

12

実世界におけるプランニング と行為

ここでは、より豊かな表現とよりインタラクティブなエージェントアーキテクチャが、どのように実世界で役に立つプランナを導くかを見していく。

前章ではプランニングにとって最も基本的な概念、表現、そしてアルゴリズムについて述べた。実世界で使われるプランナはハッフル宇宙遠鏡を使った観測スケジュール、工場の運営、あるいは、軍事行動の兵站業務(logistics)を扱うタスクなど、もっと複雑である。そのため、記述言語(representation language)の基礎とプランナが環境と相互作用する方法を拡張する。この章では、12.1節において時間と資源制約(resource constraint)がある場合のプランニングとスケジューリングを述べ、12.2節において事前に定義された副プランによるプランニングを述べる。12.3章から12.6章では、不確かな環境に対するために設計されたエージェントアーキテクチャについて述べる。12.7節では、環境に他のエージェントが存在する場合にどのようにプランするかを示す。

12.1 時間、スケジュール、資源

STRIPS表現は、どの行為(action)を実行するかを表している。しかし、その表現は状況計算(situation calculus)に基づいているため、ある行為が別の行為の前か後ろかを除いては、どのくらい長く行為を実行するのか、あるいは、いつ行為が起ころのかを表現できない。しかし、ある分野では、行為がいつ始まつていつ終わるかを知りたい。たとえば、貨物配達分野では貨物を載せた飛行機が飛行し到着することではなく、いつ到着するのかを知りたいのである。

時間は、ジョブシップスケジューリング(job shop scheduling)とよばれる種々のアプリケーションでは重要である。ここでタスクは一組のジョブの完了を要求し、個々のジョブは一連の行為からなる。さらに各行為は実施する期間(duration)をもち、なんらかの資源を必要とする場合がある。ここでの問題は、資源制約を考慮しながら、すべてのジョブを完了するために必要な総時間を短縮するスケジュールを決めることである。

図12.1にジョブシップスケジューリング問題の例を示す。これは簡略化された自動車組立問題であり、自動車 C_1 と C_2 を組み立てるという二つのジョブを考える。個々のジョブは三つの行為、すなわち、エンジン追加、結果検査からなる。エンジン

```

Init(Chassis(C1) ∧ Chassis(C2)
    ∧ Engine(E1, C1, 30) ∧ Engine(E2, C2, 60)
    ∧ Wheels(W1, C1, 30) ∧ Wheels(W2, C2, 15))
Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e, c),
    PRECOND: Engine(e, c, d) ∧ Chassis(c) ∧ ¬EngineIn(c),
    EFFECT: EngineIn(c) ∧ Duration(d))

Action(AddWheels(w, c),
    PRECOND: Wheels(w, c, d) ∧ Chassis(c) ∧ EngineIn(c),
    EFFECT: WheelsOn(c) ∧ Duration(d))

Action(Inspect(c),
    PRECOND: EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c),
    EFFECT: Done(c) ∧ Duration(10))

```

図12.1 2台の自動車を組み立てるジョブシップスケジューリング問題。Duration(d)表記は、行為の実行に d 分かかるという意味を表している。Engine($E_1, C_1, 60$)はエンジン E_1 をシャーシ C_1 に取り付けるのに 60 分かかることを意味している。

は最初に取り付けなければならない（なぜなら、前輪を取り付けるとエンジン部分を扱えないからである）。そして、検査はどちらか最後に行われるべきである。

図12.1の問題は、今までに見てきたどんなプランナでも解決できる。图12.2は（もし数字だけを無視すれば）半順序プランナPOPが見いたした解を示している。この問題をプランニング問題よりもしろスケジューリング問題として捉えるためには、行為の順序(ordering)だけでなく期間(duration)も考慮しながら、各行為がいつ始まりいつ終了するかを決めなければならない。また、行為を実行しているDuration(d)表記は、行為を完了するのに d 分かかることを意味している。 $(d$ は数字でなければならない $)$ 。

クリティカルパス法

クリティカルパス法(critical path method: CPM)を適用して、各行為の可能な開始時間と終了時間を見定すことができる。半順序プランのパス(path)は線形に順序化された一連の行為であり、Startで始まりFinishで終わる(たとえば、图12.2では半順序プランにおいて二つのバスが示されている)。

クリティカルバス(critical path)はバスの合計期間が一番長いものを指し、それは全体プランの期間を決定するので“重要(critical)”である。つまり、他のバスを短縮しても全体プランの短縮にはならないが、クリティカルバス上のどんな行為の開始の遅延は全体プランを遅らせる。图におけるクリティカルバスは太線で表されている。最短時間で全体プランを完了させるためには、クリティカルバス上の行為はそれぞれの間で遅れなしに実行されなければならない、クリティカルバス上にない行為はいくらかの余地、すなわち、行為の実行開始時間に幅(window)がある。その幅は最早開始可能時間(earliest possible start time)ESと最遅開始可能時間(latest possible start time)LSを用いて特定される。 $LS - ES$ は行為の余裕(slack)として知られている。图12.2では全体プランを実行するのに 80 分かかり、いつもそうであるようにクリティカルバス上の各行為の余裕は 0 であるが、 C_1 の組立てに関する行為は開始時間に 10 分の幅がある。このように全行為におけるES時間とLS時間は、問題のためのスケジュール(schedule)を構成する。

余裕

スケジュール

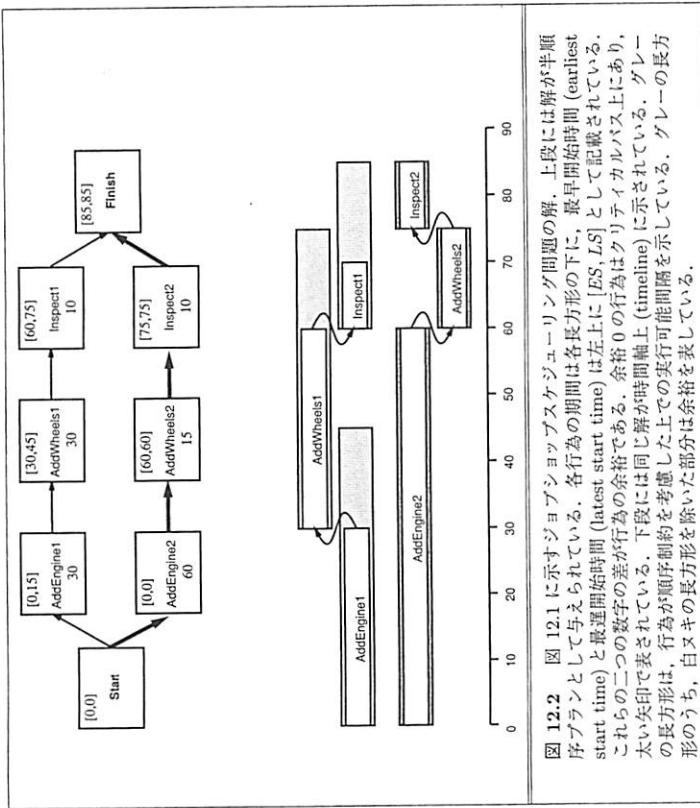


図 12.1 に示すジョブショップスケジューリング問題の解。上段には解が半順序プランとして与えられている。各行為の期間は各長方形の下に、最早開始時間 (earliest start time) と最遅開始時間 (latest start time) は左上に [ES, LS] として記載されている。これらの二つの数字の差が行為の余裕である。金額 0 の行為はクリカルバス上にあり、太い矢印で表されている。下段には同じ解が時間軸上 (timeeline) に示されている。グレーの長方形は、行為が順序制約を考慮した上で実行可能箇所を示している。白又キの長方形のうち、白又キの長方形を除いた部分は余裕を表している。

次の式は、 ES と LS の定義を与え、それらを計算するための動的計画アルゴリズム (dynamic programming algorithm) の概略を提供する：

$$\begin{aligned} ES(Start) &= 0 \\ ES(B) &= \max_{A \prec B} ES(A) + Duration(A) \\ LS(PFinish) &= ES(PFinish) \\ LS(A) &= \min_{A \prec B} LS(B) - Duration(A) \end{aligned}$$

まず、 $ES(Start)$ に 0 を割り当てるところから始める。次に、行為 B を実行し、行為 B の直前に起る全行為に ES 値が割り当てられる ($ES(B)$)。この $ES(B)$ は行為 B より前に起こった行為の中で最早終了時間 (earliest finish time) の最大値を設定する。ここで、行為の最早終了時間とは最早開始時間にその期間を加えたものとして定義される。このプロセスは、すべての行為に ES 値が割り当たるまで繰り返される。 LS 値もこれと似たような方法で計算され、 $PFinish$ 行為から後向きに設定される。詳細は練習問題に譲ろう。

クリカルバスアルゴリズムの複雑さはほんの $O(Nb)$ である。ただし、 N は行為の数、 b は行為への最大分岐数 (maximum branching factor)、あるいは、行為からの最大分岐数とする (これを理解するには、 LS と ES が行為ごとに計算され、各計算は高々 b 個の他の行為数分を繰り返しだけであること注意されたい)。それゆえ、最小期間スケジュールを見つける問題は、半順序行為が与えられれば容易に解決できる。

資源制約ありスケジューリング

資源のスケジューリング問題は、資源 (resource) の制約によって複雑になる。たとえば、自動車にエンジンを取り付けるためにはエンジン釣り上げ装置 (engine hoist) が必要になるが、もしその装置が 1 台しかなければ、自動車 C_1 にエンジン E_1 を取り付けることは同時にできず、図 12.2 に示されているスケジュールは実行不可能になる。このようなエンジン釣り上げ装置は、行為を実行している間は“占有”されているが、行為が終了すると再び利用可能になるという意味で、再利用可能な資源 (reusable resource) の例となる。ここで、再利用可能な資源は前提条件 (precondition) と結果 (effect) という点で行為の標準的な記述では扱えないことに注意しよう。なぜなら、行為完了後に利用可能な資源の量は変化しないからである。¹ このような理由から、RESOURCE($R(k)$) というフォームを含むように表現を拡張する。これは、行為が k ユニット (unit) の資源 R を要求することを意味する。このような資源要求は、資源が使用不可であれば行為は実行できないという意味で必須であり、また、行為が k ユニット分減少するという意味で一時的な結果 (temporary effect) である。図 12.3 は、“エンジン取り付けに必要なエンジン釣り上げ装置”，“車輪をつけるための車輪ステーション”，“2 人の検査官”という三資源を含むように拡張されたエンジン組立て問題を示す。また図 12.4 は、最早完了時間 (fastest completion time) が 115 分という解を示す。この値は資源制約なしのスケジュールで要求される 80 分よりも長いが、2 人の検査官を同時に必要とする時間がないので、2 人のうちの 1 人をすぐにもっと生産性のある場所に移動できることに注意されたい。

$Inspector(I_1)$ と $Inspector(I_2)$ のように名づけられた実体よりむしろ $Inspectors(2)$ のように数量として資源を表現する方法は、集約 (aggregation) と呼ばれる一般的な技術である。集約の主たる考え方は、目的に關してオブジェクトを区別できないとき、個々のオブジェクトを量として扱うことである。たとえば、組立て問題では、どの検査官が自動車を検査するかは問題にはならず、区別する必要がない (同じような考え方方が、練習問題 3.9 に示す宣教師と人食い人種 (missionaries-and-cannibals) の問題に使える)。集約は複雑性を減少させるために必須である。ここで、同時に $Inspect$ 行為が 10 個必要となるスケジュールが提案されたときに検査可能な検査官が 9 人の場合、何が起こるかを考えてみよう。量として表された検査官を用いると、すぐにな足が見つかるので別のスケジュールを試すためにバックトラック (backtrack) する。しかし、個人単位で表された検査官を用いると、検査官を $Inspect$ という行為に割り当てる 101 通りを試すためにバックトラックするが、大部分は無駄になる。

このような利点にもかかわらず、行為間の相互作用が新たに導入されると、資源制約はスケジューリング問題をさらに複雑にする。クリカルバス法を用いた制約のないスケジューリングは簡単であるが、最早可能完了時間で資源制約スケジュールを見いだすことは NP 困難である。この複雑性は理論と同様、実際問題においてもよくみられる。1963 年に、10 台の機械と 100 個の行為をもつ 10 個のジョブにおいて最適なスケジュールを見つけるという挑戦的な問題が提出されたが、23 年間解決されなかつた (Lawler et al., 1993)。特に、分枝限定法 (branch-and-bound), 焼きなましま法 (simulated annealing), タ

¹ これに対して、エンジンを組み立てるボルトなどの消費可能資源 (consumable resources) は標準的な記述で扱うことができる。練習問題 12.2 を参照のこと。



```


$$\begin{aligned}
& \text{Init}(\text{Chassis}(C_1) \wedge \text{Chassis}(C_2) \\
& \quad \wedge \text{Engine}(E_1, C_1, 30) \wedge \text{Engine}(E_2, C_2, 60) \\
& \quad \wedge \text{Wheels}(W_1, C_1, 30) \wedge \text{Wheels}(W_2, C_2, 15) \\
& \quad \wedge \text{EngineHoists}(1) \wedge \text{WheelStations}(1) \wedge \text{Inspectors}(2)) \\
& \text{Goal}(\text{Done}(C_1) \wedge \text{Done}(C_2))
\end{aligned}$$


```

```


$$\begin{aligned}
& \text{Action}(\text{AddEngine}(c, c), \\
& \quad \text{PRECOND: Engine}(e, c, d) \wedge \text{Chassis}(c) \wedge \neg \text{EngineIn}(c), \\
& \quad \text{EFFECT: EngineIn}(c) \wedge \text{Duration}(d), \\
& \quad \text{RESOURCE: EngineHoists}(1)) \\
& \text{Action}(\text{AddWheels}(w, c), \\
& \quad \text{PRECOND: Wheels}(w, c, d) \wedge \text{Chassis}(c) \wedge \text{EngineIn}(c), \\
& \quad \text{EFFECT: WheelsOn}(c) \wedge \text{Duration}(d), \\
& \quad \text{RESOURCE: WheelStations}(1)) \\
& \text{Action}(\text{Inspect}(c), \\
& \quad \text{PRECOND: EngineIn}(c) \wedge \text{WheelsOn}(c), \\
& \quad \text{EFFECT: Done}(c) \wedge \text{Duration}(10), \\
& \quad \text{RESOURCE: Inspectors}(1))
\end{aligned}$$


```

図 12.3 2 台の自動車を組み立てる資源ありジョブショップスケジューリング問題。利用可能な資源は一つのエンジン組立てステーション、一つの車輪組立てステーション、2 人の検査官である。RESOURCE 表記は、資源 r は行為実行中に使用されるが、行為が完了すると再び使用可能になるという意味である。

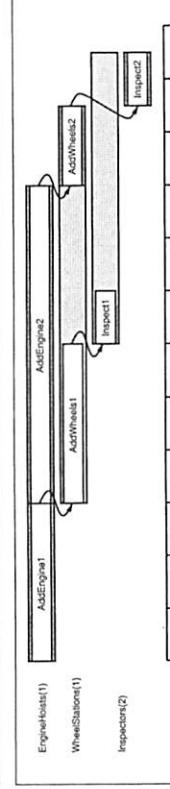


図 12.4 図 12.3 に示す資源ありジョブショップスケジューリング問題の解。資源名は左に三つ記載され、行為は各資源ごとに水平に示されている。ここでは、二つの可能なスケジュールがあり、どちらが先にエンジンステーションを使用するかによって決まる。最適解は 115 分である。

ブー検索 (tabu search)、制約充足 (constraint satisfaction)、そして、II 部に示した他の手法など、数多くのアプローチが試されてきた。シンプルだが有名なヒューリスティック手法として、最小余裕 (minimum slack) アルゴリズムがある。このアルゴリズムは欲張り (greedy) な方法で行為をスケジュールする。具体的には、各試行で現在スケジュールされていない行為の中で、次にスケジュール可能な行為に着目し、それをできるだけ早く開始させるために最小限の余裕でスケジュールする。そして、その影響を受けた各行為の ES と LS 時間を更新し、(このサイクルを) 繰り返す。このヒューリスティック手法は制約充足において最も制約された変数 (most-constrained-variable) に関するヒューリスティックと同様理論に基づいている。これは実際よく機能するが、図 12.4 に示す組立て問題の

られる。

“純 (pure)” HTN プランニングでは、プランは連続した行為分解によってのみ作られる。それゆえ、HTN ではプランニングを（状態空間や半順序プランニングのように）空の行為から行為記述を構築するプロセスとして見なすよりもしろ、行為記述を具体化するプロセスとして見なす。STRIPS における行為記述は行為記述は行為記述によって変換することができます（練習問題 12.6 を参照）。その半順序プランニングは純 HTN プランニングの特殊なケースとして見なすことができる。しかし、特に “目新しい (novel)” 運言ゴールなど、あるタスクにとっては純 HTN の見方は、どちらかといえば不自然である。そのため、未解決条件 (open condition) を設定し、順序制約を加えることによって競合を解決する標準的操作に加えて、行為分解を半順序プランニングにおけるプラン洗練として使用するハイブリッドアプローチを見出すほうが好ましい。(HTN プランニングを半順序プランニングの拡張版と捉えると、すべて新しい表記方法を導入する代わりに同じ定式表記 (notational convention) を使用できるという利点がある)。以上より、行為分解を詳しく述べることから始めよう。続いているように半順序プランニングアルゴリズムを修正しなければならないかを説明し、最後に完全性、複雑性、現実性に関して議論する。

行為分解の表現

行為分解法における一般的な記述はプランライブラリ (plan library) に保存され、構築されるプランの必要性に応じて記述は抽出 (extract) され、具体化 (instantiate) される。個々の方法は $Decompose(a, d)$ という形式で表現される。ここで行為 a は、11.3 節で述べられているように半順序プランとして表現されたプラン d へ分解される。

