BDIモデルに基づく自律エージェントの行動決定アルゴリズムの拡張および応用

2012年1月12日

奈良女子大学大学院 人間文化研究科
博士後期課程 複合現象科学専攻

藤田 恵

要　旨

近年、実世界において目的達成のために自律的に振る舞うロボットの実現が重要な研究課題となってきている。特に、そうしたロボットには周囲の環境の多様な変化に適応して、複数の目的を整合的に達成しようとする能力が要求されることが指摘されている。

これまでのロボット制御のためのエージェント技術としては、ホンダ社の ASIMO、産業技術総合研究所などによる HRP-2 や HRP-4C、大阪大の石黒らによるジェミノイドなどが広く知られる。しかしこれらは、二足歩行などの基本的な行動を精密に行うことはできるようになってきているが、その行動決定は人間による指令、または行動計画モジュールを用いた状態遷移などにより実現されたものであり、基本的に単一の目標達成に特化したものである。そうした技術のみでは、動的環境下で複数の目的を並行に管理して、整合的に達成することは難しい。特に、状態やモジュールの単純な切り替えによる制御では、並行に達成すべき目的の数が増えるほど、状態遷移の記述が困難になる。

本論文で我々は、このような実世界で複数の目標を達成するロボットの実現には、Rao らによる BDI モデルを適用した BDI ロボットが有効であることを示し、また実世界への適応に必要な改良についても述べる。

BDI モデルとは、信念 (B)、願望 (D)、意図 (I) と呼ばれる 3 つの心的状態とその時間的変化を用いて、Bratman の「意図の理論」に基づく合理的な行動決定を行う自律エージェントをモデル化したものである。BDI モデルによるエージェントアーキテクチャを BDI アーキテクチャと呼び、BDI アーキテクチャに基づいて実装されたエージェントを BDI エージェントと呼ぶ。BDI エージェントは「意図の理論」で扱われている人間の行動様式を模倣し、動的な環境下において、限られた信念・推論能力のもとで行動計画を部分的に具体化し達成することが可能である。また、意図の概念を明示的に持つことで、合理的な行動決定、一貫性のある行動、外部から見た行動の説明などを行うことも可能となる。

特に、BDI エージェントを用いると、上述した「意図」の概念を並行に保持することにより、複数の行動を切り替えながら行動をすることが可能となり、タスク切り替えのコードを人間が明示的に記述することなく、BDI エージェントの本来持つ機構によってタスクの切り替えをすることができる。これは、実世界で複数の目標を達成するためには非常に有用な性質である。

また、BDI モデルは BDI logic と呼ばれる様相論理体系による形式化を持ち、これによりエージェントの性質や振る舞いなどに関する形式的な議論を行うことが可能である。従って、これに基づいて作られたエージェントの動作を形式的に検証することができ、これが BDI モデルの大きな利点の 1 つである。

本論文ではまず、BDI モデルとそのエージェントアーキテクチャについて述べる。次いで、Nau らによるプランナ SHOP2 を我々が改造した動的プランナと BDI エージェントとの結合によって、BDI モデルそのものには与えられていないエージェントの行動方針決定アルゴリズムを柔軟に提供する方法について述べる。またこの方式が、実世界上で生じるロボットのセンサデバイスの誤認に起因するエージェントの信念の誤りによって行動に支障を生じた場合の対処に有効であることを実験で示す。次いで、実世界においてはゴール達成の過程で現れる例外に対処するために動的に生じるゴール「オンデマンド型サブゴール」への対処が必要であることを述べ、これに対して動的にプランを生成することによる対処を示し、その有用性をシミュレーション実験を通じて示す。さらに、ロボット制御への適用において、多様性に富む動的な世界において複数の目標を整合的に達成するロボットの実現に BDI モデルが有効なことを実験を通して示す。また、以上に述べた我々の結果を BDI logic によって形式化することについても述べ、実際にエージェントを作成する上で BDI logic における形式化が動作検証の上で効果的であることを示す。

目 次

第1章　はじめに

近年、実世界において目的達成のために自律的に振る舞うロボットの実現が重要な研究課題となってきている。特に、そうしたロボットには、周囲の環境の多様な変化に適応し、複数の目的を並行して扱う能力が要求されることが、Pfeifer らにより指摘されている [21]。

これまでのロボット制御のための技術としては、ホンダ社の ASIMO[31]、産業技術総合研究所などによる HRP-2[40] や HRP-4C[37]、大阪大の石黒らによるジェミノイド [14] などが広く知られる。

ASIMO は、[39] で実装されているような行動ベースのアーキテクチャ[1] を使用した行動計画モジュールを階層的に配置し、多数の行動遷移をもとに行動決定を行っているが、全体を統括するモジュールがなく、「歩くことによってある目的地に行く」などの動作は行えるが、複数の目標を並行して整合的に扱って行動しているわけではない。

HRP-2 は、人間との協調作業のためのロボットであるが、このロボットの制御は、音声指示装置で司令を与えることで行われている。また、HRP-4C はエンターテイメントの面での活躍、すなわちコンテンツ技術としてのヒューマノイドロボットを目的としており、制御方法はロボットのモーションの振り付けをすべてプログラマが用意する形で実装されている。いずれも、ロボットが目標を持って自律的にこれを解決することを目指したものではない。

ジェミノイドは人間の外見とほぼ同じようなロボットを作成し、人間とロボットとのコミュニケーションを図るための研究を行っているが、このロボットの制御方法も人による遠隔操作であり、ロボットの自律行動は目指していない。

このように、現在主流になっているロボットの制御方法は、二足歩行などの基本的な行動を精密に行うことはできるようになってきているが、主に遠隔操作、ないしは「踊る」というような単一の目標達成に特化したものがほとんどである。実世界のロボットが、冒頭に述べたような、動的環境下で複数の目的を扱うという要請に応えるには、むろん上述のような能力も必要であるが、そうした技術のみでは難しく、複数の目的を並行に管理、あるいはそれらを動的に生成・放棄するなどして、整合的に達成しようとする能力が要求される。特に、状態やモジュールの単純な切り替えによる制御では、並行に達成すべき目的の数が増えるほど、状態遷移の記述が困難になる。

我々は、このような実世界で複数の目標を達成するロボットの実現には、BDI モデル [23] を実装したロボット (本論文では BDI ロボットと呼ぶ) が有効であることを本論文で示す。

BDI モデルとは、信念 (B)、願望 (D)、意図 (I) と呼ばれる 3 つの心的状態とその時間的変化を用いて、Bratman の「意図の理論」[4] に基づく合理的な行動決定を行う自律エージェントをモデル化したものである。BDI モデルによるエージェントアーキテクチャを BDI アーキテクチャ[27] と呼び、BDI アーキテクチャに基づいて実装されたエージェントを BDI エージェント [27] と呼ぶ。BDI エージェントは「意図の理論」で扱われている人間の行動様式を模倣し、動的な環境下において、限られた信念・推論能力のもとで行動計画を部分的に具体化し達成することが可能である。また、意図の概念を明示的に持つことで、合理的な行動決定、一貫性のある行動、外部から見た行動の説明などを行うことも可能となる。

特に、BDI エージェントを用いると、上述した「意図」の概念を並行に保持することにより、複数の行動を切り替えながら行動をすることが可能となり、タスク切り替えのコードを人間が明示的に記述することなく、BDI エージェントの本来持つ機構によってタスクの切り替えをすることができる。これは、実世界で複数の目標を達成するためには非常に有用な性質である。

また、BDI モデルは BDI logic[23, 24] と呼ばれる様相論理体系による形式化を持ち、これによりエージェントの性質や振る舞いなどに関する形式的な議論を行うことが可能である。従って、これに基づいて作られたエージェントの動作を形式的に検証することができ、これが BDI モデルの大きな利点の 1 つである。

本論文ではまず、BDI モデルとそのエージェントアーキテクチャについて述べる。次いで、HTN プランナ [12] を我々が動的環境向けに改良したプランナ (本論文では動的プランナと呼ぶ) と BDI モデルとの結合によって、BDI モデルそのものには与えられていないエージェントの行動方針決定アルゴリズムを柔軟に提供するアルゴリズムについて述べる。また、プランナの扱うゴールの概念を拡張し、目標達成に対する障害が生じた場合に、それを解決するというゴールを動的に生成して対処する手法についても述べる。さらに、ロボット制御への適用において、多様性に富む動的な世界において複数の目標を整合的に達成するロボットの実現に BDI モデルが有効なことを示す。

本論文の構成を以下に示す。まず 2 章に BDI モデルの説明、次いでエージェント行動決定アルゴリズムを拡張する際に使用したプランニングの概念及びプランナについて 3 章で述べる。これを元に作成した動的プランナとそれによるエージェントシステムの拡張として、4 章で BDI 連携動的プランナ、5 章でプランライブラリを拡張するためのプランナとしてオンデマンド型サブゴールを用いたプランナについて述べる。また、6 章において実世界におけるロボット制御への問題点を述べ、これについて BDI ロボットを実際に用意し実証することにより、実世界上における BDI エージェントを適用することが有用であることを示す。また、それぞれの拡張ルーチンがすべて BDI logic により形式化可能であり、それをそれぞれの章の最後に記す。

第2章　BDIモデル

本章では我々が自律エージェントシステムを構成するためのモデルとして採用した、Rao らによる BDI モデル [23] について述べる。

エージェントシステムのアーキテクチャとして、古典的には単純反射エージェント、記憶を持つ反射エージェント、ゴール主導エージェント、効用主導エージェントなどが知られる [25]。前 2 者は知覚に直接反応するものであるが、ゴールを持たないため、例えば交差点において、目的地がどこであるかに応じて左右折・直進を選択するといったことができない。ゴール主導エージェントは、ゴールを達成するような動作を選択するものであるが、それのみでは、ゴールを達成するようないくつかの動作がある場合、より良い動作を選ぶには不十分である。これに対し効用主導エージェントは、ある行動によって達成される状態の効用を評価する機構を持ち、同じゴールを達成する行動のうちでも最も好ましい状態に到達するものを選ぶなど、より柔軟な行動選択ができる。

しかしこれらのいずれも、実世界で行動するために必要と考えられる、複数のゴールから現在達成すべきものを選ぶ機構や、2.1.4 で述べるコミットメント戦略など、現実世界で必要と考えられる能力を、それ自体では備えておらず、実世界で行動するロボットのためのエージェントアーキテクチャとしては能力が不足である。

人間の行動決定過程の分析として知られる Bratman の「意図の理論」[4] に基づくと、人間は周囲の環境に関する自分の信念に基づき自分の目標を定め、その目標を達成する手段を選んでそれを実行する意図を形成し、その意図にそって行動を定めている。BDI モデルは、この「意図の理論」が分析した人間の行動決定過程を模倣するエージェントモデルで、信念 (B)、願望 (D)、意図 (I) の 3 つの心的パラメータとその時間変化を元にエージェントの行動決定過程を記述するモデルである。このモデルに基づく自律エージェントのアーキテクチャを BDI アーキテクチャ[27] と呼ぶ。以降に示すように、BDI モデルには意図の概念があることによって、上述したような現実世界で必要と考えられる能力を実現している。

BDI モデルに基づくエージェント (BDI エージェント [27]) の形式化には BDI logic[23, 24] が用いられる。エージェントの合理性やコミットメント戦略などの性質は BDI logic で記述できる。

2.1　意図の理論

意図の理論とは、実世界における合理的エージェントの意図に関する哲学的な分析である。以下、実践的推論、計画立案、意図の整合性と再考慮、およびコミットメント戦略について述べる。

2.1.1　実践的推論

意図の理論によれば、われわれ人間は、非常に複雑で常に変化する実世界において、複数の目的を達成するために計画立案する合理的エージェントである。われわれは、目的を達成するためには、まずどのような様態 (目標) を達成すべきであるかを決定し、次に、その目標をどのような

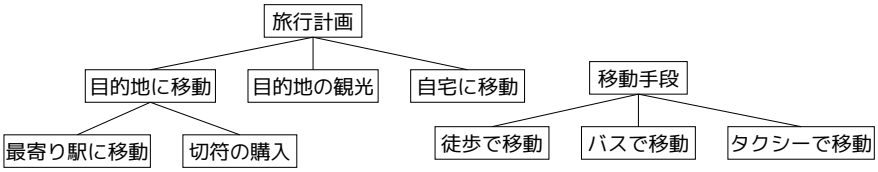

図 2.1: 旅行計画

手段を用いて達成するかを決定する。前者を熟考、後者を目的-手段推論と呼び、これら二つを合わせて実践的推論と呼ぶ。また、現在形成されている複数の意図の中から、どの意図を次に実行するか推論することも熟考と呼ぶ。

意図を OS のタスクに例えると、熟考はタスクのスケジュール管理に該当する。つまり熟考は、意図間の優先順位や整合性を維持したタスク管理であり、言い換えれば「メタプラン」を行っていることになる。

2.1.2 計画立案

われわれは、実世界では、計画立案に必要な信念 (belief) を全ては保持できない (信念の不完全性)。これら信念から未来の予測を全て計算することはできない (推論能力の有限性)。また、その都度の状況に応じて瞬時に判断を下して行為を繰り返すことで複雑な目的を達成することは難しい。このように、われわれの能力は限られたものであり、未来において行うこと全てを特定するプランを一挙に立てることは不可能である。また、われわれが住む実世界は常に変化しているため予期せぬ事態が発生し、事前に立てた計画は全く無駄に終わるかもしれない。

そこで、われわれは最初からプラン本体を全て具体的に実行できる行為列で埋めたプランではなく、環境の変化に柔軟に適応できる程度の粒度からなる副目標を用いたプランを立案した後、そのプランを意図として形成した後、実行に移す。そして、まだ埋められていない副目標を実行する時が来たと判断したとき、その状況に適応した、副目標を達成する (サブ) プランを目的-手段推論し、そのプラン本体を手段として実行する。このように状況の予測が容易でないときは、副目標の達成手段の推論を先送りすることで、多様な環境変化に柔軟に適応した意思決定を行うことができる。

たとえば、図 2.1 のように、「旅行計画」を達成するためには、「目的地に移動」、「目的地の観光」、「自宅に移動」などの達成すべき副目標を前もって定めることができる。その後、グループで「調整」し、目的地や日程などを特定して「目的地の観光」に埋め込む。次に、交通機関の時刻表を調べるなど「予備的な手続き」を行い、「目的地に移動」に移動手段などを埋めていく。そして移動当日、まだ具体的な移動手段を埋めていない「最寄り駅に移動」を、たとえば電車の予定出発時刻までの時間間隔に応じて「徒歩で移動」か「バスで移動」か「タクシーで移動」かの「移動手段」を埋めた後、最寄り駅まで移動する。このように、われわれは、そのままでは実行できない副目標からなる部分的なプランを前もって定め、調整や問題 (未来の予期せぬ事態など) に直面したときに限って、手段や予備的な手続き、または (副) 目標を特定して部分的な計画に埋め込み、意図持続の戦略であるコミットメント戦略 (後述) に基づき意図を達成しようとする。

2.1.3 意図の整合性と再考慮

合理的エージェントは整合的でない意図を形成しない。つまり、別の意図 q の達成が本来の意図 p の達成を不可能にするような場合は q を意図として形成しない。

また、以下のような場合は、意図の再考慮が行われる。

1. 計画時に想定した環境の状況と現状とに相違点がある場合。つまり、計画した時点で誤った信念を持っていたことに気がついた、または、現状が計画時の予想とは異なる場合
2. エージェントの欲求または価値意識に問題となるような変化があった場合
3. 他の意図のいくつかに問題となるような変化があった場合。つまり、意図間の整合性が取れなくなった場合

2.1.4 コミットメント戦略

次いで、意図の持続と破棄に関する「コミットメント戦略」[23] について紹介する。

(a) Blind (盲目的): エージェントは、意図はすでに達成されていると信じるまで、その意図を持続する。

(b) Single-minded (一意専心): 意図はすでに達成されていると信じるか、もしくは、その意図の達成が可能であると信じなくなるまで、その意図を持続する。

(c) Open-minded (心の広い) : その意図を形成した欲求を実現するという状況でなくなるまで、意図を持続する。つまり、その意図はすでに達成されたと信じるか、その欲求を実現する理由がなくなったので取り下げるまで、その意図を持続する。

合理的エージェントは、このようなコミットメント戦略に基づいて意図の持続や破棄を行う。たとえば、甲子園の高校野球観戦を考えてみよう。高校野球日和の炎天下では、「生ビールが飲みたい」という欲求が生じたので、「生ビールを入手する」という目的を達成するために「生ビールを入手する」という意図を形成する。Blind コミットメントでは、あらゆる手段を講じて生ビールを入手できるまでこの意図を持続する。Single-minded コミットメントでは、生ビールを入手するために「生ビールの売り子を待っていた」が来てくれなかったので、「球場内の近くの売り場に生ビールを買いに行く」が生ビールは売り切れていた。そこで、別の売り場に生ビールを買いにいく。入手できれば、この意図を破棄する。しかし、全ての売り場で売り切れていれば入手できないことを確信して、この意図を破棄する。Open-minded コミットメントでは、俄にスコールのような雨が降り出して、とてもビールを飲みたいというような状況ではなくなったので、ビールを入手するという意図を破棄する (ただしスコールが止めばこの意図は復帰するはずである)。

コミットメント戦略を用いれば、以下のように、基本行為レベルや目標レベルでの再考慮も可能となる。たとえば、基本行為が失敗したときは、その基本行為を再実行するか、またはその基本行為が実行できなければ、現在目指している目標を達成する別の手段を目的-手段推論して、その目標を持続することができる。さらに、この目標の達成手段がなくなれば、目標レベルで再考慮し、意図を達成する別の目標を熟考して、その目標を達成するよう意図を持続する。このような再考慮は意図を持たない (たとえば目標だけを持つ) エージェントで実現することは難しい。

2.1.5 BDI アーキテクチャ

BDI アーキテクチャに基づくエージェント (BDI エージェント) は、一般的に図 2.2 のようなループ (BDI インタプリタ [27]) で実装される。ここで、B は信念、D は欲求、I は意図をそれぞれ表すデータ構造である。ただしエージェントが自ら欲求を生成することは考えづらいため、BDI エージェントの欲求とは、具体的には、エージェントに外部から与えられた達成すべき「目標」と考える。

```
BDI-interpreter
initialize-state();
do
   options := option-generator(event-queue, B, D, I);
   selected-options := deliberate(options, B, D, I);
   update-intentions(selected-options, B, D, I);
   execute(I);
   get-new-external-events();
   drop-successful-attitudes(B, D, I);
   drop-impossible-attitudes(B, D, I);
until quit.
```

図 2.2: BDI インタプリタ

初期状態が与えられると、まず、event-queue を読み、エージェントに対する依頼 (目標) であれば、実践的推論を行って、(未来に実行すべき) 意図を形成する。そして現在、保持している意図の中から今の状況にて実行可能な意図の候補を options に返す。次に deliberate を用いて、次の時刻に実行すべき意図を options から熟考して一つ決定する。次に、選択された意図が持つプラン本体 (行為列) から次に実行すべき行為を決定するために、意図の更新を update-intentions にて行う。そして、決定された行為が基本行為であれば execute にて実行する。そうでなければそれは副目標であり、新たな目標 (欲求) として event-queue に追加する。次に、get-new-external-events により環境知覚器から環境に関する情報を得て、event-queue に追加する。最後に、コミットメント戦略に基づき、成功裏に目標を達成した意図を drop-successful-attitude にて破棄する。また、不整合の原因と判断された意図や、single-minded や open-minded コミットメント戦略に基づいた意図の破棄を drop-impossible-attitude にて行う。BDI エージェントは、このループを繰り返すことにより、複数の目標を達成する。

BDI アーキテクチャは大まかには図 2.3 のように、環境に関する信念と自らの目標から、実行すべき意図を生成して並行に複数保持し、選択しつつ実行するアーキテクチャと捉えられる。また、並行実行のメカニズムは、BDI インタプリタ (図 2.2) 中のループの 1 サイクル毎に、execute(I) の箇所で、図 2.4 のように、選択された意図の先頭の行為を 1 つ実行 (あるいは先頭のサブゴールを新たなゴールとするイベントを発行) することで実現されている。

2.2 BDI logic

BDI モデルの特長の 1 つは、モデル化を形式的に行う手段として BDI logic と呼ばれる様相論理体系を持つ点である。このため、エージェントの動作や性質に関する厳密な議論を行うことができ、BDI モデルが受け入れられる理由の 1 つともなっている。ここでは、BDI logic について簡単に述べる。

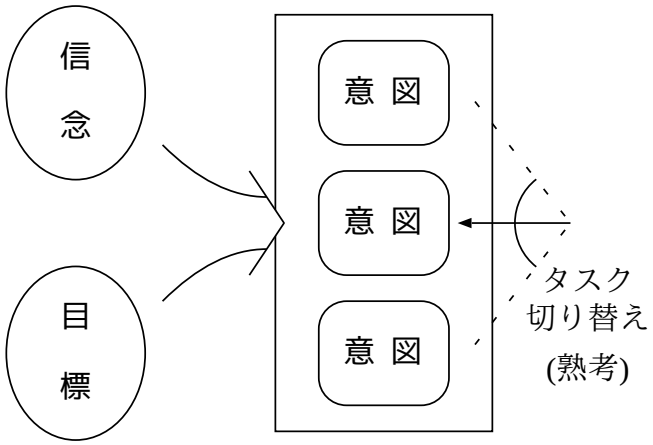

図 2.3: BDI アーキテクチャ概略図

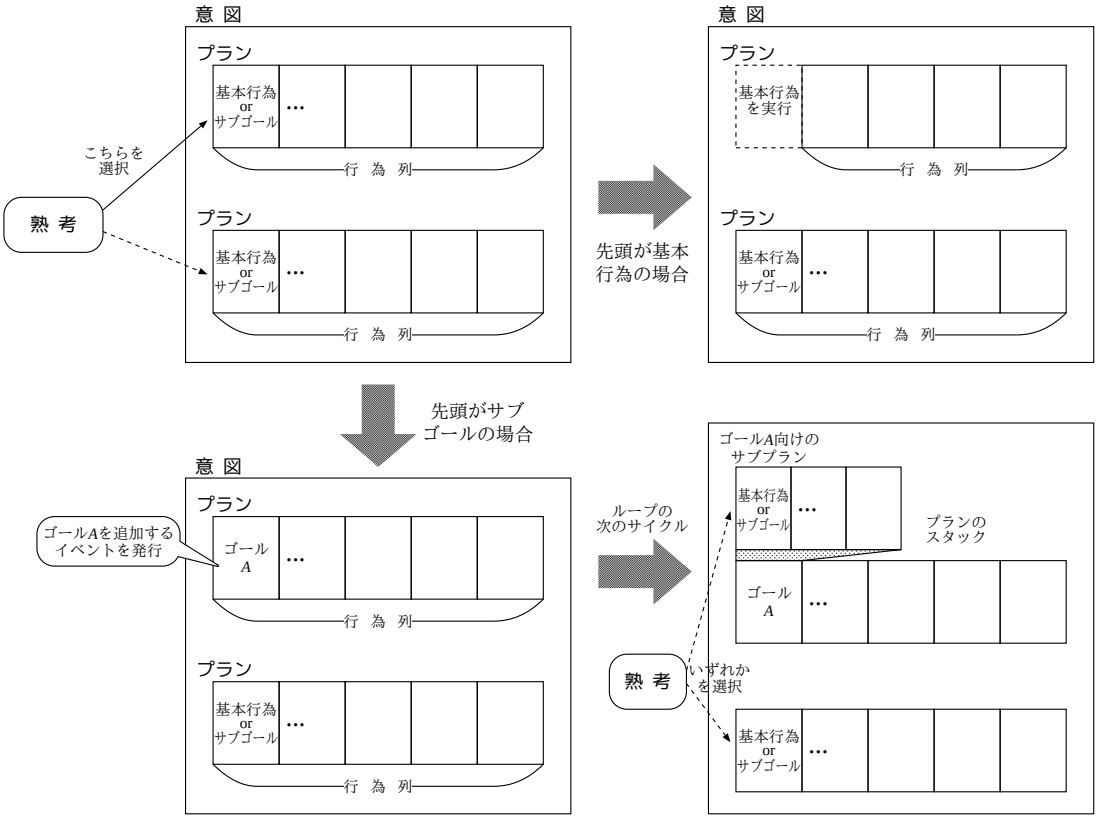

図 2.4: 並列実行のメカニズム

2.2.1 構文

BDI logic は、未来方向に分岐する時間構造を持つ (すなわち、未来の可能性が複数通りありうる) 時相論理として知られる CTL* [10] を、述語論理に拡張し、さらに信念・目標 (願望)・意図の 3 つの心的状態を表す様相オペレータを加えた論理体系である。BDI logic の論理オペレータを表 2.1 に示す。例えば $\mathsf{AX}\,\mathsf{BEL}\,p$ は「どの次の時刻にも『(現時刻で) p が成り立つ』と信じる」を表

