

事例に基づくプランニングを用いた対話管理手法

荻野 浩司 上原 邦昭 前川 禎男

神戸大学工学部情報知能工学科

神戸市灘区六甲台町 1-1

内容梗概： 本稿では、従来の対話管理手法の問題点を議論し、それらの問題点を解消するための代替案として、事例の逐次修正に基づくプランニングを用いた対話管理手法の枠組を提案する。本手法の利点として、抽象的談話構造モデルにおけるプランオペレータの利用に比べて、記述が容易であることが挙げられる。また、過去に構築したプラン事例が再利用できるので、システムは同じような推論過程を辿らずに、即座に近似解を導くことができ、計算負荷が軽減できる。さらに、事例の変形操作という考え方によって、対話の局面に応じて、現在利用中のプラン事例を逐次的に修正できるため、プラン実行段階でのユーザとの様々なインタラクションにも対応できる。

A Case Based Approach to Text Planning

Hiroshi Ogino Kuniaki Uehara Sadao Maekawa

Department of Computer Science and Systems Engineering

Kobe University

Nada, Kobe, 657 Japan

Abstract : In this paper, we will discuss the three main problems in current dialog systems, and propose a new text-planning model by using reactive revision of text plan case. This is a case-based approach that has the following advantages: (1) construction of case base is easier than that of plan operators in an abstract discourse structure model, (2) computational cost can be reduced since sub-optimal solutions may be found out without tracing complex inference which had done previously, (3) as discourse situation changes, dialog system can re-plan explanations gradually, which allow various interactions with user, by reactively revising current text plan case.

1 はじめに

現在研究開発されている多くの自然言語対話システム [1][5][7] では、対話管理方式として抽象的談話構造モデル [3] を用いたプランニングが採用されている。抽象的談話構造モデルを用いた手法の一つとして、ユーザの入力文より抽出されたゴールからプランオペレータを用いてトップダウンに展開するプランニング（以下、古典的プランニング）がある。古典的プランニングは、発話に至る理由を明確化しながらプランニングを進めるために、定型的なスキーマを用いる方法よりも比較的柔軟性が高いことが指摘されている [7]。

しかしながら、実システムの構築を念頭においた場合、古典的プランニングの枠組は以下のような点を十分に踏まえているとはいえない。

記述性 システム開発段階で知識の記述が容易であるか

効率性 システムの応答時にユーザがストレスを感じないために、いかに効率的な処理を行って計算コストを軽減するか

柔軟性 ユーザとの複雑なインタラクションに対応した柔軟な対話を実現できるか

本稿では、上記の観点から従来の対話管理手法の問題点を議論し、それらの問題点を解消するための代替案として、事例の逐次修正に基づくプランニングを用いた対話管理の枠組を提案する。本手法の利点として、抽象的知識であるプランオペレータに比べて、事例は記述が容易であることが挙げられる。また、過去に構築したプラン事例が再利用できるので、システムは同じような推論過程を辿らずに、即座に近似解を導くことができ、計算負荷が軽減できる。さらに、事例の変形操作という考え方によって、対話の局面に応じて、現在利用中のプラン事例を逐次的に修正できるため、プラン実行段階でのユーザとの様々なインタラクションにも対応できる。

2 従来の対話管理手法の問題点

記述性の問題

プランオペレータ展開の推論方式は、あるゴールを満たすためのサブゴールを探索する目標指向型推論である。換言すれば、プランオペレータはゴールが満足されるための十分条件を記述したルール型の抽象的な知識とみなすことができる。古典的プランニングでは、抽象的な知識としてのプランオペレータがあらかじめ完全に記述できることを前提とした上で、妥当なプラン体系の構築を図っている。しかしながら、人間が発話に至る綿密な推論過程を分析することは困難であるために、実際に対話システムの構築を試みる場合には、プランオペレータ設計は開発者の直観によるところが大きい。つまり、プランオペレータ設計の際の認知的負担が大きいために、知識獲得作業の緩和が重要な問題となる。

効率性の問題

一般に、ルール型の知識を用いた探索型問題解決においては、探索の際に組合せの爆発が生じ、計算負荷が大きくなる。古典的プランニングの場合には、プランオペレータを適用する際に制約条件や前提条件が充足されているかを調べる必要があるため、計算コストはさらに増大する。

また、古典的プランニングでは、同じゴールが入力されれば過去に行われた推論過程を再び辿りながらプランニングが行われるという問題がある。このため、一度構築したプランが再利用できず効率が悪いと問題がある。

柔軟性の問題

古典的プランニングを含めた従来のアプローチでは、個々のユーザの知識状態や信念状態を推論し、獲得されたユーザモデルを利用して、ユーザにとって最適な応答プランを構築することを行っている。しかしながら、複雑な推論過程によって導かれたユーザモデルが妥当であるとは限らない。また、実際には、ユーザはシステムが全く関知し得ない外界において、第三者からの教示や忘却などによって知識や信念を動的に変化させることがある。したがって、ユーザモデルの内容を常に正確な状態で維持することは不可能に近い。