家を建てるという例は良い具体例なので、行為分解の概念を説明するのに使用しよう。図 12.5 では、*BuildHouse* という行為を四つの下位レベルの行為に分解する一例を示す。また、図 12.6 では、*BuildHouse* の分解と同じように、問題を解くための行為記述のいくつかを示している。なお、ライブラリには他の可能な分解があるかもしれない。

分解に関する *Start* 行為は、他の行為によって導びかれないプラン上の行為の前提条件をすべて与える。これらの前提条件を外的的前提条件 (external precondition) とする。この例では、分解に関する外的的前提条件は *Land* と *Money* である。同様に、*Finish* 行為の前提条件である外的結果 (external effect) は、他の行為によって否定されないプラン上の行為の金結果である。例にある *BuildHouse* の外的結果は *House* と $\neg Money$ である。HTN プランナのいくつかは、*House* のような主要結果 (primary effect) と $\neg Money$ のような副次結果 (secondary effect) を区別している。ゴールを達成するために主要結果のみが使用される場合もあるが、主要と副次の結果が他の行為との競合を引き起こす可能性もある。しかし、このような競合が探索空間を大いに減少させる。²

分解は行為を正しく実行しなければならない。行為 a の前提条件が与えられた状態で、もしプラン d が a の結果を達成するための完全かつ競合のない半順序プランであれば、プラン d は行為 a を正しく実行する。もし健全な半順序プランを実施した結果であれば、明

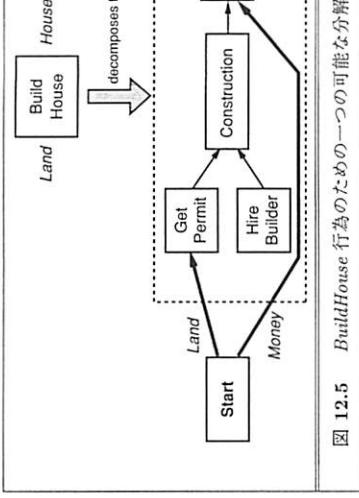


図 12.5 *BuildHouse* 行為のための一つの可能な分解。

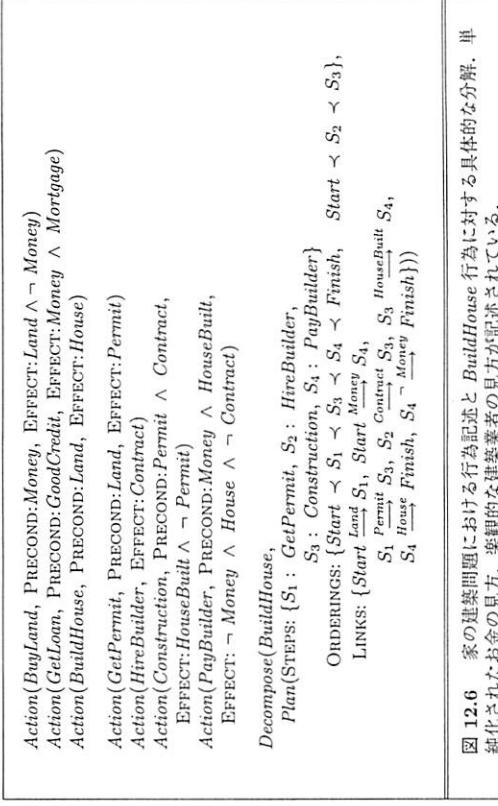


図 12.6 家の建築問題における行為記述と *BuildHouse* 行為に対する具体的な分解。單純化されたお金の見方、楽観的な建築業者の見方が記述されている。

らかに分解は正しく行われるだろう。

プランライブラリでは、与えられたどんな上位レベルの行為に対する分解も保存できる。たとえば、素手で岩石や芝草を材料に家を建てるプロセスを記述した *BuildHouse* のための分解方法もあるだろう。個々の分解は正しいプランであるべきだが、上位レベルの行為記述で決められている前提条件や結果以外のことを附加できる。たとえば、図 12.5 に示される *BuildHouse* の分解は *Land* に加えて *Money* を必要とし、 $\neg Money$ という結果を導いている。一方、自分で家を建築する場合はお金が必要としないが、素手に入る *Rocks* (岩石) と *Turf* (芝草) を必要とし、*BadBack* という結果に終わるだろう。

BuildHouse のような上位レベルの行為が種々に分解できるならば、*Traps* の行為記述では、分解で生じる前提条件や結果のいくつかが隠蔽されることは避けられない。上位レベルの行為の前提条件は外的的前提条件と共通部分 (intersection) があり、結果は外的結果と共通部分があるべきである。別な言い方をすれば、上位レベルの前提条件と結果は、原

² これは予期しないプランの発見を妨げている。たとえば、倒産手続きに直面している人は、家を買ったり建築することで流動資産 (すなわち、 $\neg Money$ の達成) を除くことができない。現在の法律は貸し主によつて主要な住宅の差押さえが起らぬないようにしてあるため、このプランは有効である。

始的に実行された真の前提条件と結果の部分集合であることが保証される。情報隠蔽の二つの形式に注目されたい。まず第一に、上位レベルの記述では分解による *Permit* と *Contract* という一時的な内部結果 (internal effect) は完全に無視される。たとえば、*BuildHouse* の記述では上位レベルの前提条件と結果を導く³。第二に、上位レベルの “内的” 行為の時間間隔が明示されない。たとえば、(例題のモデルでは) *GetPermit* が実行されるまで前提条件 *Land* は真であることが必要とされ、*House* は *PayBuilder* が実行されたあとになる。

もし階層的プランニングが複雑性を減少させることなく、上位レベルの行為を推論できるからである。しかし、これには代償を払う必要がある。たとえば、ある上位レベルの行為における内部条件と別の内部行為との間に競合が起こるかもしれないが、これを上位レベルの記述から発見する方法はない。この問題は HTN プランニングアルゴリズムにとって重要である。簡単に言えば、原始行為はプランニングアルゴリズムによってポイント事象 (point event) として扱われるのに対して、上位レベルの行為はすべての種類の事象が起こる一時的な範囲として扱われる。

分解のためのプランナ修正

ここでは、HTN プランニングを組み込むために POP をどのように修正するかを示そう。具体的には、現在の部分的プラン P に分解方法を適用させるために POP 後者関数 (successor function) (396 ページ) を修正する。まず P におけるいくつかの非原始的行為 a' を選択し、次に $a' = d' = \text{SUBST}(\theta, a)$ に置き換えることによって新しい後続プランを形成する。このような処理は、 θ を用いて a と a' を統合するような *Decompose*(a, d) に似て行われる。図 12.7 に例を示す。一番上に手に入れるというプラン P を示し、上位レベルの行為である $a' = \text{BuildHouse}$ が分解のために選択されている。分解 d は図 12.5 から選択され、*BuildHouse* はこの分解によって置き換えられる。*GetLoan* という追加ステップは、分解ステップで生成された *Money* という新しい未解決条件を解決するために導入される。このように行為を分解されたもので置き換えることは、移植手術のようなものである。つまり、新しい副プランをパッケージ (*Start* と *Finish* ステップ) から取り出し、それを挿入し、すべてを適切につなげなければならない。これを実現するには種々の方法があるが、より精密にするため、個々の可能な分解 d' に対して次のように実施する。

1. 関係 a' を P から取り除く。それから、分解 d' における各ステップ s に対して、 s の役割を選択し、プランに加える。これは s の新しい具体化、または s と統合する P にある既存ステップ s' になります。たとえば、*MakeWine* 行為の分解は、*BuyLand* することを暗示しているかもしない。ひょっとすると、すでにプラン上有あるのと同じ *BuyLand* 行為を使用することができますがもしれない。これをサブタスク共有 (subtask sharing) とよぶ。

図 12.7 では共有の機会がないので、行為に関する新しい具体例が生成される。一度行為が選ばれると、 d' 上の内部制約のすべてはコピーされる。たとえば、*GetPermit* サブタスク共有

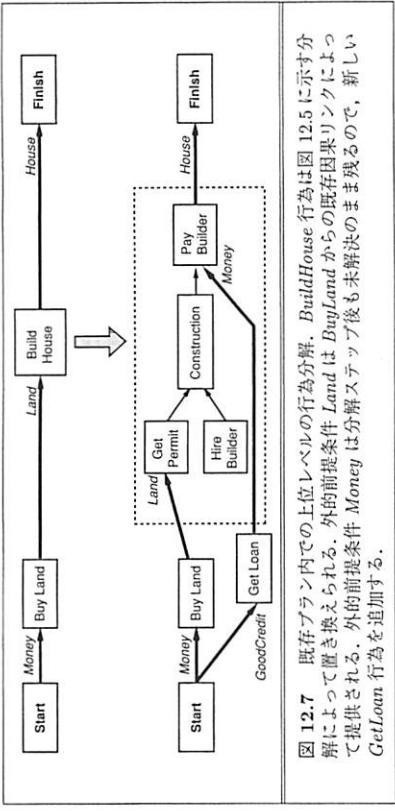


図 12.7 既存プラン内の上位レベルの行為の分解。*BuildHouse* 行為は図 12.5 に示す分解によって置き換えられる。外的的前提条件 *Land* は *BuyLand* からの既存因果リンクによって提供される。外的的前提条件 *Money* は分解ステップ後も未解決のままで、新しい *GetLoan* 行為を追加する。

は *Construction* の前であるとか、*Construction* の前提条件として *Permit* を与える二つのステップの間に因果リンクが存在する、などである。これによつて、 a' を $d\theta$ の具體化されたもので置き換えるタスクが完了する。

2. 次のステップは、元プラン上の a' に対する順序制約を d' 上のステップに取り付けることである。まず、 $B \xrightarrow{\theta} a'$ という P 上の順序制約を考えてみよう。 d' 上のステップに關して、どのように B は順序づけられるのだろうか？ 最も明らかな解法は B を d' 上の全ステップの前に順序づけることであり、それは d' 上の *Start* ～ *s* という全ての制約を $B \xrightarrow{\theta} s$ という制約に置き換えることで達成できる。しかし、このアプローチは厳しそうかもしれない！たとえば、*BuyLand* は *BuildHouse* の前にこなければならぬが、拡張されたプランでは *BuyLand* は *HireBuilder* の前にくる必要はない。厳しい順序を課すことには解の発見の妨げになるだろう。それゆえ、最も良い解法は個々の順序制約に対して理由を記録することである。これによつて、上位レベルの行為が拡張されるとときに、元の制約に対する理由と矛盾しない範囲で、新しい順序制約をできるだけ緩めることができることができる。まったく同じような考え方だが、 $a' \xrightarrow{\theta} C$ の制約を置き換えるときにも適用できる。

3. 最後のステップは因果リンクをつなぐことである。もし、元プランにおいて $B \xrightarrow{p} a'$ が因果リンクであったならば、そのリンクを B から前提条件 p をもつ d' 上の全ステップ (すなわち、 p が外的 前提条件である d' 上の *Start* ～ *s* の因果リンク) の集合と置き換える。なお、前提条件 p は分解された d' 上の *Start* ～ *s* の因果リンクによって与えられる。たとえば、*BuyLand* $\xrightarrow{\theta}$ *BuildHouse* という因果リンクは *BuyLand Land Permit* というリンクに置き換えられる (なお、元プランでは *BuildHouse* に *Money* を供給する行為がないため、分解された *PayBuilder* に対する *Money* という前提条件は未解決となる)。同様に、プラン上の $a' \xrightarrow{p} C$ という個々の因果リンクに対して、そのリンクを p を提供する d' 上のステップ (すなわち、外的 結果としての p をもつ d' 上のステップ) から分解された d 上の *Finish* ～ *C* につながっている因果リンクの集合と置き換える。たとえば、*BuildHouse* $\xrightarrow{\theta}$ *Finish* リンクは *PayBuilder House Finish* リンクで置き換えられる。

³ *Construction* すると *Permit* は取り消されるとともに、同じ群可が多く家の家を建築するために使用されるからである。しかし、*Construction* が完了しても *Contract* は古いまま残る。なぜなら、我々はまだ *PayBuilder* しなければ *Contract* を終了できないからである。

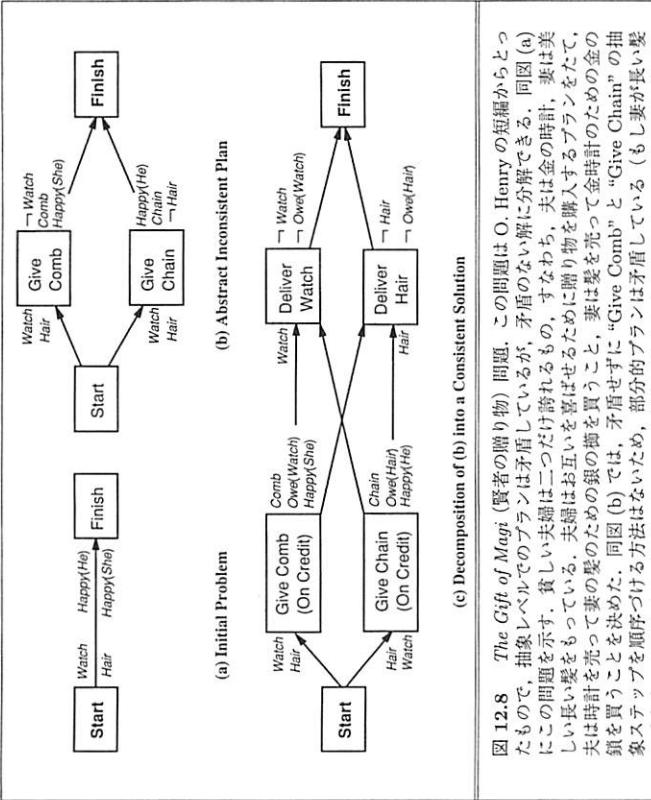


図 12.8 *The Gift of Magi* (質者の贈り物) 問題。この問題は O. Henry の短編からとつたもので、抽象レベルでのプランは矛盾しているが、矛盾のない解に分解できる。同図 (a) にこの問題を示す。美しい夫婦は二つだけ離れるもの、すなわち、夫は金の時計、妻は美しい髪をもっている。夫婦はお互いを喜ばせること、妻は梳(くし)を買うために金の鎖を買うことを決めた。同図 (b) では、矛盾せずに “Give Comb” と “Give Chain” の抽象ステップを順序づける方法はないため、部分的なプランは矛盾している（もし妻が長い髪の毛をもつていなければ、“Give Comb” 行為は彼女を幸せにするという結果を目指せないので、“Give Comb” 行為は “Hair” という前提条件が必要であることを仮定する。“Give Chain” も同様である）。同図 (c) では、“Give Comb” ステップでは、夫は支払のため時計をあわせて購入する約束で、梳(くし)を手に入れて妻に渡す。次のステップで、時計は約束どおり手離される。同様な方法で “Give Chain” も分解される。プレゼントするステップが配達の前にある限り、この分解によつて問題は解決される（これは、時計のための鎖や髪のための梳(くし)を使う幸運があとできさえ存続するようにならねばならないことによる）。

この方法は、POP プランナ分野における分解のために必要な追加部分を完了させることで、POP プランナは最終の原始的実行に関する情報を隠すため、POP アルゴリズムの上位レベルの行為は最終の原始的実行に関する情報を持たない。特に、元の POP アルゴリズムでは、もし現プランの中に解消できない競合が含まれていれば失敗して戻る。つまり、もし行為が因果リンクと競合し、そのリンクの前にも後にも順序つけられなければ失敗して戻る（図 11.9 に例を示す）。一方、上位レベルの行為を用いれば、競合する行為を分解したり、そのステップをインテリープすることによって、明らかに解けない競合を時々解決できる。その例を図 12.8 に示す。このようにして、完全で矛盾のない原始的プランが存在しないときでさえ、分解によって完全で矛盾のない原始的プランが得られるかもしれない。この可能性は、標準 POP プランナが利用できる多くの簡潔な機会を、完全 HTN プランナは捨てるこを意味している。あるいは、余分なものを取り除き、解が見落とされないように望むしかない。

⁴ 上位レベルの行為を用いた競合解決を図つた文献を調べていただきたい。この章の最後にあげた文献を参考してほしい。

ディスカッション

まず、悪いニュースから始めよう。(唯一許されているプランの洗練は分解のみである) 純 HTN プランニングは、たとえ基礎をなす状態空間が有限であつたとしても非決定的である！これは HTN プランニングの要点が効率を上げることなので、とても憂鬱になる。この難しさは行為が分解が再帰的(recursive)になる可能性からきており、そのためには HTN プランは任意に長くなる。たとえば、散歩は、一步踏み出し、それから散歩するという組合せで実現できる。特に、最も短い HTN の解は任意に長くなりうるので、決められた時間のあとに探索を終わらせる方法がない。しかし、これに対して少なくとも次の三つの有望な方法がある：

1. はどんなどの分野で必要とされない再帰は排除できる。その場合、すべての HTN プランは長さが有限であり、数えることができる。
2. 解の長さを限定することができる。状態空間は有限なので、状態空間にある状態よりも多いステップをもつプランは、同じ状態を 2 度訪れるループを含んでいるはずである。この種の HTN の解は排除しても失うものはほとんどなく、探索を制御できる。
3. POP と HTN プランニングを組み合わせるハイブリッドアプローチを採用できる。プランが存在するかどうかを決めるのは半順序プランニングで十分なので、ハイブリッド問題は明らかに決定的である。

上記の中で三番目の解答には少し注意を払う必要がある。POP は任意な方法で原始的行為をつなげるの、理解が困難で HTN プランの良い階層的組織ももたない解しか見いただせない。そのため、適切な妥協としては、原始的行為が加えられる前に任意に長い HTN プランが作成されない程度に、行為分解が新しい行為の追加よりも優先されるようなハイブリッド探索の制御が必要となる。これに対する一つの方法は、分解によって導入される行為を割り引く(discount)コスト関数を使用することである。割引きが大きければ大きいほど、接率は純 HTN プランニングと同等になり、解は階層的になるだろう。通常、階層的プランは現実的な設定での実行が容易であり、うまくいかないときは修正しやすい。