し、BEL AX p は「『どの次の時刻にも p が成り立つ』と (現時刻で) 信じている」を表す。

表 2.1: BDI logic のオペレータ

オペレータ	直感的意味
$\wedge, \vee, \neg, \supset$	かつ、または、否定、ならば
A ϕ	全ての未来でϕ
E ϕ	ある未来でϕ
X ϕ	次の時刻にϕ
F ϕ	未来のいつかϕ
G ϕ	現在以後ずっとϕ
ϕ U ψ	ψ が最初に成り立つまで ϕ が成り立つ (until)
ϕ N ψ	ψ が最初に成り立つとき ϕ が成り立つ (atnext)
BEL ϕ	ϕ を信じる
DES ϕ	ϕ を願望する
INT ϕ	ϕ を意図する

2.2.2　意味論

BDI logic の意味論は、図 2.5 のように、多数の世界のそれぞれの中に時刻の木があるという構造で表される。

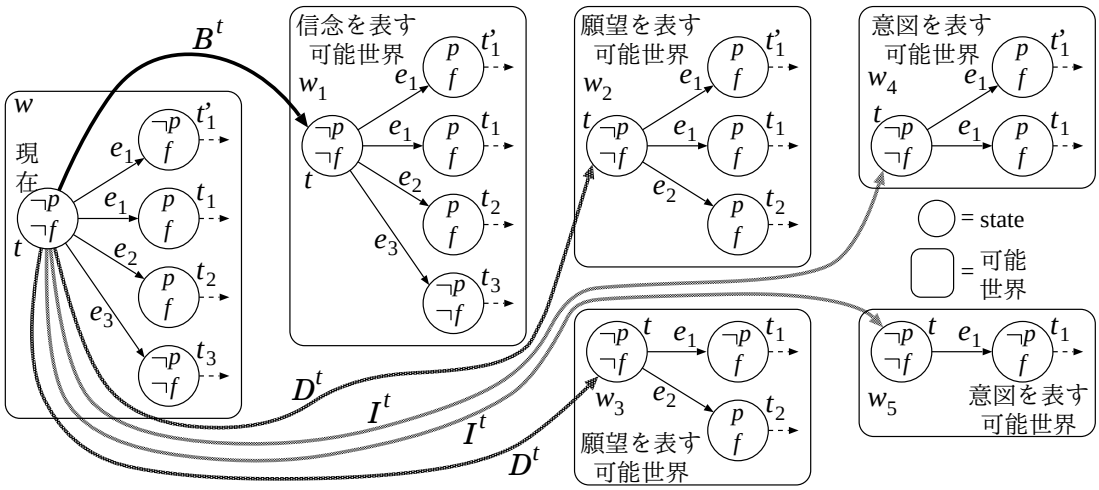

図 2.5: BDI logic の解釈に用いる世界構造

図中、ある可能世界 w のある時刻 (state) t で BEL ϕ が成り立つとは、w から関係 B^t で到達可能などの世界でも時刻 t で ϕ が成り立つことを意味する。DES, INT についても同様である。また、ある時刻 t で AX ϕ が成り立つとは、時間の流れ (図中の細い矢印で表される) において t の次の時刻のどれでも ϕ が成り立つことを意味する。他の時相オペレータについても同様に解釈する。例えば図中の世界 w の時刻 t では、AX$(p \supset f)$ や DES AX f が成り立つ。

図 2.5 は、BDI logic によって BDI エージェントの意思決定過程をモデル化した例でもある ([23] の例の一部改変)。BDI エージェントが行為を決定する過程は、自分の信念内のありえる時間の流

れ (path) から自分にとって望ましいもの (願望) を選び、さらにその中から自分の意思で選択する path を特定していく (意図) 過程と捉えることができる。図 2.5 の例では、エージェントにとって実際の世界が w、現時刻が t とし、p, f はそれぞれ「虫歯の治療のため歯が痛む」「虫歯を直してもらえる」を表すとする。また、時間の流れの矢印につくラベルはイベントを表し、イベント e_1, e_2, e_3 はそれぞれ「歯医者その 1 に行く」「歯医者その 2 に行く」「虫歯を放置してショッピングに行く」を表す。この例では、どの可能世界でも、ショッピングに行けば治療で歯が痛むことはない代わり歯は直らないが、歯医者に行けば歯は直る。また、エージェントは「歯医者で治療すれば必ず歯が痛む」と信じているが (信念の世界 w_1 ではそうなっている)、現実には歯が痛まずに歯が直る可能性もある (世界 w)。

このとき、エージェントは信念の可能世界 w_1 に存在する時間の流れのうち、歯医者に行く時間の流れだけがある世界を願望する (w_2)。すなわち、歯医者に行って歯を治療することを願望する。次いでその世界のうち、歯医者その 1 に行く時間の流れだけがある世界を意図する (w_4)。すなわち、歯医者その 1 に行くことを意図し、その結果それを実行する。この過程は、信念の世界 w_1 の subworld (一部の時間の流れだけを取り出して構成される世界) であるような願望の世界 w_2 があること、および願望の世界 w_2 の subworld であるような意図の世界 w_4 があることによって表現される。

2.2.3 形式的表現の例

BDI logic によって、BDI エージェントの様々な性質を形式的に記述できる。ここではその例を示す。

コミットメント戦略

$\mathsf{INT}\,\mathsf{AF}\,\phi \supset \mathsf{A}(\mathsf{INT}\,\mathsf{AF}\,\phi\,\mathsf{U}\,\psi)$ は、「ϕ をどの未来においてもいつか必ず達成することを意図したならば、ψ が成り立つまでその意図が継続する」ことを表し、「意図の理論」で述べられている意図の持続性を表している。ここで、ψ をどう選ぶかによって意図の持続性の強さは様々に変わるので、それによる意図の持続性の選択を「コミットメント戦略」と呼ぶ。

例えば $\psi = \mathsf{BEL}\,\phi$ の場合は「ϕ が成り立っていると信じる時まで (ϕ の達成を信じるまで) 意図が継続する」ことになり、この戦略を「blind コミットメント」と呼ぶ。この他に、$\psi = \mathsf{BEL}\,\phi \vee \neg\,\mathsf{BEL}\,\mathsf{EF}\,\phi$ の場合の「single-minded コミットメント」(ϕ の達成を信じるか、達成が将来可能だと信じなくなるまで意図を継続)、$\psi = \mathsf{BEL}\,\phi \vee \neg\,\mathsf{DES}\,\mathsf{EF}\,\phi$ の場合の「open-minded コミットメント」(ϕ の達成を信じるか、将来の達成を望まなくまで意図を継続) がよく用いられるコミットメント戦略である。

心的状態の整合性

BDI エージェントの重要な性質である、信念と意図の整合性も BDI logic で表現できる。

2.2.2 節の図 2.5 では、w_1 の時刻 t で $\mathsf{AG}(f \supset p)$ が成り立つため、その subworld である w_4 の時刻 t では、$\mathsf{EF}(f \wedge \neg p)$ を満たし得ない (もし満たすと、将来 $f \wedge \neg p$ を満たすような時間の流れが w_4 に、従って w_1 にもあることになるが、これは w_1 で「どの時間の流れでも将来ずっと $f \supset p$ である」ということに反する)。これは「(世界 w で) エージェントが $\mathsf{BEL}\,\mathsf{AG}(f \supset p)$ を成り立たせているなら、$\mathsf{INT}\,\mathsf{EF}(f \wedge \neg p)$ とはなり得ない」、すなわち「治療すれば常に痛む」という信念

に反して「治療するが痛まない」の達成を意図することはないということを意味し、「意図の理論」で述べられる「達成可能と信じられないことを意図することはない」という、信念と意図との整合性を表現している。

一方、w_1 の subworld でない意図の世界 w_5 もあることから、エージェントは $\mathsf{INT}\,\mathsf{AG}(f \supset p)$ を成り立たせてはいない。すなわち、「治療すれば常に痛む」という信念があっても、「治療すれば常に痛む」ことを意図するわけではない。このようにして、いわゆる「副次効果問題」が回避される。

第3章　プランナ

プランナとは、初期状態から最終状態でのゴールが与えられた際に、基本行為を組み合わせて初期状態からゴールへの行動計画を自動的に行うプログラムのことを言う。我々がエージェントに導入した動的プランナは、Hierarchical Task Network(HTN) プランナ [13] を改良したものである。ここでは HTN プランニング (階層的プランニング) と HTN プランナについて説明する。

3.1　HTN プランニング

HTN プランニングとは、初期状態で達成すべきタスクが与えられると、タスクをいくつかのさらに小さなサブタスクに分解し、これを繰り返すことにより、与えられたタスクを基本行為に分解し、初期状態から与えられたタスクを達成するための行動計画を生成する問題である。

プランニングを行う際に与える問題定義をドメインと呼び、ドメインには以下のものが記述される。

1. 基本行為 ··· これには次状態へ追加される状態と削除される状態が記述されている
2. メソッド ··· 与えられた問題の解決方法 (基本行為またはサブタスクに分解することによる問題解決方法)

HTN プランニング問題を解決するプランナを HTN プランナと呼ぶ。HTN プランナにおいて、ドメインを用いて問題のための状態の定義を行った上で、問題解決ルーチンを実行すると、ドメインのメソッドを用いてタスクを分解し、プランを生成することが可能となる。

3.2　SHOP2

本論文で使用しているプランナは Nau らによる SHOP2[19] と呼ばれる HTN プランナであり、基本的なアルゴリズムは上に述べた通りになっている。SHOP2 とは Simple Hierarchical Ordered Planner の略で、version2 版である。SHOP2 は静的な問題を取り扱い、初期状態からゴールまでのプランを全てメソッドで分解することにより得ることができる。SHOP2 は Common Lisp で実装されており、後に紹介する我々の動的プランナも Common Lisp を用いて拡張をおこなっている。

ドメインは `defdomain` と呼ばれるマクロで定義することができ、問題解決を行うために `defproblem` マクロによる状態の定義づけと、問題解決ルーチンを実行する `find-plans` 関数の実行により、プランを生成することが可能である。

例として、図 3.1 ではドメイン定義と問題解決における状態定義の部分と問題解決ルーチン実行部分を挙げる。ドメイン部分には、`:operator` により基本行為を記述することができ、`:method` によりメソッドを記述することが可能である。基本的な記述方式は以下の通りである。

(`get-operatorpart` *name def*)

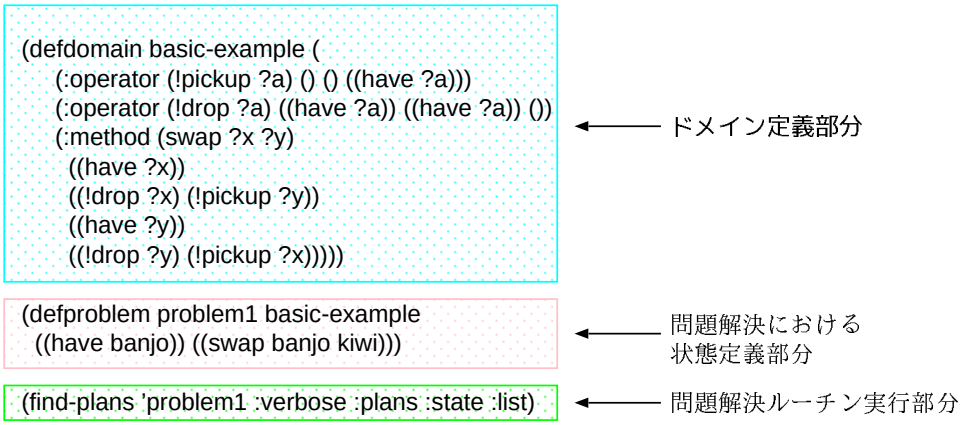

図 3.1: SHOP2 のドメインと問題解決ルーチン実行のための記述部分 (例)

で、*name* という問題のドメインが *def* で定義されることを示す。

`(:operator` *act*`)`

は、*act* という基本行為が使えること、

`(:method` *subgoal cond body cond body* ⋯`)`

は、副目標 *subgoal* が (条件 *cond* を満たしていれば) 行為の列 *body* に分解できることを表す。

例では、`:method` の中では `swap` という問題の解決のために、`(have ?x)` ならば `(!drop ?x)(!pickup ?y)` を実行、そうでなければ `(!drop ?y)(!pickup ?x)` を実行することになっている。このとき、`(!drop ?x)` は基本行為として `(:operator (!drop ?a)⋯)` と記述されており、これに従って基本行為が生成される。さらに、`defproblem` のところで、状態が `(have banjo)` であることを明記して現在の状態を渡し、`(swap banjo kiwi)` という問題を解く設定にしており、最終的に `find-plans` 関数により問題解決ルーチンを実行している。

これの実行結果は以下の通りになる。(図 3.2)

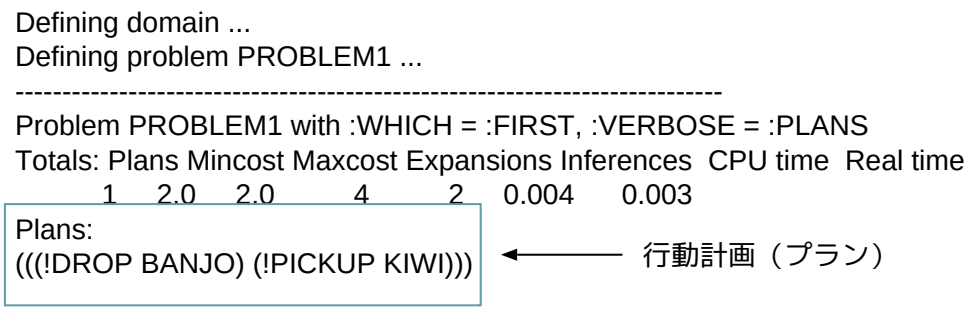

図 3.2: SHOP2 による問題解決のためのプラン生成の例

第4章　熟考ルーチンとしての動的プランナ

本節ではプランニングシステムによる BDI エージェントにおける熟考ルーチンの構築をおこなう。

今回作成した動的プランナ (図 4.1) は、3.2 節で述べた SHOP2 と呼ばれる固定環境向けの優れたプランニングを行うことができるプランナを動的環境向けに改造したものである。

このようなプランナをエージェントに取り付けることにより、動的環境でロボットの行動方針を柔軟に切り替えることができる。

4.1　BDI 連携動的プランナ

本節では、我々が提案する、BDI エージェントとの連携向けの動的プランナ (以後「BDI 連携動的プランナ」と呼ぶ) について述べる。

BDI 連携動的プランナは、先に述べた通り階層的プランナとして知られる SHOP2[19] を改造したものであり、SHOP2 のエンジンを使用している。

SHOP2 は静的な問題向けのプランナであるため、達成すべき目標を与えられると、それを基本行為にまですべて分解し、目標達成のための一連の行為列を作る。しかし、動的環境においては、一度立てたプランが状況の変化などにより続行不能となることもあるため、基本行為にまで分解することは現実的でない。

人間は、動的環境においては、まず部分的で大まかなプランを立て、必要に応じてそのプランの各部分をより具体化しつつ実行する。動的環境向けのプランナにも、そのように一度大まかな粒度にタスクを分解することが求められる。すなわち、プランナは一段階目の分解でプランの分解を中断して、それをサブゴールの列として蓄積しておき、エージェント側にはその先頭のサブゴールを目標とさせ、それを分解してタスクを生成し、そのタスクをエージェントに与えて目標達成への行動方針とする。もし、サブゴールに基づくタスクを方針としてそのサブゴールを達成できた場合には、次のサブゴールを目標とし、エージェントの目標もそちらへ切り替え、次の方針を与える。しかしながら、与えた方針でエージェントが目標を達成できなかった場合には、その時点の状態を元に再プランニング (その時点でのサブゴールを再び分解し基本行為を作成する) を行い、新たな方針を立ててそれをエージェントに与え直す。BDI 連携動的プランナは以上のような設計になっている。

4.2　BDI 連携動的プランナの実装方法

BDI 連携動的プランナの実装の一部を図 4.2 に示す。ここでは、3.2 節で用いているドメイン記述オペレータ `get-operatorpart` の定義の概要を示す[1]。

[1]get-operatorpart はマクロとして定義されているが、内部で所定の処理を行った上で `t` を返しており、本来の意味の Lisp マクロではない。本来は関数として定義すべきものであるが、マクロとした理由は、SHOP2 本来の問題定義オペレータの書式と揃えるため、第 1 引数や第 2 引数にドメイン名やドメイン定義内容をクォートされない形で書けるようにしたためである (関数にすると引数が評価されてしまう)。

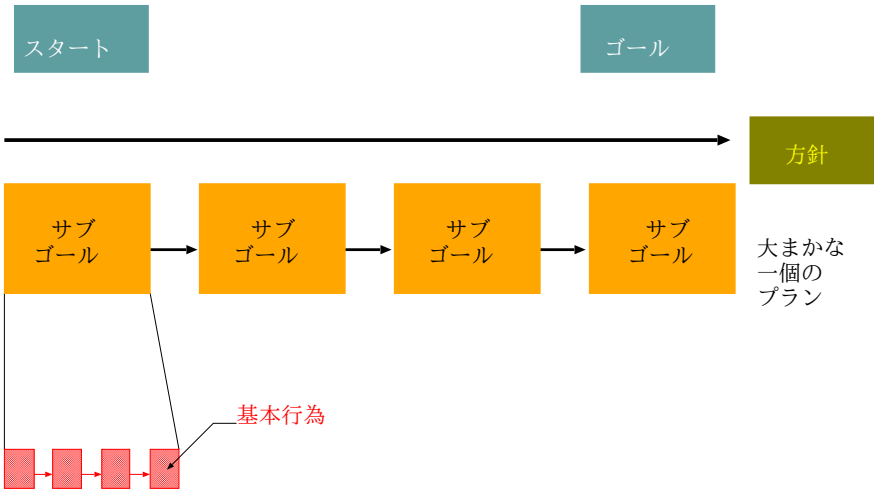

図 4.1: プランナの模型図

プランナは、このオペレータによりドメイン記述を与えられると、その中のメソッド (サブゴール) 定義を加工し、それを基本行為と見なすような定義を作り出した上で、SHOP2 の問題定義ルーチンに引き渡す。これにより、プランナは目標を与えられると、サブゴールを完全に基本行為にまで分解せずに、第 1 段階目の分解で中断してサブゴールの列を返すことができる。

この他に、分解したサブゴール列を保存しておくコードなどがあるが割愛する。

図 4.2 におけるマクロ `get-operatorpart` では `name` という名前のドメインの定義 `def` を引数として受け取り、ループ内で `def` を一行ずつ `part-subplan` に格納し、その部分の中に`:method` があればその行はメソッドの記述として判断しても良いので、`!`を先頭に付加して基本行為扱いに変更して`:operator` を作成し、メソッドの記述もその作成された基本行為に直接分解するように書き換えてこれをすべて `method1` に格納する。また、`part-subplan` の中が`:operator` であればそのまま `method1` に格納する。これによって、一段階目で停止することが可能であるドメインを作成することが可能となる。

そして作成したドメインの名前に `subdomain` の属性をつけて、ドメインの名前に対して属性 `subdomein` で必要な情報を取り出せるようにしている。元のドメインには `domain` の属性をつけている。

4.3 Jason

BDI エージェントの実装において、BDI モデルに基づくエージェントアーキテクチャである BDI アーキテクチャ[27] の実装プラットフォーム、Jason[3] を用いた。

本節では Jason について述べ、次節以降で BDI エージェントとのプランナとの連携における具体的な実装について述べる。

4.3.1 Jason インタプリタ

Jason[3] とは、BDI アーキテクチャに基づくエージェント記述言語 AgentSpeak[22](の拡張) の処理系であり、2.1.5 節に述べた BDI インタプリタの実現である。Jason では、エージェントのプランや信念は Prolog に似た文法を持つルールの形で宣言的に記述し、環境モデルの定義は Java 言

```
; nameというドメインの定義defを与える
(defmacro get-operatorpart (name def)
  (let ((method1 ())
        (method2 def))
   (loop
     (if (null def) (return))
     ; リストdefの各要素part-subplanを見ていく
     (let ((part-subplan (car def)))
     ; それがメソッドの記述であれば
       (if (eq (car part-subplan) ':method)
           (progn
             (let* ((atommethod
                            ; シンボル名に「!」を付加して
                            ; 基本行為扱いに
                     (subst (symbol-add-exclamation (car (nth 1 part-subplan)))
                            (car (nth 1 part-subplan))
                            (nth 1 part-subplan))))
               ; 基本行為扱いに変更した定義をmethod1に追加
               (setq method1
                     (cons `(:method ,(nth 1 part-subplan) () ,atommethod)
                           method1))
               (setq method1 (cons `(:operator ,atommethod () () ())
                                   method1))
               ))
         (setq method1 (cons  part-subplan method1)))
       (setq def (cdr def))))
    ; 引数defのままのドメイン定義をdomainの属性として与える
   (setf (get name 'domain) `(defdomain ,name ,method2))
    ; 引数defに上記の加工をしたドメイン定義をsubdomainの
    ; 属性として与える
   (setf (get name 'subdomain)`(defdomain ,name ,method1))
   t
   ))
```

