 述語，obj 述語によって表現している．それぞれの情報で利用されるリテラルの例を以下に示す．

コマンド情報：
delete, move, output, change, create, modify etc.

オプション情報:
recursive, interactive, assign, numeric etc.

オブジェクト情報:
file, direcory, printer, display, stream etc.

日本語による発話は，上記の定義にしたがってシステムの内部表現に翻訳される．たとえば，ユーザの発話「ファイルに行番号をつけて表示したい」は，「ファイル」が file，「行番号をつけて」が numeric，「表示」が output および display に翻訳される．翻訳されたリテラルは，それぞれの情報ごとに述語形式で表現され以下の記述となる．

「ファイルに行番号をつけて表示したい」

```
user_goal(act(output),
          opt(numeric),
          obj([display,file]))
```

4.2 事例表現

ユーザの発話は前述のような述語形式によって記述されるが，対話を表現するためには，発話の流れを考慮した表現形式を検討しなければならない．本研究では，対話状況の遷移を状態遷移グラフ (図 2) によって表現している．状態遷移グラフのノード部分はシステムの発話を表し，このノードを発話ノードと呼ぶ．アークのラベル部分はユーザの発話を表している．対話状況の遷移は，START ノードから END ノードにいたる，アークとノード間の遷移によって表現される．

従来の Assist-R では，この他に発話ノードに関連付けられた根拠ノードが用いられていた．根拠ノードは，ユーザの発話を正当付けるための領域知識を記述したものである．しかしながら，前章で述べた問題のために，本稿で述べる新たな Assist-R では根拠ノードを用いていない．

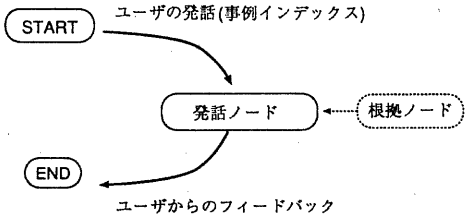

図 2: 事例表現

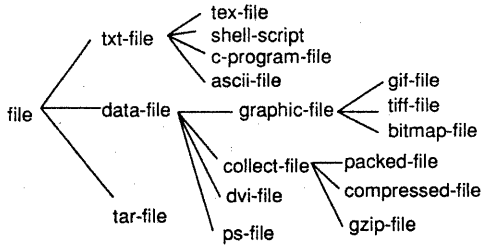

図 3: 概念階層木

事例ベースに格納された対話事例を検索するためには，インデックスが必要となる．本システムでの対話はユーザの発話によって起動されるために，事例インデックスにはユーザの発話の内部表現を用いている．たとえば,「ファイルに行番号をつけて表示したい」というユーザの発話に関する対話事例には，4.1 節に示した発話の内部表現が事例インデックスとして利用されている．

5 事例検索手法

事例検索とは，ユーザの発話に対する応答プランを得るために，事例ベースから適切な事例を検索するプロセスである．Assist-R の事例検索は 5.2 節で述べる類似度計算に基づいて行なっている．類似度の計算法を説明する前に，類似性の尺度に利用される概念階層木について説明する．

5.1 概念階層木

概念階層木は，図 3 で示すように，より抽象的な記述から特殊な記述にいたる階層型の木構造として表している．たとえば,「テキストファイル (txt-file)」という概念は,「 TEX 形式のファイル (tex-file)」,「シェルのスクリプトファイル (shell-script)」,「アスキーファイル (ascii-file)」などのファイル概念よりも上位の概念であることを表している．

概念階層木は，対話に有効であると思われる属性を中心に構成されている．たとえば，概念 “file” は「日付」,「ファイル名」,「ファイルサイズ」,「ファイルフォーマット」などの多くの属性を持っている．これらの属性のうち,「コマンドに関する問い合わせ」に関しては,「ファイルフォーマット」の属性を利用することが適切だと思われる．このため，図 3 は「ファイルフォーマット」の属性を中心にした概念階層木として構成されている．

5.2 類似度計算

類似度は，検索キーとなるユーザの発話と事例インデックスから計算される．検索キーと事例インデックスの類似度は，概念階層木における重みつき距離によって計算される．類似度の計算方法は次のように定義している．