ユーザモデルの獲得が困難である以上、システムがあらかじめ計画したプランが実行時に成功するという保証はない。例えば、システムが計画した説明方法では、ユーザは理解しにくいかもしれない。このような場合は、ユーザはもっと分かりやすく説明するように求めるであろう。人間の会話では、このような発話者同士のインタラクションがしばしば発生する。

従来のアプローチでは、ユーザモデルを信頼し過ぎているために、立案されたプランが実行段階において失敗しないという非現実的な仮定に立脚した上で、いかに前もって完全なプランを構築するかという方法論のみに終始している。つまり、プランの実行段階におけるユーザとのインタラクションを重要視しておらず、協調的な会話を実現することができないという問題がある。

上記のインタラクションについて、Moore [5]、Maybury [6]、Causey [7] らは、古典的プランニングの範囲での解決を試みている。しかしながら、目標指向型探索である古典的プランニングの範囲では、元来、あらかじめ予測困難なユーザからのインタラクションによるプランの逸脱を考慮していないために、柔軟性の点で限界がある。

3 事例に基づくプランニングによる対話管理の枠組

事例に基づくプランニング (Case-Based Planning) は、過去のプランを検索して現在の問題に適合させることにより新しいプランを立案する枠組である。Birnbaum [8] によると、事例に基づくプランニングの概括的な手続きは図 1 のようなものになる。

```
begin Case-Based-Planning
  ゴールから失敗を予測 (anticipation);
  ゴール、失敗から事例を検索 (retrieve);
  現在の問題に事例を適応化 (adaptation);
end Case-Based-Planning
```

図 1: 事例に基づくプランニング

まず、ゴールの特徴から予測ルールによって失敗を予測する。次に、予測された障害の回避を新たなゴールとしてゴール集合に加え、ゴール集合の中で最も多くのゴールを満たす事例を検索する。さらに、修正のためのルールによって、検索されたプランを現在の問題に適合化する。

我々は、以上のような事例に基づくプランニングの枠組が対話管理にも有効であると考え、対話管理に適用するために枠組の拡

張とともに、メカニズムの詳細設計を行った。

対話においては、ユーザとのインタラクションを考慮することが必要となる。そこで、本手法では、プランの実行時にも常に現在の対話の局面を観測し、プランを柔軟に変更できるように Birnbaum の枠組を拡張している。本研究で提案する対話管理の枠組を図 2 に示す。

```
begin Text-Planning
  失敗事例の検索 (失敗の予測);
  プラン事例の検索;
  if プラン事例 = nil
    古典的プランニング;
    プラン木の実行形式変換;
  endif
  失敗事例とプラン事例の適合化;
  失敗事例とプラン事例の統合化;
  do while 会話未終了
    プランを 1 ステップ実行;
    フィードバック観測;
    Tweak Rule 適用;
  endwhile
end Text-Planning
```

図 2: 本対話管理の枠組

まず、ユーザの入力文から抽出された発話意図 (対話ゴール) が入力される。次に、ユーザのゴールに関連して起こりうる失敗 (誤解やケアレスミスなど) を予測する。ここで、起こりうる失敗は過去の失敗事例に基づいて予測するようにしている。つまり、システムは過去の失敗事例を蓄積しているため、過去に失敗に至った状況が現在の状況と類似しているならば、失敗事例を検索することによって類似した将来の状況下での失敗が予測される。失敗事例を用いることにより、予測のためのルールを陽に記述する必要がなくなる。失敗が予測されると、その失敗を事前に回避するような戦略でプランニングを行うことができる。構築したプランが実行段階で成功する保証はないが、過去の失敗を基にして将来の失敗を予測することにより、プランの実行時における失敗がある程度吸収できる。

続いて、プラン事例の索引とゴールとの類似度に基づいて、プラン事例を検索する。類似性に着目するのは、問題領域を制限しない柔軟な検索を可能にするためである。プラン事例が検索できなかった場合は、まず古典的プランニングによってプラン木を構築する。次に、プラン木において、実際の発話行為である葉 (leaf) のみを時系列的に構造化し (中間ノードフィルタリング)、現在用いるための暫定的なプラン事例とする。システムが経験を重ねるにしたがって、プラン事例も蓄積されるので、以前に経験した問題に対しては再び古典的プランニングを行う必要がなくなり、プラン事例の直接的な適用により高速な応答が可能となる。

ただし、得られた失敗事例とプラン事例は現在の問題領域に完全に適合しているとは限らない。このため、現在の問題に失敗事例とプラン事例を適合化する。さらに、適合化された失敗事例とプラン事例を一つのプランとして統合化する。統合化は、失敗事例から予測される事態を回避できるようにプラン事例を修正することによって行われる。

この段階で構築されたプランは、対話がシステムの期待に沿って理想的に進行するという前提でプランニングされている。しかしながら、プランの実行段階では、ユーザとの様々なインタラクションが発生し、必ずしも先の前提が満たされるとは限らない。このため、プランを一度に実行してしまわずに、構築されたプランを 1 ステップずつ実行するたびに、ユーザからのフィードバック