図 4.2: プランナの実装の一部

語で行う。後者にはエージェントと環境の相互作用に関する記述、特にエージェントの環境知覚や基本行為の定義も含まれる (図 4.3)。定義された環境知覚や基本行為はプラン内で用いることができる。

プラン記述は基本的に

目標
: プランを適用するための前提条件
<- プラン本体 (基本行為または副目標の列)

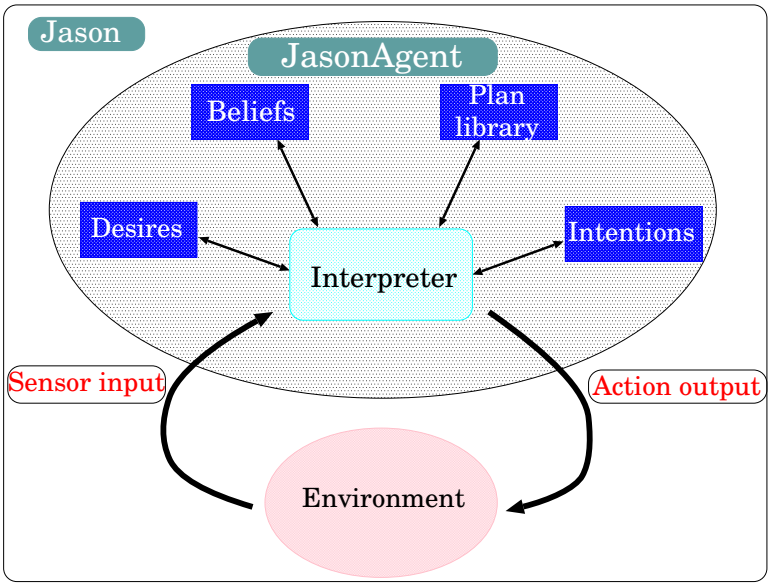

図 4.3: Jason と環境との相互作用

という形をとる。プラン頭部の「目標」は実際には、その目標を生じさせるイベントの発生として記述される。Jason インタプリタは、イベントが発生していれば、それに合致する頭部を持つプランの中から 1 つ選んで (「目的-手段推論」)、エージェントの意図の集合 (図 2.2 の options) に追加する。次いで現在の意図のうち 1 つを選んで (「熟考」、同図の deliberate)、その本体の行為 1 つ (基本行為または副目標) を実行する。これを繰り返す。

4.3.2　エージェントとの連携

Jason では環境の記述を Java で行えるため、環境プログラムが起動する際に、Java から BDI 連携動的プランナのサブプロセスを起動している。すなわち、プランナは Jason とは独立したプロセスであり、Jason のエージェントプログラムが終了するまでは生存する。両者はプロセス間通信により連携を行っており、Jason 側から情報を送ってプランナにプランニングを行わせ、生成したプランを Jason 側で受け取ってエージェントの目標達成のための方針を決めることができる。

以下に、プランナとの結合を実現している Java コードの概略を示す。まず、エージェントは基本行為 `ask_planner` によりプランナに情報を伝える。プランナはこれを元に解の探索を行って出力し、Jason 側で受け取る。getPlan() の部分はプランナから解が来るまで自動的に待つので、ユーザがプランナと同期を取る必要はない。

4.4　ロボットを用いた例題

本節では、例題による実験によって、我々の提案手法が実世界ロボットの制御に有効であることを示す。

問題は、ロボットがグリッドワールドにおいて、壁の位置を知覚して信念に加えつつ、オブジェクト (以下「宝」と称する) を探し、探索後元の位置に戻るというものである。ただし、ロボットはセンサの誤差による知覚の誤りのため、壁の位置に関する誤った信念を持つことがある。

```
…
} else
// Jason側で基本行為ask_plannerが実行
// されたら
if(action.getFunctor().
          equals("ask_planner")){
    // ask_plannerの引数を取り出す
    input = action.getTerm(0).toString();
    // プランナに入力を送る
    if(input.equals("null")){
    // 引数がnull(初期状態)のとき
        planner.send("(())");
    } else {
    // 引数がロボットの状態を表す
        planner.send("(("+input+"))")
    }
    // プランナが出力した解を受け取る
    output = planner.getPlan();
}
…
```

図 4.4: Jason とプランナのインタラクション部分

ロボットは普段は、(知覚にはコストがかかるため) 一旦得た壁の位置に関する信念を信頼して探索を続ける方針をとる。しかしその場合、信念の誤りのため目的地に到達できないことが起きる。

このとき、プランナは壁の位置を再知覚しつつ探索する方針を提示し、その方針によって新たに選択したプランでロボットは探索を続行することができる。その結果、未探査の場所を新たに見つければ、ロボットは目標達成不能状態を脱したと判断し、元の方針に戻しての探索を続ける。

4.4.1 本研究で使用したロボットと知的制御のための分散処理方法

本節では、エージェントからのロボットの制御方法について述べる。

今回使用したロボットは LEGO 社の MINDSTORMS NXT[11] で、前輪 2 つと後輪 1 つのタイヤによる移動ロボットになっている。また、NXT の正面に超音波センサがあって、障害物までの距離判定を行うことができ、壁の認識のための知覚情報はこれをもとに得る。NXT には Python による既存の制御ライブラリが公開されており、我々はこれを利用してロボット制御プログラムを作成した。

エージェント側には、ロボットの動作にかかわる基本行為として「1 マス前進」、「右 90 度回転」、「左 90 度回転」、「壁を知覚」などがある。しかし、NXT のレベルでは、モータの回転数を指定する、センサが障害物までの距離を表す整数の信号を返す、などの低レベルの命令しかない。

そこで我々は、エージェント側の基本行為や知覚を Python のメソッドとして実現した。各基本行為 (ロボット側にとっては高レベル命令) に対応するメソッドが、これをロボット側の低レベル命令に変換してロボットを制御し、ロボット側のセンサからの信号は壁の知覚の有無に変換されてエージェント側に返される。Python プログラムと NXT は Bluetooth で通信を行う。

以上の設計を図 4.5 に示す (図には Jason エージェントと BDI 連携動的プランナの結合部も含んでいる)。

このように、エージェント制御では高レベル命令を扱い、ロボット制御には低レベル命令を扱うことにより、エージェント側ではロボットがどのような言語・ライブラリおよび通信手段によって制御されているかを気にすることなく、行動レベルでの制御のみを行えばよい利点が生まれる。また Python プログラム側も、行動制御の精度の改善などを上位とは独立に行える。例えば、本論文では扱わないが、ロボット制御にはモータ出力の不正確さなどに起因する行動の誤差が伴い、これに関する改善をこの部分で行うことができれば、本論文の提案の有効性も高まる。

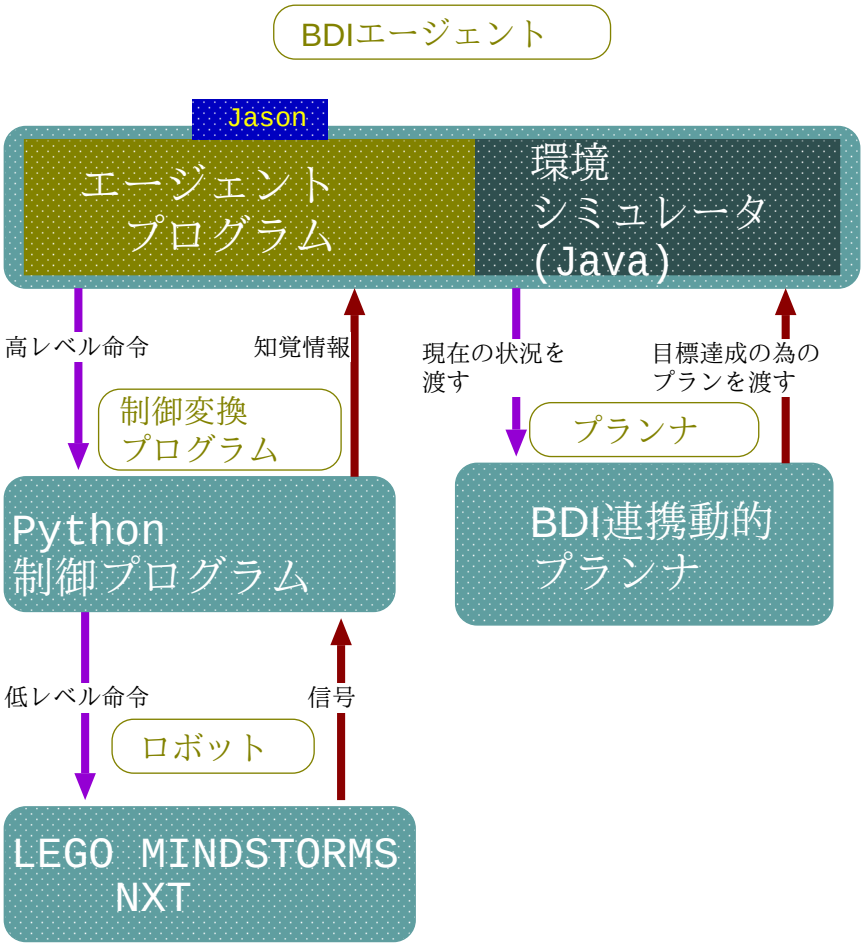

図 4.5: 設計の全体図

4.4.2　エージェントプログラムの流れ

エージェントは「宝を見つけて元の位置に戻る」という目標を持ち、最初の副目標「宝を見つける」のために行動方針をプランナに生成させる。プランナは「通常探索」という方針をエージェントに渡し、これを受けエージェントは「通常の方法で探索する」というプランを選んでこれを意図とし、実行する。このプランは、知覚で得た壁の情報を信念に加えつつ探索を行う。探索済みでなくかつそこへ動ける場所がなくなればこのプランは失敗するが、その場合、プランナに問

い合わせ、プランナは「再探索」の方針を与え、その結果エージェントは「誤認識があった可能性を考慮して探索する」というプランを選んで意図とし実行する。このプランは、既に知覚で得ている壁の情報を疑い、再度知覚しつつ探索を行う。その結果、探索済みでない場所を見つければ、誤認識による目標達成不能状態は脱したので、再度プランナに問い合わせ、「通常探索」の方針をプランナから得たエージェントは「通常の方法で探索する」というプランを選び直して意図とする。

この後、宝を見つけたら次の副目標「元の位置に戻る」が実行されるが、この部分については略する。

4.4.3 プラン

探索プランは概ね以下のような構成になっている。

- x 軸正負、y 軸正負の 4 方向に向かって壁の知覚を行う
- 進める方向を 1 つ選び、動いた経路を記録しながら進む
- 動ける場所がすでにすべて探索済である場合 (あるいは目標達成不能状態を脱した場合)、プランナに問い合わせて、行動方針の決定を行う
- プランナから返された方針 (通常探索/再探索) に基づき、探索プランを新たに選び直す

以下に各プランのコードの概要を示す。コードは Jason のプラン記述部の文法によっており、Prolog の文法に近い。1 つのプランは

トリガ : 前提条件 <- 本体

の形で書かれる (前提条件や本体は省略可)。述語名の先頭に付く記号は表 4.1 のような意味を持つ (Achievement goal とは達成を要する目標、Test goal とは検査を行うだけの目標を指す)。

表 4.1: Jason のプラン記述での記号の意味

記号	意味
+	信念の追加
-	信念の削除
+!	Achievement goal の追加
-!	Achievement goal の失敗
+?	Test goal の追加
-?	Test goal の失敗
!	目標 (プランの本体に現れれば副目標)
?	信念の参照
何もなし	基本行為

1. 通常の方法で探索するプラン

   ここでは探索プランそのものの他、関係するサブプランも示す。

   - 初期願望
     起点 $(0, 0)$ からの探索をするという願望 (目標) を与えておく。

```
!how_to_search(0,0).
```

- 行動方針を決める

  プランナに問い合わせるのみである。ask_planner の実装は 4.3.2 節に示す。

```
+!how_to_search(X,Y) <-
    // プランナに問い合わせる
    ask_planner(null).
```

- 行動の方針を変える

  プランナから与えられた方針としての信念 go_search を得て、行動の方針を変更する。

```
+go_search <-
    !search(X,Y).
```

- 探索用プラン

```
+!search(X,Y) <-
    // まず回り4方向の知覚をしておく
    !plusX(X,Y);
    !plusY(X,Y);
    !minusX(X,Y);
    !minusY(X,Y);
    // 進める方向へ進む
    !action(X,Y).
```

- 知覚を行うプラン

  ここでは x 軸正方向の知覚を行うプランのみ示し、他の 3 方向のものは略する。

```
+!plusX(X,Y)
  // その方向に壁があるという
  // 信念がなければ
  : not wall(X,Y,X+1,Y) <-
    // 現在の方向Dから0度方向へ向く
    rotate(D,0);
    // 知覚を行う(得られた知覚に従って
    // 壁の信念も追加される)
    percept(X,Y,X+1,Y);
    // 進める場所の記録を行う
    !can_go(X+1,Y).
```

- 進める場所の記録を行うプラン

  経路に含まれていない場所であれば進むことが可能である場所として記録する。

```
+!can_go(X,Y)
  // その場所が既知の経路に含まれなければ
  : not path(X,Y,_) <-
    // 進める座標として記録
    +can_go(X,Y).
```

- 進める方向へ動くプラン 1

  動ける方向を探して動き、経路を記録したあと、search ゴールを再帰的に呼んで探索を続行する。4 方向に対するものが 1 つずつあるが、ここでは x 軸正方向へ動くもののみ示す。

```
+!action(X,Y)
  // その方向に壁があるという信念がないか
  // 再確認(知覚で壁の信念が追加された
  // 可能性があるため)
  : not wall(X,Y,X+1,Y) <-
    // 指定された方向へ回転し進む
    rotate(Z,0);
    go;
    // 動いた経路を記録する
    +path(X+1,Y,N+1);
    // 再び探索をする.
    !search(X+1,Y).
```

- 進める方向へ動くプラン 2

  進める方向へ動くプラン 1 がいずれも適用できない場合、後戻りして別な経路を試す余地があれば、後戻りする。

```
+!action(X,Y) <-
    // 進める場所として記録されている
    // 場所がまだあるかどうか判定
    ?test_can_go;
    // そうであれば後戻りして、
    backtrack(X,Y,X1,Y1,D);
    // 経路からそのマスを除外
    -path(X,Y,N);
    // failのタグを付けて記録
    +path(X,Y,fail);
    // 探索し直しをする
    !search(X1,Y1).
```

- 進める場所として記録としてされているかどうかのチェック

  進める場所として記録されている場所があれば真となる。

```
+?test_can_go : can_go(_,_).
```

- 探索すべき場所がなくなったことの検知と対処

  すでにすべてのマスに移動して他に目標達成のための経路が存在しない場合、+?test_can_go が失敗することによってそのことが検知され、このプランを使う。プランナを呼び出し、失敗の対処を行う。

```
-?test_can_go <-
    ask_planner(failure_search).
```

2. 誤認識を考慮して探索するプラン

   ここでも、探索プランそのものの他、関係するサブプランも示す。1 節の場合との主な違いは再知覚用プランにある。1 節の知覚プランは、壁に関する信念が既にある場合は知覚を行わないが、本節のものは常に知覚を行う。

   - 行動の方針を変える

     プランナから与えられた方針としての信念 go_research を得て、行動の方針を変更する。

     ```
     +go_research <-
         !research(X,Y).
     ```

   - 再知覚を行いながら探索

     すでにある信念は正しいと思わずに、再知覚を行いながら行動する。

     ```
     +!research(X,Y) <-
         !plusX_again(X,Y);
         !plusY_again(X,Y);
         !minusX_again(X,Y);
         !minusY_again(X,Y);
         !action(X,Y).
     ```

   - 再知覚用プラン

     知覚を行い、壁が本当にあるかどうか判定する。1 節と同様、4 方向分あるもののうち 1 つのみ示す。

     ```
     +!plusX_again(X,Y) <-
         percept(X,Y,X+1,Y);
         ?not_deletewall(X,Y,X+1,Y).
     ```

   - 壁があるかどうかの判定

     壁は存在しており、最初の知覚に問題がない場合、何もしない。

     ```
     +?not_deletewall(X,Y,X1,Y1)
       : not deletewall.
     ```

     知覚情報に問題があったが、すでに経路にその道が含まれている場合も、問題なしとして特に信念の修正は行っていない。

     ```
     -?not_deletewall(X,Y,X1,Y1)
       : path(X1,Y1,_).
     ```

     知覚情報に問題があり、経路にも含まれておらず、壁と誤認したことがわかった場合、再行動決定を行う。

     ```
     -?not_deletewall(X,Y,X1,Y1)
       : not path(X1,Y1,_) <-
         // プランナに現状を伝えて
         // 方針を決め直す
         ask_planner(found_new_route).
     ```

4.4.4 プランナ側の記述

ここではプランナに与えるこの問題におけるドメイン (問題定義) ファイルの概略 (図 4.6) を示す。

```
(get-operatorpart move-agent (
 ;エージェントに与える行動方針
 (:operator (!go_search))
 (:operator (!go_research))
 ;行動方針を決めるための解決法
 (:method (search object)
   ;探索失敗かつ宝を
   ;発見できなかった場合
   (and (failure_search)
        (not (find_object)))
   (!go_research)
   ;再探索の後新しいルートを発見、
   ;若しくは探索を開始し、まだ
   ;宝を発見していない場合
   (or (found_new_route)
       (not (find_object)))
   (!go_search)
 )
))
(get-operatorpart back-track (
 ;エージェントに与える行動方針
 (:operator (!go_back_track))
 (:operator (!go_back_track_again))
 ;行動方針を決めるための解決法
 (:method (back_track robot)
   ;宝を発見した場合
   (find_object)
   (!go_back_track)
   ;最初の地点に戻れない場合
   (not (reached_start))
   (!go_back_track_again)
 )
))
```

図 4.6: 今回の問題におけるプランナのドメインファイル

ドメインは 2 つ存在し、エージェントが探索を行う際に行動方針を決めるためのドメインと、エージェントが探索を終えて元の位置に戻るためのドメインが存在する。

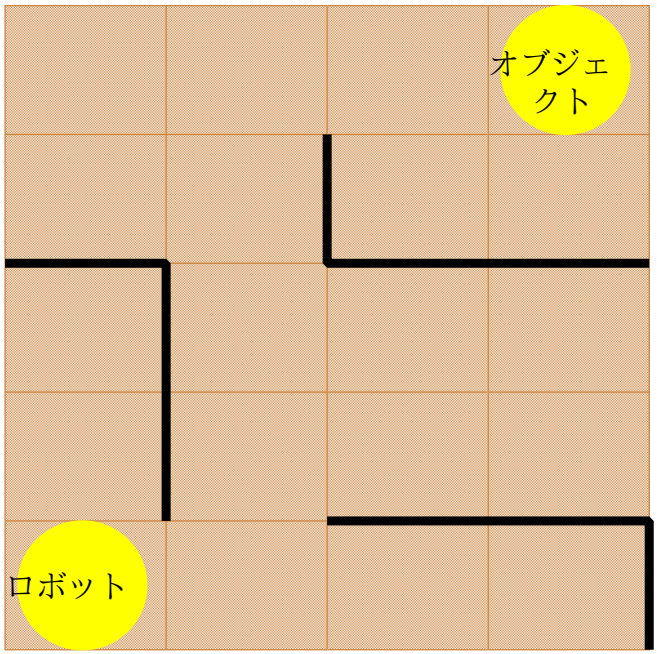

図 4.7: 実験でのマップ

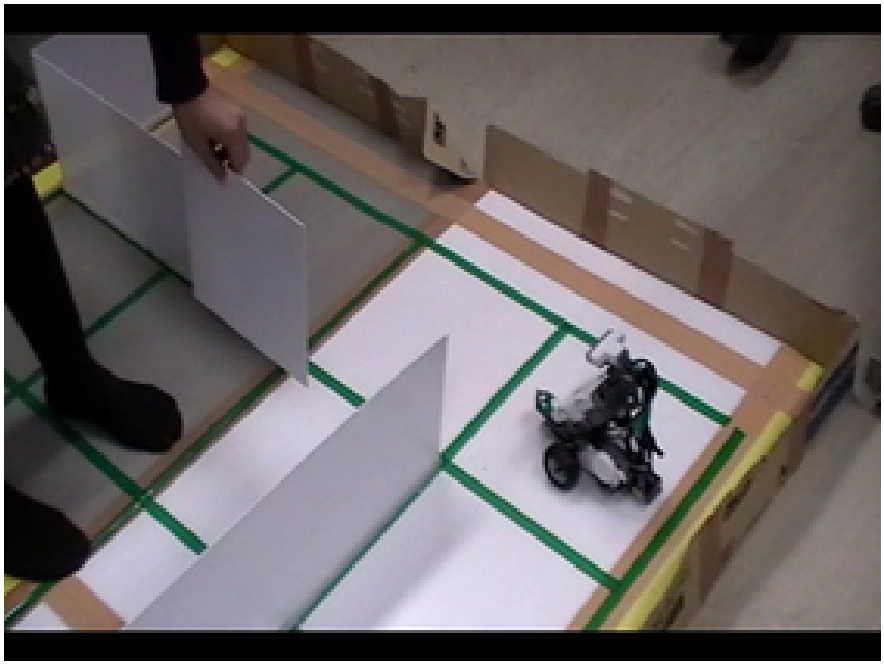

図 4.8: 実験の様子

4.5　実験と結果

4.5.1　実験

実験は図 4.7 のような 4×5 の 2 次元グリッド空間でおこなった。実験の様子は図 4.8 のようになっている。

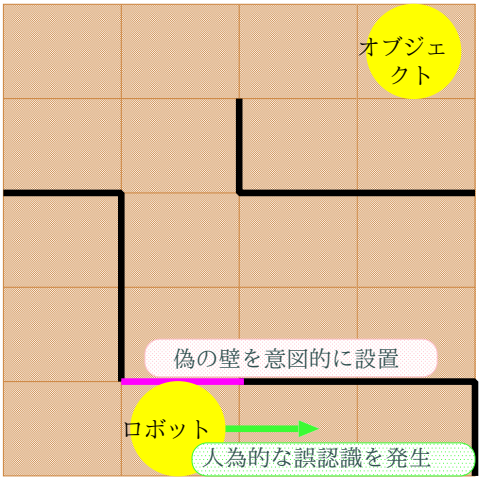

図 4.9: 壁を意図的に誤認させた様子

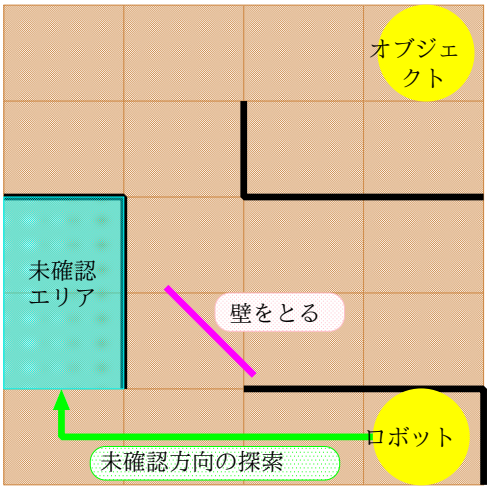

図 4.10: 誤った信念を元にそのまま探索を続行する様子

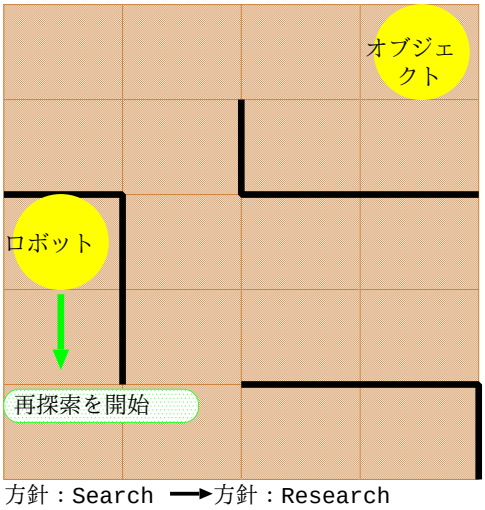

図 4.11: すべて探索を行い八方塞がりになりプランナに問い合わせ、行動方針を切り替える様子

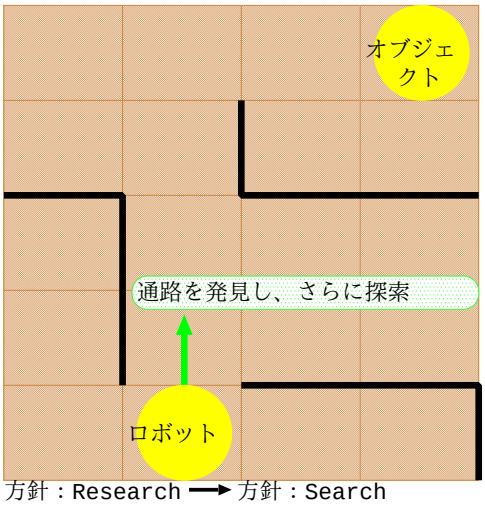

図 4.12: 誤認した場所を発見し、再びプランナに問い合わせ探索方針を切り替え再び探索をする様子

エージェントは最初、グリッド空間で表されたマップの枠しか知識がなく、ロボットはマップのマス目単位で行動して前方の壁の知覚を行うことにより、エージェントの信念ベースに壁の信念が追加されていく。