それぞれ N 個のリテラルからなる検索キー $K = \{k_1, \cdots, k_N\}$ と事例インデックス $D = \{d_1, \cdots d_N\}$ が存在するものとする．このとき，全体の類似度は対応するリテラル間の類似度の平均によって定義される．

$$R = \frac{1}{N} \sum_{i=0}^{N} dist(k_i, d_i)$$

リテラル間の類似度 $dist(k_i, d_i)$ は，概念階層木の k_i と d_i を結ぶ経路に存在する枝ごとに定義された重み $(w_1, w_2, \cdots)$ の積である．

$$dist(k_i, d_i) = \Pi w_i$$

検索された事例のうち，最も類似度の高いものが Assist-R の事例適合部に渡される．もし類似度が等しいものが複数ある場合は，それらのうちからランダムに 1 つの事例が選択される．

6 複数事例適合メカニズム

6.1 事例適合手法

事例適合メカニズムは，検索によって得られた最も類似度の高い事例に基づいて，新たなプランを

生成する枠組みである．本研究で提案する事例適合メカニズムは図4のようになる．

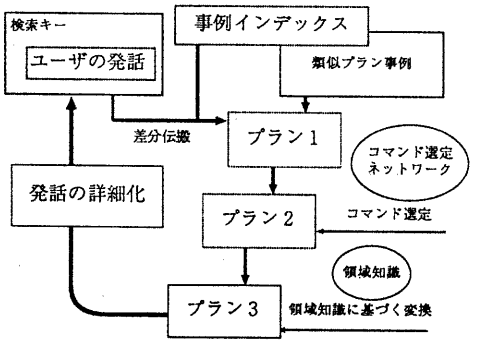

図 4: 複数事例に基づく適合メカニズム

本メカニズムは，単一の事例を利用して対話プランを生成するための**代替適合**と，複数の事例を利用してプランを生成するための**変換適合**からなる．代替適合は，図4の「プラン1」から「プラン3」までを生成する過程で，事例の内部表現を書き換える手続きである．変換適合は，代替適合が失敗した場合に，ユーザの発話を複数の発話に分解し，それぞれの発話を個別に代替適合しながら新たなプランを生成する手続きである．変換適合におけるユーザの発話の分解は，複数の発話意図を内包したユーザの発話を，複数の詳細な発話に書き換えることになるので，特に発話の詳細化と呼ぶ．

6.2 代替適合

代替適合は，図4に示されるように，差分による書き換え，コマンド選定ネットワークによる書き換え，領域知識による書き換えからなる3ステップを通して，新たなプランに書き換える手続きである．以降では，それぞれの詳細な書き換えの手続きについて述べる．

6.2.1 差分による書き換え

差分とは，ユーザの発話と事例インデックスの異なる部分を抽出したものである．たとえば，図5の「ファイルを移動したい」という事例と「ファイルを削除したい」というユーザの発話の差分について考える．

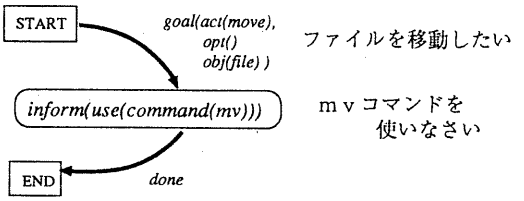

図 5: ファイル移動の対話事例

ユーザの発話「ファイルを削除したい」の内部表現は次のように得られるので，ユーザの発話と事例インデックスの差分は delete と move の対になる．

```
user_goal(act(delete),
          opt(),
          obj(file))
```

この差分によって，事例中の move を delete に書き換えると，図4における「プラン1」が生成される．

6.2.2 コマンド選定ネットワークによる書き換え

新たな Assist-R の適合メカニズムは，従来の Assist-R のような根拠ノードを持たないので，ユーザの発話と利用するコマンドを関連付けるための知識が別に必要となる．このような知識として，図6に示すコマンド選定ネットワークを定義している．

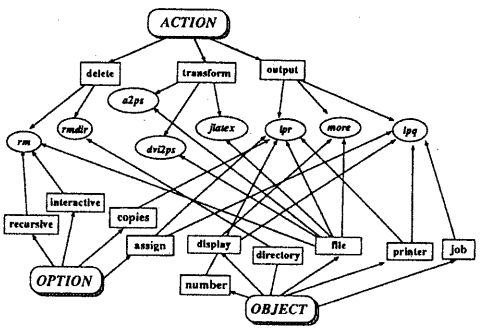

図 6: コマンド選定ネットワーク

コマンド選定ネットワークは，4章で述べた3種類の情報(コマンド情報，オプション情報，オブジェクト情報)ノードと UNIX のコマンドからなるネットワークによって構成される．ユーザの発話に対応