を観測する。もし、フィードバックがシステムの期待を逸したものであれば、Tweak Rule を用いて既に立案したプランを修正変更する。Tweak Rule は、フィードバックとプランの修正操作を関連付けたルールである。

対話終了時には、次のプランニングのためにプラン履歴を事例として保存する。

4 本対話管理機構の構成法

本章では、事例に基づくプランニングによる対話管理機構の構成法を各段階毎に定式化する。

4.1 事例に基づくプランニング

4.1.1 発話プリミティブ

システムの対話管理部に入力されたユーザのゴールやフィードバック、対話管理部から出力されるシステムの応答プランは、プランニングにおいて最も下位レベルの発話行為として取り扱われる。これらの発話行為を発話プリミティブと定義する。発話プリミティブは述語論理形式のリテラルによって表現している。また、発話プリミティブの構成要素 (述語記号や項) をエンティティと定義する。

4.1.2 実行プランフロー

過去にシステムが実行したプラン履歴は、実行プランフローという表現形式で蓄積される。実行プランフローは対話局面の遷移状態をグラフで表現したものである (図 3)。グラフのノードはシステムの発話プリミティブを表し、アークは各種状態 (対象領域やユーザモデルの状態) の遷移情報とユーザの発話プリミティブを表している。これらのノードとアークをそれぞれ発話ノード、遷移アークと呼ぶ。また、grounds というラベルのアークの始点ノードでは、システムの発話 (終点ノード) に至るための根拠となる知識を記述している。このようなノードを根拠ノードと呼ぶ。

実行時には、START ノードから出発し、各時点での現在の状況に基づき遷移アークを選択しながら辿りついた発話ノードの発話プリミティブを実行し、END ノードで終了する。なお、ラベルが存在しない遷移アークについては、その遷移アークの前後のプランを続けて実行する。続けて実行されるプラン列の実行単位を 1 ステップとする。

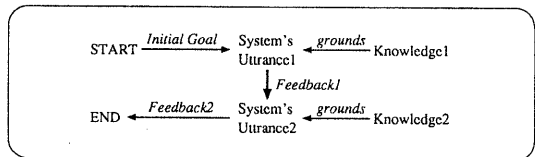


図 3: 実行プランフローの基本概念

以下では、簡単なために、特に必要がない限り実行プランフロー中の根拠ノード、遷移アークの状態遷移情報の記述を省略する。

4.1.3 事例の表現

事例には、ユーザのゴールを充足するために行われた対話の局面の遷移履歴を表すプラン事例と、ゴールを充足するための様々な障害回復の履歴を表す失敗事例がある。

プラン事例 プラン事例は実行プランフローの形式で表現される。また、プラン事例はユーザのゴールによって発火されるので、START ノードから出るアークのラベルにより索引付けされる。

失敗事例 ユーザが目標を達成するための様々な障害を失敗という。本研究では、失敗を図4のようなクラスに分けて定義している。

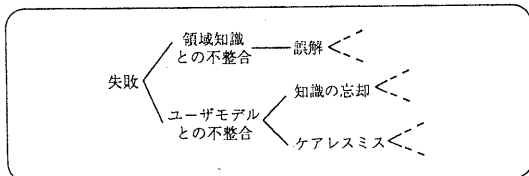


図4: 失敗のサブクラス

対話管理においては、失敗を解消したり未然に回避するために失敗事例を利用している。失敗事例は図5のようなスロットと値によって表現される。

スロット	値
Bug	失敗の内容
Situation	失敗に至る状況
Solution	失敗の解決策
Prevention	失敗の予防策

図5: 失敗事例の表現

図5では、Situation で記述された状況において Bug が発火する可能性があることを表現している。したがって、Situation スロットが失敗事例の索引となる。Solution と Prevention は対話を通じて失敗を解決したり予防した履歴であり、実行プランフロー形式で記述される。Prevention スロットによって失敗を未然に防ぐことができる。

4.1.4 事例の検索

前述のように、事例は現在の問題と完全にマッチしなくても類似したものであれば検索できる。検索は、現在の検索キーである発話プリミティブと、事例の索引中の発話プリミティブの特徴に着目して行われる。また、両者の類似度は、概念階層木におけるそれぞれの特徴の値の距離（最短経路の長さ）によって算出するようにしている¹。さらに、検索時に重要な特徴には重みを設定することができるようにしている。したがって、各特徴の類似度の重み付き総和が、二つの発話プリミティブ間の類似度となる。類似度の算出方法は図6のようになる。