HTN プランのもう一つの重要な特徴は、サブタスクの共有ができることがある。ここで、サブタスク共有とは、分解されたプラン上の二つの異なるステップを実施するために同じ行為を用いることであることを思い出そう。もしサブタスク共有が許されていないければ、分解 d の各々の具体化は多くいうよりは一つの方法でしかなされず、その結果、大いに探索空間を切り詰めることができ。通常、このような切詰めは時間を節約し、悪くとも最適より少し長い解を導く。しかし、いくつかの場合では問題となる。たとえば、“ハネムーンを楽しんで家族を育てる”というゴールを考えよう。プランライブラリは“結婚してハイウェイに行く”を最初のサブゴールとして提案し、“結婚して 2 人の子供をもつ”を二つ目のサブゴールとして提案するかもしれない。サブタスク共有がなければ、このプランは二つの異なる結婚という行為を含むことになり、それはしばしば望ましくないと考えられる。

サブタスク共有のコストと利益に関する興味深い例として、最適化コンパイラがある。ここで、 $\tan(x) - \sin(x)$ という式をコンパイルする問題を考えよう。ほんどのコンパイラーは、二つの独立したサブルーチンコールを単純に合わせることによって実現し、tan 計算は sin 計算を行う前にすべて行われる。しかし、sin と tan に対する次のティラー級数近

似 (Taylor series approximation) を考えてみよう：

$$\tan x \approx x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315}; \quad \sin x \approx x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040}$$

サブタスク共有可能な HTN プランナでは、 \sin 計算にかかる多くのステップは \tan の既存ステップで実行できるので、もつと効果的な解を生成できる。しかし、すべての種のステップは \tan の既存プランには時間がかかりすぎるため、多くのコマバイラではこの種の相互手続 (interprocedural) 共有は行わない、その代わり、個別のサブプランを作成し、ピープホールオプティマイザ (peephole optimizer) を使って結果を修正する。

行為分解を導入することによって生じる付加的な複雑さがすべて分かっても、なぜ我々は HTN プランニングが効果的であると信じるのだろうか？ 複雑性の根源を実際に解析するのは難しいので、理想的な場合を考えよう。たとえば、 n 個の行為を用いてプランを作成したい場合、各状態で b 個の行為を許可した非階層的前向き状態空間プランナ (forward state-space planner) ではコストは $O(b^n)$ となる。HTN プランナでは、 d 種類の分解によって、個々の非原始的行為が次の下位レベルである k 個の行為に分解される正规分解構造 (regular decomposition structure) を仮定すると、どれだけ多くの異なる分解木 (decomposition tree) がその構造であるかを知りたい。ここで、もし n 個の行為が原始的レベルであれば、根 (root) から下にあるレベル数は $\log_k n$ なので、内部のノード数は $1 + k + k^2 + \dots + k^{\lfloor \log_k n - 1 \rfloor} = (n-1)/(k-1)$ となる。個々の内部ノードは d 種類に分解可能なので、 $d^{(n-1)/(k-1)}$ 個の正规分解木を作成できる。この数式を調べてみると、 d を小さく k を大きく保つことが大きな節約につながることがわかる。つまり、もし b と d が同等であれば、本質的には非階層的コストの k 番目の根を採用する。一方、少ない数ではあるが長く分解でき、どんな問題も解決できるプランライブリをいつも作成できるとは限らない。言い換えれば、広範囲の問題に利用できる長いマクロはたいへん貴重である。他に、HTN プランニングが効果的であると信じられる良い理由は HTN プランニングが実際に機能していることである。HTN プランニングでは大規模プランを少ない計算コストで作成するために、人間の専門家から複雑なタスクを処理する重要な知識を提供してもらえるため、多くの大規模応用プランナは HTN プランナである。たとえば、O-PLAN (Bell and Tate, 1985) は HTN プランニングとスケジューリングを組み合わせたもので、日立の製品プランを発展させたために使用されてきた。典型的な問題は、350 個の異なる製品ライン、35 個の組立て機械、そして 2000 個を超える作業からなる。プランナは、何百万ステップを含む 1 日 8 時間 3 回交替制のスケジュールを 30 日分生成する。

HTN プランニングにとって重要なことは、複雑な上位レベルの行為を実行するための既知の方法は、問題解決の経験から学習することである。ゼロからプランを作成する一つの方法は、問題解決の経験から学習することである。ゼロからプランを作成する一つの方法は、問題解決の経験を経ると、タスクによって定義された上位レベルの行為を実行する方法のように、プランをライブリに蓄えることができる。このように古い方法の上に新しい方法を構築しながら、段々と有能になっていく。このような学習プロセスの重要な側面は、問題事例固有的詳細部分 (たとえば、builder の名前や土地の住所) を除き、プラン上の基本的要素だけを残しながら、構築された方法を一般化する (generalize) 能力である。この種の一般化を達成する方法は 19 章で述べているが、我々人間がこのようなメカニズムがなくても同じように有能であることは信じられない気がする。

12.3 非決定的領域におけるプランニングと行為

今まで完全観察、静的、かつ決定的な古典的プランニング (classical planning) の領域だけを考え、さらには行為の記述も正しく完全であると仮定してきた。このような状況では、エージェントはまずプランを作成し、次に “目を開じたままで” そのプランを実行できる。反対に不確かな環境では、エージェントはプラン実行中に何が起こっているのかを知るために知覚しなければならず、もし何か予期せぬことが起こればプランを修正もしくは置き換える必要がある。

エージェントは、不完全な情報と不正確な情報の両方を扱わなくてはならない。特に、不完全性 (incompleteness) は世界が部分観測、非決定的、あるいはその両方の特徴をもつことからくる。たとえば、オフィスサプライ・キャビネットへのドアはロックされているかもしれないし、それでいないかもしない、また、もしドアがロックされていれば、私のもつている鍵の一つで開けられるかもしないし、聞けられないかもしないし、気づいていません。私は自分の知識を使ってこのようなく完全性に気づいているかもしないし、聞けられないかもしない、このように、私の世界モデルは不十分 (weak) であるが正しい。一方、不正確性 (incorrectness) は私の世界モデルと実世界との必ずしも一致しないことから起こる。たとえば、私の鍵でキャビネットを開けられると信じるかもしないが、鍵が変われば私は間違っている。このような不正確な情報を扱う能力がなければ、最終的にエージェントはフンコロガシ (37 ページ) が糞の玉を取り上げられたのに、なお、糞の玉で自分の鼻をふざごうとすると同じぐらい無知になる。

完全で正しい知識をもつ可能性は、世の中にどれぐらい不確定性 (indeterminacy) が存在するかにかかっている。限定不確定性 (bounded indeterminacy) があると、行動は予期せぬ結果を導くが、起こりうる結果を行為記述公理 (action description axiom) に記録できる。たとえば、硬貨をはじくと結果が Head か Tail になるのは道理に合っている。エージェントはすべての起こりうる状況で機能するプランを作成することによって限定不確定性に対応できるようになる。一方、非限定不確定性 (unbounded indeterminacy) があると、起こりうる前提条件や結果の集合は未知か完全にはあまりに大きすぎる。たとえば、ドライブ、経済プラン、軍事戦略のようなとても複雑かつ動的な問題領域がこれに当てはまるだろう。エージェントはプラン、知識ベース、あるいはその両方を変更する準備がなされている場合のみ、非限定不確定性は 10 章で論じた制限条件記述問題 (qualification problem) と密接に関係している。10 章では、意図された結果を得るために実世界の行為に必要な前提条件をすべて記憶することの不可能性を議論している。

不確定性を扱うプランニングの方法には次の四つがある。最初の二つは限定不確定性に適しており、あとの二つは非限定不確定性に適している。

- ◇ センサレスプランニング (sensorless planning) : 順応プランニング (conformant planning) ともいう。この方法は知覚なしに実行できる標的的な連続プランを作成する。センサレスプランニングアルゴリズムは、真の初期状態や実際の行為結果にもかかわらず、すべての起こりうる状況でプランのゴール達成を保証しなければならない。
- ◇ センサレスプランニング (sensorless planning) : 順応プランニング (conformant planning) ともいう。この方法は知覚なしに実行できる標的的な連続プランを作成する。センサレスプランニングアルゴリズムは、真の初期状態や実際の行為結果にもかかわらず、すべての起こりうる状況でプランのゴール達成を強制 (coercion) という考え方に基づいている。強制

限定不確定性
非限定不確定性

とはエージェントが現状態にに関して部分情報しかもつてないときでさえ、世界を与えられた状態と見なす考えである。強制はいつも可能であるとは限らないので、センサレスプランニングはしばしば適用できない。信念状態空間の探索を含むセンサレス問題解決については3章で述べている。

- ◇ 条件付きプランニング (conditional planning)：偶発性プランニング (contingency planning)ともいう。この方法は、発生した異なる偶発事象に対して異なる枝を持つ条件付きプランを作成することによって、限定不確定性に対処する。古典的プランニングと同じように、エージェントはまずプランを立て、次に立てたプランを実行する。そして、エージェントは適切な条件を検査するために知覚行為 (sensing action) をプランに含め、プランのどの部分を実行すべきかを見いだす。たとえば、航空運輸業界では、“SFO 空港が使用できるかどうかチェックし、もし使用可能であればそこへ飛行せよ。そうでなければ、Oakland へ飛行せよ”というプランが考えられる。条件付きプランニングは12.4節において説明する。
- ◇ 実行モニタリングと再プランニング：この方法では、エージェントはプラン作成のために前述したどのプランニング技術（古典的、センサレス、あるいは条件付き）も使用できるが、現状態に対するプランが実行できるか、あるいはプランを変更する必要があるかどうかを判断するために、実行モニタリング (execution monitoring) も使用する。再プランニング (replanning) は何かうまくいっていないときに起こる。このように、エージェントは再プランニングを用いて非限定不確定性を扱う。たとえば、再プランニングエージェントが SFO 閉鎖の可能性を予期しなかった場合でも、エージェントはそれが起こった状況を認識でき、ゴールへの新しいバスを見いだすために再びプランナをよぶ。再プランニングエージェントについては12.5節において説明する。

◇ 運続プランニング (continuous planning)：今まで見てきたプランナはすべてゴールを達成し、そして止まるようになりますが、運続プランナはライフタイムを越えて存続するように設計されている。そのため、たとえエージェントがプラン作成中に予期せぬことが起こっても、その環境に対処できる。また、連続プランナはゴールを放棄したり、ゴールの定式化 (goal formulation) によってゴールを追加することもできる。連続プランニングについては12.6節において説明する。

これらエージェントの違いを明らかにするために例をあげよう。その例とは、イスとテーブルを同じ色に塗る問題である。
古典的プランニング (classical planning) エージェントは、初期状態が完全には明記されていないので、この問題は対処できない。センサレスプランニング (sensorless planning) エージェントは、プラン実行中にセンサーを要求しなくともうまくいくプランを見つけなければならない。ここで解説は、(エージェントは何かわからっていないが)、どれかのベンキの缶を開けてイスとテーブルの両方に塗り、強制的 (coercing) に同じ色にすることである。このような強制は計画することが高価だったり、知覚できないときに最も適している。たとえば、医師は条件的プランである血液検査をし、その結果を待つてから特別の抗生素を処方するよりも、ごく一般的な抗生素を処方する。それは、血液検査にかかる遅延と費用が大きすぎるからである。

条件付きプランニング (conditional planning) エージェントは、次のような良いプランを作成できる。まずイスとテーブルの色を知覚し、もし二つの色がすでに同じであればプランを完了する。色が異なれば缶のラベルを知覚し、どちらかの家具と同じ色の缶があれば、どんな色でも二つの家具に塗る。

再プランニング (replanning) エージェントは条件付きプランナと同じプランを作成できるだけなく、最初はほとんど（条件）分岐を作れなくても、実行時に必要に応じて埋めることができます。また、エージェントは行為記述の不正確性にも対処することができます。たとえば、*Paint(obj, color)* 行為が *Color(obj, color)* という結果を決定的に導くと信じられていると想定しよう。条件付きプランナではないん行為が実行されるとその結果が生じることが仮定されているが、再プランニングエージェントでは結果をチェックして、それが正しくなければ（たとえば、エージェントが不注意で塗る場所を見出すなど）、その場所を塗り直すために再プランすることができる。446ページでこの例を再び使う。

連続プランニング (continuous planning) エージェントは、予期せぬ出来事の対処に加えて、適切にプランを変更できる。たとえば、もしうーマークルで夕食をとるというゴールを追加すると、ベンキ塗装プランは延期される。

実世界では、上記の方法を組み合わせて使用する。自動車製造業者はスペアタイヤやエアバッグを販売しているが、それらはパンクや衝突に対応するよう設計されている条件付きプランの枝の具体例である。一方、多くのドライバはスペアタイヤやエアバッグの可能性を決して考えないが、再プランニングエージェントのようにパンクや衝突に対応する一般的に、エージェントは重要な結果や失敗するときの些細なきつかけを導く骨髄性に対してだけ、条件付きプランを作成する。だから、サハラ砂漠を横断旅行を計画しているドライバが故障の可能性を考えるのはもっともだが、近所のスーパーでマーケットへ行くのは進歩したプランニングは必要としない。

本章で述べたエージェントは不確定性に対応できるように設計されているが、プラン成功の可能性とプラン作成のコストの間のトレードオフを解決する能力はない。16章にこれらの事柄を扱う付加的ツールを紹介する。

12.4 条件付きプランニング

条件付きプランニングは、事前に決められたプラン上のポイントにおける環境で実際に何が起こっているのかをチェックすることによって、不確定性に対応する方法である。条件付きプランニングの説明は、完全観測環境 (fully observable environment) が最も簡単なので、その場合から始めることにする。部分観測 (partially observable) はより難しいが、さらに興味深い。

完全観測環境における条件付きプランニング完全観測性 (full observability) とは、いつもエージェントが現状態を知っていることを意味する。しかし、もし環境が非決定的であれば、エージェントは行為の結果を予想することはできない。そこで、条件付きプランニングエージェントは、次に何をすべきかを決め

るために（実行時に）環境状態をチェックする条件付きステップを（プラン作成時に）プランに組み込むことによって非決定性を処理する。ここで問題となるのは、どのようにこのような条件付きプランを作成するかということである。

例題として、決定的な場合の状態空間が 65 ページに掲載されている掃除機の世界（vacuum world）を使う。まず、利用できる行為が *Left*, *Right*, そして *Suck* することを思い出そう。そして、状態を定義するために次の命題（proposition）を用いる。もしエージェントが左（右）の状態⁵にいれば *AtL* (*AtR*) は真となり、もし左（右）の状態が空（clean）であれば、*CleanL* (*CleanR*) は真になる。ここで、最初に行うべきことは、非決定性を扱えるように STRIPS 言語を拡張することである。そのためには、行為が実行されるときはいつでも二つまたはそれ以上の異なる結果を導くという選択的結果（disjunctive effect）を行為に導入する。たとえば、*Left*への移動が時々失敗すると想定すると、通常の行為記述である

*Action(Left, PRECOND:*AtL*, EFFECT:*AtL*)*

は、次のように選択的結果を含むように修正されるべきである。

*Action(Left, PRECOND:*AtL*, EFFECT:*AtL* ∨ *AtR*)*

また、行為の結果が行為を実行する状態に依存するという条件付き結果（conditional effect）を導入することも有益である。条件付き結果は、行為の EFFECT スロットに“when <condition> : <effect>”という文法で記述される。たとえば、*Suck* 行為をモデル化することは、次のように選択的結果を含むように修正されるべきである。

*Action(Suck, PRECOND: EFFECT: *AtL* : *CleanL*)*

条件付き結果は不確定性を取り入れないが、それをモデル化することに役に立つ。たとえば、きれいな場所のみに時々ゴミを残すような掃除機があるとしよう。これは次のように選択的かつ条件付き⁶記述でモデル化できる。

*Action(Left, PRECOND:*AtL*, EFFECT: *AtL* ∨ when *CleanL*: *¬CleanL*)*

条件付きプランを作成するには条件付きステップ（conditional step）が必要となる。そこで、“if <test> then *planA* else *planB*”という文法を使つて条件付きステップを記述する。ここで、<test> は状態変数に関するゴール関数である。たとえば、掃除機の世界にとっての条件付きステップは“if *AtL* ∧ *CleanL* then *Right* else *Suck*”と記述できるだろう。このようなステップの実行は明白なやり方で進められ、条件付きステップを入れ子（nesting）にすることによってプランは木となる。

我々は行為の結果が実際に起こるかどうかにかかわらず、うまくいく条件付きプランを望む。以前にこの問題を別の方で扱つたが、2 人ゲーム（6 章）でも相手の手に關係なく勝つ手が望まれる。このような理由で、非決定的プランニング問題はしばしば自然を相手にしたゲーム（game against nature）とよばれている。⁵

掃除機の世界特有の例を考えよう。ロボットがきれいな世界の右の区画にいる状態を初期状態とする。完全観測環境なので、エージェントは *AtR* ∧ *CleanL* ∧ *CleanR* という完全状態記述を知つていい。ゴール状態はロボットがきれいな世界の左の区画にいる状態とする。

⁵ 明らかに、*¬AtL* が真である場合およびその場合に限つて *AtR* は真となる。逆も同じである。読みやすさを向上させるためにこの二つの命題を使つてある。

⁶ 条件付き結果である *when* *CleanL*: *¬CleanL* は少しおかしく見えるかもしれない。しかし、*CleanL* は行為を実行する前の状態であり、*¬CleanL* は行為を実行した後の状態であることを思い出してほしい。