したコマンドは，発話から得られる3種類の情報にしたがってネットワークのノードをたどりながら選定される．なお，図6において，ACTIONはコマンド情報，OPTIONはオプション情報，OBJECTはオブジェクト情報のそれぞれの始点ノードを指している．

たとえば,「ファイルを削除したい」というユーザの発話を考える．この発話のコマンド情報はdeleteであり，オブジェクト情報はfileであるので，コマンド情報からは *rm*, *rmdir* のコマンドノードに，オブジェクト情報からは *more*, *lpr*, *rm*, *a2ps*, *dvi2ps* のコマンドノードに到達する．このため，コマンド情報とオブジェクト情報から共通に到達できる *rm* がユーザの発話意図を満たすコマンドとして選定される．

選定されたコマンドを事例のコマンド部分と置き換えれば，図4の「プラン2」にあたる事例が生成できる．なお，コマンドが複数選定されるような場合には，いずれもユーザの発話を満たしていることから，ランダムにいずれか1つのコマンドを選定するようにしている．この問題については以降で検討する．

6.2.3 領域知識による書き換え

コマンドはユーザの曖昧な発話に基づいて選定されるので，現在の対話状況において，そのコマンドが実際に利用できるとは限らない．そのため，コマンドの領域知識を用いて，コマンドの利用可能性について検討する必要がある．コマンドの領域知識は，図7に示すようなコマンドを実行するための最低限の情報によって定義される．

command-name	*rm*
object	*file directory*
option	*interactive(i)*

command-name	*rm*
object	*directory*
option	*recursive(r)*

図 7: コマンドの領域知識

ただし，同じコマンドでもオプションなどによって実行の状況が変化する場合には，複数の領域知識によって定義している．図7の例では，*rm* コマンドの利用において，再帰的に実行できるオブジェクトはディレクトリのみであること，対話的にコマンドを実行できるオブジェクトはファイルとディレクトリであることを示している．新たなAssist-Rでは，コマンドの領域知識を参照して，発話ノード中のリテラルがすべての領域知識を満していれば適合を終了し，オプションの記述などの細部を書き換えて「プラン3」を生成するようにしている．

6.3 変換適合

前述の代替適合が失敗するのは，検索キーとなるユーザの発話と実際の事例インデックスの間にギャップがあり，そのままの形では事例と適合することができないことによる．このようなギャップを埋めるためには，ユーザの発話を詳細化し，複数の事例を用いてユーザの発話意図を満たす応答プランを生成しなければならない．このような問題を解決するために，新たなAssist-Rでは，メタルールによってユーザの発話を詳細化している．

メタルールは，ユーザの発話と代替適合時の不適合部分の情報から，もとのユーザの発話意図に相当する複数の発話に分解するためのルールである．

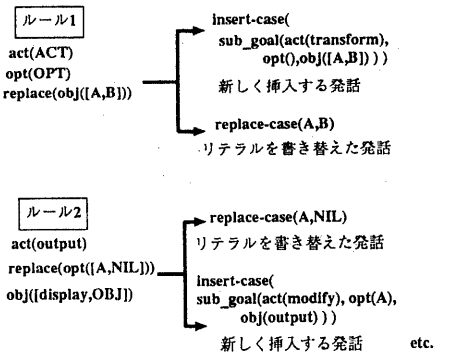

図 8: メタルール

たとえば，図8のルール1は，ユーザの発話におけるコマンド情報ACTと適合に失敗したオブジェクト情報A，Bに基づいて,「AをACTしたい」という発話を「AをBに変換したい」と「BをACTしたい」という発話に詳細化するためのルールである．ルール2は，ユーザの発話からコマンド情報が「表示」に関するものであることが分かった場合に，オブジェクト情報OBJと適合に失敗したオ

プション情報 A に基づいて,「OBJ を A という状態で表示したい」という発話を「OBJ を表示したい」と「出力を A という状態に整形したい」という発話に詳細化するためのルールである.