N 個の特徴をもつ発話プリミティブ U_k, U_i における i 番目の特徴 F_i の値を f_{ki}, f_{ii} 、概念階層木を T として、 T における f_{ki} と f_{ii} の距離 D_i が求まる。 T において隣接するノード間の類似度を W_i ($0 < W_i < 1$)、特徴 i の重みを Wf_i (ただし $\sum_{i=1}^N Wf_i = 1$) とすると、 U_k, U_i 間の類似度 S は

$$S = \sum_{i=1}^N Wf_i \cdot W_i^{D_i}$$

となり、 S は $(0,1)$ の値域をとる。

図6: 類似度の算出アルゴリズム

類似度がある閾値を上回った事例のうちで、最も類似度が高いも

¹単語の類似性の判定にシソーラス上の概念距離を利用する方法 [9] と同様である。

が選択される。類似度の導入により、事例選択のためのヒューリスティックスを特に設定する必要がなくなるという利点が生じる。

4.1.5 事例の適合化

事例は現在の問題に完全にマッチしなくても検索できることを許しているため、検索した事例を一部修正して現在の問題に適合化しする必要がある。事例の適合化は図7のように行われる。

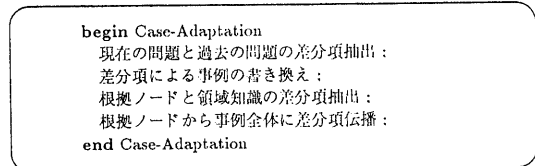


図7: 事例の適合化

まず、検索の際にキーとして用いた発話プリミティブと、検索された事例の索引を比較し、両者の差分項（一致しないエンティティ）を抽出する。次に、抽出された差分項に基づいて、索引を現在の問題に合わせて書き換える。同様にして、抽出されている差分項に相当する事例中の全てのエンティティを書き換える。続いて、事例中の全ての根拠ノードの知識と領域知識（対話の対象領域の知識）を照合する。先の書き換えによって根拠ノード内の知識が変更されている可能性があるが、領域知識自体が変更されることはない。したがって、照合の結果得られた差分項によって根拠ノード内で修正すべきエンティティが発見できる。根拠ノードの修正後は、修正箇所を発話ノード、遷移アークに伝播させて事例全体を修正する。失敗事例の場合は、上記に加えて各スロットの値も修正する。

なお、事例の適合化は領域知識との矛盾を回避するための必要最低限な修正であり、実際にはプランの妥当性はプラン実行段階にのみ確認できる。したがって、プラン実行段階においてはユーザからのフィードバックを考慮したプランの修正が必要となる。

4.1.6 事例の統合化

適合化されたプラン事例と失敗事例は、一つのプランとして統合化しする必要がある。統合化は、プラン事例内に失敗事例を組み込むことによって行われる。まず失敗事例中の Situation スロットで記述された状況を参照する。この状況下で失敗が発生する可能性があるため、対応する状況をプラン事例の遷移アークから探す。次に、失敗事例における Prevention スロットの値である実行プランフローを、発見された遷移アークの部分に挿入する。

遷移アークが発見できなかった場合は、対話中に失敗が発生する時点が特定できないので、事例を統合化せずにプラン事例のみを実行する。ただし、失敗が生じた時点で即座に修復できるように、プラン実行段階では常に失敗事例を保持しておくようにしている。

4.2 事例数が少ない段階での処理

4.2.1 古典的プランニング

事例が検索できなかった場合は、古典的プランニングによって最初からプランを構築する。古典的プランニングでは、ゴールを展開するためにプランオペレータが用いられる。プランオペレータの記述例を図8に示す。

```

planop {
  Name: explain_by_analogy.
  Effect: explain (CON),
  Constraint: analogous(CON, ANALOGUE),
  Precondition: know(user, ANALOGUE),
  Decomposition:
    [introduce_analogy (CON, ANALOGUE),
     describe_difference (CON, ANALOGUE)]
}.

```

図 8: プランオペレータの例

図 8 のプランオペレータでは、概念の類似性による説明戦略を記述している。Effect はオペレータの適用効果を表している。ゴールを満たすためには、オペレータの適用によりどのような効果が得られるかを知ればよいので、オペレータの検索は Effect とゴールとのマッチングにより行われる。また、オペレータは Constraint で指定された制約条件が満たされていない限り適用できない。Decomposition はサブゴールを記述している。ただし、Precondition で指定された前提条件が満たされていないければ、サブゴールの充足に先だってこの前提条件を充足する必要がある。Effect をさらに Nucleus と Satellites に分類したり [5]、Precondition を Essential と Desirable に分類することにより [6]、高機能化をめざしたものもあるが、本研究では簡単のため、以上のような記述にしている。

プランオペレータを用いて、古典的プランニングは図 9 のようにして行われる [6]。

```

begin Text-Planning-By-Plan-Operator
  ゴールが入力される ;
  ゴールを満たすプランオペレータ検索 ;
  変数の例示と制約条件、前提条件検査 ;
  オペレータに優先順位をつける ;
do while ゴール未充足
  最も優先度が高いオペレータを選択 ;
  サブゴールの提示 ;
endwhile
end Text-Planning-By-Plan-Operator