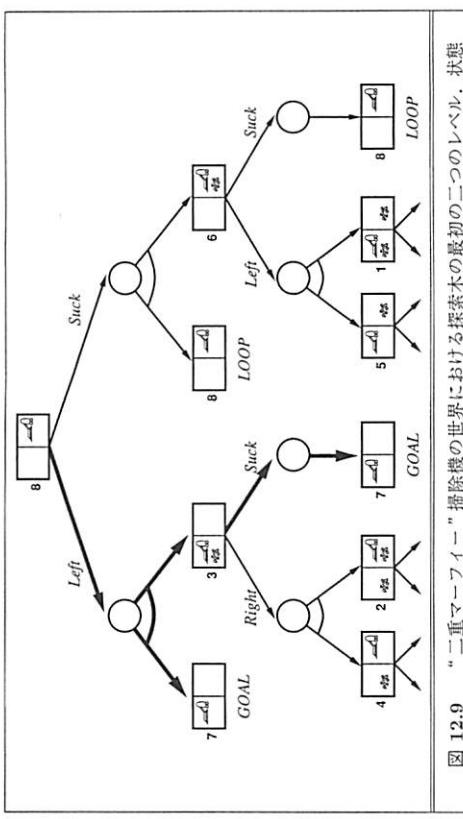


図 12.9 “二重マーフィー”掃除機の世界における探索木の最初の二つのレベル。状態ノードは OR ノードで、何かの行為が選択されなければならぬ。丸で示された機会ノードは AND ノードで、枝をつなぐ弧に示されるように全結果を扱わなければならない。解は太い線で示されている。

る。ここで、掃除機がきれいな区画へ移動するときに時々ゴミを残したり、あるいは、きれいな区画で *Suck* するときに時々ゴミを残したりするような“二重マーフィー（double Murphy）”の掃除機がなければ、この問題はしまらないものになるだろう。

この環境に対する“ゲーム木”を図 12.9 に示す。ロボットは木の中の“状態（state）”ノードで行爲し、自然（nature）は丸で示されている“機会（chance）”機会ノードでどのようないくつかを決定する。ここでの解は、(1) すべての葉（leaf）でゴールノードをもち、(2) 各“状態”ノードで一つの行為を明記し、(3) 各“機会”ノードで全結果の枝を含む、という部分木（subtree）となる。その解は図に太線で示されており、[Left, if *AtL* ∧ *CleanL* then *CleanR* then [] else *Suck*] というプランに相当する（今のところは、状態一空間プランナを使用しているので、条件付きステップの検査は完全状態で記述できるだろう）。

ゲームの正確な解答のために、ミニマックスアルゴリズム（minimax algorithm）を使用し（図 6.3），条件付きプランニングのために次の二つの典型的な修正を行ふ。まず第一に、MAX と MIN ノードを OR と AND ノードとする。直観的に、プランは到達した全状態で何かの行為をとる必要があるが、実行した行為から生じる全結果も扱わなくてはならない。第二に、アルゴリズムは、單なる一つの移動よりも条件付きプランを提供することが必要となる。OR ノードにおけるプランは、どんなことが起ころうとも選択され行爲からなり、一方、AND ノードにおけるプランは、個々の結果に対するサブプランを示す if-then-else ステップの連続入れ子状態となる。これらのステップの検査は完全状態で記述できる。⁷

形式的に言えば、我々が定義した探索空間は AND-OR グラフ（graph）である。AND-OR グラフは 7 章の命題ホーン節推論（propositional Horn clause inference）の中で述べられて いる。ここでの枝は論理推論のステップといいうよりは行為であるが、アルゴリズムは同じである。図 12.10 に、AND-OR グラフ探索のための再帰的な深さ優先（depth-first）アルゴ

⁷ そのようなプランは case 文（case construct）を用いても記述できる。

```

function AND-OR-GRAPH-SEARCH(state, problem, path) returns a conditional plan, or failure
    OR-SEARCH(INITIAL-STATE[problem], problem, [])
    if GOAL-TEST[state] then return the empty plan
    if state is on path then return failure
    for each action, state...set in SUCCESSORS[problem](state) do
        plan ← AND-SEARCH(state.set, problem, state | path)
        if plan ≠ failure then return [action | plan]
    return failure

function AND-SEARCH(state.set, problem, path) returns a conditional plan, or failure
    for each si in state.set do
        plani ← OR-SEARCH(si, problem, path)
        if plan = failure then return failure
        return [if si then plani else if s2 then plan2 else ... if sn-1 then plann-1 else plann]
    return failure

```

図 12.10 非決定的環境で生成された AND-OR グラフを探索するアルゴリズム、
SUCCESSORS は、起こりうる結果の集合と関連する行為のリストを返す。このアルゴリズムの目的は、あらゆる環境においてゴール状態に到達する条件付きプランを見いだすことである。

リズムを示す。

このアルゴリズムの一つの重要な点は、しばしば非決定的プランニング問題（たとえば、時には行為が何の結果も生まないとか、意図せぬ結果が訂正されるとときに起こる問題）で現れる周期に対処する方法にある。つまり、もし現状態が根から発生したバス上にある状態と同じならば、アルゴリズムは失敗を返す。これは現状態からの解がないことを意味しているわけではない。もし非周期的な解があるならば、現状態より前のところから到達できることを意味しており、新たな状態は放棄可能である。このチェックにより、アルゴリズムはあらゆる有限状態空間において終了することを保証する。なぜなら、すべてのバスはゴール、行き止まり、あるいは環返し状態に必ず到達するからである。なお、このアルゴリズムは現状態が根から発生した他のバス上の状態であるかどうかはチェックしていないことに注意していただきたい。練習問題 12.15 にてこの問題を扱う。

AND-OR-GRAPH-SEARCH から返されるプランは、枝を決めるための完全状態記述を検査する条件付きステップを含んでいる。多くの場合、網羅的でない検査で済む。たとえば、図 12.9 に示す解プランは、[Left, if CleanL then [] else Suck] と簡単に記述できる。なぜなら、エージェントが検査後どのような状態にいるのかを正確に知るには、一つの検査である CleanLにおいて AND ノードの状態を二つの単集合 (singleton set) に分けるだけ十分だからである。事実、单変数の if-then-else 検査は、状態が完全観測であるという条件下状態集合を単集合に分けるだけで十分である。だから、我々は一般性を損失せずに単変数の検査に制限することができたのである。

最後に、非決定的領域でしばしば起くる厄介な問題がある。それは物事はいつも最初からうまくいかず、再びやり直さなければならないことがある。たとえば、(以前に述べた習性に加えて) 命令されたときに時々移動に失敗する“三重マーフィー”掃除機を考えてみると、Left は式 (12.1) に示すように選言的結果である $Alt \vee Alt$ をもつことができる。そうすると、[Left, if CleanL then [] else Suck] というプランはもうまく保証はな

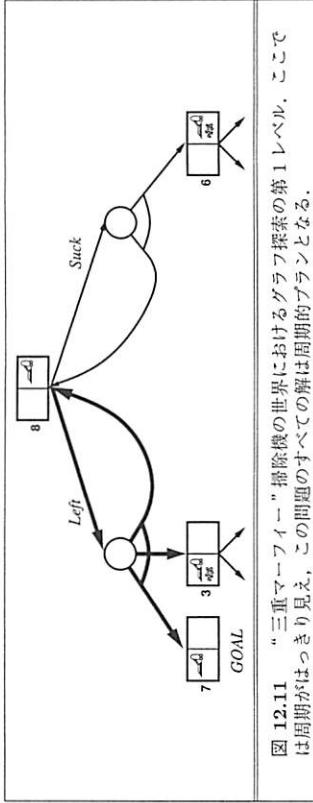


図 12.11 “三重マーフィー”掃除機の世界におけるグラフ探索の第 1 レベル。ここでは周期がはっきり見え、この問題のすべての解は周期内のプランとなる。

い、図 12.11 に探索グラフの一部分を示すが、確かにものは非周期的 (acyclic) な解は存在せず、AND-OR-GRAPH-SEARCH は失敗として返すだろう。しかし、うまく今まで Left を試しつづけるという周期的な解 (cyclic solution) は存在する。このような解はプランの一部を示すラベル (label) を加え、プラン自身を繰り返す代わりにそのラベルを使用することで表現できる。したがって、周期的な解は次のようになる。

$[L_1 : Left, \text{if } Alt \text{ then } L_1 \text{ else if } CleanL \text{ then } [] \text{ else Suck}]$

(このプランのループ部分における良い文法は、“while Alt do Left”になるだろう)。AND-OR-GRAPH-SEARCH に必要な修正は、練習問題 12.16 でカバーされている。重要な理解としては、状態 L に戻る状態空間上のループは状態 L のサブプランが実行される時点まで戻るプラン上のループに解釈されることである。今やプログラムのように見える複雑なプランを条件やループと統合することができるのが、不幸にもここでこのループは潜在的に無限ループである。たとえば、三重マーフィー世界の行為表現では $Left$ は最終的に成功するとは言えない。だから、周期内のプランは非周期的なプランよりも望ましくない。しかし、個々の葉がゴール状態であり、プラン上のある止まるポイントから達成できるという条件で、周期的なプランは解として考えてよいかもしない。

部分観測環境における条件付きプランニング

前節では完全観測環境を扱ってきたが、その環境では条件付き検査を用いればどのように疑問に対しても確実に答えが得られるという利点がある。しかし、実世界では部分観測のほうがより一般的である。そのような部分観測プランニング問題の初期状態では、エージェントは現状態についてのみいくらか知っている。この状況をモデル化する最も簡単な方法は、初期状態が状態集合 (state set) に属することである。ここでの状態集合 (belief state) は、エージェントの初期信念状態 (belief state) を記述する手段である。⁸

掃除機の世界でのエージェントが右側の区画にいて、その区画がきれいなことは知っているが、他の区画にゴミがあるかないかは感知できない状況を想定しよう。この場合、エージェントの知る限りでは、左側の区画はきれいか汚いかの二状態のうちどちらかである。この信念状態は図 12.12 において A と記されている。この図は“代替二重マーフィー”

⁸ このような概念は 3.6 節で導入されており、読者はこの章を読む前に調べたいと願うかもしれない。

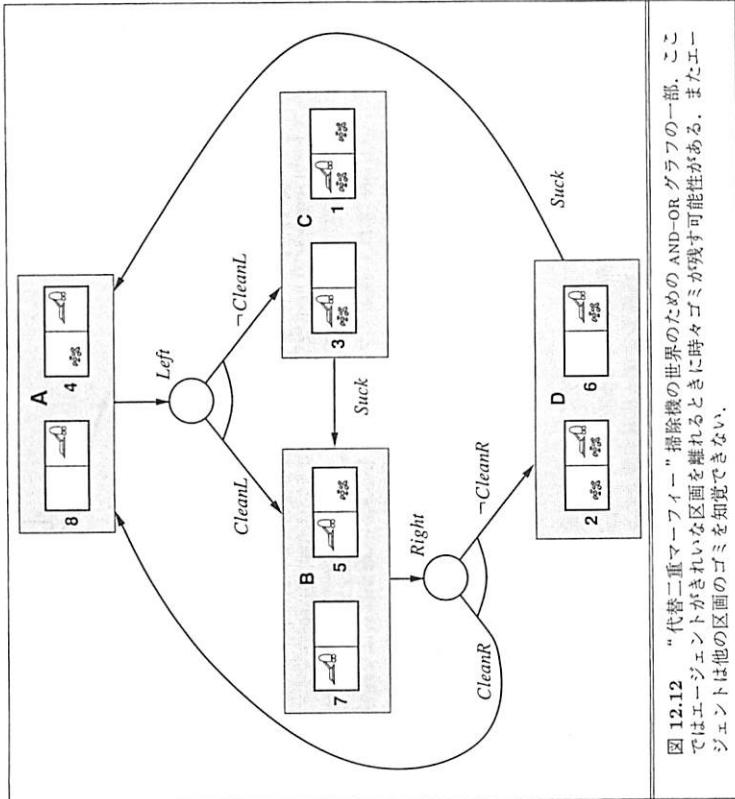


図 12.12 “代替二重マーフィー”掃除機の世界のための AND-OR グラフの一部。ここでエージェントは他の区画のゴミを知覚できない。

掃除機の世界のための AND-OR グラフの一部を示しているが、ここではエージェントがきれいな区画を離れるときに時々ゴミを残す可能性があるとする。⁹もし世界が完全に観測できるなら、エージェントは“両方の区画がきれいで、かつ、左の区画にいる状態になるまで、左右に動いてゴミがあれば吸いとることを繰り返す”という周期的な解を作成できる（練習問題 12.16 を参照のこと）。しかし、不幸なことに局所的なゴミ知覚では“両方の区画がきれい”かどうかの検査において真値が得られないため、このプランは実行できない。それでは、AND-OR グラフがどのように作成されるかを見てみよう。まず、信念状態 A から $Left \rightarrow CleanL$ へ移動した結果を示す（他の行為ではない）。エージェントはゴミを残す可能性があるので、二つの起こり得る初期世界は B と C に示されるようになります。この世界では利用できるセンサ情報によって二つの異なる信念状態が形成される。¹⁰ B ではエージェントは $CleanL$ を知り、 C では $\neg CleanL$ を知る。ここで、 C にてゴミをきれいにすると信念状態 B に移行する。また、 B において右へ移動するとゴミを残すかもしれないし残さないので、 $CleanR$ (A に戻る) か $\neg CleanR$ (信念状態 D) というエージェントの知識によって分離された四つの起こりうる世界に再びなる。

⁹ 幼い子供ももつて観察はこのような状況をよく知っているだろう。いつものように謝る。

¹⁰ 信念状態はエージェントが移動したときにゴミが吸っているかないかによって分類されたものではないことに注意していただきたい。信念状態間ににおける分歧は代替の知識的結果によって起ころるもので、代替の物理的結果によるものではない。

要するに、非決定的かつ部分観測環境では信念状態では記述することがができる。それゆえ、AND-OR-GRAPH-SEARCH という完全観測の場合とまったく同じアルゴリズムを用いることによって、条件付きプランを見つけることができる。また、何が行われているかを知る他の方法は、エージェントの信念状態はいつも完全に観測できる、すなわち、エージェントは何を知っているかを理解することである。“標準的な”完全観測問題解決は、あらゆる信念状態が一つの物理的状態を含む単集合である特別な場合である。

これでやるべきことは完了したのだろうか？いや、まだだ！たとえば、我々はどのように信念状態を表現すべきか、どのように知覚がはたらくのか、また、新しい設定でどのように行為を記述すべきかを決める必要がある。

そこで、以下に信念状態のための三つの基本選択を示す：

1. 完全状態記述の集合。たとえば、図 12.12 に示す初期信念状態は次のように記述できる。

$$\{(AtR \wedge CleanR \wedge CleanL), (AtR \wedge CleanR \wedge \neg CleanL)\}$$

この表現は簡単に実施できるが、たいへんコストがかかる。もし状態を定義したゴール命題が n 個あれば、各物理的状態記述のサイズを $O(n)$ とした場合、信念状態は $O(2^n)$ 個の物理的状態記述を含む。このような信念状態は、エージェントが命題の断片のみを知っているときはいつでも指数的に大きくなる。つまり、知っている量が少なければ少ないほど、より可能な状態が信念状態に入る。

2. 信念状態における可能世界の集合を正確に記述した論理文 (logical sentences)。たとえば、初期状態は次のように記述できる。

$$AtR \wedge CleanR$$

明らかに、すべての信念状態は一つの論理文によって正確に記述することができる。もし、しなければならないなら、すべての連言状態記述 (conjunctive state description) の選言を使用できるが、我々の例はもとと簡潔な文である。

一般的な論理文の欠点は、同じ信念状態を記述する文の中で論理的に等しいが異なるものが多くあるので、グラフ探索アルゴリズムを用いてチェックされた繰返し状態は一般的な定理証明が必要になることである。この理由から、我々は個々の信念状態がそれぞれ一つの文に対応する通常の表現を好む。¹¹ そのような表現では、命題名で序列化されたリテラルの連言を用いる。 $AtR \wedge CleanR$ はその例である。これは、ちょうど 11 章における開世界仮説 (open-world assumption) のもとでの標準的な状態表現となる。ただ、すべての論理文が必ずしもそのような型式で記述できるわけではない。たとえば、 $AtL \vee CleanR$ を表現する方法はないが、多くの問題領域で扱われている。

3. エージェントの知識を記述する知識命題 (knowledge proposition) (7.7 節も参照)。初期状態として、次のような知識命題をもつ。

$$K(AtR \wedge K(CleanR))$$

¹¹ 一般的な命題文にとつて最もよく知られた通常の表現は二分決定グラフ (binary decision diagram: BDD) (Bryant, 1992) である。

なお、 K は“知っていること”を表し、 $K(P)$ はエージェントが P が真であることを知っていることを意味する。¹² 我々は閉世界仮説 (closed-world assumption) のもとで知識命題を用いるので、もし知識命題がリストになければ偽と見なされる。たとえば、 $\neg K(CleanL)$ と $\neg K(\neg CleanL)$ は上記の文で暗に示されているので、エージェントは $CleanL$ の真値を知らないという事実に直面する。

二番目と三番目の選択肢は大まかに言って同じであるが、ここで三番目の知識命題を使用する。なぜなら、知識命題は知覚についての鮮明な記述ができる、かつ、我々は閉世界仮説を用いてどのように STRIPS 記述を書くかをすでに知っているからである。