今回の実験ではセンサが認識を誤った状況を模擬再現するため、4.5.2 節に示すように、人為的に誤認識を発生させることによって、意図的にロボットが目標達成をできない状態にし、この状態で先述のメカニズムがどのように働くかを検証した。

4.5.2　結果

実験の結果としては、図 4.9, 図 4.10, 図 4.11, 図 4.12 のようになった。

ロボットは図 4.7 の迷路を探索していきオブジェクトの探索を始めた。

最初の探索方法は、壁を確認して壁が存在すればエージェントの信念ベースに追加、なければ信念ベースには何も追加されず、その知識を元に探索を進めていく search という探索方針で探索を進めた。search では壁が認知されるとその壁が存在する方向について、反対方向から知覚を行わない。つまり、信念に壁が加わると、その壁にまつわる方向への壁の認識を行わないのですべての方向の探索は行われない。これにより、全探索をしなくて済み、探索の効率がよくなる。

そして探索をする過程において、実験者が通路の一部に壁を意図的に設置し、誤認識を人為的に行わせた。(図 4.9)

この結果、ロボットは間違った信念を持ったまま探索を進めて (図 4.10)、結果的に探索できる場所が存在しないことを認識し、この結果を元にプランナに問い合わせ、他の行動方針である、research という自身の信念を疑いながら探索を行うという方針に変更した。(図 4.11) research は search とは違って、すべての方向に対して知覚を行う、全探索方針である。

これにより、もう一度すべての壁を探索し始め、現在の信念と照らし合わせながら行動を行い、間違いを発見して新たな通路を発見すると、その場所の信念を訂正し、その方向へ移動した。このとき、さらにプランナにこの事実を伝えることにより、プランナはエージェントが現在正しい信念を元に行動をすることができると踏まえて、search の方針に切り替えるタスクを行動方針として与えた。(図 4.12)

そしてその先の未開拓なエリアをさらに探索するという結果となった。

また二箇所通路を塞ぐ実験も行ったが、このときも探索の方針がプランナに問い合わせることにより変更され、全ての方向を探索する research の方針に従って行動を始め、塞がれていない通路を見つけ出すことにより、目標達成不能状態になることを免れることができた。

これらの知見から誤認識における問題を解決することができたと言える。

4.5.3　実行時の出力

実験実行時のプランナの出力を図 4.13, 図 4.14, 図 4.15, 図 4.16 に示す。

図 4.13 はエージェントプログラム実行開始と共にプランナが実行されサブゴール列が出力される時を表す。4.5.2 節で示した図 4.11 におけるプランナの実行出力が図 4.14 に相当し、4.5.2 節、図 4.12 におけるプランナの実行出力が図 4.15 にあたる。最終的に宝を発見し、それによって最初のサブゴールが達成され、元に戻るための方針を示され、初期位置に戻ったという情報を渡されプランナのプロセスが終了したのが図 4.16 である。

4.6　考察

これまでに述べたように、我々の手法を実装したロボットは、「宝を見つけて元の場所に戻る」というトップレベルの目標を保持しつつ、状況に応じて行動方針を切り替えることにより複数のプランを使い分け、意図の破棄と再選択を繰り返しつつ、目標を達成することができた。

また、行動方針をプランナに決定させ、それによってプランを選ぶことにより、プランナが熟考の働きを担っている。1 節で述べたように、BDI エージェントを用いると行動切り替えのコードは記述しなくて済むが、熟考アルゴリズムは記述する必要があった。我々の手法では、この部分

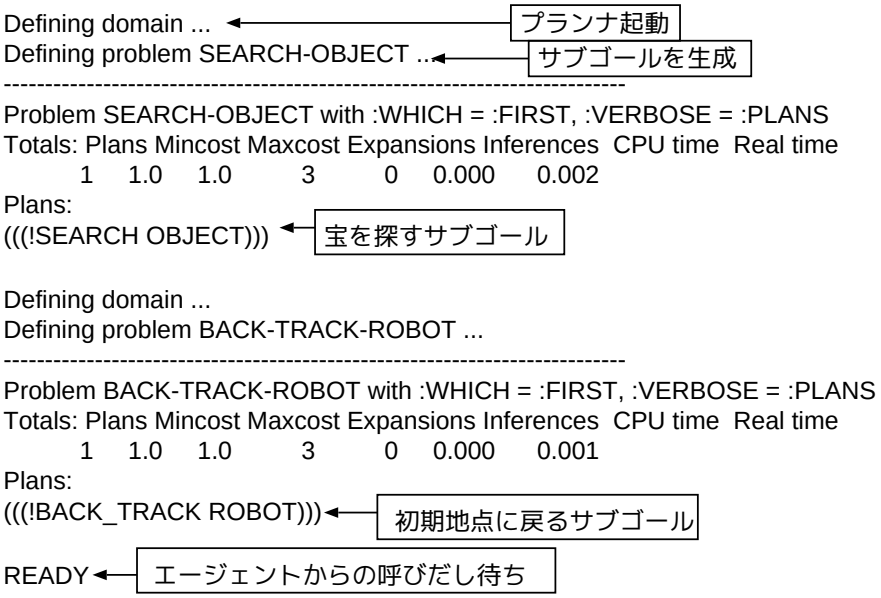

図 4.13: 起動時のエージェントの行動に対応したサブゴール列生成過程

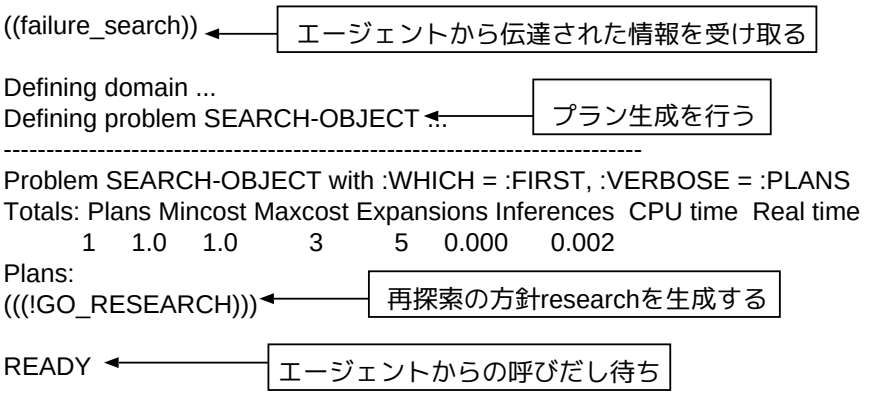

図 4.14: 探索に失敗したときのプランナの処理

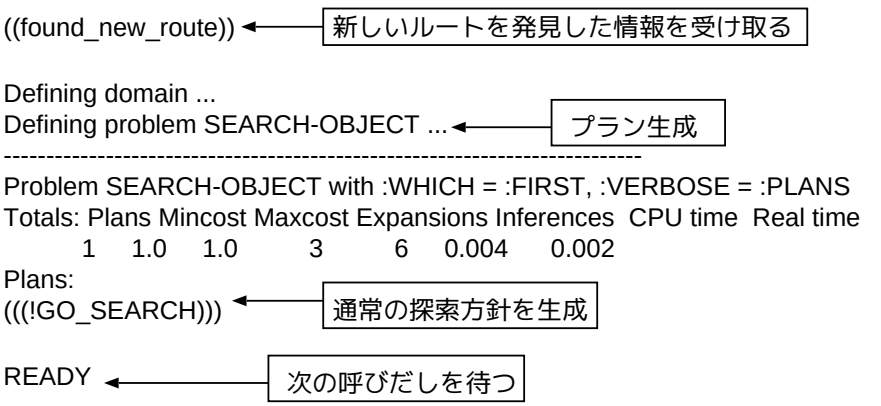

図 4.15: エージェントが新しいルートを発見したので、元の探索方法を指示したところ

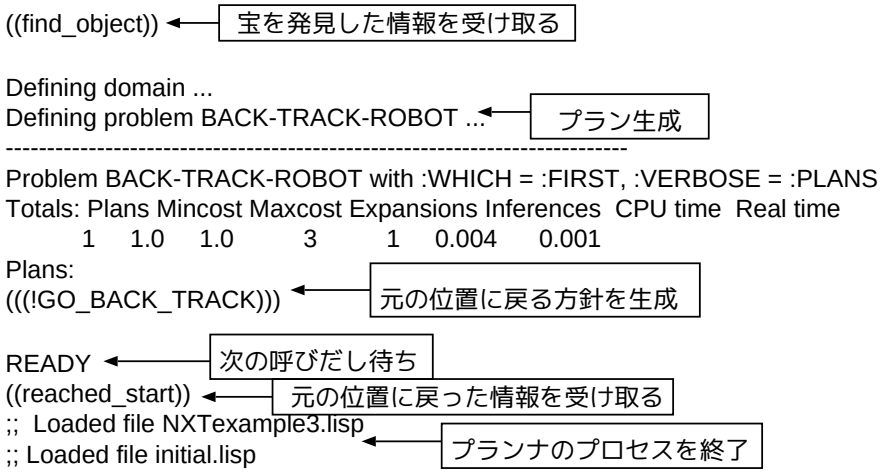

図 4.16: 宝を発見し、元の位置に戻り全体の目標を達成しプロセス終了

をプランナに任せることにより、熟考アルゴリズムを手続き的に記述する必要がなくなり、エージェント側からの現状を示す情報を与えるのみで、専念すべき行動方針を一つのプランとして決定できる。さらに、手作業でのコーディングでは、プランのコードとプランを選択するコード (メタプラン) が混在してしまうことから、状態爆発が発生することが懸念されるが、我々の提案するプランナによる熟考ルーチンシステムにおいては、(ロボット実験のような例では効果が現れにくいが) 大規模な応用で、実世界上での効率のよいプラン選択を行うことが期待できる。

さらに、我々の手法は BDI エージェントの実装基盤としての Jason の欠点を補うものでもある。

Jason では一度プランが失敗すると、失敗用のプランを実行することはできるが、失敗用のプランが失敗した場合、再度目標達成することができない。すなわち、失敗を考慮した設計になっていない。しかし、現実世界での目標達成においては、失敗が伴う。何度も失敗して行動を選択し直した後に目標を達成できる場合も考慮する必要がある。

実際、4.5.2 節で述べた、2ヶ所通路をふさぐ実験においては、失敗からのリカバリが 1 回しかできないと、目標が達成できない。我々の手法によれば、失敗したときに他のアプローチからの目標達成のための方針を立て直すことが可能であり、従って失敗しても再度目標達成を試みることが可能である。

エージェント実装プラットフォームとしては、BDI エージェントの他によく知られているものとして JADE[28] が挙げられる。JADE は、エージェントの振る舞いを「ビヘイビア」という形態で記述し、ビヘイビアインスタンスを複数走らせることで並行タスクを実現する。しかし、ビヘイビアの起動に条件を指定することはできないため、目標を与えてそれを達成する方法を選択させるということが実現できない。BDI エージェントを基盤にした我々の手法では、目標指向の動作を自然に実現できる。

また、他のプラットフォームと比較しての BDI エージェントの利点の 1 つは、BDI logic[23] という形式化の手段が用意されていることである。我々はこれにプランナの記述を導入する拡張を提案しており [34]、これを用いた提案手法に対する形式的検証の可能性も期待できる。

4.7 形式化

2.2 節で述べたように、他のプラットフォームと比較しての BDI エージェントの利点の 1 つは、BDI logic[23] という形式化の手段が用意されていることである。我々は、BDI logic にイベントの選択、確率的状態遷移、不動点演算子、およびエージェント毎の心的状態オペレータを導入して拡張した論理体系 TOMATOes[35, 33, 36] を提案し、BDI エージェントへの外部の行為決定機構 (強化学習など) の結合を、TOMATOes によって形式化することを行っている。ここでは、BDI エージェントへの HTN プランナの導入を TOMATOes で形式化することについて簡単に述べる。

4.7.1 TOMATOes

まず TOMATOes の導入を行う。BDI logic と比べての TOMATOes の拡張点 (のうち本節で使うもの) は以下の通りである。

- 心的状態オペレータがエージェント毎にある。例えば $\mathsf{BEL}^a\,\phi$ は「エージェント a が ϕ という信念を持つ」を表す。
- あるイベント e が現時刻で実行可能であることを表す原始論理式 $\mathsf{pos}(e)$ を持つ。
- イベントを伴う時相オペレータがある。例えば $\mathsf{AX}^e\,\phi$ は「イベント e の実行によって行ける次の時刻の全てにおいて ϕ が成り立つ」を表す。

4.7.2 HTN プランニング問題の記述

ここでは HTN プランニング問題におけるドメイン (問題) 定義を以下のように捉える。なお、簡単のため、メソッドの本体の制約は半順序のみとし、サブタスク同士が並列に動く場合についてはここでは考えないものとする (そのような場合も含めたより一般的な議論は [34] に譲る)。

メソッドの集合 $\mathcal{M}$ とアクションの集合 $\mathcal{A}$ が与えられており、1 つのメソッド $m \in \mathcal{M}$ はタスク t^m、前提条件 ϕ_m、本体となるサブタスクまたはアクション $t_1^m, \cdots, t_{n_m}^m$ とその間の半順序 (実行順序に関する制約) $\mathcal{R}_m$ の組、1 つのアクション $e \in \mathcal{A}$ は前提条件 ϕ_e とその効果 ξ_e の組とする (簡単のためここではタスクやアクションの引数は省いて書く)。

すると、各タスク (またはアクション) t に対し、t の開始・終了を示す新たな述語 *start_t*, *done_t* を導入することにより、タスクの実行過程は

$$\neg\phi_m \supset \neg \mathit{start\_t}^m \tag{4.1}$$

$$\mathit{start\_t}^m \supset \mathsf{AF}\,\mathit{start\_t}_i^m \quad (\text{for } 1 \le i \le n_m) \tag{4.2}$$

$$\mathit{start\_t}^m \vee \bigvee_{1\le i\le n_m} \mathit{done\_t}_i^m \supset \bigvee_{1\le i\le n_m} \mathit{start\_t}_i^m \vee \mathit{done\_t}^m \tag{4.3}$$

$$\mathit{done\_t}_i^m \supset \mathsf{A}(\mathit{done\_t}_i^m\ \mathsf{U}\ \mathit{done\_t}^m) \quad (\text{for } 1 \le i \le n_m) \tag{4.4}$$

$$\bigwedge_{1\le i\le n_m} \mathit{done\_t}_i^m \supset \mathit{done\_t}^m \tag{4.5}$$

$$\neg\mathit{done\_t}_i^m \supset \neg\mathit{start\_t}_j^m \quad (\text{for } t_i^m\ \mathcal{R}_m\ t_j^m) \tag{4.6}$$

のように書ける。これらは順に、「前提条件が満たされなければタスクは開始されない」「タスクが開始されれば、そのタスクのサブタスクはいつかは開始される」「あるサブタスクが終われば、

次のサブタスクを開始する」「サブタスクが終了されると、親タスクの終了までそのことを覚えておく」「全てのサブタスクが終わったことをもって、親タスクの終了とする」「2 つのサブタスクに順序制約がある場合、先に行うべきタスクが終了しないうちは後のサブタスクは開始できない」を表し、全体としては「タスク t_m の本体の全てのサブタスクを、実行順序の制約 $\mathcal{R}_m$ を満たした上で実行完了すれば、t_m の実行も完了する」ことを記述している。また、アクションの実行過程は

$$\phi_e \supset \mathsf{pos}(e) \wedge (\mathit{start\_e} \supset \mathsf{AX}^e(\mathit{done\_e} \wedge \xi_e))$$

のように書け、これは「アクション e の前提条件が満たされればアクションは実行可能であり、実際に開始されればその次の時刻にそのアクションは終了して終了条件が満たされる」を表す。

エージェント a がタスク t を達成するという目標を持ち、それに対する既存プランがプランライブラリ中にない場合、プランナは素朴には、目標 $\mathsf{DES}^a\,\mathsf{AF}\,\mathit{done\_t}$ を与えられて、これらの制約を満たすアクションの実行系列 $e_1, e_2, \cdots, e_k$ を発見する制約充足器と捉えられ、その系列が見つかれば、

$$\mathsf{INT}^a(\mathit{start\_e_1} \wedge \mathsf{AX}^{e_1}(\mathit{start\_e_2} \wedge \mathsf{AX}^{e_2}(\cdots \mathsf{AX}^{e_k}(\mathit{done\_t})\cdots)))$$

という意図 (アクション $e_1, e_2, \cdots, e_k$ を順に実行し、完了すれば t が達成される) の生成によりそれを実行に移す。

しかし、動的環境におけるエージェントが行うプランニングの場合、既に 4.1 節で述べたように、基本行為にまで分解することは現実的でないため、タスクを初めから基本行為にまで分解し切らずに大まかなプランのみ生成し、実行の段階でより精密なプランに分解するのが適切である。

例えば、タスク t に対するプランとして $e_1; t_1; e_2$ という列 (e_1, e_2 はアクション、t_1 はサブタスク) を生成したとする。この場合、それを実行する意図は

$$\mathsf{INT}^a(\mathit{start\_e_1} \wedge \mathsf{AX}^{e_1}(\mathit{start\_t_1} \wedge \mathsf{A}((\mathit{start\_e_2} \wedge \mathsf{AX}^{e_2}\,\mathit{done\_t})\,\mathsf{N}\,\mathit{done\_t_1}))) \tag{4.7}$$

と書ける (e_1 を実行、次の時刻に t_1 を開始し、それが終了したとき e_2 を実行すれば、その次の時刻に t が達成される)。

また、このプランの実行途中で、このプランで t が将来達成できるという信念を失った場合、信念世界の subworld (2.2.2 節参照) である意図の可能世界でも t は将来達成できないため (図 4.17)、式 4.7 も真でなくなる。すなわち、現在のプランを実行する意図を放棄することになる。

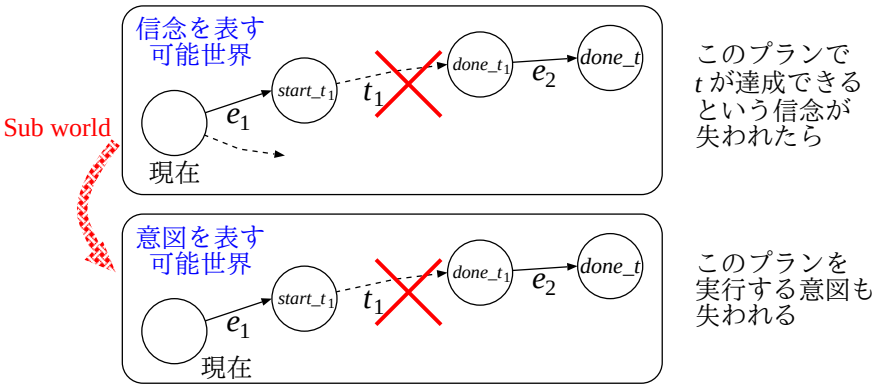

図 4.17: 達成に関する信念を失った場合の意図の放棄

BDI モデルとプランナの結合の形式化には、[26, 8] などもある。こちらは CANPLAN という言語で操作的意味論を定義するもので、プランナの動作もその記述の一部としてある程度特定さ

れて取り込まれる。一方、ここで述べた我々の形式化は公理的なもので、プランナの内部動作の特定を要しない。このため、プランナと BDI エージェントの疎な結合、すなわちプランナの特定の実装に依存しない形でのエージェントとの結合を扱える。

第5章　プランライブラリを拡張するための プランナ

本節では、プランライブラリを拡張するためのプランナについて述べ、特にサブゴールから派生される多様な行動を並行に生成することが可能である、「オンデマンド型サブゴール」の取扱いについて述べる。