```

図 9: 古典的プランニング

以上のようにして、プランは木構造形式で構築される。プラン木の根 (root) がユーザのゴールを、葉 (leaf) がシステムの発話プリミティブを示している。また、古典的プランニング終了後には、完成されたプラン木や適用可能な未使用のオペレータなどのプラン履歴が保存される。

4.2.2 プラン木の実行形式変換

構築されたプラン木は、中間ノードをフィルタリングして実行プランフローに変換される (図 10)。変換においては、プラン木の葉が発話ノードとなるようにしている。また、プラン木の葉である発話プリミティブの実行から、ユーザの反応を期待して得られた発話プリミティブが遷移アークとなるようにしている。つまり、この段階では、会話がシステムにとって理想的に進行することを仮定している。

変換後の実行プランフローはプラン事例として取り扱われ、直ちに実行される。また、以降同様のゴールが入力された場合には最初から古典的プランニングを行う必要がなく、プラン事例の検索によって即座に近似解を導くことができる。

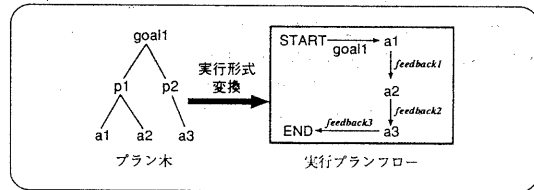


図 10: プラン木の実行形式変換

4.3 プランの実行

2 章で指摘したように、妥当なユーザモデルの構築は困難であり、あらかじめ計画したシステムの発話プランがユーザの心理状態に与える効果を完全に予測することはできない。したがって、本対話管理手法では、あらかじめ構築したプランに過度の信頼を寄せずに、むしろ実行段階におけるプランニングをより重要視するようにしている。

すなわち、本手法におけるプラン実行段階では、あらかじめ構築したプラン列をまとめて実行せずに、1 ステップの実行毎にユーザの反応 (フィードバック) を観測して再プランニングを行うようにしている。なお、再プランニングは立案したプランの修正変更によって実現している。このため、プランの修正変更のために、フィードバックに基づく事例の修正方法を記述したルール (Tweak Rule) を導入する。

4.3.1 フィードバックの観測

本研究では、Maybury [6] が挙げた 13 種類のフィードバックを追加変更して取り扱っている。フィードバックはユーザのゴールを達成するための表層的な発話行為であるといえるので、従来手法では、ユーザの反応から推定されたゴールを基に新たにプランニングを行うものが多い。本手法では、既に立案されたプランのフィードバックに基づく修正変更というアプローチをとっている。このため、ユーザの意図推定のための複雑な推論過程を軽減できる。

本対話管理機構で用いるフィードバックの例を図 11 に示す。

フィードバック	例
understanding	わかりました。なるほど
affirmation	その通りです
elaboration(X)	なにそれ? X とは何ですか?
confusion	はあ? よく分かりません
rejection	黙れ。もう十分です
goal(X)	X を達成したい

図 11: フィードバックの例

本手法では、以上のようなフィードバックを発話プリミティブとして取り扱っている。また、同一カテゴリのフィードバックはグループ化して定義することができる。例えば、understanding, affirmation は positive として、elaboration(X), confusion, rejection は negative としてグループ化することができる。このようなフィードバックのグループ化により、Tweak Rule の記述の一般化を図っている。

システムが発話に対するユーザのフィードバックが受理できれば、観測は成功したといえる。このような観測を受動的観測という。しかしながら、寡黙なユーザとの対話においてはフィードバックが得られないために、実行したプランの成否の裏付けが困難な場合がある。また、ユーザのフィードバックは曖昧な口語表現で

入力されることが多いので、ユーザの発話文から妥当なフィードバックを一意に絞りこむことが困難な場合もある。以上のような場合には、プランニングの際の情報収集のために、「わかりましたか？」などのフィードバック獲得戦略を利用して観測する必要がある。このような観測を積極的観測という。

4.3.2 プランの逐次修正

実行段階でのプラン修正は、実行プランフローの変形操作によって行われる。変形操作は、図 12 のような基本変形オペレータを組み合わせて行われる。基本変形オペレータは図 12 の他にも分岐、置換などがある。

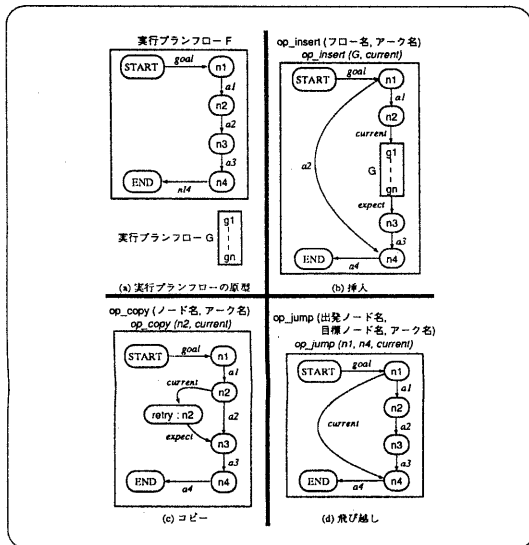


図 12: 基本変形オペレータ

例えば、図 12 (b) は *current* という遷移アークの後に実行プランフロー G を挿入するという基本変形操作を表現している。ここで、*current* は実行プランフローにおける現在の対話局面へのポインタを示すラベルである。