両方の選択肢において、個々の命題記号は肯定 (positive), 否定 (negative), あるいは、未知 (unknown) のうち一つとなる。それゆえ、この方法で記述される可能な信念状態は 3^n 個存在する。ここで、信念状態の集合は物理的状態の集合 (全部分集合の集合) であることを考えると、 2^n 個の信念状態があるので、 2^{2^n} 個の選択肢は信念状態の表現といふ点で制約がある。しかし、これは今のところ避けないと信じられている。なぜなら、すべての可能な信念状態を表現できるどんなスキーム (scheme) も、最悪の場合には個々の状態を表すために $O(\log_2(2^{2^n})) = O(2^n)$ ビットを必要とするからである。しかし、我々の簡単なスキームは圧縮のための表現を用いることによって、個々の信念状態を表すのに $O(n)$ しか必要としない。特に、もし前提条件の不明な行為が起こると、結果として生じた信念状態は正確には表されないだろうし、行為の結果もわからなくなる。

次に、どのように知覚するかを決める必要がある。ここに二つの選択肢がある。まず、各ステップでエージェントが知覚可能なものをすべて得る自動知覚 (automatic sensing) である。図 12.12 では場所と局所的なゴミに関して自動知覚できることを仮定している。これの代わりに、*CheckDirt* や *CheckLocation* のような特殊な知覚行為 (sensory action) の実行によってのみ知覚できる能動知覚 (active sensing) もある。これから順番に個々の知覚を扱おう。

知識命題を使用する行為記述を書いてみよう。代替二重マーフィー世界において、自動局所ゴミ知覚をもつエージェントが $Left$ に移動することを考えると、エージェントはその世界のルールにしたがって、もし区画がきれいだったならゴミを残すかもしれないし残さないかもしれません。これは物理的結果としては選択的であるが、知識結果としては單に $CleanR$ というエージェントの知識を消すことになる。そのエージェントは局所ゴミ知覚によって $CleanL$ が真かどうかを知るだろうし、そして $AttL$ にいることも知るだろう：

```
Action(Left, PRECOND:AttL,
      EFFECT: $K(AttL) \wedge \neg K(AtR) \wedge$  when  $CleanL: \neg K(CleanR) \wedge$ 
            when  $\neg CleanL: K(CleanL) \wedge$ 
            when  $\neg CleanL: K(\neg CleanL))$ )
```

前提条件と when 条件は、知識命題ではなく普通な命題であることに注目していただきたい。行為の結果がまさに実世界に依存しているので、これは当然のことである。しかし、我々のもつているすべてが信念状態であるとき、それらの条件の真値をどのように調べるのだろう? もしエージェントが現信念状態の中で、たとえば $K(AtR)$ などの命題を知っていられるかを聞きわめるために知覚したものを調べる。マーフィーの法則によれば、ネズミ、人間、そして条件付きプランニングによる最もすばらしいプランでさえ、しばしば失

なるならば、その命題は現物理的状態において真であるに違いないし、実際に行為を適用できる。もしエージェントが when 条件である $CleanL$ などの命題を知らなければ、信念状態は $CleanL$ が真である世界と $CleanL$ が偽である世界を含まなければならない、これは行為の結果として多様な信念状態を引き起こす。つまり、もし初期状態が $(K(AtR) \wedge K(CleanR))$ であれば、 $Left$ に移動後、 $(K(AttL) \wedge K(\neg CleanL))$ と $(K(AtL) \wedge K(\neg CleanL))$ という二つの信念状態が生成される。どちらの信念状態でも、 $CleanL$ の真値はわかるので、 $CleanL$ の検査はプラン上で使用できる。

エージェントは、(自動知覚に相対する) 能動知覚を用いて新しい知覚を得るために、 $Left$ に移動後、エージェントは左側の区画が汚いのかどうかを知らず、最後の二つの条件付き結果もはや式 (12.2) の行為記述には現れない。そこで、エージェントは区画が汚いかどうかを知るために、*CheckDirt* を用いる：

```
Action(CheckDirt, EFFECT:when AttL  $\wedge$  CleanL:  $K(CleanL) \wedge$ 
      when  $AtR \wedge \neg CleanL: K(\neg CleanL) \wedge$ 
            when  $AtR \wedge CleanR: K(CleanR) \wedge$ 
            when  $\neg AtR \wedge \neg CleanR: K(\neg CleanR))$ )
```

能動知覚設定における *CheckDirt* の前に実施した $Left$ が、自動知覚設定において実施した $Left$ と同じように、二つの信念状態を導くことを示すのは容易である。能動知覚を用いた場合、物理的行为が信念状態を一つつ続の信念状態に写像する (map) のはいつものことである。特別な知識を与え、それによって条件付きテストをプラン上で可能にする知覚行為によってのみ多様な信念状態が導入される。

今まで、状態空間の AND-OR 探索に基づいた条件付きプランニングに対する一般的なアプローチを述べてきた。このアプローチは、いくつかのテスト問題ではないへん効果的であるが、他の問題では手に負えないことがわかつってきた。理論的には、条件付きプランニングは古典的プランニングよりも困難な複雑性クラスに属することを示すことができる。ここで、NP クラスでは候補の解が本当に解かどうかを多項式時間で見きわめることができることを思い出すと、古典的プラン (少なくとも多項式規模のもの) に対してはこれは真であるので、古典的プランニング問題は NP である。しかし、条件付きプランニングでは、すべての可能な状態に対してゴールに到着するプラン上のバスがいくつか存在するかどうかを見きわめるために、候補の解を調べなければならない。しかし、“すべてあるいは一部”の組合せを多項式の時間内に検査することはできないので、条件付きプランニングは NP よりも困難なものとなる。唯一の方法は、プランニングフェーズに起こる偶発的なものをいくつか無視し、実際に起こったときだけ対処することである。このようなアプローチを次節で追求する。



自動知覚

知覚行為

能動知覚

¹² これは、7 章で示した巡回に基づくエージェント (circuit-based agent) のために使用された表記法と同じである。ある著者はそれを “ P が真であるかどうかを知っている” という意味で使う。二つの表現の間の移行は簡単である。

敗する。問題は非限定不確定性にある。つまり、エージェントの行為記述が不正確であるため、いくつかの予期せぬ状況がたえず起きる。それゆえ、実環境では実行モニタリングが必要なのである。それでは、次の 2 種類の実行モニタリングを考えよう。一つ目は簡単であるが、行為モニタリング (action monitoring) とよばれる弱い方法であり、エージェントは次の行為が実施されることの確認のために環境を調べる。もう一つはもっと複雑であるが、プランモニタリング (plan monitoring) とよばれる効果的な方法であり、エージェントは残りのプラン全体を確認する。

再プランニング (replanning) エージェントは、何か予期せぬことが起こったときに何をすべきかがわかつており、ゴール到達のための新プランを提案するためにプランナーエージェントをよぶ。プランニングにあまりにも長い時間を費さないために、たいてい古いプランを修正し、予期せぬ現状態からもとのプランに戻る。この世界ではエージェントがきれいな区画に移動すると時々そこにゴミが残されるが、もしエージェントがそのことを知らないか、それを気にしなければはどうなるだろうか？ そのときは、エージェントは簡単な解決である [Left] を提案するだろう。もしプランが実際に実行されているときにゴミが落ちていないれば、エージェントはゴール達成を発見する。さもなくなければ、Finishステップの前提条件である CleanL が満たされていないために、エージェントは [Suck] という新プランを作成する。このように、このプランの実行はいつも成功する。

実行モニタリングと再プランニングとはともに次のよほうな一般的な戦略を作成する。それは、完全と部分観測環境に適用でき、かつ、状態空間、半順序、条件付きプランを含む多様なプランニング表現が可能な戦略である。図 12.13 に状態空間プランニングへの一つの簡単なアプローチを示す。プランニングエージェントはゴールを目指して出発し、それを達成するための初期プランを作成する。それから、エージェントは一つずつ行為を実行する。また再プランニングエージェントは他のプランニングエージェントとは異なり、残りの実行していないプラン部分である plan と初期プラン (action monitoring) を用いる。つまり、plan 上の次の行為を実行する前に、プラン上のどの前提条件も期待に反して満たされないかどうかを確かめるために知覚したものを調べる。もし満たされなければ、エージェントは whole_plan 上のあるポイントに戻る一連の行為を再プランニングし、そのポイントに戻る。

図 12.14 にそのプロセスの図式的な例を示す。まず再プランナは plan 上の最初の行為の前提条件が現状態に合わないことに気づく。そして、再プランナはプランをよび、現状態から whole_plan 上のある状態 s へ到達する repair とよばれる新たな副プランを提案する。この例では、状態 s は残りのプランから 1 ステップ戻ったところである（これは、残りのプランよりも全体のプランの跡をたどるからである）。一般的には、現状態からできるだけ近い s を選択する。repair と s 以降の whole_plan の部分の連結する continuation によって新プランを作成し、再び実行できるようになる。

さて、今回は再プランニングを通してイスとテーブルを同じ色に塗る例題に戻ってみよう。完全観測環境であると仮定し、初期状態ではイスは青、テーブルは緑、青と赤のペンキの缶があるとする。このことから、次のように問題を定義できる：

```
function REPLANNING-AGENT(percept) returns an action
static: KB, a knowledge base (includes action descriptions)
plan, a plan, initially []
whole_plan, a plan, initially []
goal, a goal
```

```
TELL(KB, MAKE-PERCEP-SENTENCE(percept, b))
current ← STATE-DESCRIPTION(KB, t)
```

```
if plan = [] then
  whole_plan ← plan ← PLANNER(current, goal, KB)
  if PRECONDITIONS(FIRST(plan)) not currently true in KB then
    candidates ← SORT(whole_plan, ordered by distance to current)
    find state s in candidates such that
      failure ≠ repair ← PLANNER(current, s, KB)
      continuation ← the tail of whole_plan starting at s
      whole_plan ← plan ← APPEND(repair, continuation)
  return Pop(plan)
```

図 12.13 行為モニタリングと再プランニングを行うエージェント、エージェントは PLANNER とよばれる完全状態空間プランナーとアルゴリズムをサブルーンとして使用する。もし次の行為の前提条件が満たされなければ、エージェントは PLANNER がプランできるパスを探しながら、whole_plan 上の可能なポイント *p* を見いだす。このパスを *repair* とよぶ。もし PLANNER が *repair* をうまく見つければ、エージェントは *repair* は *p* 後のプランに付け加え、新たなプランを作成する。それで、エージェントはプラン上の最初のステップに戻る。

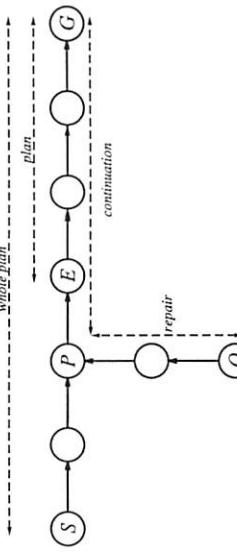


図 12.14 プランナはプランを実行する前、*S* から *G* に到達するために whole_plan とよばれるプランを立てる。エージェントは *E* とマークされたポイントまでプランを実行する。そして残りの plan を実行する前に、エージェントはいつものように前提条件を調べる。今は状態 *E* でなく状態 *O* にいることを見いだす。そのとき、エージェントはプランナーとアルゴリズムをよび、*O* からもととの whole_plan 上のある *P* に進む *repair* を作成する。新たな plan は *repair* と continuation (もともとの whole_plan の再開) が連結したもののになる。

行為モニタリング

12 章 実世界におけるプランニングと行為

```

Init(Color(Chair, Blue) ∧ Color(Table, Green)
    ∧ ContainsColor(BC, Blue) ∧ PaintCan(BC))
    ∧ ContainsColor(RC, Red) ∧ PaintCan(RC))
Goal(Color(Chair, x) ∧ Color(Table, x))

Action(Paint(object, color),
    PRECOND:PaintCan(can) ∧ ContainsColor(can, color)
    EFFECT:Color(object, color))

Action(Open(can),
    PRECOND:HavePaint(can)
    EFFECT:HavePaint(color))

EFFECT:ContainsColor(can, color)

```

また、エージェントの PLANNER は次のプランを立てるべきである。

```

[Start; Open(BC); Paint(Table, Blue); Finish]

今、エージェントはプランを実行する準備ができる。ここで、エージェントは青のベンキの缶を開けてテーブルに塗り、すべてがうまくいくとしている。前節のエージェントはプラン上のステップを完了させた時点での勝利宣言をするだろう。しかし、実行モニタリングエージェントは二つが同じ色であるという Finish ステップの前提条件を調べなければならない。ここで、エージェントが緑のテーブルの一部を塗り損ねて、二つは同じ色ではないと知覚することを想定しよう。このとき、エージェントは whole_plan 上で目的の位置を理解し、そこに到達するように一連の行為を修正する必要がある。エージェントは現在状態と Paint 行為をする前の前提条件が同じであることを確認後、repair のために空列を作成し、その plan を以前試みた [Paint, Finish] 列と同じにする。実行モニタリングは正しい場所での新プランを再開し、Paint 行為は再び試される。しかし、このループはプラン上の明示的ことを知覚するまで、この行為は繰り返される。しかし、このループによって構築されることに注意なループではなく、プラン-実行-再プランという loop だ。
```

今、エージェントは実行モニタリングのとても簡単な方法であるが、ときどき知的でない行為を導く。たとえば、エージェントがイスとテーブルを赤に塗って塗料問題を解決するプランを作成することを考えよう。ここでは、エージェントは赤のベンキの缶を開け、イスに塗るだけの塗料しかないと発見する。しかし、行為モニタリングはイスを塗り終えたあとに *HavePaint(Read)* が偽になるまで、プラン失敗に気づかない、我々が本当にやるべきことは、残りのプランがもはやうまくない状態になつたときに失敗を検出する。このプランの中で他のステップによって達成される前提条件を除いて、おのとのステップでの前提条件を調べる。プランモニタリング (*plan monitoring*) は残りのプランが失敗するまで続ける。つまり、このプランの各ボイントに、残りのプランが成功するための前提条件の注釈をつけるように

13 プランモニタリングはファンコロガシよりもエージェントを知的にする (37 ページ参照のこと)。エージェントは翼をなくしたことには気づき、他の玉を使って自分の巣をふさぐという再プランを立てるだろう。

14 アナバチ属のスズメバチ (*sphex wasp*) によって示された行動は、まさに無駄なプラン修正の繰返しだある。

15 ベージュの「間違った行為記述に対するもう一つの解決策は学習 (Learning) である。学習エージェントは何回か試みたあとに、そのキーがドアを開けるという行為記述を修正できなければなりません。一般的にばらつきのある修正プランが侵立つ。

16 エージェントがゴール到達に向けて繰り返した試みが無駄などき、つまり、その試みがある前提条件もしくは知らない結果によって阻止されるとときに問題が起くる。たとえば、もしエージェントがホテルの部屋の間違ったカードキーをもつているなら、いくらそのキーを使つてもドアは開かない。¹⁴ これに対する一つの解決策は、同じ計画を何度も試みるよりも、可能性のある修正プランが有効なアルゴリズムに選択することである。この場合、ホタルのフロントに行って、部屋に合うカードキーをとつくるという修正プランが有効な代替案となる。エージェントが真に非決定的な場合と無駄な場合を区別できることを考へると、一般的にばらつきのある修正プランが侵立つ。

プランモニタリング

13 プランモニタリングはファンコロガシよりもエージェントを知的にする (37 ページ参照のこと)。エージェントは翼をなくしたことには気づき、他の玉を使って自分の巣をふさぐという再プランを立てるだろう。

14 アナバチ属のスズメバチ (*sphex wasp*) によって示された行動は、まさに無駄なプラン修正の繰返しだある。

(37 ページ参照のこと)。

ればならない。その時点では、再プランナは新しいキーをとつて代替プランを作成するだろう。この種の学習は21章で述べる。

これらの人可能性のある改善策すべてをもつても、再プランニングエージェントにはまだいくつかの欠陥がある。まず、実時間環境では遂行できない、また再プランニングに費やす時間に制限がない、行為決定にかかる時間の制限もない、さらに、自分のための新たなゴールを定式化できなければ、現ゴールに加えて新しいゴールを受け入れることもできないので、複雑な環境ではエージェントは長く活躍できない。これらの欠陥については次の章で取り組む。

12.6 連続プランニング

本節では、環境の中で無期限に存続するエージェントを設計する。そのエージェントは与えられた単一ゴールを達成するまでプランを立て、行為するという“問題解決器(problem solver)”ではなく、むしろ常に変化するゴールの定式化、アランニング、そして、行為という一連のフェーズを通して存続する、つまり、アランナと実行モニタリングを別々のプロセスとして捉え、どちらかの結果を他方に渡すものと考えるのはなく、連続プランニングエージェント(continuous planning agent)における單一のプロセスとして考える。

このようなエージェントは、生涯存続するための壮大プランを実行する方法の一部として捉えられる。それには、実行可能なプランにおける数ステップの実行、未解決前提条件充足や矛盾解消のためのプランの洗練、そして、実行中に得られた追加情報を考慮したプランの修正がある。明らかに、エージェントは新たなゴールを定式化するときに実可行な行為がないので、部分的プランを作成するのに時間がかかる。しかし、エージェントは達成すべき独立のサブゴールをもつている場合、プランが完了する前に実行を開始することは可能である。連続プランニングエージェントはたえず世界を監視し、たとえ思考中だつたとしても新しい知覚からその世界モデルを更新する。

それでは、例題を示した後、意図的行動を表現するために半順序プランを使用するPOP-CONエージェントプログラムを説明しよう。これは、ここでは表現を簡単にするために、完全観測環境を仮定する。同じ技術は部分観測環境にも拡張できる。

用いる例は積み木の世界の問題である(11.1節)。開始状態は図12.15(a)に示すとおりである。必要な行為はMove(x, y)であり、もし両方の積み木の上に何もなければ積み木 x を y に移動する。その行為は次のとおりである。