このようなプランナをエージェントに取り付けることにより、必要に応じて合成したプランをプランライブラリが拡張されることができ、プランライブラリを固定された概念で留まるものではなく、拡張性のあるものにすることが可能である。

5.1　オンデマンド型サブゴール

我々のプランナは抽象レベルでサブゴールを保持し、そのゴールから派生する多様な行動を並列に扱うことが可能である。特に、そのゴールを達成しようとする過程において、何らかの障害が発生した場合、それを解決するための新たなサブゴールを動的に発生し、そのためのプランを生成することができる。このようなサブゴールに関して、我々は「オンデマンド型サブゴール」と以降呼ぶことにする。

例えば、5.4 節で取り扱う、農園に行ってトマトを収穫するエージェントの例で、エージェントがサブゴール「農園へ向かう」を保持し、農園へ向かう行動を開始したとする。このとき、エージェントは農園へ向かうことと並行して、その行動を継続するために必要なさらに小さい多様なサブゴールを並行して持つことができ、プランナはそれらのプランを生成した上で、農園に向かう行動の一環として管理することができる。

その一つとしてエージェントの行動の邪魔をするキウイがいる場合、エージェントはキウイを捕獲するプランをプランナに生成してもらい、それを元にキウイを捕まえることができる。(図 5.1)

しかし、これと同時に仮にエージェントが喉の渇きを欲して水筒を取り出して水を飲みたいという願望が発生しても、それが農園へ向かうというゴールから派生するものとして扱われる限り、並行して水筒を取り出すプランを生成することは可能である。

ここで、例えば「キウイを捕獲する」のようなサブゴールを生成してそれのみを管理することを行うと、キウイを捕獲することと独立に水筒を取り出して水を飲むようなプランを生成して、合理的に管理することはお互いが無関係なゴールであるため管理が複雑化し好ましくない。我々のプランナは両者の派生元である「農園へ向かう」という抽象レベルでオンデマンド型サブゴールを管理することにより合理性を保つことを容易に行うことができる。

5.2 適切な行動生成のためのプランの生成方法

行動の生成に関しては、同じゴールに対しても、現在の信念を用いた条件記述によって、生成するプランを絞りこみ、現状との矛盾などが起こらないように調整することが可能である。

5.4 節で取り扱う問題で言えば、キウイを捕まえるプランを知らない場合には、プランナにキウイを発見し捕まえる方法を知らないという情報を与え、キウイを捕まえるプランが生成できるようになっている。

このときエージェントに与えられるプランの内容は、バンジョー (楽器) を保持している場合には、バンジョーを置いてキウイを捕まえる[1]という手順の組合せになっている。

しかし、もし、バンジョーを置いている間にキウイが見えなくなってキウイを捕獲するプランが失敗に終わった場合、その後、キウイを発見した時にプランナにエージェントがキウイを捕獲するプランの提供を求めたとき、エージェントはバンジョーを持っていないという情報をプランナに渡す。

このとき、プランナはバンジョーを置くという手順を外したプランを生成したプランを生成し、それをエージェントに渡す。

いずれの場合もエージェントから与えられる情報が的確であればその場面に応じたプランを生成することが可能であり、キウイを捕獲する手段も場合によって変化し、現状を把握したプランを生成することが可能である。

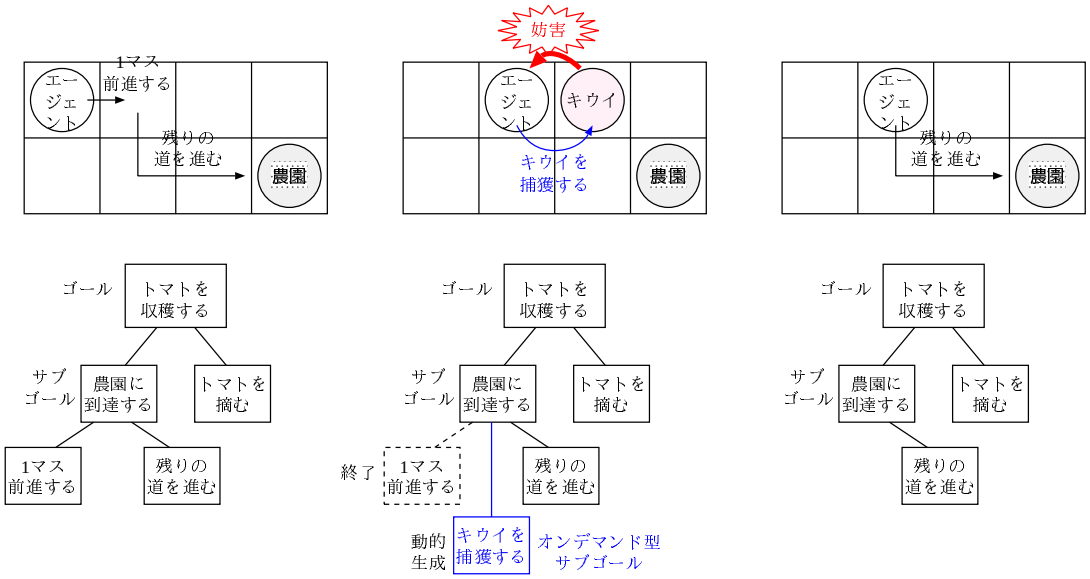

図 5.1: 動的な環境に適応するプランナ概要図

5.3 統合方法

我々は、Jason の環境プログラムが Java で記述されていることを考慮し、その Java プログラムにプランナのプロセスを生成して Jason とプロセス間通信する機能を実現した。次に、BDI エージェントのプランライブラリ内に呼出しコマンドを設け、それによってプランナにエージェント

[1]以後「キウイを捕獲する」とは必要に応じてバンジョーを置いてからキウイを捕まえることを指し、「キウイを捕る」とは捕まえる行為そのものを指す。

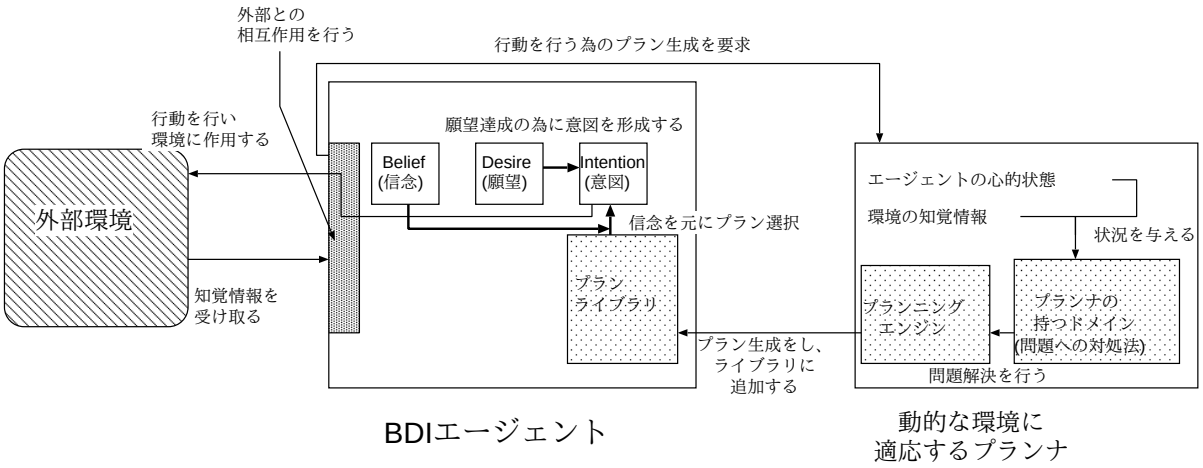

図 5.2: 実装全体図

の状況を渡し、プランナがプランを生成、それをエージェントが受け取り、エージェントの内部イベント標準ライブラリである.add_plan コマンドによりプランをプランライブラリに追加するという形で実装を行った。

全体図としては、図 5.2 のようになる。

5.4　実験

我々が提案するエージェントとプランナの統合の実装の有用性を検証するため、以下のような実験を行った。

実験の内容は、エージェントが農園でトマトを収穫するために農園に向かってトマトを収穫するというものであり、その道中キウイがエージェントを見つけるとその行動の邪魔をする。エージェントはキウイの妨害を受けてそのままではゴールを達成しづらい状況になると、キウイを捕獲しようとし、そのプランをプランナから獲得して実行する。

以下に実験の詳細と、実験の評価を記す。

5.4.1　実験の設定

図 5.3 のようなシミュレータを使用して実験を行った。

実験の内容は、エージェント (robot) が右画面下にある農園 (farm) に向かって道 (road) にそって動き、農園に着いたらトマト (tomato) を収穫するというものである。

このとき、中央にいるキウイ 3 羽 (kiwi) がエージェントの行動の邪魔をするので、キウイを見つけたら捕獲するゴールが発生するが、最初そのプランを持っておらず、そのことに気づいてプランナに依頼し、プランを生成してもらう。

エージェントはバンジョー (楽器) を持っており、これを持ったままではキウイを捕まえられないので、キウイを捕獲するプランとして、もしバンジョーを持っていればバンジョーを置いてからキウイを捕る行動を取るというプランを用いる。

キウイの動きは最初はランダムになっているが、エージェントを発見したらエージェントに近寄り、一定の確率で攻撃を行う。エージェントは攻撃を受けると、行動の一回分その場から動けなくなる。

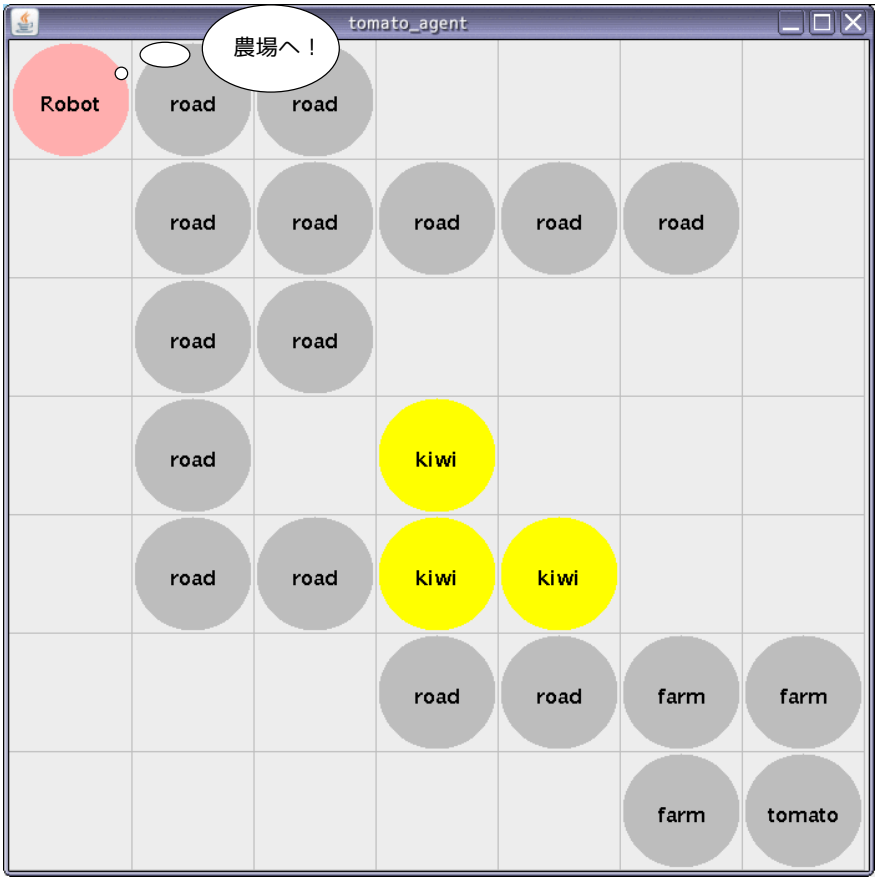

図 5.3: 実験の画面 (初期状態)

5.4.2 実験における検証

検証として、実際にシミュレーションを行った結果、最初にエージェントが持つ信念は図 5.4 のようになり、エージェントは!get(tomato) というゴール (トマトを収穫する) を行うために!at(robot,farm) というサブゴール (農園に到達する) を達成するという意図を生成している。これはエージェントプログラム内にある、プランライブラリを参照した結果得られた意図であり、これを全体の目標としてエージェントは行動を行う。

このとき、エージェントプログラムの起動と同時にプランナのプロセスも生成され、プランナの持つサブゴールは農園へ向かうことをサブゴールとして保持しエージェントの持つ意図と同期をとっている。

図 5.5 では、実際にエージェントがキウイに遭遇し妨害を受けた際に、オンデマンド型サブゴールとしてキウイを捕るプランをプランナに生成してもらうことにより、エージェントが現在のサブゴール「農園に到達する」ことを達成するように対処している様子が伺える。

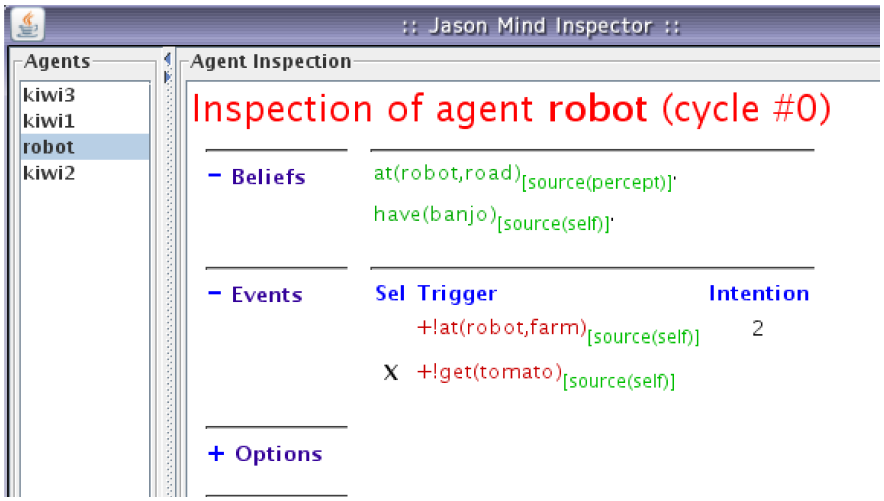

図 5.4: エージェントの初期信念

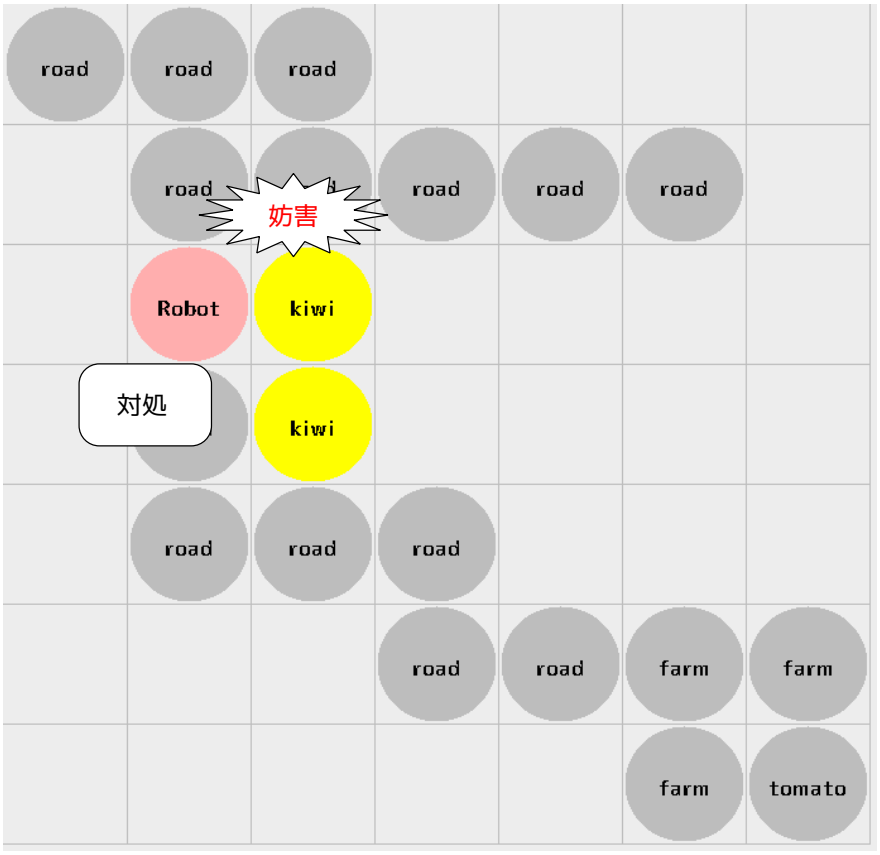

図 5.5: プランニングを必要とする状況

これによりキウイを捕獲する方法であるプランをさらにプランナに生成してもらい、キウイを捕獲することができるようになった。

図 5.6 においてこのときにおこなわれたプラン生成の流れを示す。

農園 (farm) にたどり着いた時点 (図 5.7) での信念は図 5.8 のようになり、バンジョーを置いて

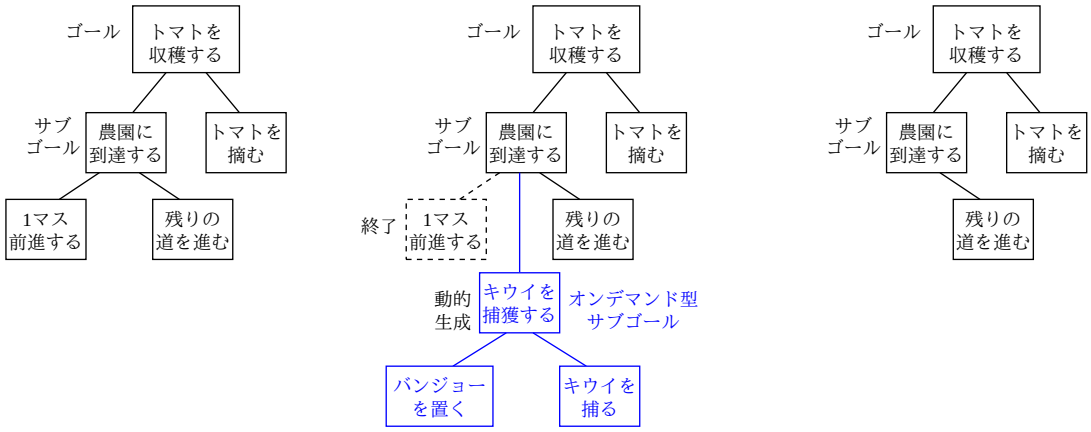

図 5.6: オンデマンド型サブゴールを取り扱った状況におけるプランニングの流れ

(drop(banjo)) 既にキウイを 2 羽捕獲していることがわかる。また、キウイから攻撃を受けてエージェントは一時的に行動を行えないようになっているという、can_not_move(robot) という信念が追加されている。

この後トマトを収穫することを達成し、一連のサブゴールを達成し、目標を達成することができた。

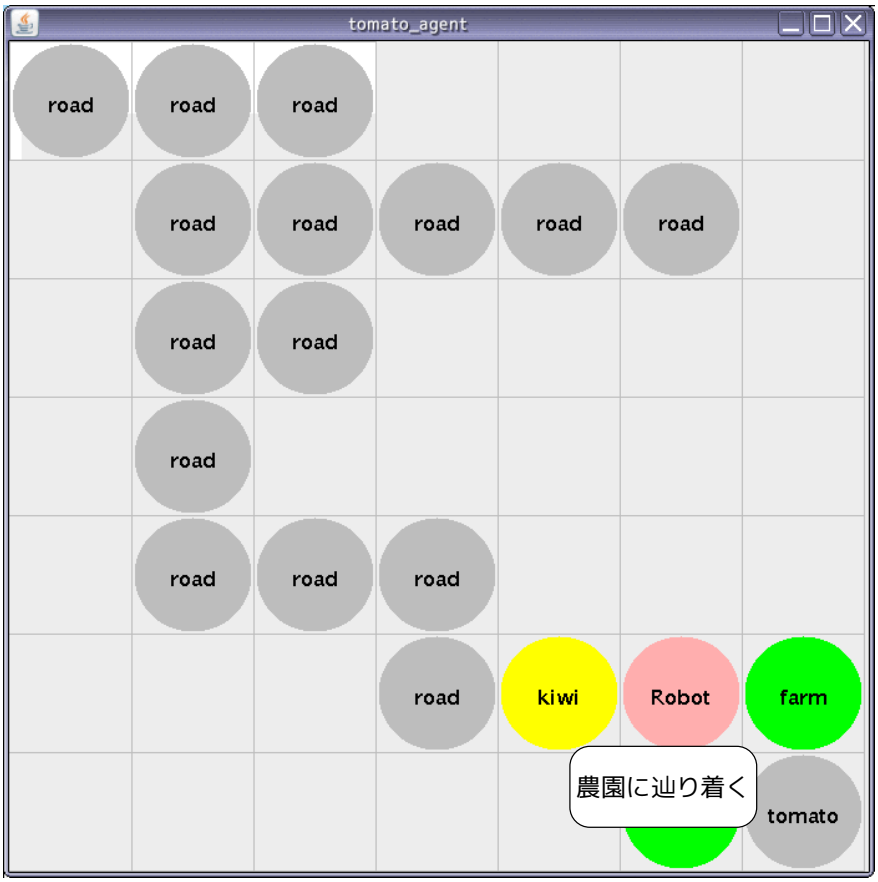

図 5.7: シミュレーション画面 (農園に辿り着いた時点)

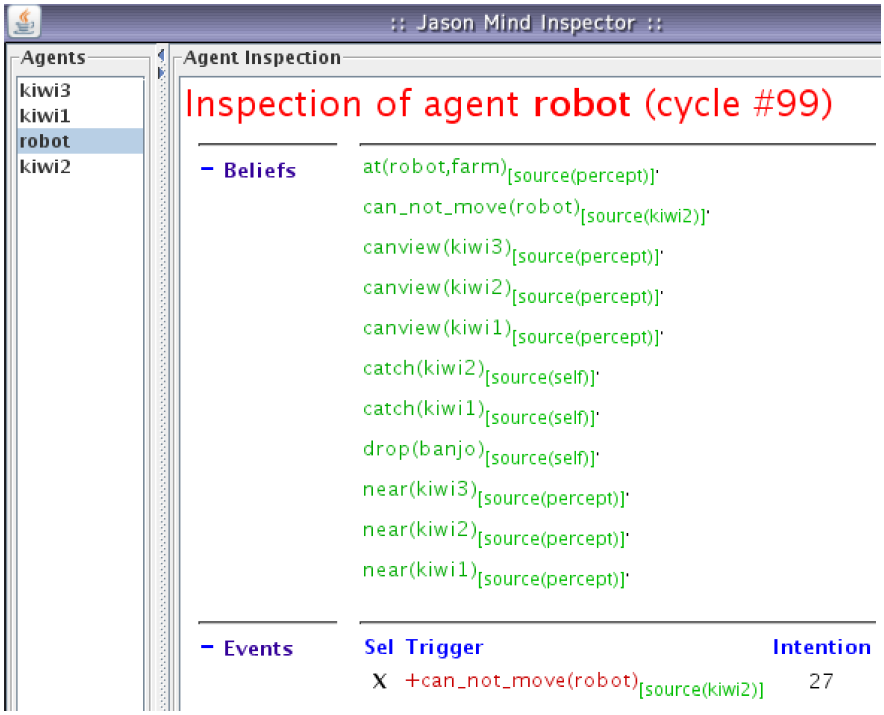

図 5.8: エージェントが農場に辿り着いた時の信念

5.5　考察

5.4 節で述べたようにプランナはエージェントの行動生成を状況に合わせて行い、これによってエージェントは自身のプランライブラリにある行動以外の行動をとることができ、プランライブラリの拡張を可能とした。