観測されたユーザのフィードバックに基づいて、実行プランフローの変形操作を決定するためのルールを Tweak Rule という。Tweak Rule により、プラン実行段階におけるフィードバックを考慮したプランの逐次修正が可能となる。Tweak Rule は図 12 の基本変形オペレータを用いて図 13 のように定義される。

フィードバック	変形操作
positive elaboration(X)	do_nothing op_insert(explain(X), current) & op_copy(previous, next)
confusion rejection	op_insert(re_explain(X)), current) op_jump(current, Next-Topic)

図 13: Tweak Rule の例

例えば、フィードバックが elaboration(X) である場合は、現在の対話局面の後に X を説明するための実行プランフローを挿入し、フィードバックの原因となったプランを挿入後の位置にコピーするという変形操作が行われる。

5 実行例

本章では、UNIX の利用に関する対話を例題として、事例に基づくプランニングによる対話管理過程を示す。初めに、システムが図 14 のようなプラン事例のみをもっていると仮定する。

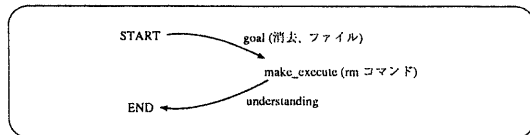


図 14: プラン事例

図 14 のプラン事例では、図 15 の対話 1 のような発話プランが構築できる。

ユーザ (1) ファイルを消したい。
システム (1) rm コマンドを使ってください。

図 15: 対話例 1

次に、ユーザが図 16 の対話例 2 においてユーザ (1) のような発話文を入力したとする。

ユーザ (1) 間違えてファイルを消してしまいました。
システム (1) 復活はできません。
ユーザ (2) ええっ!
システム (2) こんな時のために、今後は念のために rm に -i オプションをつけた方がいいですよ。
ユーザ (3) するとどうなるの?
システム (3) rm でファイルを消す時に確認のメッセージが表示されます。

図 16: 対話例 2

この発話文はシステムの文解析部によって、

$\text{not}(\text{intend}(\text{消去}(\text{ファイル})) \wedge \text{fact}(\text{消去}(\text{ファイル})))$

と認識される。これは「ファイルを消去するつもりはない」「ファイルを消去した」という二つの概念の and 結合である。ここで、システムは

$\text{not}(\text{intend}(X)) \wedge \text{fact}(X) \rightarrow \text{careless_miss}(X)$

という矛盾発見のルールを持っている。これは、「X を実行する意図がないにも関わらず X を実行したならば、X はケアレスミスによる行為である」という意味である。図 4 の定義からケアレスミスは失敗であるので、ユーザは失敗に遭遇していると認識される。認識された失敗は、図 4 の木において認識されたノードのパス構造であり、このパス構造を *bug₁* とおくことにする。

ここで、ユーザが自分自身の失敗を自覚しているかどうかを検証する。この例の場合はユーザ自身が失敗を明言しているために、自分自身の失敗を自覚しているといえる。このため、ユーザのゴールは失敗の解消、つまり goal(do_debug(*bug₁*)) となる。ユーザが自分自身の失敗を自覚していない場合は、ユーザは誤解をしていると判断できるので、誤解を解消するためのゴールを作る。

次に、得られたゴールをキーとして、失敗を回避するために失敗事例を検索する。残念ながら、この場合は失敗事例が検索できなかったために、古典的プランニングによって図 17 のように始めからプランを構築する必要がある。

ここで、構築されたプラン木は実行プランフローの形式に変換され、逐次的に実行される(システム (1), (2))。対話例 2 の終了後、実行プランフローは図 18 のような失敗事例形式に変換され、蓄積される。

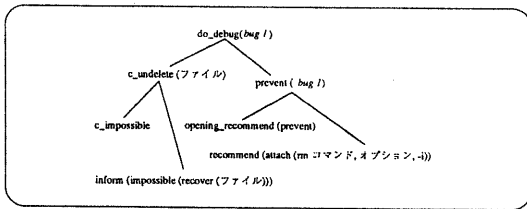


図 17: 新規失敗解消のためのプラン木生成

スロット	値
Bug	bug ₁
Situation	消去 (ファイル)
Solution	inform(impossible(recover(ファイル)))
Prevention	opening_recommend (prevent), recommend(attach(オプション,rm,-i))