7 実行例

本章では，複数事例適合メカニズムを導入した新たな Assist-R におけるプランの生成過程を示す．現在，事例ベース中には図 9 に示す「ファイルの出力」に関する対話事例 A と「DVI ファイルの変換」に関する対話事例 B が存在しているものとする．ただし，図 9 中の左部分は事例の内部表現を示し，右部分はこの事例から生成される対話を示している．

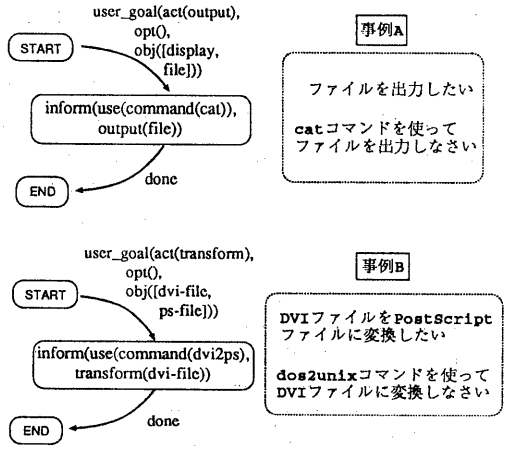

図 9: 保持事例

まず，ユーザが「アスキーファイルを印刷したい」と発話すると，Assist-R はユーザの発話を以下のような内部表現に翻訳する．

```
user_goal(act(output),
          opt(),
          obj([printer,ascii-file]))
```

このユーザの発話は，直接的には「ファイルの印刷」に関する要求を意図したものであるが，ポストスクリプトプリンターではアスキーファイルを印字することができない．このため，一旦，ファイル形式をアスキーファイルからポストスクリプト形式のファイルに変換しなければならない．言い換えると，ユーザの発話に対する応答プランを生成するためには，2 つの UNIX コマンドが必要となる．従来の Assist-R の適合メカニズムでは，複数事例について考慮していなかったために，このような応答プランを生成することはできなかった．

7.1 事例検索

まず Assist-R は過去の類似した事例を獲得するために，ユーザの発話と事例ベース中の事例インデックスにより類似度検索を行なう．前述の類似度計算法にしたがうと，図 9 の事例 A の類似度は 0.93，事例 B の類似度は 0.55 と算出される．類似事例の閾値は 0.8 以上としているので，ユーザの発話に類似した候補として事例 A が検索される．

7.2 代替適合

つぎに，Assist-R は検索された事例 A を適合メカニズムにより，ユーザの発話意図を満すプランに書き換える．適合過程の最初のステップでは，差分を用いた書き換えが行なわれる．まず，ユーザの発話と事例 A のインデックスから [display, printer], [file, ascii-file] という差分が得られる．これらの差分から，図 10 中の差分伝搬の矢印で指し示された部分で display が printer に，file が ascii-file に書き換えられる．

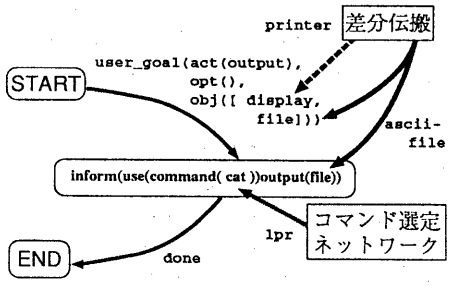

図 10: 差分による変換事例

つぎのステップでは，事例中に存在するコマンド部分がユーザの発話の意図を満たすコマンド群の中に含まれているかどうかの検証が行なわれる．Assist-R は，ユーザの発話からコマンド情報 act(output) とオブジェクト情報 obj([printer, ascii-file]) を取り出し，図 6 のコマンド選定ネットワー

クを用いてコマンド *lpr* を選定する．このとき，事例中のコマンド部分 *cat* はユーザの「アスキーファイルを印刷したい」という発話意図を満たしていないことが分かる．したがって，図 10 で示されるコマンド部分はコマンド *lpr* に置き換えられる．

最後のステップでは，コマンドの妥当性の検証が行なわれる．Assist-R が選定したコマンドが実行可能なものであるかどうかは，図 11 に示すコマンド *lpr* の領域知識を利用して検証される．

command-name	*lpr*
object	*ps-file number*
option	*assign(P)* *copies(#)*

図 11: コマンド lpr の領域知識

ユーザの発話を領域知識によって検証すると，オブジェクト情報の不適合が認識される．すなわち，ユーザのオブジェクト情報 ascii-file は領域知識で許される ps-file の概念と同等あるいは下位概念に含まれていないので，適合は失敗していることが分かる．

7.3 変換適合

代替適合が失敗したのは，ユーザの発話と対話事例の間にギャップがあったためである．適合に失敗した時の状態を図 12 の左に示す．

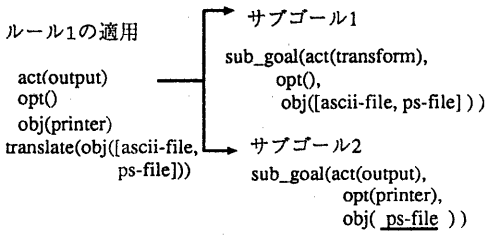

図 12: 発話の詳細化 1

このギャップを埋めるために，メタルールを用いた発話の詳細化が行なわれる．この例では，ユーザの発話のオブジェクト情報が ps-file でなく ascii-file であるという情報から，メタルール 1 が適用され，「ascii-file を印刷したい」という発話は「ascii-file を ps-file に変換したい」と「ps-file を印刷したい」という 2 つの発話に詳細化される[1]．