現時点でのプランナにまつわる問題点としては、サブゴールに定義された行動がいかなる状況であっても、条件さえ適合すれば生成することが可能が挙げられ、たとえエージェントにとって状況を考えるとその行動を取ることがふさわしくない場合 (時間の制約がある場合など) にでも現時点ではプランナはそのプランの生成を行う。

これには、行動の優先度やより詳細な状況把握を行えるようなシステムを必要とするので、今後の課題である。

また、今回取り扱ったシミュレーション実験の規模は比較的小規模であるがゆえに、このプランナの取り扱うドメインが小規模で済んだが、実際にプランナにより複雑な問題を与えることによる性能評価をすることも課題として残っている。

現在考慮している評価方法としては、[6] のようなオープンソースになっている 3D ゲームに我々のプランナ及び BDI エージェントを搭載し、動かしてみることが挙げられる。

5.6　関連研究との比較

先に述べた [32] では、親エージェントと子エージェントを用意し、親エージェントが目標達成のための大まかなプランを生成し、子エージェントがその親エージェントが作成した大まかなプ

ランの中のサブゴールについて基本行為までのプランニングを行うといったものであり、親エージェントと子エージェント間の動的な連携による同期をとることが難しいのが問題点とされる。これに対し、本研究では大まかなプランは抽象的レベルで記述することにしており、大まかなプランを破棄することはプラン全体を破棄することが起こらない限り本質的に必要はなく、また我々のプランナは比較的大きなプランから小さなプランまで生成することが可能であり、プランナ自体を二分化する必要もなく、同期をとるという問題自体が発生しない。

また、[26][30] では、基本的に HTN プランナを BDI エージェントに結合したものであるが、実装においては全順序型プランを生成し、これを意図としてエージェントに与え、エージェントはこれに従って行動をする。このプランが失敗した場合、その時点から再プランニングを行い、再び専念するという仕組になっているが、我々のシステムでは実装が、現在のプランを継続するのに支障するなどの状況が信念に加わった場合 (キウイを発見したなど)、それが引き金となってその解消を達成するためのプランニングを行い、現状に必要なプランを渡すという仕組みになっており、それが終了すれば元のプランを続行できる。よって最初から最後までプランニングを行うことがなく、失敗しても不要なプランニングをすることが最小限になるようにとどめることが可能である。

[9] では、プラン失敗によってプランナを呼出し、その場面にあったプランを生成してもらう設計である。

このプランナはその使用用途 (救急車の適切な配置方法など) を決めて設置して呼び出すため、使用用途外のプラン生成は行えない。従って、複数の事態に対処するような一連の行動の一貫性を元にした動的なプラン生成を行うことはできない。

これに対して、我々のプランナは一連の行動の一貫性を保持しつつ、現状を元にした動的なプランニングを行うことが可能である。

5.7 形式化

ここでは、4.7 節で述べた拡張 BDI logic である TOMATOes を用いて、我々の提唱するオンデマンド型サブゴールの形式化を示す。

4.7.2 節の定義を少し変更し、メソッド m の構成要素にオンデマンド型サブゴールの集合 $Odsg_m$ を加える。ここで、$Odsg_m$ は $\langle \phi, \psi, t \rangle$ の形の 3 つ組を要素とする有限集合で、ϕ は前提条件、ψ は事後条件、t はタスクを表す。直感的には、$Odsg_m \ni \langle \phi, \psi, t \rangle$ とは「メソッド m の実行中は、ϕ が成り立てば t を達成せねばならず、その達成後は ψ が成り立つ」を表す。

すると、4.7.2 節の式 4.1〜4.6 のうち 4.3 だけを

$$\mathit{start\_t}^m \vee \bigvee_{1 \le i \le n_m} \mathit{done\_t}_i^m \vee \bigvee_{\langle \phi, \psi, t \rangle \in Odsg_m} \mathit{done\_t} \supset \bigvee_{1 \le i \le n_m} \mathit{start\_t}_i^m \vee \bigvee_{\langle \phi, \psi, t \rangle \in Odsg_m} \mathit{start\_t} \vee \mathit{done\_t}^m$$

に変更し (本来のサブタスク、あるいはあるオンデマンド型サブゴールのタスクのいずれかが終了すれば、他のサブタスクあるいはオンデマンド型サブゴールのタスクのいずれかが開始する)、さらに式

$$\phi \supset \mathit{start\_t} \quad (\text{for } \langle \phi, \psi, t \rangle \in Odsg_m)$$
$$\mathit{done\_t} \supset \psi \quad (\text{for } \langle \phi, \psi, t \rangle \in Odsg_m)$$

を加える (オンデマンド型サブゴールのタスクの開始・終了条件) ことで、オンデマンド型サブゴール込みでのタスクの実行過程が書ける。

また、4.7.2 節の例と同様、あるタスク t が $e_1; t_1; e_2$ という大まかなプラン (e_1, e_2 はアクション、t_1 はサブタスク) に分解され、かつオンデマンド型サブゴールが 1 つだけ ($\langle \phi, \psi, t' \rangle$ とする) あるとき、その実行過程は

$$\mathsf{A}((\phi \supset \mathit{start\_}t') \,\mathsf{U}\, \mathit{done\_}t) \wedge \mathsf{A}(\xi_1 \,\mathsf{N}\, \neg\phi)$$

$$\text{ここで} \quad \xi_1 \equiv \mathit{start\_}e_1 \wedge \mathsf{AX}^{e_1}\, \mathsf{A}(\xi_2 \,\mathsf{N}\, \neg\phi), \quad \xi_2 \equiv \mathit{start\_}t_1 \wedge \mathsf{A}(\mathsf{A}(\xi_3 \,\mathsf{N}\, \neg\phi) \,\mathsf{N}\, \mathit{done\_}t_1)$$

$$\xi_3 \equiv \mathit{start\_}e_2 \wedge \mathsf{AX}^{e_2}\, \mathit{done\_}t$$

と書ける。t の達成の過程では、ϕ が成り立てば t' を達成せねばならないこと、および、ϕ が成り立たない時がきたらまず e_1 を実行し、その後再び ϕ が成り立たない時が来るのを待って t_1 を実行し…という過程が書かれている。

このように、TOMATOes を使えばオンデマンド型サブゴールを伴うプランニングを形式的に記述することができる。

第6章　ロボット制御へのBDIモデルの適用

これまでに実用化されているロボットの多くは、単一の目標達成に特化されたもので、1節で述べたように、たとえば二足歩行エンターテインメントロボットなどはいかにダンスなどの振る舞いをうまく表現するかに重点がおかれており、多様な環境変化に適応しながら行動をするものではない。このようなエンターテインメントロボットなどのようなロボットは身体性を持っているが、アクションの状態遷移を網羅してハードコーディングされているのものがほとんどである。つまり実世界を一定の仮想世界と仮定してロボット制御が行われている。

一方、人間と協調的に日常生活を指向したロボットを作るのであれば、人間や環境の多様な変化に適応して複数の目的を整合的に達成する必要がある。このようなロボットを制御するにおいて、複数の目的を達成すべく状態遷移におけるハードコーディングを行うとすれば、これは状態爆発が避けられない結果となる。

そこで、この章からは2.1節にて紹介した「意図の理論」をベースにしたBDIモデルに基づいたロボットの意思決定が、そうした多様性に富む実世界において複数の目的を整合的に達成するために有効であることを示す。

また、本論文ではこのようなBDIモデルを実装したロボットをBDIロボットと呼び、実際にロボット実験を行い、検証を行う。

6.1　実世界のさまざまな問題

実世界におけるさまざまな問題として、たとえば心理学者戸田正直が提案した「キノコ食い」のような完全エージェント [29] が身体性を有して、実世界で長い期間生き延びて機能するためには、仮想世界とは異なる以下のような問題が生じる [21]。

1. 情報獲得に時間がかかる。
2. 極めて限られた情報しか得られない。
3. 物理デバイスは外乱や故障から逃れられない。
4. 実世界を離散的には記述できない。
5. 実世界のエージェントは、常に複数のことを並行に行う必要がある。
6. 実世界は固有のダイナミックスで常に変化している。
7. 実世界は極めて複雑なダイナミカルシステムであり、その非線形特性と初期値に対する鋭敏性ゆえに、本質的に予測不可能である。

たとえば、身体性を持ったエージェント、つまりロボットは、(1) 隣の部屋にプリンタがあるのかどうか知りたければ、行ってみるか、だれかに聞くか、あるいは、ネットワーク構成図をみなければならないなど、実世界から情報を取り出すには時間を要し、(2) カメラの視界内の部分的な

情報しか得られず、(3) 暗闇では、輝度が不足して障害物検出ができない、(4) スムーズな二足歩行の振る舞いを、離散的な状態に分けて記述することは難しく、(5) 看護ロボットであれば、患者の話し相手をしながら、飲み物を運んだり、ベッドに寝かせたり、体温や脈を測りながら主治医へ連絡するなど、患者の状態や要求に応じて、複数の目的を並行して行う必要があり、(6) 坂道での二足歩行は、自らの重さに重力が働き、バランスを崩すと転がりはじめ、上手く起き上がるのは難しい、(7) サーファーロボットであれば、荒い波のぶつかり合いなどによる多様な波の変化は予測不能である、などの問題に直面する。

6.2　実世界の諸問題と意図の理論の対応

上記のような実世界の諸問題と意図の理論との対応を考えてみる。まず 6.1 節の (1) に対しては、実世界の情報を得るプランを意図として保持し、情報が必要になれば実行することで、実世界と相互作用を行い所望の情報を得る。または他のエージェントに依頼することで得ることもできる。

次に同節 (2) に対しては、行為列を実装していない副目標を用いたプランを意図として保持・実行することで、状態が分かるまで保留するという場合が考えられる。また、センサの性能で自力では正確な情報が得られない場合は、正確な情報を得ることができるエージェントに依頼するプランを意図として保持・実行することで得ることができる。

次に、同節 (5) に対しては、複数の目的を状況に適応して実行するには、それぞれの目的を達成する意図を保持し、並行に実行することができる。

以上のような考察に基づき、我々は、上記 (1), (2), (5) のような実世界の問題への対応として、意図の理論に基づく上記のように振る舞う BDI ロボットを実装し、実世界の多様性に対して、BDI モデルの行為選択の柔軟性および意図の整合性と再考慮が有効であることを、次節以下の実験を通じて示す。また、その他の問題への対応は 6.7 節にて述べる。

6.3　BDI ロボット

本節では実験に用いたロボットの構成、および BDI ロボットの実装について述べる。

我々は今回、2 台のロボット、Explorer と Forklift (以下、Lift) を用意した。Explorer の目的は、掃除を行うこと、荷物を置く台を発見した場合に Lift に伝えること、Lift に依頼されたときに台の識別を行うことであり、Lift の目的は荷物を置く台に自身のもつ荷物一個をその台に置くことである。

初期状態ではこれらの BDI ロボットは台がどこにあるかは知らず、台の位置に関する信念はロボットの知覚行為によって加わっていくこととなる。

6.3.1　ロボットの構成

我々が用意した 2 台のロボットは、LEGO 社が開発した商用教育用のロボット LEGO MINDSTORMS NXT を用いて作成した。NXT は安価で比較的制御が簡単であり、また、多数のパーツを組み立てて一つのロボットを作成するためロボットの形状は様々なものに仕上げることが可能である [11] ので、実験に用いるには適したロボットである。2 台の作成にあたっては NXT の組み立て用 Web ページ [20] を参考にした。2 台の主な特徴を以下に述べる。

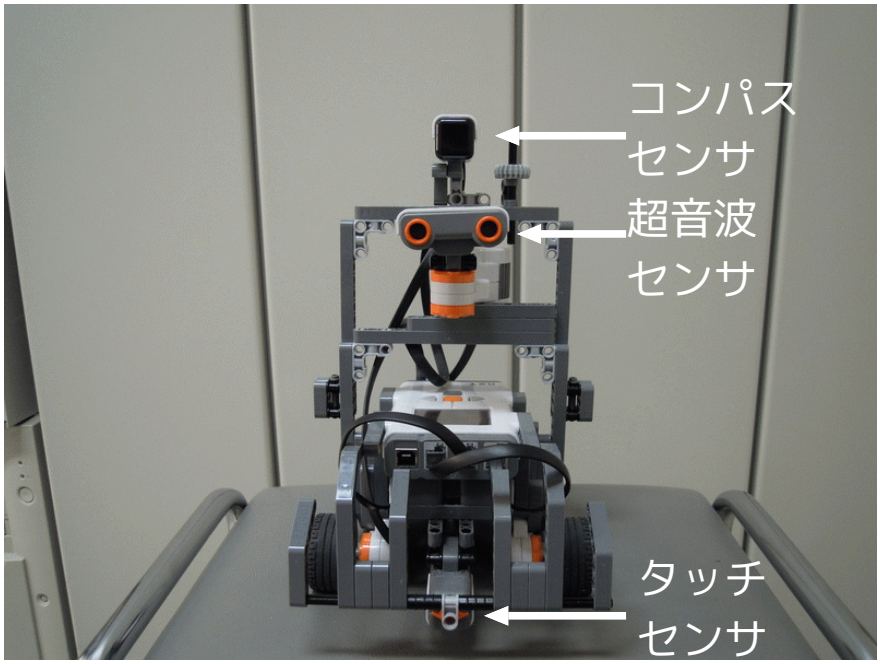

図 6.1: Explorer 正面図

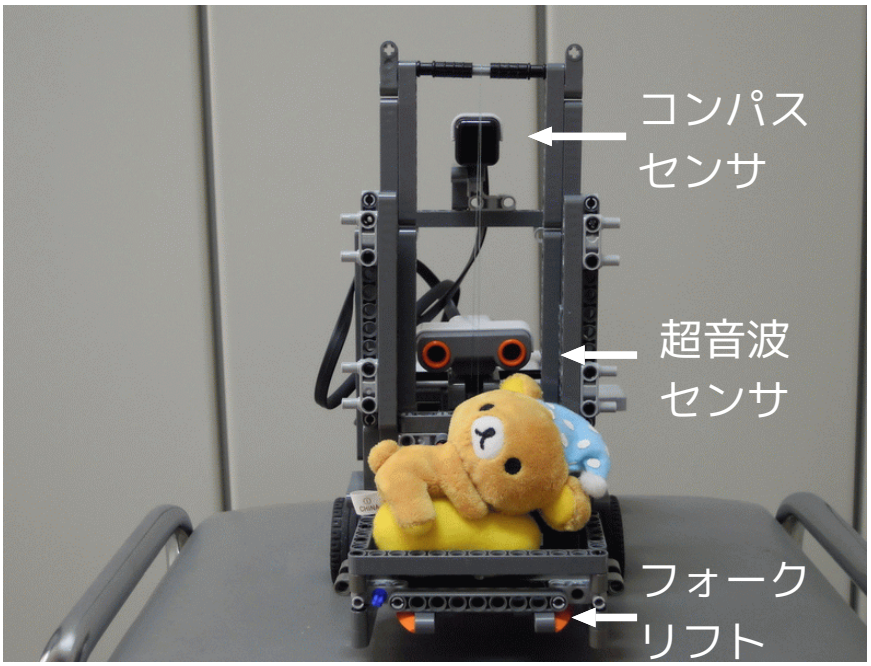

図 6.2: Lift 正面図

- Explorer

  Explorer は図 6.1 のような形状をしたロボットであり、

  - コンパスセンサ

  - 超音波センサ
  - タッチセンサ

  を持つ。

- Lift

  Lift は図 6.2 のような形状をしたロボットであり、

  - コンパスセンサ
  - 超音波センサ
  - フォークリフト

  を持つ。

6.3.2　ロボットの基本行動

ロボットの基本行為としては、グリッドワールドでの行動をにらみ、次のようなものを用意する。これには Python によるライブラリ NXT_Python[17] を用いている。

- 1 マス前進
- コンパスセンサを用いた現在の方向の知覚、及び 90 度単位の回転 (右、左)
- 超音波センサによるロボットの前方方向への知覚
- タッチセンサによる衝突認識 (Explorer のみ)
- フォークリフトの上にあらかじめ載っている箱を台の上に置く (Lift のみ)

6.3.3　環境設定

実験の環境設定は以下のものである。

- 実世界上にグリッドワールド状の盤を設置 (図 6.4 は配置の初期状態の例で、大きさが 5×4、左上のマスの座標が $(0, 0)$、右下が $(4, 3)$)
- Explorer と Lift と台をグリッドワールド上のマス目に配置
- 台は、荷物を置く台 (stand) と障害物 (obstacle) の 2 種類がある
- 荷物を置く台は 1 個だけ、障害物は複数個

また、以下の理由により Lift は台を見つけた場合、それが荷物を置く台であるかどうかの判定は Explorer に委託する必要がある (図 6.3)。これらは 6.1 節の (1) や (2) への対応を示すための設定である。

- Explorer の超音波センサは荷物を置く台より高いため、障害物のみ認識できるが、タッチセンサは荷物を置く台を認識できるため、両センサの併用で台の種類を見分けられる
- Lift は超音波センサのみ持ち、それは荷物を置く台より低いため、台を認識はできるが、台の種類を見分けることはできない

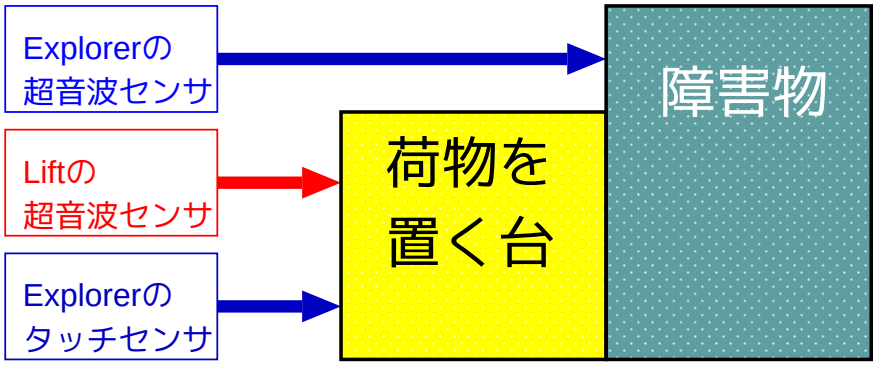

図 6.3: ロボットのセンサと台の高さの関係

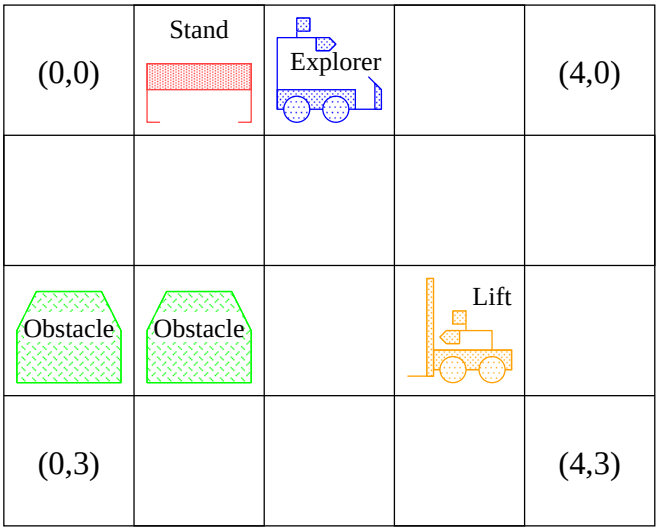

図 6.4: 実験の環境

6.3.4 熟考ルーチンの拡張

Jason であらかじめ (デフォルトで) 用意されている熟考 (deliberate) ルーチンは、現在実行中の意図をラウンドロビンで選ぶ (そして選ばれたプランを 1 ステップずつ実行する) というものだけである。しかし、実世界で複数の目的を整合的に達成していくにあたっては、より柔軟な熟考ルーチンを必要とすることがある。例えば、2 つのプランが競合しており同時には実行できない場合に、一方を優先して実行するといった判断を要する場合があり、本論文での実験にもその例が現れる。こうした処理は、プランの選択に関するメタプランとして、エージェントの信念内に書ける必要がある。

そこで我々は、Jason の熟考処理部を拡張し、熟考ルーチンをプラン記述と同様、Prolog 風のルール形式によるメタプランとして記述できるようにした。

具体的には、Jason が熟考を行う際に、`delib` という 3 引数述語が呼ばれるようにした。その第 1 引数には、現在実行中で選択の候補となっている意図のリスト、第 2 引数には、現在存在するが保留 (suspend) 状態となっている意図のリストが入る。ユーザは、この述語の第 3 引数に、選択された意図 1 つが代入されるようにこの述語を定義することで、熟考ルーチンを記述できる。この述語が失敗すれば、デフォルト通りラウンドロビンで意図が選択される。

例えば、特定の 2 つのプラン A と B について、A と B が選択の候補となっていれば A を優先するよう delib を記述することにより、A の実行中は B の実行を抑制させることができる (図 6.5)。プラン A 内にプラン B を一時停止させるよう記述するのとは異なり、プランの選択に関する記述をプランそのものの記述と混在させるのではなく、メタレベルの記述として分離することができる。