図 18: 得られた失敗事例

ユーザ (1) ディレクトリを移動したい
 システム (1) mv コマンドを使えばよいのですが、念のために -i オプションをつけた方がいいですよ。
 ユーザ (2) (mv -i コマンドを実行)
 ユーザ (3) 移動できない。
 システム (3) ファイルシステムを越えた移動を行っていますね?
 ユーザ (4) ファイルシステムとは何ですか?
 システム (4) ハードディスクを論理的に切り分けたそれぞれの部分をファイルシステムといいます。
 ユーザ (5) はあ?
 システム (5) 例えば、df コマンドを実行してみてください。
 ユーザ (6) (df コマンドを実行)
 システム (6) 各行の最初に出力されているのが、切り分けられたそれぞれのファイルシステムです。
 ユーザ (7) ふむふむ。
 システム (7) 各行の最後にはディレクトリが表示されていますね?
 ユーザ (8) はい。
 システム (8) 移動したいディレクトリは、どのディレクトリの下にありますか?
 ユーザ (9) /usr の下にあります。
 システム (9) 移動先のディレクトリは?
 ユーザ (10) /home の下です。
 システム (10) /usr の下のディレクトリから /home の下のディレクトリへの移動は、ファイルシステムを越えた移動ということになります。
 ユーザ (11) なるほど。
 システム (11) ファイルシステムを越えたディレクトリの移動はできません。
 ユーザ (12) じゃあどうすればいいの?
 システム (12) cp コマンドでディレクトリのコピーを行ってから rm コマンドで元のディレクトリを消して下さい。

図 19: 対話例 3

続いて、図 19 における対話 3 の処理を考える。まず、発話ユーザ (1) は

goal (移動 (ディレクトリ))

というゴールとして認識される。次に、このゴールをキーとして過去の失敗事例を検索する。この場合、図 18 の失敗事例が最も類似度が高い事例として検索される。また、過去のプラン事例を検索し、図 14 の事例が選択されるので、得られたプラン事例と失敗事例を適合化する。適合化段階では、事例中のエンティティである「消去」が「移動」に、「rm」が「mv」にそれぞれ置換される。さらに、プラン事例中で失敗事例の Situation がマッチする部分に失敗事例を挿入して二つの事例を統合化する。以上のようによって、図 20 のようなプランが構築される。

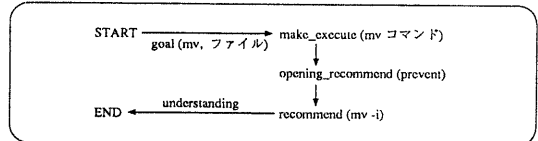


図 20: 統合化された事例

プラン実行段階においては、まず最初の発話プランであるシステム (1) を実行する。このようにして、対話 2 でユーザが失敗した経験から、システム (1) の実行によって同様の失敗を事前に回避している。

ここで、ユーザのフィードバックとしてシステムが期待する understanding に反してユーザ (3) が観測されているために、新たなゴール

recover_to_fail (移動 (ディレクトリ))

を生成してプランニングを行う必要がある。

recover_to_fail (移動 (ディレクトリ)) と照合する事例は検索されなかったため、古典的プランニングによって図 21 のようなプラン木を生成し、図 22 のようなプラン事例形式に変換する。

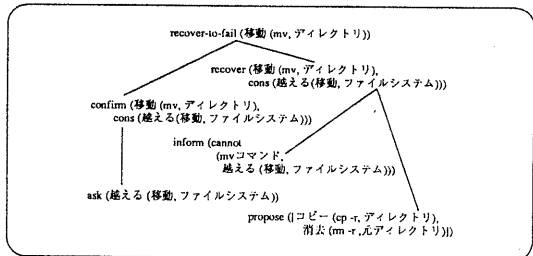


図 21: 新規ゴール充足のためのプラン木生成

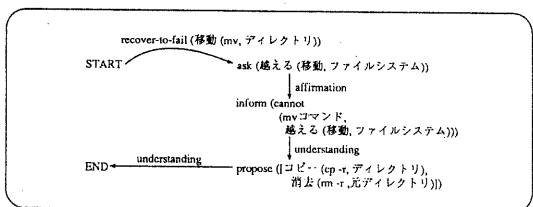


図 22: 得られたプラン事例

図 22 の事例により、システム (3) のプラン事例が実行できる。しかしながら、ユーザは再びシステムの期待を裏切り、フィードバック elaboration (ファイルシステム) を発話する (ユーザ (4))。このため、Tweak Rule を利用して現在のプラン事例を図 23 のように修正する。

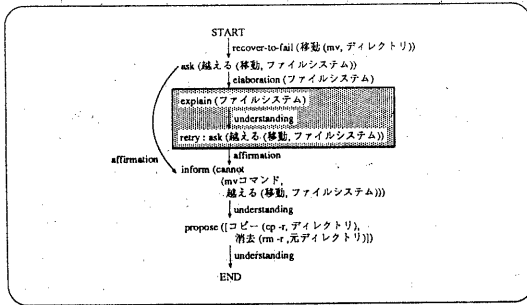


図 23: 修正されたプラン事例

同様に、修正されたプラン事例に基づいて、システムはプラン実行を続けることができる。ユーザ (5) では再び Tweak Rule を適用して、図 24 のようなプラン事例に修正されている。