$\text{Action}(\text{Move}(x, y))$

PRECOND: $\text{Clear}(x) \wedge \text{Clear}(y) \wedge \text{On}(x, z)$,
EFFECT: $\text{On}(x, y) \wedge \text{Clear}(z) \wedge \neg \text{On}(x, z) \wedge \neg \text{Clear}(y)$

エージェントは自分のためにまずゴールを定式化する必要がある。ここでゴールの定式化について議論するつもりはないが、その代わりにエージェントが $\text{On}(C, D) \wedge \text{On}(D, B)$ というゴールを達成するよう言われた(あるいは、自分自身で決定した)ことを想定しよう。エージェントはこのゴールのためにプラン作成を始める。プランナが問題に対する完全な解を見いだすまで知覚を遮断する他のすべてのエージェントとは異なり、連続プランニン

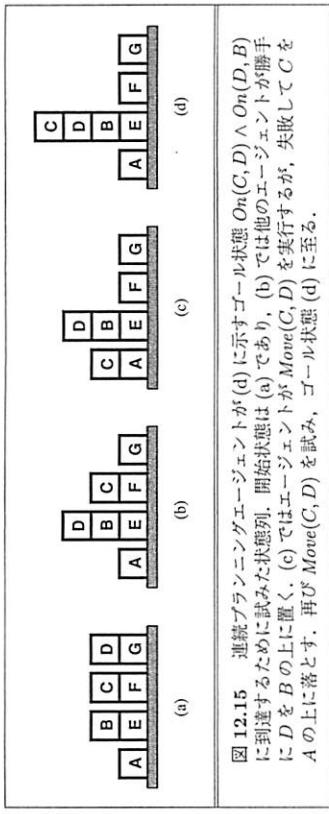


図 12.15 連続プランニングエージェントが(d)に示すゴール状態 $\text{On}(C, D) \wedge \text{On}(D, B)$ に到達するために試みた状態列。開始状態は(a)であり、(b)では他のエージェントが勝手に D を B の上に置く。 (c) ではエージェントが $\text{Move}(C, D)$ を読み、ゴール状態 (d) に至る。

エージェントは一定の時間をかけて少しずつプランを作成する。そのプラン作成ごとに、エージェントは行為として $NoOp$ を返し、知覚内容を再度調べる。ここで、エージェントの知覚内容は変わらず、すばやく図 12.16 に示すプランを作成することを考えよう。ここで注意することは、両方の行為の前提条件は $Start$ によって満たされているが、 $\text{Move}(C, D)$ の前に $\text{Move}(D, B)$ という順序制約があることである。これは $\text{Move}(D, B)$ が完了するまでに $\text{Clear}(D)$ が真であることを確認する。連続プランニングのプロセス中、 $Start$ はいつも現状態のためのラベルとして使用される。エージェントはおのおの行為のあとで、状態を更新する。

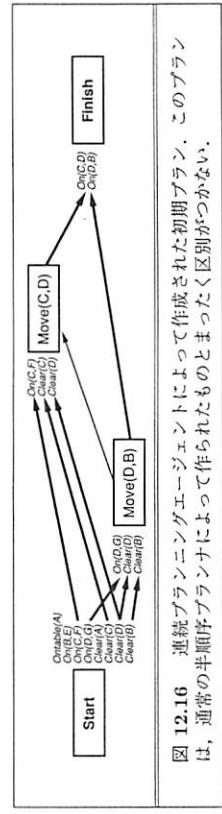


図 12.16 連続プランニングエージェントによって作成された初期プラン。このプランは、通常の半順序プランによって作られたものとまったく区別がない。

今、プランは実行可能であるが、エージェントが行為する前に自然が介入することを考へる。外部のエージェント(たとえば、イライラしたエージェントの教師)が D を B の上に動かし、世界は図 12.15(b)に示す状態になる。ここで、エージェントはそれを知り、 $Clear(B)$ と $\text{On}(D, G)$ はもはや現状態では真ではないことを認識し、それに応じて現状態のモデルを更新する。このとき、 $\text{Move}(D, B)$ 行為の前提条件である $Clear(B)$ と $\text{On}(D, G)$ を与える因果リンクは無効となるので、そのリンクはプランから除去しなければならない。新しいプランを図 12.17 に示す。 $Start$ はいつでも現状態を示しているので、ここで $Start$ は前図とは異なる。ここでプランは不完全であることに注意しよう。つまり、 $\text{Move}(D, B)$ の前提条件のうちの二つは未解決であり、特に前提条件 $\text{On}(D, y)$ は具体化されていない。なぜなら、もはや G からの移動を仮定する理由がないからである。エージェントは、 $\text{Move}(D, B) \xrightarrow{\text{On}(D, B)} \text{Finish}$ という因果リンクが $Start$ から Finish への直接リンクで置き換え可能だと気づくことによって、“役に立つ”介入の利点を得ることができる。このプロセスは因果リンクの拡張(extension)とよばれ、あとステップではなく前のステップによって新たな矛盾を引き起こさずに条件が満たされるときはいつでも

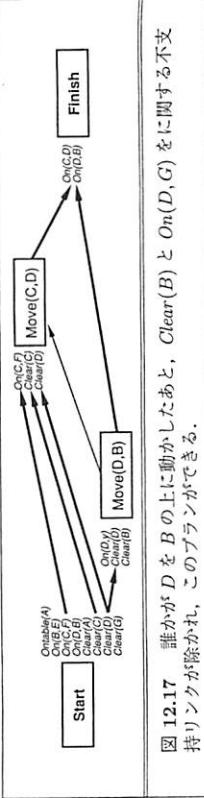


図 12.17 誰かが D を B の上に動かしたあと、 $Clear(B)$ と $On(D, G)$ をに関する不支持リンクが除かれ、このプランができる。

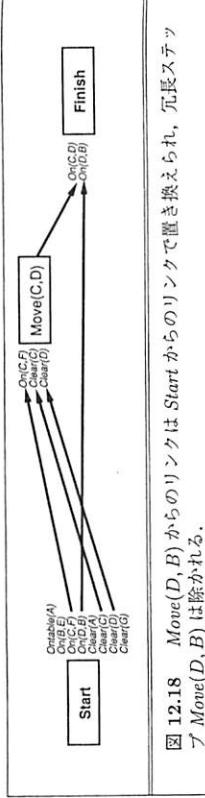


図 12.18 $Move(D, B)$ からのリンクは $Start$ からのリンクで置き換えられ、冗長ステップ $Move(D, B)$ は除かれる。

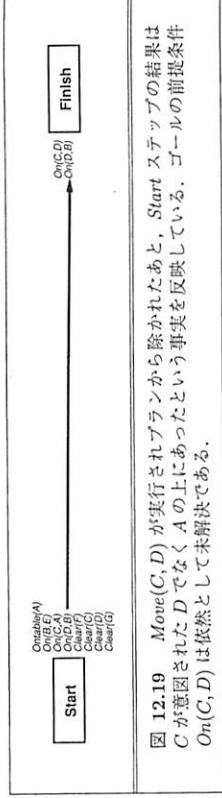


図 12.19 $Move(C, D)$ が実行された D ではなく A の上にあつたという事実を反映している。ゴールの前提条件 $On(C, D)$ は依然として未解決である。

冗長ステップ

$Move(D, B)$ から $Finish$ への古い因果リンクが取り除かれた後、 $Move(D, B)$ はいかなる因果リンクも与えず、冗長ステップ (redundant step) となる。そして、すべての冗長ステップとそれらを与えるいかなるリンクはプランから除かれ、図 12.18 に示すプランになる。ここで、 $Move(C, D)$ は実行可能である。なぜなら $Move(C, D)$ の前提条件が $Start$ ステップで満たされ、他のステップはプランから削除され、また、プラン上のどのリンクとも矛盾しないためである。そこで、このステップは不器用で C を D ではなく A の上に落とし、図 12.15 (c) に示す状態になる。新しいプランの状態を図 12.19 に示す。プラン上にはないが、 $Finish$ ステップのための条件がまだ未解決であることに注意しよう。

そこで、エージェントは未解決条件のためのプランを作成を決める。再び $Move(C, D)$ はゴール条件を満たす。その前提条件は $Start$ ステップからの新しい因果リンクによって満たされる。新たなプランを図 12.20 に示す。

これより、再び $Move(C, D)$ は実行可能となる。今回ようまくいき、図 12.15 (d) に示すゴールの状態に至る。このステップがプランから削られるとき、ゴール条件 $On(C, D)$ は再び未解決となる。しかし、 $Start$ ステップは新たな世界の状態を反映するように更新されるので、ゴール条件は $Start$ ステップからのリンクによってすぐに満たされる。これは、行為が成功するときのよくある成行きである。最終的なプランの状態を図 12.21 に示す。全てのゴール条件は $Start$ ステップによって満たされ、残された行為はないので、エージェントのゴール条件は $Start$ ステップによって満たされる。

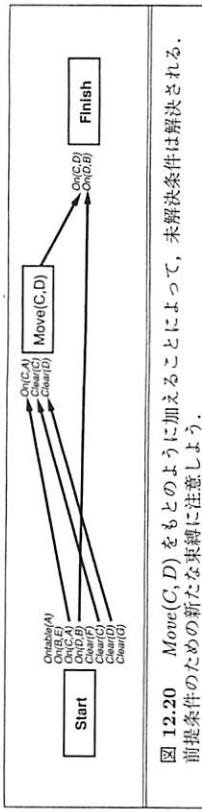


図 12.20 $Move(C, D)$ をもとのように加えることによって、未解決条件は解決される。前提条件のための新たな東縛に注意しよう。

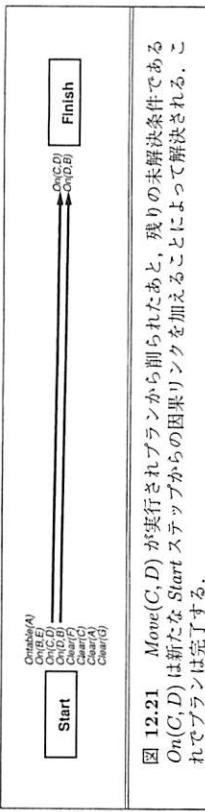


図 12.21 $Move(C, D)$ が実行されプランから削られたあと、残りの未解決条件である $On(C, D)$ は新たな $Start$ ステップからの因果リンクを加えることによって解決される。これまでプランは完了する。

ントは今や自由に $Finish$ からゴールを削除し、新たなゴールを定式化できる。この例から、連続プランニングは半順序プランニングといへん似ていることがわかる。各試行では、そのアルゴリズムは修正を必要とするプラン、いわゆる次隙プラン (plan flow) を見だし、修正する。POP アルゴリズムは未解決前提条件と因果衝突という二つの欠陥を除去するアルゴリズムとして捉えることができる。一方、連続プランニングエージェントは次のように幅広い範囲での欠陥を扱う：

- 行方不明ゴール：エージェントは $Finish$ ステップに新しいゴールを一つもしくは複数加えることができる（連続プランニングでは、名称を $Finish$ から $Infinity$ に、また $Start$ から $Current$ に変えるほうがよりわかりやすいかもしないが、ここでは伝統にしたがう）。
- 未解決前提条件：（POP にあるように）新たな行為もしくは現存の行為を選択し、未解決前提条件に因果リンクを加える。
- 因果衝突：因果リンク $A \xrightarrow{p} B$ と結果 $\neg p$ を導く行為 C が与えられたなら、(POP にあります) 衝突を解決するために順序制約が変数制約を行う。
- 不支持リンク：もし、 p が $Start$ においてもはや真でないときに因果リンク $Start \xrightarrow{p} A$ があれば、そのリンクを取り除く（これは前提条件が偽である行為の実行を防ぐ）。
- 允長行為：もし、行為 A が何の因果リンクも与えないならば、それとそのリンクを取り除く（これによって、思いがけないものを発見する能力を得られた出来事を利用できる）。
- 非実行行為：もし、行為 A ($Finish$ 以外の行為) の前提条件が $Start$ において満たされ、その行為の前に順序化された他の行為 ($Start$ を除いて) もなく、因果リンクとも衝突しないならば、 A とそのリンクを取り除き、実行されるべき行為として返す。
- 不必要な歴史的ゴール：もし、（全因果リンクが直接 $Start$ から $Finish$ に向かうよう）にプラン上に未解決前提条件も行為もなければ、現在のゴール集合は達成されている。新たなゴールのために、達成されたゴールとそれにつながっているリンクを

取り除く。

図12.22にCONTINUOUS-POP-AGENTを示す。このエージェントは“知覚、欠陥除去、行為”的サイクルからなる。つまり、エージェントは知識ベースに永続するプランを持ち、各順番ごとに一つの欠陥をプランから取り除く。そして、(しばしばNoOpになるであろう)行為を実施し、そのループを繰り返す。このエージェントは、450ページにある再プランニングエージェントの議論でリストアップした多くの問題を扱うことができる。特に、実際に時間で行為し、思いがけないものを発見する能力を扱い、自分のゴールを定式化でき、さらにもう一度プランに影響を与える予期せぬ出来事をも扱うことができる。

```
function CONTINUOUS-POP-AGENT(percept) returns an action
static: plan, a plan, initially with just Start, Finish
action  $\leftarrow$  NoOp (the default)
EFFECTS[Start] = UPDATE(EFFECTS[Start], percept)
REMOVE-FLAW(plan) // possibly updating action
return action
```

図12.22 連續半順序プランニングエージェントであるCONTINUOUS-POP-AGENT。エージェントは知識後、たえず更新されるプランから欠陥を取り除き、行を返す。実際には、NoOpを返す欠陥除去プランニングにおいてしばしば多くのステップがかかる。

12.7 マルチエージェントプランニング

今まで、エージェントが1人である単一エージェント環境(single-agent environment)を扱ってきた。環境に他のエージェントがいるとき、基本的なアルゴリズムを変えなくて、その環境のモデルの中にエージェントを簡単に組み込むことができる。しかし、多くの場合、他のエージェントの対処は自然の対処とは同じでない。貧弱(poor)なバフォーマンスを導くだろう。特に、自然是エージェントの意図¹⁵とは無関係である(と仮定する)が、他のエージェントはそうではない。この節では、このような問題を扱うためにマルチエージェントプランニングを紹介する。

2章において、マルチエージェント環境は協調的(cooperative)もしくは競争的(competitive)であることを見てきた。ここでは、テニスにおけるダブルスマッシュという簡単な協調的の例から始めよう。チームを組む両プレイヤーの行為を明記したプランを構築できるので、そのようなプランを効率的に構築する技術を述べる。しかし、効率的なプラン構築は役に立つが、必ずしも成功するとは限らない。なぜなら、エージェントは同一のプランを使用することに同意しなければならないからである。これはコミュニケーション(communication)によって達成されるる何らかの調整(coordination)を必要とする。

¹⁵ちょっとしたピクニック計画でも雨を考慮する英国の居住者は、同意しないかもしれない。

```
Agents(A, B)
Init(At(A, [Left, Baseline])  $\wedge$  At(B, [Right, Net])  $\wedge$ 
Approaching(Ball, [Right, Baseline])  $\wedge$  Partner(A, B)  $\wedge$ 
Partner(B, A))
Goal(Returned(Ball)  $\wedge$  At(agent, [x, Net]))
Action(Hit(agent, Ball),
PRECOND: Approaching(Ball, [x, y])  $\wedge$  At(agent, [x, y])  $\wedge$ 
Partner(agent, partner)  $\wedge$  At(partner, x, y))
EFFECT: Returned(Ball)
Action(Go(agent, [x, y]),
PRECOND: At(agent, [a, b]),
EFFECT: At(agent, [x, y])  $\wedge$  At(agent, [a, b]))
```

図12.23 ダブルステニス問題。2人のエージェントが一緒にプレイし、[Left, Baseline], [Right, Baseline], [Left, Net], [Right, Net]の四つのうち一つの場所にいる。1人のプレイヤーが正しい場所にいれば、ボールを打ち返すことができる。

協調：共同ゴールとプラン

ダブルスのテニスにおいてチームでプレイする2人のエージェントは、試合に勝つという共同ゴールをもっているが、そのゴールはいろいろなサブゴールを導く。ここで、チームのある時点自分で自分たちに打たれたボールを打ち返し、少なくとも2人のうち1人がネットでカバーするという共同ゴールをもつことを想定しよう。この答えをマルチエージェントプランニング(multiagent planning)として図12.23に示す。

この表記のために二つの新しい特徴を導入する。まず、*Agents(A, B)*はプランに参加している2人のエージェントAとBがいることを明言している(この問題では、相手プレイヤーはエージェントとして考えられない)。次に、どっちのエージェントが何をするかの軌跡をたどる必要があるので、個々の行為にエージェントの名前をバラメータのように明示的に記載する。

共同プラン マルチエージェントプランニング問題に対する解は共同プラン(joint plan)であり、それは個々のエージェントの行為から成り立つ。個々のエージェントが割り当てられた行為を行ってゴールが達成されるならば、共同プランは解となる。次のプランはテニス問題に対する一つの解である：

```
PLAN 1:
A: [Go(A, [Right, Baseline]), Hit(A, Ball)]
B: [NoOp(B)]
```

もし両方のエージェントが同じ知識ベースをもち、それが唯一の解であれば、すべてはうまくいくだろう。その場合、エージェントはお互いに解を決定し、それを共同で実行できる。しかし、エージェントにとって不幸なことに(その理由はすぐわかる)，上記と同じようにゴールを満たす別のプランが存在する：

```
PLAN 2:
A: [Go(A, [Left, Net]), NoOp(A)]
B: [Go(B, [Right, baseline]), Hit(B, Ball)]
```

もしAがプラン2、Bがプラン1を選べば、だれもボールを打ち返さないだろう。反対に、もしAがプラン1、Bがプラン2を選べば、おそらく2人は互いに衝突し、だれもボール