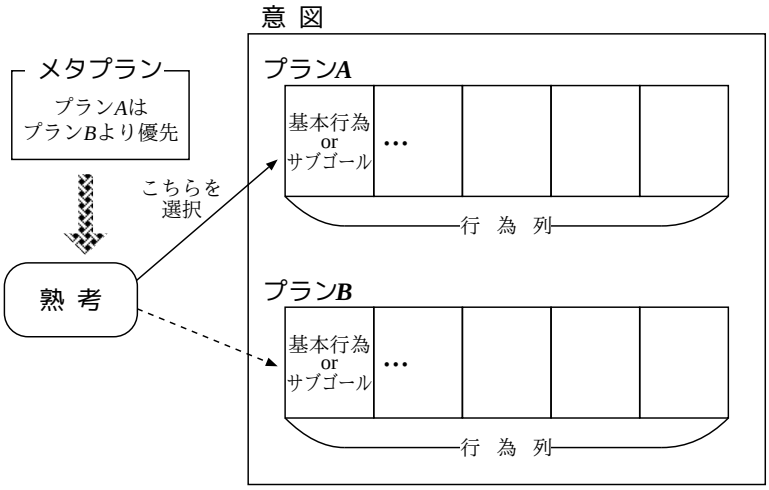

図 6.5: 熟考の拡張

6.4　BDI ロボットの目的と目標

2.1.3 節で述べた意図の再考慮や 2.1.4 節で述べたコミットメント戦略などの振る舞いを実現するために、BDI ロボットの目的及び目標を記述する。各目標の最後に書かれている「(プラン名)」は、それぞれの目標を達成するための、6.4.1 節にて記述されているプランの名前である。

まず、Explorer の目的は以下のように 3 つあり、それぞれの目標とともに列挙する。「•」で始まるものは目的、「– 」で始まるものはその目的を達成するための目標である。

- 盤上を掃除して回ること
  - 電池が消耗するまでの間、盤上を掃除して回る[1](プラン (a))
  - 電池が消耗してきた時、掃除のタスクは終了され初期位置に戻る (プラン (d))
- 荷物を置く台を発見したら Lift に伝えること
  - 掃除をしながら盤上を移動している途上において、荷物を置く台を発見した場合は、Lift に台を発見したことを伝える (プラン (c))
- 荷物を置く台かどうかの識別を行うこと
  - Lift に台の識別を依頼された場合は最優先して台の識別を行う (プラン (b))

次に、Lift の目的は 1 つである。「∗」で始まるものは副目標である。

[1]実験では実際は単に動き回っているだけだが、モップか何かを持って清掃して回っているという想定である。

- 荷物を置く台に一個荷物を置くこと
  - 荷物を台に置く目的を達成するために自身の持つ荷物を荷物を置く台に載せる (プラン (e))
    - まず荷物を置く台の探索を行う
    - 台の発見と共に Explorer に台の識別を依頼
    - 台に荷物をおくことができたならば、タスク終了とみなし、初期位置に戻る

    この目標を達成するために 2.1.4 節で述べた失敗した場合の処理を考え、意図の保持によって柔軟に対処することが可能であることを示すために、次のような失敗にも対処可能としている。
    - 環境の変化による失敗
      1. Explorer が先に荷物を置く台を発見した場合に、Lift にそれを知らせる (プラン (c)、プラン (g))
      2. Lift がその台の位置に向かう間に台がなくなってしまうことにより (環境の変化)、荷物を置くという目標の達成に失敗する (プラン (g) における失敗)
  - Explorer から荷物を置く台の発見を知らされた場合には、その場所に向かい荷物を置く (プラン (g))
  - Explorer の電池が消耗してきた時、Explorer はすべてのタスクを放棄するため、自身の依頼も受け入れてもらえなくなることから、Lift は荷物を置くタスクを達成不能と判断し、荷物を置くことをあきらめる (プラン (i))
  - Lift が現状の探索の範囲に偏りがあると判断したとき、探索の戦略を変更する (プラン (h))

6.4.1 BDI ロボットの目標を達成するプラン

上に述べた BDI ロボットの目標を達成するプランを以下に擬似コードで記す[2]。Lift のプランについては図にも示す (図 6.6、プラン (i) を除く)。

- Explorer のプラン

  (a) 掃除しながら盤上を動くプラン

```
clean_around
:  true
<- look_around; // 周囲のマスの知覚
   move; // 動けるマスを1つ選んでそこへ移動
   clean_around. // 再帰
```

  (b) 台の識別をするプラン ((a) より優先)

```
identify_rack
:  receive(lift, identify_request(X,Y))
   // Liftから識別依頼が来たら
```

[2]Jason で用いられる、イベントの追加などを表す記号 (「+!」など) は省いた。

```
<- wait(current_pos(X1,Y1)); // 自分の現在位置の取得
   calculate_route_to_next(X1,Y1,X,Y,Route);
   // (X,Y)の隣までの経路の計算
   move_along(Route); // 移動
   judge(X,Y,Type); // 識別
   send(lift, type(X,Y,Type)). // Liftに結果を伝える
```

(c) 荷物を置く台を発見したことを Lift に伝えるプラン

```
notify_of_stand
:  perceive_touch(X,Y) // タッチセンサが感知
<- if(percept(X,Y)){ // 超音波センサで知覚し直す
     // 知覚があればそれは障害物
   } else { // 何も知覚できなければ荷物を置く台と判断
     send(lift, type(X,Y,stand)). // Liftに結果を伝える
   }
```

(d) 電池が消耗してきて意図を放棄するプラン

```
abandon
:  timeout // 電池消耗により時間切れ
<- drop_all_desires; // すべての目標を放棄
   send(lift, abort); // Liftに放棄したことを告げる
   return. // 初期位置に戻る
```

- Lift のプラン

(e) 荷物を台に置くプラン

```
put_baggage
:  true
<- search_stand(X,Y); // 荷物を置く台を見つける
   if(percept(X,Y)){ // 念のため超音波センサで知覚し直す
     put_baggage_at(X,Y); // 荷物を置く
     return; // 初期位置に戻る
   } else {
     fail; // 失敗
   }
```

(f) 荷物を置く台を見つけるプラン (自分で見つける場合)

```
search_stand(X,Y)
:  not already_put_baggage // 荷物をまだ置いていない
<- find_object(X1,Y1); // 台を探す
   send(explorer, identify_request(X1,Y1));
   // explorerに識別を依頼
   wait(type(X1,Y1,Type));
   if(Type != stand){ // 荷物を置く台ではなかった
     search_stand(X,Y); // 再帰
```

```
  } else {
    X=X1; Y=Y1; // 荷物を置く台があった
  }
```

(g) 荷物を置く台を見つけるプラン (Explorer に教えてもらう場合。(f) より優先)

```
search_stand(X,Y)
:  receive(explorer, type(X,Y,stand))
<- wait(current_pos(X1,Y1)); // 自分の現在位置の取得
   calculate_route_to_next(X1,Y1,X,Y,Route);
   // (X,Y)の隣までの経路の計算
   move_along(Route). // 移動
```

(h) マップの左半分優先で台を探すプラン ((f) のサブプラン。右半分優先の同様のプラン (h') もある)

```
find_object(X,Y)
:  more_unseen_places_on_lefthalf // 未見の地が左半分に多い
<- set_search_strategy(left_prior);
   // 探索方針変更; look_aroundやmoveに影響を及ぼす
   look_around; // 周囲のマスの知覚
   if(found_object(X,Y)){ // 台があった
     // そのまま終了
   } else {
     move; // 動けるマスを1つ選んでそこへ移動
     find_object(X,Y); // 再帰
   }
}
```

(i) Explorer の電池が消耗してきた時に Lift がとるプラン

```
give_up
:  receive(explorer, abort)
<- wait(current_pos(X,Y)); // 自分の現在位置の取得
   drop_all_desires; // すべての目標を放棄
   return; // 初期位置に戻る
```

プラン (b) を (a) よりも優先する働きは、6.3.4 節に述べた拡張熟考ルーチンで実現される。

これらの中で用いられている `move` や `look_forward` などのサブプランは、最終的には 6.3.2 節で述べた基本行為 (と Jason の内部アクション) を用いて実現されている[3]。

6.4.2 BDI ロボットの行動

6.4.1 節に述べたプランのもとでの、各ロボットの振る舞いを以下に述べる。特定の初期条件のもとでの振る舞いの詳細については、6.5 節の実験結果において述べる。

[3]例えば move は、基本行為の「1 マス前進」と、Jason の内部アクションである他エージェントとの信念のやりとりなどを用いて実現されており、自分が進もうとしているマスの占有を他のエージェントに伝えることで、エージェント同士の衝突を防ぐように作られている。

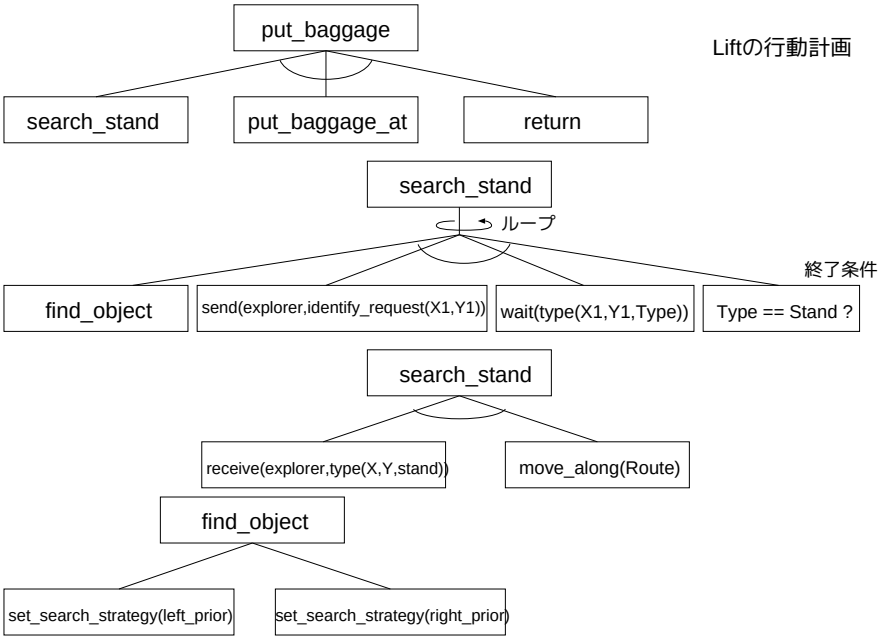

図 6.6: Lift の行動計画図

Explorer は初期目標として `clean_around` を与えられ、プラン (a) によって掃除をしながら盤上を動き回る。ただし、Lift から台の識別の依頼が来た場合は、プラン (b) が優先されて (a) が保留され、台の識別に向かう。

動き回っている最中に台を発見した場合、プラン (c) によりそれを識別し、荷物を置く台であった場合は Lift に知らせる。このプランの終了後は、元のプラン (a) が継続され引き続き動き回る。

電池消耗により時間切れとなった場合、目標 `abandon` が発生してプラン (d) によりすべての目標を放棄する。このときリフトにその旨を伝える。

Lift は、初期目標 `put_baggage` を与えられ、プラン (e) を実行開始。まず副目標 `search_stand(X,Y)` により、プラン (f) で荷物を置く台を見つけようとする。これに成功すれば、`(X,Y)` に実際に荷物を置く台があることを確認の上、荷物を置いて (e) は終了する。プラン (e) の途上で、Explorer が電池切れによる目標放棄を伝えてきた場合、Lift は台の識別の手段を失うので、プラン (i) によって自らの目標を放棄する。

プラン (f) により荷物を置く台を見つけるには、まず台を探してから Explorer に識別を依頼する。台を探す目標 `find_object(X1,Y1)` のためのプランは (h) と (h') の 2 つあり、その時点での状況 (未見の地が盤の左半分と右半分のどちらに多いか) に応じていずれかを選ぶ。また、プラン (f) の途中で、Explorer が台の発見を伝えてきた場合、`search_stand(X,Y)` を達成する意図をプラン (g) に切り替え、こちらで達成しようとする。

6.5　実世界の多様性への適応実験

6.3.3 節で述べたような設定のもと、実際に我々のロボットが実世界の多様性に適応できることを示すため、以下に述べるような 5 種類の配置および状況のもとで実験を行った。

I)　図 6.8 の配置

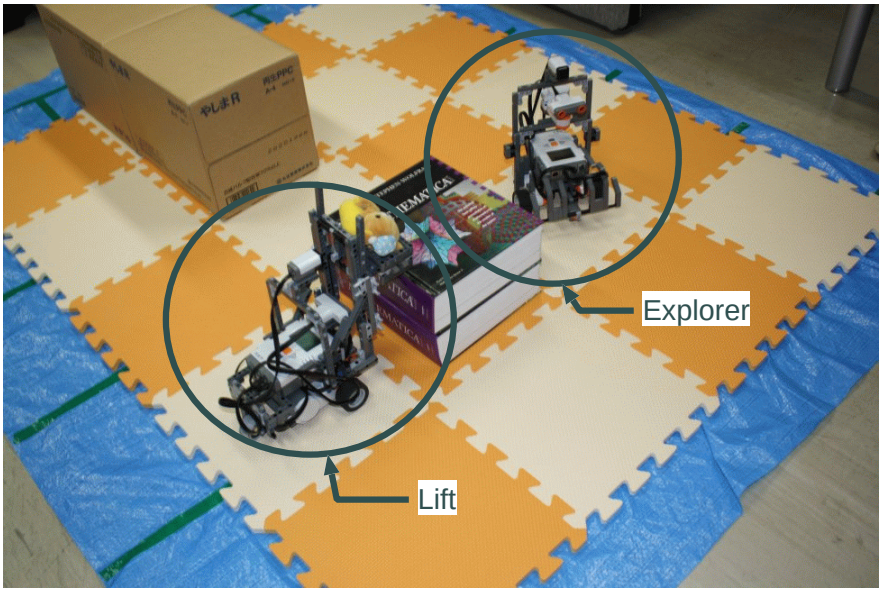

図 6.7: 実験の様子

II) 図 6.9 の配置

III) 図 6.10 の配置

IV) 図 6.8 の配置だが、Explorer が Lift に台の発見を伝えた後に台が移動してしまうという状況 (6.4 節で述べた環境の変化が起きるケース)

V) 図 6.11 の配置

各実験の目的および結果を表 6.1 でまとめる。図 6.7 は II) の配置 (図 6.9) からスタートした場合の実験の 1 場面 (Lift が台に荷物を置いているところ) である。

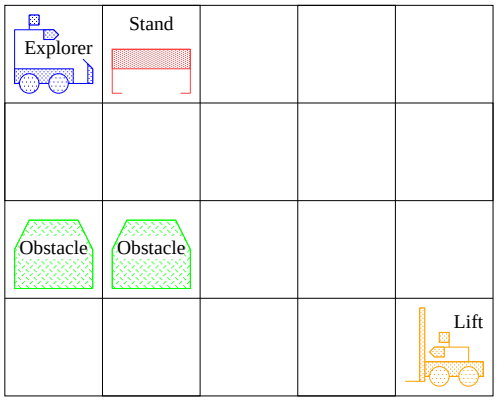

図 6.8: Explorer による発見の場合の配置図

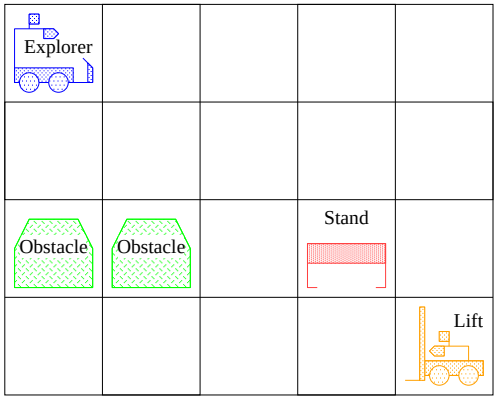

図 6.9: Lift による発見の場合の配置図

表 6.1: 実験の結果の表

実験の目的	初期配置	観測されたロボットの行動 (括弧内は 6.4.1 節の該当プランの番号)	評　　価
意図の再考慮 (2.1.3 節) が行えること	I) 図 6.8	Explorer は掃除を開始 (a) したが、$(1, 0)$ に荷物を置く台を発見し、Lift に伝えて (c) 掃除を再開。Lift は荷物を台に置くプラン (e) を開始していたが、台を見つける副目標のための意図 (f) を中断し、Explorer から伝えられた台のところへ行く (g) ことで同目標を達成。(e) の残りを実行して荷物を置き、初期位置に戻った	Lift が予期と異なる状況に直面し、目標を達成するための意図を切り替えることができたことで、意図の再考慮が行えることを示した
情報の不完全さ (6.1 節の (2)) や時間がかかること (同 (1)) への対応、および複数の意図の並行 (同 (5))	II) 図 6.9	Lift は $(3, 2)$ の台を発見 (f) し、Explorer に識別を依頼。Explorer は掃除のタスク (a) を中断し、プラン (b) によって台の識別を行った。その結果、荷物を置く台であることがわかり、それを Lift に伝えて (b) は完了、(a) を再開した。Lift は荷物を置く台だったことを知って (f) を完了、台を見つける副目標を達成。以後は I) と同じ	Lift は、台を見つけても識別はできない情報の不完全さ、および識別情報が来るまで時間がかかる事象に対処した。また、Explorer は掃除の意図を保持したまま台の識別の意図も達成しており、こちらは複数意図の並行の実現を示している
コミットメント戦略 (2.1.4 節の (b)、(c)) の実現	III) 図 6.10	Explorer は (a) の最中、台がないため台を見つけられぬうちに制限時間がきた。よって (d) により全タスクを放棄し Lift に通告、初期位置に戻って終了。Lift は通告を受け、(i) により全タスクを放棄し初期位置に戻り終了	Explorer は探索の達成を放棄して終了したため Open-minded、Lift は荷物を置く目標の達成不能 (台の識別ができないため) により終了したため Single-minded の各コミットメント戦略を実現している
意図の継続 (2.1.4 節) による失敗への対処が行えること	IV) 図 6.8 の配置で、Explorer が Lift に台の発見を伝えた後に台が移動	I) と同様に、Lift は Explorer から伝えられた台のところへ行ったが、台がなくなっていて (e) が失敗。しかし、荷物を台に置く目標は残っているので、この目標を達成するために再度プラン (e) が選ばれて意図され、台を探し直し目的を達成した	目標達成の手段が失敗しても、目標が残っていれば新たな手段を選ぶことで意図を持続できることを示した。また、新たに (f) を実行する際、`find_object` の達成手段はその時の状況によって変わる (左/右のどちらを先に探すかはその時になってから選ぶ) ので、この部分は 2.1.2 節で述べた副目標の達成手段の推論の先送り (部分的なプランニング) の例でもある
現状での問題点	V) 図 6.11	Lift が $(3, 2)$ の台の識別を依頼し、Explorer は (b) が (a) を抑止した格好で識別に向かうが、その途中で $(3, 1)$ の台を見つけて (c) が割り込む。(b) と (c) の間には排他関係が書かれていなかったため両者が競合し、正常に行動できなくなった	割り込みの入れ子のような場合における割り込み同士の競合には (通常の開発同様) 慎重な検討が必要である

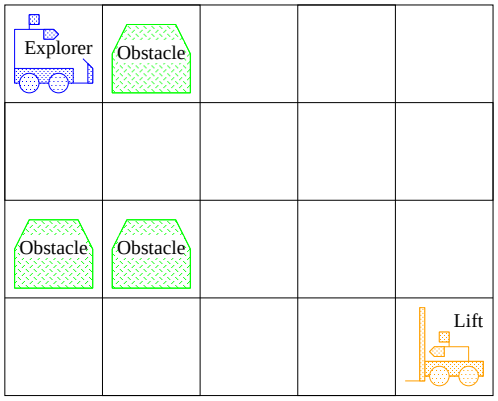

図 6.10: すべて障害物であるの場合の配置図

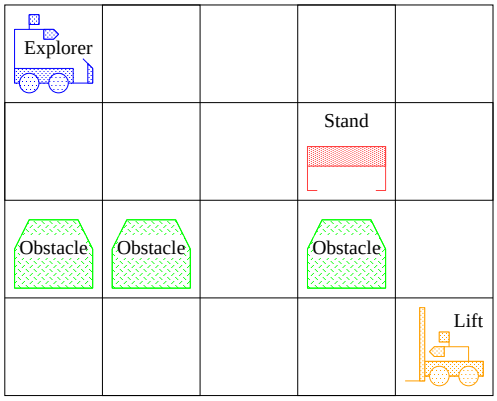

図 6.11: 例外処理が二重に起こる場合の配置図

6.6 考察

6.6.1 BDI ロボットの柔軟性

2 節で述べたように、BDI ロボットは 6.1 節の (1), (2), (5) に対処しつつ、多様な環境変化に柔軟に対応して複数の目的を整合的に達成できる。6.5 節の実験でも、Explorer は II) で複数の目的に対する意図を並立させており、また Lift は IV) で荷物を置く台がなくなっているケースに対応して台を探し直したり、I) で状況の変化に直面して目標達成の手段を切り替えたりすることができた。

この柔軟さには、6.3.4 に述べたようなプランとメタプランの分離によって、それらが互いに独立していることも寄与している。

この独立性は、実行時の動的環境変化への対処という意味での柔軟さだけでなく、エージェントの設計の柔軟な変更を可能とする利点ももたらす。例えば、6.4.1 節のプラン (b) を (a) より優先させている熟考ルーチンを変更し、「Explorer が Lift から “何か奢ってあげる” 旨伝えられているという信念を得た場合のみ (b) を優先し、さもなければ (a) の方を優先する」よう書き直すこともでき、その場合 Explorer は、何か奢るぞと言われていない限り Lift の台識別依頼を無視する。この変更にあたり、プラン (a) や (b) そのものの変更の必要はない。

また、プラン選択がプランから分離されていることから、プラン同士も互いに干渉しないように記述できるため、例えばプラン (a) で盤面探索アルゴリズムのみ変更することもでき、この際に (b) との干渉を気にする必要はない。これに対し、(b) からの割り込みを処理するルーチンが (a) の中に書かれるような実現法だと、(a) 内の変更においても干渉に注意する必要が出る。

このように、エージェントの信念 (に含まれているプランやメタプラン) を局所的に変更するだけで、柔軟な振る舞いの変更ができる点も、BDI モデルの利点の 1 つである。

6.6.2 意図を持つことの利点

BDI モデルの特徴の 1 つは、意図という心的状態を明示的に持つことであり、本論文のような実世界ロボットにおけるその意義を 2.1 節にて論じた。意図を持つことには他にも多くの利点があり、それをここで論じる。

タスク管理システムとしての BDI モデル

2.1.1 節で述べたように、BDI モデルにおいては、意図を単位としたタスク管理に相当することが BDI のフレームワーク自体で行われる。従って、他のプランへの切り替えをプラン自体が管理したり記述に含めたりすることは原則的になくて済む。

例えば、表 6.1 の実験 II) でプラン (b) の実行の間プラン (a) を保留する (そして (b) が終了すれば (a) を再開する) 処理は、(b) に「(a) を一時停止する」のように書かれることはなく、4.3.1 および 6.3.4 節に述べたような、熟考という BDI フレームワークの持つ働きで実現される[4]。また、実験 IV) のケースでも、台がなくなっていて荷物を置くことに失敗した後、Lift が再度探索プランを開始するのは、プランにやり直せと書かれているからではなく、目標を達成する意図が残っているために目的-手段推論が再度行われることによる。