さらに，Assist-R は 2 つの詳細化されたユーザの発話に対して代替適合を行ない，新たなプランの生成を試みる．「ps-file を印刷したい」という発話には，類似度検索により事例 A が選ばれ「ascii-file の印刷」と同様の適合過程をたどって新たなプランが生成される．「ascii-file を ps-file に変換したい」という発話に対しては，類似検索によって事例 B が検索される．この場合，ユーザのコマンド情報は act(transform)，オブジェクト情報は obj([ascii-file, ps-file]) であるので，コマンド選定ネットワークより *a2ps*，*dvi2ps*，*jlatex* がコマンド候補として選択される．たとえば *dvi2ps* が選択される場合には，領域知識から *dvi2ps* はオブジェクト情報として dvi-file を必要とするので適合に失敗することが分かる．このようにしてコマンド候補が絞り込まれていくが，*a2ps* はオブジェクト情報の ps-file，ascii-file を満たしているので，最終的に *a2ps* が書き換えられるべきコマンドとして決定される．このようにして生成されたプランの内部表現を図 13 に示す．

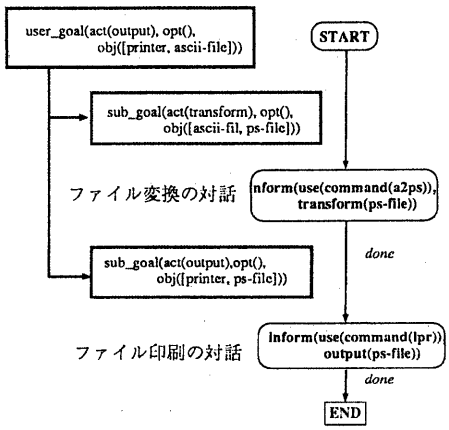

図 13: 新たな生成プラン C

生成されたプランを用いれば，Assist-R は対話

[1] 詳細化された発話は，システムから見ればプランを実行するのためのゴールであるので，サブゴールとも呼ぶ．

例1を実現できるようになる．対話の終了後，生成されたプランは新たな対話事例Cとして事例ベースに加えられる．また，サブゴールもそれぞれ部分的な事例インデックスとして格納されるので，たとえば「アスキーファイルをポストスクリプト形式に変換したい」という発話には，事例Cの部分的な「ファイル変換」に関する部分対話が個別に利用可能となる．

対話例1

ユーザ:	アスキーファイルを印刷したい
システム:	a2psコマンドを使ってポストスクリプトファイルに変換しなさい
ユーザ:	a2psコマンド実行
システム:	lprコマンドを使ってポストスクリプトファイルを印刷しなさい
ユーザ:	lprコマンド実行

8 不適切な事例の生成問題

本研究における複数事例適合メカニズムによって，事例の利用範囲が拡張されているが，事例を扱う上で発生するすべての問題が解決されたわけではない．たとえば，類似事例が複数存在する場合には，事例候補をランダムに決定すると述べたが，選択される事例によっては不適切な対話が行なわれてしまうことがあるという問題が残されている．例として，ユーザが「アスキーファイルを表示したい」という発話を行なった場合を考える．この例からは以下のような内部表現が得られる．

```
user_goal(act(output),
          opt(),
          obj([display,ascii-file]))
```

類似検索を行なうと，発話と類似した事例として事例Aと事例Cが求まる．事例Aとユーザの発話は，displayとprinterが同一レベルの抽象階層となっているので両者の距離が2となり，全体の類似度は0.93となる．事例Cでは，図3よりfileとascii-fileの距離が2となるので，全体の類似度は0.93となり，2つの事例の類似度は等しくなる．両者の事例のうち，もし誤って事例Cが採用された場合には，複数事例適合アルゴリズムによって対話例2のプランが生成されてしまう．

対話例2

ユーザ:	アスキーファイルを表示したい
システム:	a2psコマンドをつかってポストスクリプトファイルに変換しなさい．
ユーザ:	コマンド実行
システム:	ghostviewコマンドでポストスクリプトファイルを印刷しなさい．
ユーザ:	コマンド実行