を打ち返さず、ネットはカバーされないだろう。だから、正しい共同プランが存在しても、ゴールが達成されることを意味しないのである。エージェントは、同じ共同プランに到達するために調整(coordination)メカニズムが必要となる。さらに、それは何か特有の共同プランを実行するエージェントの間で、共通する知識(10章を参照のこと)でなければならぬ。

マルチボディプランニング

マサアタリ調整に関することは後回しにして、本節では正しい共同プランの構築に専念する。これをマルチボディプランニング(multibody planning)とよぶ。これは、数種の各物理的実体(physical entity)に行為を命令できる単一の中央権利型エージェントが直面する本質的なプランニング問題である。真のマルチエージェントの場合では、個々のエージェントは一緒に実行されれば成功するであろう可能な共同プランを理解できる。

マルチボディプランニングに対するアプローチは、11.3節で述べた半順序プランニングに基づいている。ここで事物を簡単にするために完全順序を仮定するが、それでも單一エージェントの場合では生じない別の問題が生じる。それは、環境はもはや本当に静的(static)ではないことである。なぜなら、ある特定のエージェントが熟考(deliberate)している間でも他のエージェントは行動できるからである。それゆえ、同期(synchronization)に関する必要がある。そこで、簡化のために個々の行為は同じ時間を使い、共同プラン上の個々の時点での行為は同時に起ることを仮定しよう。

個々のエージェントはどんな時点においても、一つの行為(おそらくNoOpを含む)を実行する。同時に起る行為の集合を共同行為(joint action)とよぶ。たとえば、2人のエージェントAとBのテニス(455ページ)における共同行為はNoOp(A), Hit(B, Ball)である。共同プランは共同行為の半順序化されたグラフからなり、たとえばテニス問題のプラン2は次のような一連の共同行為として表現できる:

$$\langle Go(A, [Left, Net]), Go(B, [Right, baseline]) \\ \langle NoOp(A), Hit(B, Ball) \rangle \rangle$$

すべての可能な共同行為の集合に通常のPOPアルゴリズムを適用し、プランニングすることはできるだろう。唯一の問題はこの集合のサイズである。つまり、10個の行為と5人のエージェントでは 10^5 個となる。個々の行為の前提条件や結果を正確に明記することはつまらないし、そのような大きな集合を用いてプランニングすることは非効率である。

代替案としては、個々の行為がどのように他の可能な行為と相互作用するのかを記述することである。そこで暗黙のうちに共同行為を定義することがあげられる。このような定義は、ほとんどの行為が他の行為と独立しているので、より簡単になるだろう。つまり、実際に相互作用するほんの少數の行為リストのみが必要となる。そこで、通常のStripsやADLの行為記述に新しい一つの特徴、すなわち、同時発生行為リスト(concurrent action list)を拡張することによって上記のことを可能にする。これは、同時に実行されるべき行為、そうでない行為を記述することを除けば、状態変数を記述することよりもむしろ、行為記述の

ここで、Hit行為の実行中に別のエージェントによって他のHit行為が行われないという同時発生禁止制約(prohibited-concurrency constraint)がある。逆に、同時発生行為を要求することもできる。たとえば、飲み物がいっぱい入ったクーラーボックスをテニスコートへ運ぶために2人のエージェントが必要となるときである。このときの行為は同じクーラーボックスを同時にCarryする別のエージェントBがないければ、エージェントAはCarry行為を実行できないと記述される:

$$\begin{aligned} &\text{Action}(Carry(A, cooler, here, there), \\ &\quad \text{CONCURRENT: } \text{Carry}(B, cooler, here, there) \\ &\quad \text{PRECOND: } \text{At}(A, here) \wedge \text{At}(cooler, here) \wedge \text{Cooler}(cooler) \\ &\quad \text{EFFECT: } \text{At}(A, there) \wedge \text{At}(cooler, there) \wedge \neg \text{At}(A, here) \wedge \neg \text{At}(cooler, here) \end{aligned}$$

この表現を用いると、POP半順序プランナにとても近いプランナを作ることができるが、次の三つの違いがある:

1. $A \prec B$ という時間的順序関係に加えて、“同時発生”, “同時発生あるいはそれ以前”を意味する $A = B$ と $A \preceq B$ を導入する。
2. 新しい行為が同時発生行為を要求するとき、プラン上の新しい行為か既存の行為を用いて同時発生行為を具体化しなければならない。
3. 禁止された同時発生行為は追加制約である。個々の制約は衝突する行為を前か後ろにうつすことによって、解決されなければならない。

この表現は、マルチボディ領域に対してPOPと同等である。最後の二つの章を通して、このアプローチをHTN、部分観測、条件、実行モニタリング、再プランニングに拡張できるだろう。しかし、これらはこの本の範囲外である。

調整メカニズム

エージェントのグループにおいて共同プランの同意を確実にする最も簡単な方法は、共同行動に從事するより先に協定(convention)を採用することである。協定は共同プランの選択に関する制約であり、もし全エージェントがそれを採用するならば、共同プランがうまくいくための基本的な制約の範囲を越えるものとなる。たとえば、“あなたのコートに張りついていなさい”という協定は、ダブルスのパートナーにプラン2を選ばせる。一方、“1人のプレイヤーは常にネットにいる”という協定は、社会的法則(social law)と考えられている。人間の言語もまた協定どみなすことができる。

前の段落の協定は問題領域特有であり、行為記述から協定違反を排除することによって実施できる。さらに一般的なアプローチは、問題領域独立な協定を使用することである。たとえば、もし個々のエージェントが同じ入力に対して同じマルチボディプランニアーアルゴリズムを実行すれば、他のエージェントも同じ選択をするだろうと確信して、最初に



見つけた実行可能な共同プランを実行する協定にしたがうことができる。さらにはロバストだがよりコストのかかる戦略は、すべての共同プランを構築し、その中から一つを選ぶ、たとえばアルファベット順で最初のものを選ぶことである。

協定は進化の過程でも発生する。たとえば、群れで生活する社会性昆蟲のコロニーでは、コロニーに住む昆蟲に共通な遺伝構造(genetic makeup)によって容易になった精巧な共同プランが実行される。また、協定からの逸脱は進化的な適合性を減少させるという事実から、どのような実行可能な共同プランも安定平衡になるために協定一致(conformity)はさせられる。同様の考えは人間の言語の発達にも適用できるが、ここで重要なことは各人がどの言語を話すべきかということではなく、すべての人々が同じ言語を話すという事実である。最後の例は鳥の群れ行動の例をあげる。もし、個々の鳥エージェント(時々、birdoidとかbirdとよばれる)が、次の三つのルールを組み合せて実行するなら、もっともらしいシミュレーションを得られる。

1. 分離(separation)：とても接近したときは、近所の鳥から離れなさい。

2. 疋集(cohesion)：近所の鳥の平均的な位置に向かなさい。

3. 整列(alignment)：近所の鳥の平均的な方位(方向)へ向かなさい。

もしすべての鳥が同じ方針を実施すれば、その群れは時間がすぎても散乱せず、一定の密集度をもった擬似剛体(pseudo-rigid body)として飛行する創発的行動(emergent behavior)をみせる。昆虫のときと同様に、個々のエージェントは他のエージェントの行為をモデル化した共同プランを実施する必要はない。

典型的に、協定は個々の問題に対して新たに開発されるよりはむしろ、個々のマルチエージェントプランニング問題の世界をカバーするために適用される。しかしこのことはダブルスにおいてボールが2人のプレイヤからほぼ同じ距離にあるときに時にみられるように、剛直(inflexibility)や崩壊(breakdown)を導く可能性がある。適用できる協定がない場合は、エージェントは実行可能な共同プランの共通知識を得るために、コミュニケーション(communication)を使用することができます。たとえば、ダブルステニスのレイヤは好ましい共同プランを指示するために、“私のボール”とか“あなたのボール”と叫ぶことができる。コミュニケーションメカニズムについては22章でもっと深く見ていくが、コミュニケーションは必ずしも言葉の交換を含む必要はない。たとえば、1人のプレイヤは共同プランの最初の部分を単に実行することによって、もう1人のプレイヤに好ましい共同プランを伝えることができる。テニス問題では、もしエージェントAがネットに向って進むならば、エージェントBはボールを打つためにベースラインへ戻る必要がある。なぜなら、プラン2はエージェントAがネットに向って進むことから始まる唯一の共同プランだからである。調整に対するこのアプローチは時々プラン認識(plan recognition)とよばれ、單一行為(あるいは短い一連の行為)によって共同プランを曖昧なく決定できる場合にうまくいく。

エージェントが成功する共同プランに到達できるかどうかは、エージェントの設計者があるいはエージェント自身による。前者の場合、エージェント設計者はエージェントがプランする前にエージェントの政策や戦略が成功することを証明すべきである。このとき、エージェント自身はもし今いる環境に対してもく機能するなら即応的に行動でき、他のエージェントにに関する明確なモデルを必要としない。後者の場合、エージェントは熟考的であり、他のエージェントの推論を考慮しながら、自分のプランが効果的であることを証

明するか実際にやってみせなければならぬ、たとえば、論理的エージェントAとBがいる環境では、両者とも次の定義をもつことができる。

$$\forall p, s \text{ Feasible}(p, s) \Leftrightarrow \text{CommonKnowledge}(\{A, B\}, Achieves(p, s, Goal))$$

これはどのような状況 s においても、もしプラン p によってゴールを達成できることがエージェント間の共通知識ならば、プラン p は実行可能な共同プランであることを示している。特別な共同プランを実行するための共同意図(joint intention)に関する共通知識を確立するには、さらに公理が必要となる。そして、そのときだけエージェントは行動を開始できる。

競争

競争

すべてのマルチエージェント環境は協調的エージェントで構成されているわけではない、矛盾する効用関数をもつエージェントは、互いに競争(competition)状況にある。この例として、チエスのように2人でプレイするゼロ和ゲーム(zero-sum game)があげられる。6章では、チエスをするエージェントは数ステップ先の相手の可能な手を考える必要があることを見てきた。つまり、競争環境にいるエージェントは(a)他のエージェントがいることを認識し、(b)他のエージェントの可能なプランをいくつか計算し、(c)他のエージェントのプランがどのように自分のプランと相互作用するのかを計算し、そして、(d)これらが相互作用から最も良い行為を決定しなければならない。だから、競争は協調のように他のエージェントのプランにおけるモデルを必要とする一方で、競争的な環境では共同プランは関係ない。

12.4節では、ゲームと条件付きプランニング問題の類似性を示してきた。図12.10における条件付きプランニングアルゴリズムは、環境に関して最悪ケースを仮定しても機能するプランを構築するので、そのアルゴリズムはエージェントが成功と失敗のみに関心がある競争的状況にも適用できる。もしエージェントとその相手がプランのコストを分配するなら、ミニマックス法が適切である。しかしながら、POPやHTNプランニングのような方法とミニマックス法を結びつけ、6章で使われた状態空間探索モデルを超える研究はほとんどない。競争に関する問題はゲーム理論を扱う17.6節で戻ろう。

12.8 まとめ

本章では、実世界におけるプランニングと行為の複雑さに取り組んできた。主要点は次のとおりである。

- 多くの行為は資金、ガソリン、原料などの資源を消費する。これらの資源はたとえば個々の硬貨や紙幣に換算して推論するよりも、数値的な尺度として扱うほうが便利である。また、行為は資源を消費あるいは生産することができるのが、部分的な計画をさらに洗練する前に、その計画の資源制約を調べることははたいで、安価で効率的である。
- 時間は最も重要な資源の一つである。時間は専門のスケジューリングアルゴリズム

によって扱うことができれば、スケジューリングがプランニングと統合されることも可能である。

- 階級的タスクネットワーク(HTN)プランニングにより、エージェントは問題領域の設計者から分解ルールの形式を用いて助言を受けることができる。これは、多くの実世界応用で求められている大規模プランの作成を可能にする。
- 標準的なプランニングアルゴリズムは、完全で正確な情報と決定的で完全観測環境を仮定している。多くの問題領域はこの仮定に違反する。
- 必要な情報を得るために知覚行為の使用をプランニングすることによって、不完全情報を扱うことができる。エージェントはプランなどの分岐をたどることを決定するために、条件付きプランはエージェントが実行中に外界を知覚した結果を用いる。ある場合は、知覚を必要とせずに動作するプランを作成するために、センサレスか順応プランニングが使用できる。センサレスと条件付きプランはともに信念状態の空間を探索することによって作成される。
- 不正確な情報は、行為とプランに対して前提条件を満たさない結果を導く。実行モニタリングはプランが成功するためには前提条件の違反を検出する。
- 再プランニングエージェントは実行モニタリングを用い、必要に応じて修繕したプランを継いでいく。
- 連続プランニングエージェントは行動しながら新たなゴールをつくり、実時間で応じる。

- マルチエージェントプランニングは、環境の中に協力、競争、あるいは、調整する他のエージェントがいるときには、マルチボディアランニングは、共同行為記述を効果的に分解して共同プランを作成するが、もし2人の協力的エージェントがどちらの共同プランを実行するかで合意するならば、マルチボディプランニングには調整のためのいくつかの形式が増やされるにちがいない。

文献と歴史ノート

連続時間を用いたプランニングはDEVISER (Vere, 1983)によって初めて扱われた。プラン上の時間の系統的表現に関する問題はFORBINシステムの中でDean *et al.* (1990)によつて取り組まれた。NONLIN+(Tate and Whiter, 1984)とSPE(Wilkens, 1988, 1990)は、プラン上の様々なステップで制限されている資源分配の論議を可能にした。HTP(Planer and Steel, 1988; Young, Pollack, and Moore, 1994; Barrett and Weld, 1994; Kamblampati, Mali, and Srivastava, 1998)は、本質で取り上げたハイブリッドアプローチを提案しており、その中の分解は半順序プランニングは下位レベルの行為の前提条件を無視することを許可する抽象的階層(abstraction hierarchy)の考えを導入した。Austin Tateの博士論文(1975b)とEarlSacerdoti(1977)による研究は、現代の形式におけるHTNプランニングの基礎的考え方を開発した。O-PLANとSPEを含む多くの実践的プランナはHTNプランナである。Yang (1990)はHTNプランニングを効果的にする行為の特性を論議している。また、Erol, Hendler, Nau (1994, 1996)は純HTNプランナにおける複雑な結果の範囲とともに、完全階層分解プランナも記述している。他の著者(Ambros-Ingerson and Steel, 1988; Young, Pollack, and Moore, 1994; Barrett and Weld, 1994; Kamblampati, Mali, and Srivastava, 1998)は、本質で取り上げたハイブリッドアプローチを提案しており、その中の分解は半順序プランニングのゴールの一つは、一般化されたプランの形で以前のプランニング経験を再使用することであった。19章に詳しく述べられる説明に基づく学習(explanation-based learning)の技術は、以前に計算されたプランを一般化する手段としてSOAR(Laird, Rosenbloom, and Newell, 1986)やPRODIGY(Carbonell, Knoblock, and Minton, 1989)などを含むいくつかのシステムに応用された。もう一つのアプローチとしては、以前に計算されたプランを原型のままで記憶し、元問題と類似した新しい問題を解くために再利用することである。これは、事例に基づくプランニング(case-based planning)(Carbonell, 1983; Alterman, 1988; Hammond,

增幅された処理能力によって実用的応用への挑戦が可能になり、時間プランニング研究が振り向いた。SAPA(Do and Kambhampati, 2001)とT4(Hashem and Geffner, 2001)はともに期間と資源をもつ行為を扱うために、洗練されたヒューリスティックスと前向き状態空間を使用した。他の方法としてはとても表現力のある行為言語の使用が考えられるが、APSEN(Fukunaga, Rabideau, Chien, and Yan, 1997), HSTS (Jonson *et al.*, 2000),そして, IXtET(Ghallab and Laruelle, 1994)によって行われているように、その言語を人間によって記述された領域特定のヒューリスティックスによって洗練する必要がある。

航空宇宙では長いスケジューリングの歴史がある。T-SCHED(Drable, 1990)は、UOSAT-II衛星における一連のミッションコマンドをスケジュールするために使用された。O-PLANに基づくOPTRIMUM-ATV(Aarup, Arentoft, Parrot, Stader, and Stokes, 1994)とPLAN-ERSI(Fuchs, Gasquet, Olakainty, and Currie, 1990)は、欧州宇宙局(European Space Agency)において宇宙船の組立と観測プランニングに使用された。SPIKE(Johnston and Adorf, 1992)はハッブル宇宙望遠鏡のためにNASAで脚脚スケジューリングシステム(Deale, Yvanovich, Schnitzius, Kautz, Carpenter, Zweben, Davis, and Dunn, 1994)は16,000人までの労働者交替のジョブプロジェクトスケジューリングを行っている。リモートエージェント(remote agent)(Muscettola *et al.*, 1998)は1990年Deep Space One探査機に搭載され打ち上げられ、宇宙船を制御する最初の自律的プランナ-スケジューラーになった。オペレーションズリサーチにおけるジョブショップスケジューリングの文献はVaessens *et al.* (1996)によって調査され、理論的結果は(Martin and Shmoys, 1996)によって発表された。