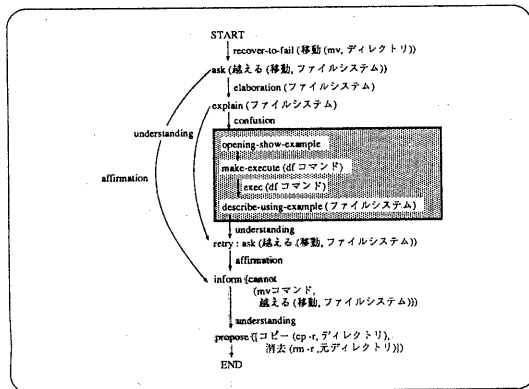


図 24: 再び修正されたプラン事例

このような処理過程によって、会話 3 のシステム (5) 以下の対話例を実現することができる。対話終了後の実行プランフローは、新たな事例として蓄積される。

6 おわりに

本稿では、対話管理機構として事例に基づくプランニングを採用するという新しいパラダイムを提案した。システム設計者は単に人間の対話例を事例として記述すればいいので、人間らしい自然な対話が可能である。ただし、システム設計者が直接事例を記述する際には、根拠ノードを注意深く設計する必要がある。しかしながら、事例の登録時に事例から後向き探索によってプラン木を構築し、得られたプラン木中の情報から根拠ノードを自動に生成できるようにすれば、根拠ノードの記述を省略できる。

また、古典的プランニングによって構築されたプランは、プラン事例形式に変換できるために効率性の問題も克服できる。しかしながら、システムが経験を重ねるにしたがって蓄積された事例数は増大し、一つ一つの事例も肥大化するために、事例の検索効率が低下する。このため、保持できる事例数の上限を設定し、事例数が指定された数を上回った場合は利用頻度の低い事例を削除する必要がある。同様に、一つの事例中で利用頻度の低いノードの削除も考慮しなければならない。さらに、同じような事例を一般化すれば事例数は削減できるが、事例の一般化に伴って事例の検索や適用段階での計算コストは増加するというトレードオフ

が発生する。したがって、システムの領域量や計算量を考慮して事例の評価関数を設定し、各時点での事例集合の評価値に基づいて動的に一般化や特殊化を繰り返す必要がある。以上のように、事例管理機構を新たに構築する必要がある。

さらに、プランの実行段階では、1 プランステップの実行毎に観測されたユーザのフィードバックから Tweak Rule によって現在のプランを逐次的に修正している。実行時の状況に対応して動的にプランを修正しているため、プラン実行段階での対話管理はプランニング/実行観測のインターリーブによるリアクティブ・プランニング [4][10] の枠組によって実現されていると位置づけることができる [2]。現実の対話によっては、対話局面の状態遷移と同時に対話の対象領域も動的に変化するものもあるので、対話の対象領域毎のリアクティブ・プランニングを考える必要がある。また、インターリーブ方式によるリアクティブ・プランニングでは、対象世界への追従のために、プランニング/観測実行の切り替えタイミングの決定が重要な問題であると指摘されている [10]。対話においては、長い説明をどこで区切ってユーザのフィードバックを観測するかという説明量を考慮し、現在の説明量がある限界を超えた時点でいったん観測に移るなどの管理が必要である。しかしながら、本対話管理手法においては、1 ステップ毎にプランニング/観測実行を切り替えることによって枠組を単純化している。

参考文献

- [1] 細見格, 荻野浩司, 上原邦昭, 前川禎男: 領域知識内での構造写像を用いた動的概念の説明手法, 情報処理学会, 人工知能研究会資料, 86-4, pp.25-32 (1993).
- [2] 荻野浩司, 上原邦昭, 前川禎男: リアクティブ・プランニングによる発話文生成手法, 電気関係学会関西支部連合大会, G8-41 (1993).
- [3] Mann, W. C.: Discourse structures for text generation, Proc. of the International Conference on Computational Linguistics, pp.267-330, MIT Press (1983).
- [4] Georgeff, M. P., Lansky, A. L.: Reactive Reasoning and Planning, Proc. of AAAI-87, pp.677-682 (1987).
- [5] Moore, J. D. and Swartout, W. R.: A Reactive Approach to Explanation, Proc. of AAAI-89, pp.1504-1510 (1989).
- [6] Maybury, M. T.: Communicative acts for explanation generation, International Journal of Man-Machine Studies, Vol.37, pp.135-172 (1992).
- [7] Causey, A.: Planning interactive explanations, International Journal of Man-Machine Studies, Vol.38, pp.169-199 (1993).
- [8] Birnbaum, L. and Collins, G.: A Model-Based Approach to the Construction of Adaptive Case-Based Planning Systems, Proc. of 1991 DARPA Case-Based Reasoning Workshop, pp.215-224 (1991).
- [9] 佐藤理史: MBT2: 事例に基づく誤語選択, 人工知能学会誌, Vol.6, No.4, pp.592-600 (1991).
- [10] 山田誠二: インターリーブによるリアクティブ・プランニング, 情報処理学会, 人工知能研究会資料, 79-8 (1991).