もしUNIXの専門家ならば，対話例に示すような応答は行なわず，ファイルをコンソールやターミナル上に表示することを考えて「catコマンドを使いなさい」のような発話を行なうだろう．しかしながら，ユーザの発話からは曖昧な情報しか得られないので，相手が初心者の場合には，このような対話プランでもユーザの意図を反映している可能性はありうる．言い換えると，対話の不適切さは，必ずしも事前に判断できるものではなく，対話の中でユーザからの応答によって初めて認識できるものである．このような考え方から，本研究の枠組みにおいても，プランの実行中にユーザのフィードバックから不適切さを認識し，プランを逐次的に修正する立場を取っている[3]．

一方，同じ発話に対して常に同じ応答を繰り返していたのでは，対話がスムーズに進まない．ユーザから「間違っている」といった否定的なフィードバックが観測されれば，プラン生成部では次回からの検索を避けるために，事例インデックスを変更しなければならない．このような問題を解決するためには，検索時に用いる事例インデックスに重みを与え，ユーザのフィードバックから重みを逐次的に修正し，不適切な事例候補を取り除く手法が考えられる．また，ユーザのフィードバック観測から得られる意図的な情報を事例インデックスに反映させて，不適切な事例候補を取り除く手法も考えられる．

前者の重みを与えるヒューリスティックな手法では，事例に余分な情報を付加して計算量を増加させてしまう．また，ある事例の重みが変化すると，他の事例インデックスの重みも相対的に変化するので，さほど有効でない事例を適合してしまう可能性も生じる．逆に，ユーザの意図的な情報を考慮して

事例インデックスを変更する手法では，上記の対話例の場合,「コンソール上で表示したい」といった，オブジェクト情報を特殊なものに限定するようなフィードバックが得られると，インデックスの詳細化が可能となる．言い換えると，上記の例の場合，display 部分を display-non-console の概念で置き換えると，次回からの類似度を下げることができる．この手法では，ユーザの適切なフィードバックが得られなければ類似度を改善することはできないが，他の事例に影響を与えないという点では有効であると考えられる．

9 おわりに

本稿では，従来の Assist-R における事例の適合メカニズムを，代替適合と変換適合の枠組みから再構成し，より柔軟に事例を利用できることを示した．また，本稿で用いたコマンド選定ネットワーク，および領域知識の利用によって，設計者に依存する根拠ノードを事例から分離できることを示した．以上のことから，事例適合の可能性が高まることが期待される．

また，ユーザの発話を詳細化して事例を適合することにより，複数の事例から複合的にプランを生成する手法を示した．さらに，詳細化された発話を事例インデックスとすることで，複数の手順からなる複雑な事例を分解して部分利用できることを示した．このように，本手法は従来の Assist-R よりもプランの生成能力が向上していると考えられる．なお，本稿の 8 章に述べた不適切な事例の生成問題については今後の課題として残されている．

参考文献

[1] Moore, J. and Swartout, W.: A Reactive Approach to Explanation, *Proc. of AAAI-89*, pp.1504–1510 (1989).

[2] Causey, A.: Planning Interactive Acts for Explanations, Gneration, *International Journal of Man-Machine Studies*, Vol.37, pp.169–199 (1993).

[3] 荻野浩司, 上原邦昭, 前川禎男：事例に基づくプランニングを用いた対話管理手法, 情報処理学会, 自然言語処理研究会資料, No.99-1, pp.1–8 (1994).

[4] Uehara, K. and Ogino, H.: A Case-Based Approach to Text Planning, *Proc. of the IASTED International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, pp.123–127 (1994).

[5] Smyth, B. and Keane, M.: Retrieving Adaptable Cases: The Role of Adaptation Knowledge in Case Retrieval, *Lecture Notes in Artificial Intelligence*, Vol.837, pp.209–221 (1994).

[6] Birnbaum, L. and Collins, G.: A Model-Based Approach to the Construction of Adaptive Case-Based Planning Systems, *Proc. of Case-Based Reasoning Workshop*, pp.215–224 (1991).

[7] Sycara, K. and Navinchandra, D.: Index Transformation Techniques for Facilitating Creative Use of Multiple Cases, *Proc. of IJCAI-91*, pp.347–352 (1991).

[8] Cunningham, P., Finn, D. and Slattery, S.: Knowledge Engineering Requirements in Derivational Analogy, *Lecture Notes in Artificial Intelligence*, Vol.837, pp.209–221 (1994).