一連の原始的ステップからなる“マクロオペレータ”的macropsを学習するSTRIPSプログラムの便益は、階層的プランニング(Flikes, Hart, and Nilsson, 1972)のための最初のメカニズムに見られる。階層はLAWALYシステム(Siklossy and Dreissi, 1973)においても使用された。ABSTRACTSシステム(Sacerdoti, 1974)では、ワーキングプランの一般的構造を導くために、上位レベルのプランニングは下位レベルの行為の前提条件を無視することを許可する抽象的階層(abstraction hierarchy)の考えを導入した。Austin Tateの博士論文(1975b)とEarlSacerdoti(1977)による研究は、現代の形式におけるHTNプランニングの基礎的考え方を開発した。O-PLANとSPEを含む多くの実践的プランナはHTNプランナである。Yang (1990)はHTNプランニングを効果的にする行為の特性を論議している。また、Erol, Hendler, Nau (1994, 1996)は純HTNプランナにおける複雑な結果の範囲とともに、完全階層分解プランナも記述している。他の著者(Ambros-Ingerson and Steel, 1988; Young, Pollack, and Moore, 1994; Barrett and Weld, 1994; Kamblampati, Mali, and Srivastava, 1998)は、本質で取り上げたハイブリッドアプローチを提案しており、その中の分解は半順序プランニングのゴールの一つは、一般化されたプランの形で以前のプランニング経験を再使用することであった。19章に詳しく述べられる説明に基づく学習(explanation-based learning)の技術は、以前に計算されたプランを一般化する手段としてSOAR(Laird, Rosenbloom, and Newell, 1986)やPRODIGY(Carbonell, Knoblock, and Minton, 1989)などを含むいくつかのシステムに応用された。もう一つのアプローチとしては、以前に計算されたプランを原型のままで記憶し、元問題と類似した新しい問題を解くために再利用することである。これは、事例に基づくプランニング(case-based planning)(Carbonell, 1983; Alterman, 1988; Hammond,

事例に基づくプランニング

1989)とよばれる分野で使用されるアプローチである。Kambhampati (1994)は、事例に基づくプランニングは洗練されたプランニングとして分析されるべきだと主張し、事例に基づく半順序プランニングの形式的な基礎を築いた。

実環境の予期困難性と部分観測性は、Shakey (Fikes et al., 1972)と FREDDY (Michie, 1974)を含むプランニング技術を用いたロボティクス・プロジェクトで当初認知されたが、この問題は McDermott (1978a) の *Planning and Acting* という影響力の強い論文が発表されたあと、たいへんな注目を浴びた。

条件とループに欠けていた初期のプランナは、条件付きプランニングの概念を明白に認識していなかった。しかしそれにもかかわらず、初期のプランナは環境の不確実性に即応する強制的なやり方に時々頗っていた。Sacerdoti の NOAH は、プランナが初期状態をほとんど知らない“壁と箱”というプランニングのチャレンジ問題を解決するために強制 (coercion) を使用した。Mason (1993) は、ロボティクス・プランニングにおいてはセンサをしばしば省くことができるか、もしくは省くべきと主張し、初期の位置によらず一連の行為によって道具を机上の特定の位置に移動させる知覚なプランを示した。この考えについてはロボティクスの章で述べる(図 25.17 を参照のこと)。

Goldman and Boddy (1996)は、世界を既知の状態に強制することにより不確実性を扱うセンサレスプランナのために順応プランニング (conformant planning) という用語を導入し、センサレスプランはたとえエージェントがセンサをもついててもしばしば効果的であることを示した。適度に効果的な初期の順応プランナは Smith and Weld (1998)による順応グラフプラン (Conformant Graphplan: CGP) であった。Ferraris and Giunchiglia (2000)と Rintanen (1999)は独立に SAT プランに基づく順応プランナを開発した。Bonet and Geffner (2000)は 1960 年代に、部分観測マルコフ意思決定過程あるいは POMDP (17 章を参照)のために初めて開発されたアイデアを継りながら、信念状態空間におけるヒューリスティック探索に基づく順応プランナを示した。最近、HSCP (Bertoli, Cimatti, and Roveri, 2001a) のように最も計算の早い信念状態に基づく順応プランナは信念状態を表すために二分決定グラフ (binary decision diagram: BDD)(Bryant, 1992)を使用し、CGPよりも 5 倍早く計算できる。

WARPLAN の変種である WARPLAN-C (Warren, 1976)は条件付き行為 (conditional action) を用いた最も初期のプランナの一つである。Olawski and Gini (1990)は条件付きプランニングに関係する重要な問題をまとめている。
本章で述べられている条件付きプランニングのアプローチは、Jimenez and Torras (2000)と Hansen and Zilberstein (2001)によって開発された周期内の AND-OR グラフのための効果的な探索アルゴリズムに基づいている。Bertoli, Cimatti, Roveri, and Traverso (2001b)は、ループをもつた条件付きプランを作成する BDD に基づくアプローチを述べている。C-BURIDAN (Draper, Hanks, and Weld, 1994)は確率的な結果を起こす行為のための条件付きプランニングを扱っている。なお、ここでは POMDP (17 章を参照) の節でも扱われている。

条件付きプランニングと自動プログラム合成は密接な関係があり、9 章で多くの参考文献があげられている。しかし、これら二つの分野は計算機命令の実行コストとロボットやマニピュレータによる行為の実行コストの間で大きな差があるため、別々に探求されてきた。Linden (1991)はこれら二つの分野間での明示的な相互発達の促進を試みている。

そのためどのように拡張されたのかを理解することができ、検索に基づく技術は信念空間に基づくアプローチである。SATPLAN は確率的な SATPLAN (Majercik and Littman, 1999)と限量化されたブール論理 (Rintanen, 1999)を用いるプランニングを導き、半順序プランニングは UWL (Etzioni, Hanks, Weld, Draper, Lesh, and Williamson, 1992), CNLP (Peot and Smith, 1992),そして CASSANDRA (Pryor and Collins, 1996)を導いた。GRAPHPLAN はセンサグラフプラン (sensory graphplan) もしくは SGP (Weld et al., 1998)を導いたが、完全確率 GRAPHPLAN はまだ開発されていない。

実行モニタリング (execution monitoring) を最初に本格的に取り扱ったのは、Shakey ロボットを制御するために STRIPS プランナを用いた PLANEX (Fikes et al., 1972)であった。PLANEX は完全な再プランニングを行わずに部分的な実行による失敗から回復させるために、三角表 (triangle table) を使用した。この表はプラン上の各時点での前提条件を効果的に保存するメカニズムである。実行に関する Shakey のモデルは 25 章でさらに議論する。NASL プランナ (McDermott, 1978a) は実行とプランニングを完全に統合するために、プランニングの問題を単に複雑な行為をして扱った。NASL は複雑な行為を推論するために定理証明 (theorem proving) を用いている。

SPIE (System for Interactive Planning and Execution monitoring) (Wilkins, 1988, 1990) は再プランニング (replanning) の問題を系統的に扱う最初のプランナであった。それは、航空母艦のフライドッキにおけるプランニング操作やオーストラリアのビール工場でのジョブシップスケジューリングなどの各種領域で使用されている。また、別の研究ではプランナが挑戦する中で最も複雑な問題領域の一つである多層ビルの建設計画に SPIE を活用した。

IPEM (Integrated Planning, Execution, and Monitoring) (Ambros-Ingerson and Steel, 1988) は、連続的プランニングエンジニアメントを生みだすために半順序プランニングと実行を統合した最初のシステムであった。CONTINUOUS-P-OP-AGENT は IPEM, PUCCINI プランナ (Golden, 1998), そして CYPRESS システム (Wilkins, Myers, Lowrance, and Westley, 1995) のアイデアを融合したものである。

1980 年代の中頃には、半順序プランニングと関連技術では実世界エージェントにとって効果的な行為を早く生成することは不可能と信じられていた (Agre and Chapman, 1987)。その代わりに、即応プランニング (reactive planning) システムが提案された。それらは基本的に内部状態をもつ反射エージェントであり、条件行為ルールの様々な表現を用いて実装される。Brooks (1986) の包摂アーキテクチャ (subsumption architecture) (7 章と 25 章を参照) では、歩行ロボットや車輪ロボットの運動を制御したり、障害物を避けるために層状 (layered) の有限状態機械を用いた。Pengi (Agre and Chapman, 1987) は、現ゴルの“視覚的”な表現とエージェントの内部状態を結合させたブール回路を用いて (完全観測の) ビデオゲームで遊ぶことを可能にした。

“普遍的プラン” (universal plan) (Schoppers, 1987) は、即応プランニングのためのテーブル参照手法 (look-up-table method) として開発されたが、これは以前からマルコフ決定過程の中で使用された政策 (policy) の考え方の再発見であることが判明した。普遍的プラン (あるいは政策) では、その状態でとられるべき行為の写像 (mapping) をもつてある。Ginsberg (1989) は、即応プランニング問題のいくつかの定式化において手に負えない結果を指摘し、普遍的プランを厳しく批難した。これに対し、Schoppers (1989) も激烈に反振り返ると、重要な古典的プランニングアルゴリズムが不確実性を伴う問題領域に対処す

マルチエージェントプランニングの歴史があるが、近年、高い人気を集めている。Konolige (1982) は一階述語論理でマルチエージェントプランニングの定式化を図り、Pernaut (1986) は STRIPS スタイルの記述を提供了。エージェントが共同プランを実行する場合に重要な共同意図の考えは、伝達行為の研究 (Cohen and Levesque, 1990; Cohen, Morgan, and Pollack, 1990) からきている。マルチボディ半順序プランニングの表現は Boutiller and Braffman (2001) の研究をもとにしている。

ここでは、マルチエージェントプランニングの交渉に関する研究の表面をわざかにかじつたにすぎない。Durfee and Lesser (1989) は、交渉によってタスクがエージェント間でどうに共有され得るのかを議論した。Kraus, Ephrati, and Lehmann (1991) は交渉、連合形成と分解、そして、不正行為を必要とする外交 (Diplomacy) というボードゲームを遊ぶためのシステムを示した。Stone (2000) は、競争的、動的、かつ部分観測環境のロボットサッカーにおいて、エージェントがいかにチームメイトとして協力できるかを示した。(Weiss, 1999) はマルチエージェントの概観がまとめられた本である。458 ページにあるボイド (void) モデルは Reynolds (1987) によるものだが、彼は「バットマンリターンズ」でのモデルをコウモリとベンギンの群れに応用したとしてアカデミー賞を受賞した。

練習問題

12.1 12.1節における時間と資源の表現を注意深く調べよ。

- なぜ *Duration(d)* を行為の結果とすることが、*DURATION: d* 形式の行為に別のフィールドをもつよりも良い考え方であるのか？（ヒント：条件付き結果と選言的結果を考えよ）
- なぜ、*RESOURCE:m* は行為の結果であるよりも、行為の中の別フィールドに表現されるのか？

12.2 消費可能資源 (consumable resource) は行為によって（部分的に）使用される資源である。たとえば、自動車にエンジンを取り付けるにはボルトが必要であるが、ボルトは一度使用されると他の取付けには利用できない。

- 最初に 100 個のボルトがあり、エンジン E_1 に 40 個のボルト、エンジン E_2 に 50 個のボルトが必要とするには、図 12.3 の表現をどのように修正するかを説明せよ。+ と - の符号は資源のための結果リテラルに使用してもよい。
- 消費可能資源を扱うために、半順序プランニングにおける因果リンクと行為の間の競合 (conflict) の定義をどのように修正すべきかを説明せよ。
- たとえば、工場にボルトを再供給するとか、自動車に再度ガソリンを入れるなどのいくつかの行為は、資源の利用可能性を上げることができる。行為が資源の利用可

能性を上げなければ、資源は単調には増大しない、探索空間を切り詰めるために、この特性をどのように使うのかを説明せよ。

12.3 図 12.7 における *HireBuilder* と *GetPermit* ステップを分解 (decomposition) し、分解されたサブプランをどのように全体プランに結合させるのかを示せ。

12.4 家を建てる問題において、ステップを共有しないと一貫したプランに融合できない二つの抽象的なサブプランの例を示せ（ヒント：家において二つの物理的部品が一緒にない場所は、二つのサブプランが相互作用する傾向の高い場所である）。

12.5 HTN プランニングの利点は、非 HTN の表記では表せない問題、たとえば“ロサンゼルスとニューヨークの往復”的ように、スタートとゴール状態が同じ (*Att(L,A)*) である問題を解決できることと言われている。この問題を HTN なしに表現し、解決する方法が考えられるか？

12.6 条件 p を達成する行動を示す *Achieve(p)* 表記を用いてことによつて、どのように標準的な STRIPS 行為記述を HTP 分解として書き直せるかを示せ。

12.7 標準的なプログラミング言語におけるいくつか操作は、世界の状態を変える行為としてモデル化できる。たとえば、代入操作はある器機のメモリ内容をコピーする一方、プリント操作は出力ストリームの状態を変える。これらの操作からなるプログラムは、プログラムの仕様に記載されたゴールをもつプランと考えられる。それゆえ、プランニングアルゴリズムは、与えられた仕様を達成するプログラムを構築するために用いることができる。

- 代入操作（ある変数値を他に代入する）のための操作スキーマを記述せよ。もとの値は上書きされることを覚えておくこと。
- 一時的な変数を用いて二つの変数値を交換するプランを作成するために、プランナによってどのようにオブジェクト作成が用いられるかを示せ。

12.8 次の議論を考えよ。不確定な初期状態を評している枠組みでは、選言的結果 (disjunctive effect) は付加的な表現力の源ではなく、單なる都合の良い表記である。選言的結果 $P \vee Q$ をもついかなる行為スキーマ a に対して、それをいつも when $R: P \wedge \text{when } \neg R: Q$ という条件付き結果に置き換えて、二つの正規行為に変形できる。命題 R は初期状態では未知で、何の知覚行為もないランダムな命題を示している。この議論は正しいのか？ ブラン上に行為スキーマ a の事例が一つある場合と、一つ以上ある場合と、一つ以上ある場合と、一つ以上ある場合とを別々に考えよ。

12.9 なぜ、条件付きプランニングは限界のない不確定性を扱えないのか？

12.10 積み木の世界では、Clear述語の正確性を維持するために *Move* と *MoveToTable* という二つのSTRIPS行為の導入を余儀なくされた。一つの行為を用いて両ケースを表現するためには、どのように条件付き結果を使用するのかを示せ。

12.11 口ボットのいる場所によつてきれいになるかどうかが決まる掃除機の世界では、Suck行為のために条件付き結果が説明された。Suckが無条件記述をもつように、掃除機世界の状態を定義するために新たな命題変数の集合を考えることができるか？ あなたの命題を用いて Suck, Left, そして Right の記述を書き出し、それらが世界のあらゆる可能な状態を記述するのに十分であることを示せ。

12.1.2 きれいな目的区画に移動し、あるいは、きれいな区画で *Suck* を適用したときにゴミを残すという二重マーフィーの電気掃除機に対して、完全な *Suck* 記述を書け。

12.1.3 適度に汚いカーペットを見つけ、障害物を取り除いて、掃除機をかけよ。できる限り正確に電気掃除機の通るバスを描け。本章で議論したプランニングの形式を参照してバスを説明せよ。

12.1.4 次の引用はシャンプーボトルの裏に書かれていて、(a) “泡をたて、洗い流し、それを繰り返しなさい”。(b) “頭皮にシャンプーをつけ、数分間そのままていなさい、必要に応じて洗い流し、それを繰り返しなさい”。(c) “もし問題があれば、医者に診てもらなさい”。おののが無条件プラン、条件付きプラン、あるいは実行モニタリングプランであることを明らかにせよ。

12.1.5 図 12.10 における AND-OR-GRAPH-SEARCH アルゴリズムは、根から現状態までのバスにおいてのみ繰り返される状態を調べる。ここでは、これに加えてアルゴリズムは訪れたすべての状態を保存し、そのリストを調べることを想定する（一例として、図 3.19 における GRAPH-SEARCH を参照）。保存されるべき情報を決定せよ。また、繰返しの状態を見ついたときにはアルゴリズムがどのように保存された情報を使用すべきかを決めよ（ヒント：成功副プランが以前作成された状態と副プランが見つけられない状態とを少なくとも区別する必要がある）。副プランの複数コピーの保持を避けるために、どのようにラベル（label）を使用するかを説明せよ。



12.1.6 もし非周期的プランが存在しないなら、周期的プランを作成するために、どのように AND-OR-GRAPH-SEARCH アルゴリズムを修正するのかを正確に説明せよ。それには次の三つの課題を扱う必要があるだろう。つまり、周期的プランがプランの先頭部分に戻れるようにラベルを付けること、周期的プラン発見後に非周期的プランを探しつけるように OR-SEARCH を修正すること、そして、プランが周期的かどうかを示すためにプラン表現を増やすことである。(a) 三重マーフィーの掃除機世界と(b) 二者択一の二重マーフィーの掃除機世界において、あなたのアルゴリズムがどのように動作するかを示せ。その結果を調べるために、コンピュータで実装したいかもしれないが、標準的なループ文法を用いてケース (b) のためのプランを書くことができるかを答えよ。

12.1.7 部分観測環境に対して信念状態の更新手続きを完全に示せ。すなわち、現信念状態の表現と条件付き結果を伴う行為記述から、(知識)リストのような新しい信念状態の表現を計算する方法のことである。

12.1.8 *Right* と *Suck* 行為のために、式 (12.2) と類似する行為記述を書け。また、式 (12.3) と類似する *CheckLocation* も記述せよ。このとき練習問題 12.11 からの二者択一の命題集合を再度使用せよ。

12.1.9 450 ページに示される再プランニングエージェントができるない事柄のリストを見て、一つかそれ以上の事柄を処理できるアルゴリズムを描け。

12.20 次の問題を考えよ。脱水か病気 *D* のどちらか（両方ではない）によって引き起こされた兆候がある患者が医師のところに到着する。このとき、二つの可能な行為がある。それは、無条件に脱水をなおす *Drink* か、患者が脱水のときには好ましくない副作用が出る

が、病気 *D* を治す *Medicate* である。この問題を PDDL を用いて記述し、すべてに関連する可能性のある世界を列挙して問題を解決するセンサレスプランを図解せよ。

12.21 *Disease* が真であり、どんな場合でも *Known(CultureGrowth)* という知覚結果をもつとき、*CultureGrowth* という条件付き結果をもつ *Test* 行為を、先の練習問題 12.20 である治療問題に追加せよ。問題を解決する条件付きプランを図解し、*Medicate* 行為の使用を最小限にせよ。