ハンドコーディング手法では、プラン自体のコードにタスクスイッチの処理も書くことになり (OS が存在しない環境でのプログラミングに当たるだろう)、小規模な例ではよくてもタスク数が増えると破綻する。また、要求分析のような手法では状態遷移を構築することが多いが、状態数が増えると、例えば全ての状態に割り込みのためのコードを書くようなことが必要になればやはり破綻する。これに対し、タスク自体とその管理の自然な分離を可能とする BDI モデルは、多様性への対処には非常に有効である。

6.6.3　意図を用いた相互理解

BDI モデルでの意図概念は、単なるジョブスケジューラのみに相当するものではない。それが何かを為そうとする心的状態であることにより、人間との相互理解に寄与するものでもある。

たとえば、看護ロボットを考えてみよう。われわれは、看護ロボットが、将来、何をしようとしているのか、また、いま何をしていてその理由は何であるのかなど看護ロボットの心と行為を理解できないと何を要求していいのか分からない。逆に看護ロボットもわれわれの心と行為を理解できないと、十分なサービスはできない。

人間は、主体・他者の心と行為に関する理解は、意図概念を用いて行ってきた。そこで、人間とエージェント間の相互理解のために新たなプロトコルを創発するよりは、エージェントに意図概念を導入した方が合理的である。

本論文ではこうしたことは扱っていないが、実世界で人との親和性の高いロボットを実現する手段としての意図概念の利用は今後検討すべきであろう。

6.7　実世界ロボットでの実験の意義と現状の問題点

ここでは、シミュレーション実験と比較しての、実際に実世界ロボットで実験を行うことの意義について、および現状での問題点について述べる。

6.1 節で述べたように、実世界のロボットは、仮想世界と比べて、知覚に時間がかかったり不完全である、固有のダイナミックスで常に変化するなど、様々な点で異なる。よって、実世界のロボットの振る舞いをシミュレーションのみで開発したり検証したりしようとすることには無理がある。

[4]ただしある意図が他の意図の続行に能動的に関与することは可能ではあり (プロセスに例えれば UNIX における kill のように)、今回の例でも全ての目標 (とそれから発生している意図) を破棄する「`drop_all_desire`」が用いられているが、これを用いずに信念の変更による意図の放棄として同等のことを行うこともでき、この方がコミットメント戦略の本来の姿にも近い。

本実験でのその一例として、6.4.1 節で述べたプラン (a) のサブプラン move の実装の不適切さに起因するプラン (b) の誤動作が挙げられる。当初の実装では、move は 1 マス動いた直後に自分の現在位置に関する信念 `current_pos(X,Y)` を更新していた。しかし、これではプラン (a) の move での移動に時間がかかっている間に Lift から台の識別依頼が来た場合、プラン (b) が優先され、自分の現在位置を取得ののち台へのルートを計算して台の位置までの移動を始めてしまうが、「1 マス前進」という基本行為には割り込めないため、台の位置への移動は移動直後に始まる。しかし、台へのルートの計算の起点となった自分の現在位置は移動前のものであるため、正しく移動できない事態が起こった。しかも、この問題は移動に時間を要しないシミュレーション環境では発生しなかった[5]。

ロボット実験によってこの問題の存在を把握した結果、move の実装を、自分の現在位置の信念を削除 → 1 マス前進 (基本行為) → 新たな現在位置の信念を得る、と変更することで、プラン (b) は現在位置の取得を移動直後まで待つことになり、この問題は解消した。

このようなシミュレーションと実世界の概念のずれに起因する障害は、シミュレーションだけでは発見が困難であり (環境が変わると誤動作するプログラムのデバッグの困難さを想像すればよいだろう)、実世界でのロボット構築の実証実験の意義の一環はこうしたところにあると言える。

我々はこの点について、形式化の観点から 6.9 節にて再度論じる。

次に問題点について述べる。我々は本論文で、6.1 節で述べた実世界ロボットにとっての問題点のうち (1), (2), (5) への対処について述べた。しかしそれ以外、例えば (3) などへの対処は難しく、特に本実験で用いたような安価なロボットでは極めて困難である。本実験での具体例では、室内の電化製品などの磁場の影響で実験設備内での磁場が歪んで、コンパスセンサに誤差が生じ、正確な方向制御ができないことが起きた。また、(4) についても本実験ではモデルの簡単化のためグリッドワールドを用いたので非離散的な世界記述には対応できていないが、ロボットの行動をより柔軟にするという点からもこれは将来の課題である。

6.8 関連研究

6.8.1 他の BDI によるロボットの研究

これまで BDI モデルの応用はソフトウェアエージェントが多く、実世界ロボットへの応用はあまり見られない。その中で、文献 [18] と文献 [15] はどちらも、BDI によるマルチエージェントを小型のロボットに搭載し、実際に実験を行った例である。

文献 [18] は、複数のロボットがグリッドワールド上を探索してオブジェクトを収集するもので、ロボット間のコミュニケーションに重点をおいており、これによってロボット同士の衝突を避けたり、入札システムを介して、同じオブジェクトを複数のロボットが拾いに行くことを避けるなどを実現している。また文献 [15] は、複数のロボットがコンベア上の小包を指定位置まで運ぶというもので、ロボットはコンベアベルトに相当するものとフォークリフトに相当するものの 2 種があって、前者は小包をパレット上に置くこと、後者はそれを指定された位置に運ぶことを目標としており、これらの協力でシステムを構成している。

しかし、これらは複数の意図の並行や、状況の変化に応じて意図を柔軟に変更するといったことを扱っていないため、本論文で述べたような BDI モデルの利点について明らかにしているもの

[5]「シミュレーションでも移動に時間がかかるようにする」というアイデアは、この問題を把握していないと必ずしも当然なものとしては出てこない。現実に起こることのすべてをシミュレーションに取り込むことは不可能であり、適切なもののみシミュレーションモデルに取り込むことも、その適切さの予測があらかじめつきにくいため難しい。これは、実世界の複雑さに起因する本質的な困難といえる。

ではない。また、いずれにも中央集中的な管理機構が存在するため、本論文のように独立したロボットが必要な時のみコミュニケーションを取って協調するというものと比べ、エージェントの自律性もやや損なわれる。

6.8.2　他の適応的なエージェントプラットフォーム

状況の変化に適応して適切なふるまいをするシステムとしては、近年、自己適応システム [16] の提案もある。これは、環境との相互作用を行う最下層の部品制御層、同層からの環境変化の通知を受けて振る舞いの変更を担う中間層の変更管理層、下位からの要請で目標達成のためのプランニングを担う最上位の目標管理層の 3 層からなるアーキテクチャモデルである。

自己適応システムにおいては、状況の変化を最下層のセンサが感知すれば、中間層がそれに呼応して、下位の部品の変更や再構成を行うことで動的に振る舞いを変える。しかし、BDI モデルでは外界の変化によらず、自分の心的状態の変化によっても振る舞いの変化を起こすことがある。例えば Open-minded コミットメント戦略では、まだ達成可能な意図を、(自分が飽きたなどの) 理由で中断することがある (本論文の例でも、Explorer はタイムアウトという自己都合で探索の意図を中止している)。すなわち、BDI モデルの方が振る舞いの変化のより一般的なモデルを提供している。さらに、意図を保持したまましばらく保留して他の優先度の高いタスクを先に片付けるようなことは、振る舞いの変化という解釈では実現できない。振る舞いの解釈と変化するなら、例えば本論文の Explorer のプラン (a) から (b) への切り替えは、同じタスクの別々のフェーズのごとく捉えることになるが、本来全く別な 2 つの仕事をそのように捉えることは合理性を欠く。

また、自己適応システムの実現は研究途上の段階で、実装に関する提案はまだ十分とはいえない。実現の 1 つとして、目標指向の要求分析法である KAOS[7] によって問題を記述し、エージェント実装プラットフォーム JADE[28] によるコンポーネント記述への変換によって自己適応システムを構築する試み [38, 39] がある。しかし、KAOS での分析を JADE での実装のレベルにまで変換することは自動的には難しいという課題を抱えている。これに対し、BDI モデルを基盤にした我々の手法では、多様性に柔軟に対応した目標指向の振る舞いを自然に記述・実現できている。

状況に応じた柔軟な行動を取れる反応型エージェントを実現する例として、Bonasso らの 3T アーキテクチャ[2] も挙げられる。これは上述の自己適応システムとほぼ類似した構成を持つ 3 層アーキテクチャであるが、最下層は反射的行動に特化しており、単一ドメインの問題に特化したシステムである。しかし、複数のタスクを並行に扱うような機構は組み込まれていない点が我々のものと異なる。

センサと振る舞いモジュールの階層構造からなるロボットアーキテクチャとしては、Brooks のサブサンプションアーキテクチャ[5] も知られる。これは、複数の目標を達成するために、多数のセンサに反応し、頑健かつ漸進的に拡張可能なロボットを容易に設計できる方法論であり、以下の特徴を持つ。

1. タスクを階層的なレイヤに分け、下位のレイヤが構築されれば、それを変更することなく、上位のレイヤを構築できる。

2. 内部処理をほとんど必要としないような単純なセンサと行動の結合によって、環境と実時間での相互作用ができる。

3. 知能は中央集権化されたものではなく、複数の緩やかに結合されたプロセスから知的振る舞いが創発されると考える。

しかし、本論文で扱っている、Lift が荷物を台に載せるタスクのように、多数の副目標の達成や他者とのコミュニケーションを段階的に要するようなプロセス列を、センサとアクチュエータの組み合わせのみで環境との相互作用から創発させることは現実的でなく、そのような応用には我々のもののようにプランを明示的に扱える手法の方が向いていると言えよう。

6.9 形式化

2.2 節でも述べたように BDI モデルの利点の 1 つは、BDI logic[23] という時相論理体系をモデルの構成要素として持ち、これを用いて信念・欲求・意図といった心的状態やその時間的変化を形式的に記述したり論じたりできることである。他のフレームワークでも、要求や仕様の記述時点で形式論理を使えるものはあるが、その範囲にとどまらない一般的な振る舞いの記述を行える形式論理体系を持つものは見あたらない。

例えば 6.7 節で述べた問題において、訂正後の move の性質は BDI logic では

$$\begin{aligned}&\mathsf{BEL}\ \mathit{current\_pos}(x,y) \wedge \mathsf{INT}\ \mathit{move}(x,y,x_1,y_1) \supset \\ &\qquad \mathsf{A}((\forall w,v \neg\, \mathsf{BEL}\ \mathit{current\_pos}(w,v))\ \mathsf{U}\ \mathsf{BEL}\ \mathit{finish\_move}(x,y,x_1,y_1)) \wedge \\ &\qquad\qquad \mathsf{A}(\mathsf{BEL}\ \mathit{current\_pos}(x_1,y_1)\ \mathsf{N}\ \mathsf{BEL}\ \mathit{finish\_move}(x,y,x_1,y_1)) \qquad (6.1)\end{aligned}$$

のように書ける。これは「$move$ を意図するとその終了を信じるまでは現在位置に関する信念はなくなり、終了を信じたときに現在位置の信念が更新される」を表す。一方、プラン (b) の性質 ($calculate\_route$ を行うところまで) は

$$\begin{aligned}&\mathsf{BEL}\ \mathit{identify\_requested}(x,y) \supset\ \mathsf{A}(\phi\ \mathsf{N}\ \exists w,v\ \mathsf{BEL}\ \mathit{current\_pos}(w,v)) \\ &\qquad \text{ただし } \phi = (\mathsf{BEL}\ \mathit{current\_pos}(x_1,y_1) \supset \mathit{calculate\_route}(x_1,y_1,x,y,\mathit{Route})) \qquad (6.2)\end{aligned}$$

と書ける。いま、システムがこの両者を成り立たせるよう動いているとし、$\mathsf{INT}\ \mathit{move}(x,y,x_1,y_1)$ が成り立って以後 $\mathsf{BEL}\ \mathit{finish\_move}(x,y,x_1,y_1)$ が成り立つまでの間に Lift からの依頼が来た ($\mathsf{BEL}\ \mathit{identify\_requested}(x,y)$ が成り立った) としよう。すると、それ以降 $\exists w,v\ \mathsf{BEL}\ \mathit{current\_pos}(w,v)$ が最初に成り立つのは移動が終わった時であり、このとき $calculate\_route$ の引数の x_1, y_1 は移動後の現在位置を反映したものである。すなわち、正しい起点からの $calculate\_route$ が行われる。

一方、不適切な方の move の実装の性質は

$$\begin{aligned}&\mathsf{BEL}\ \mathit{current\_pos}(x,y) \wedge \mathsf{INT}\ \mathit{move}(x,y,x_1,y_1) \supset \\ &\qquad \mathsf{A}(\mathsf{BEL}\ \mathit{current\_pos}(x,y)\ \mathsf{U}\ \mathsf{BEL}\ \mathit{finish\_move}(x,y,x_1,y_1)) \wedge \\ &\qquad\qquad \mathsf{A}(\mathsf{BEL}\ \mathit{current\_pos}(x_1,y_1)\ \mathsf{N}\ \mathsf{BEL}\ \mathit{finish\_move}(x,y,x_1,y_1)) \qquad (6.3)\end{aligned}$$

のように書け、これは「$move$ を意図した場合、その終了を信じるまでは現在位置の信念は更新されず保たれ、終了を信じたときに現在位置の信念が更新される」を意味する。こちらの場合、上記と同じタイミングで Lift から依頼が来ても、その時直ちに $\exists w,v\ \mathsf{BEL}\ \mathit{current\_pos}(w,v)$ が成り立つため、$calculate\_route$ の引数の x_1, y_1 は移動前の現在位置を反映したものであり、すなわち、誤った起点からの $calculate\_route$ が行われてしまう。

しかし、シミュレーション環境では次の式

$$\mathsf{INT}\ \mathit{move}(x,y,x_1,y_1) \supset \mathsf{AX}\,\mathsf{BEL}\ \mathit{finish\_move}(x,y,x_1,y_1) \qquad (6.4)$$

が成り立つことがある。これは「$move$ を意図すると次の時刻には移動は終了している」を意味する。これが成り立つ限り、式 (6.3) の場合でも上のような問題は起こらない。

このように、システムの振る舞い全般に関する議論を形式的に行える手段が最初から用意されている点は、他のフレームワークには見られない長所といえる。

我々は、強化学習やプランニングなど BDI の外部の行為選択機構との結合を扱えるような BDI logic の拡張を提案しており [34, 36]、今後、ロボットの柔軟な行為選択の設計に有用であることが期待できる。

第7章　おわりに

我々はBDIエージェントに動的プランナを取り付けることによって、熟考システムの拡張をおこない、ロボットの行動方針の切り替えを行うことを可能とすることにより、ロボットの行動における誤差による認識の誤りを修正することが可能となった。

また、エージェントの行動決定に対して、プランナが生成するプランのためのゴールの構造として「オンデマンド型サブゴール」を提唱し、プラン生成においてサブゴールを達成する際に例外となるさらに小さなサブゴールに対して動的にそのサブゴールに対してプラン生成ができることにより、従来のゴールに直結したプランニングからさらに柔軟性のあるプラン生成ができることになったことから、より動的環境を配慮したプランニングをおこなえるようにすることが可能となった。

また、BDIロボットが実世界において複数のタスクをスケジューリングしながら行動を行った点で、これからの実世界における有用なロボットとして非常に有効であることも示した。

さらにBDI logicによりこれらの形式化が可能であることを示した。

今後の課題としては、BDIエージェント構築処理系Jasonのアルゴリズムの拡張を活かすため、これらのシステムをすべて統合したBDIエージェントを実装し、実際に実験を通してこのシステムの有用性を実証することである。

また、そのようなシステムをより大規模な問題やより大規模なロボットを用いて実験を行い、有用性を確かめたい。さらにBDI logicによるシステムの形式化から、ロジックからの方面での動作記述の改善を図りたい。

将来展望としては、外部環境をより具体的に認識することが可能なシステムを実装し、これを実際に記号と接地することにより、ロボットの「認識」概念を記号的に扱い、これにより推論が行えるようにすることである。これが可能となれば、現状のプランナには有限のドメインしか扱えない点が、より拡張されたドメインを扱うことが可能となり、より実世界に対応することができる動的なプランナとして実装することが可能となる。このようなプランナを実装した上で、複数のタスクをスケジューリングしながら行動のできるロボット制御を目標としたい。

謝辞

本論文は、著者が奈良女子大学大学院 人間文化研究科 複合現象科学専攻 博士後期課程に在籍中の研究成果をまとめたものです。同大学理学部 情報科学科 加古 富志雄教授には、指導教官として本研究の実施の機会を与えていただき、また、その遂行にあたって終始ご指導をいただき、支えていただきました。ここに深謝の意を表します。同大学院 人間文化研究科 複合現象科学専攻 城 和貴教授、および同大学理学部 情報科学科 山下 靖教授には、副査として本論文の細部にわたりご指導ご助言をいただきました。ここに深謝の意を表します。

近畿大学理工学部 高田 司郎准教授には、論文の執筆過程で数多くの熱心なご指導、ご助言、ご協力をいただきました。ここに深く感謝します。また、奈良女子大学大学院 人間文化研究科 情報科学専攻片山 寛子さんと小島 侑子さんには、実験の過程で多大なるご協力をいただきました。ここに感謝の意を表します。さらに、同大学 理学部 新出 尚之准教授には、三年間多大なご指導をいただいたと共に、精神的にも支えてくださり、感謝しきれないほどの御恩をいただきました。いつも先生の存在が私にとっての安心感であり、たくさんの苦労があっても私をいつも導いてくださったことに言葉にし難い感謝の思いがあり申し訳ないほどです。本当にありがとうございました。

また、物理科学科に在学中にたくさんの質問に答えてくださって、私に研究者の道の選択についても教えてくださった奈良女子大学 理学部 物理科学科 狐崎 創先生、地震の研究を遂行中に振動工学について教えてくださり、人工知能系統の研究室を私に勧めてくださった、大阪府立大学の伊藤 智博先生、新谷 篤彦先生、学会のメンタリングセッションなどで研究に対して数々の御指摘くださり、その後も就職のことや研究の方針について Skype で対応してくださった電気通信大学の大須賀 昭彦先生、東芝のインターン中で大変な中たくさんの面倒を見てくださった東芝の長 健太さん、同じく東芝で精神的に支えてくださった東芝の酒井 政裕さん、ロボットのビジョンの話を教えてくださり、共同研究も提案してくださった東芝の田崎 豪さん、ロボットの研究について相談にのってくださり、学術振興会の件でもたくさん御世話になった奈良先端科学技術大学院大学の小笠原 司先生、論文のことや将来のことをいつも教えてくださった大阪大学の和田 昌昭先生、学会のメンタリングセッションなどで将来の職業について私のことを綿密に分析してくださり、御指摘くださった日本 IBM の山本 学さん、ポスドクのことや国際会議のことなど多くのアドバイスをくださった国立情報学研究所の佐藤 健先生、皆様のおかげで私は研究者として大きな視野を持つことができ、数々の研究を行うことができたことを深く感謝いたします。

また、公私にかかわらず何かとお世話になりながら、ここにお名前を記すことのできなかった数多くの方々にも感謝の意を表します。

最後に、昨年学会で出会った京都大学 数理解析研究所の後藤 勇樹君には、本論文作成の際に多くのアドバイスと精神的な支えになっていただいたことに深く感謝します。これからの共同研究において、この論文が原点となり、共に多くの研究をおこなって発見の楽しさを分かち合えれば望外の喜びです。本当にありがとうございました。

参考文献

[1] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[2] R. Peter Bonasso, David Kortenkamp, David P. Miller, and Marc Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, pp. 237–256, 1995.

[3] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[4] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987. (門脇俊介, 高橋久一郎 (訳). 意図と行為—合理性、計画、実践的推論—, 産業図書, 1994).

[5] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14–23, 1986.

[6] Kenta Cho, Naoki Iketani, Masaaki Kikuchi, Keisuke Nishimura, Hisashi Hayashi, and Masanori Hattori. BDI model-based crowd simulation. In Helmut Prendinger, James Lester, and Mitsuru Ishizuka, editors, *Intelligent Virtual Agents*, Vol. 5208 of *Lecture Notes in Computer Science*, pp. 364–371, 2008.

[7] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, Vol. 20, No. 1-2, pp. 3–50, 1993.

[8] Lavindra de Silva, Anthony Dekker, and James Harland. Planning with Time Limits in BDI Agent Programming Languages. *ACM International Conference Proceeding Series*, Vol. 240, pp. 131–139, 2007.

[9] Lavindra de Silva and Lin Padgham. Planning on demand in BDI systems. *ICAPS-05*, pp. 37–40, 2005.

[10] E. Allen Emerson and J. Srinivasan. Branching Time Temporal Logic. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pp. 123–172. Springer-Verlag, 1989.

[11] Claudia Frischknecht and Thomas Other. LEGO Mindstorms NXT: Welcome to the NeXT generation. `http://www.tik.ee.ethz.ch/tik/education/lectures/PPS/mindstorms/sa_nxt/index.php?page=print`, 2006.

[12] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc., 2004.

[13] Malik Ghallab, Dana Nau, and Paolo Traverso. *AUTOMATED PLANNING*. MORGAN KAUFMANN, 2004.

[14] Hiroshi ISHIGURO. Studies on humanoids. *Journal of the Japan Society for Precision Engineering*, Vol. 76, No. 1, pp. 20–23, 2010.

[15] Lars Kock Jensen, Bent Bruun Kristensen, and Yves Demazeau. Flip: Prototyping multi-robot systems. *Robotics and Autonomous Systems*, Vol. 53, No. 3-4, pp. 230–243, 2005.

[16] Jeff Kramer and Jeff Magee. Self-managed systems: an architectual challenge. In *Proc. of FOSE '07*, pp. 259–268, 2007.

[17] Douglas P Lau. Nxt_python. `http://home.comcast.net/~dplau/nxt_python/index.html`, 2007.

[18] Anders Lemke, Johan Laidlaw, and Lars Zilmer-Pedersen. Developing multi-agent lego robotics. `http://www.imm.dtu.dk/upload/institutter/imm/cse%20laboratories/lemke_ea.pdf`, 2007.

[19] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, Vol. 20, pp. 379–404, 2003.

[20] Dave Parker. nxtprograms.com. `http://www.nxtprograms.com/index.html`.

[21] Rolf Pfeifer and Josh C. Bongard. *How the Body Shapes the Way We Think*. The MIT Press, 2006.

[22] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proc. of MAAMAW-96*, Vol. 1038 of *LNAI*, pp. 42–55. Springer-Verlag, 1996.

[23] Anand S. Rao and Michael P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Proc, of International Conference on Principles of Knowlegde Representation and Reasoning*, pp. 473–484, 1991.

[24] Anand S. Rao and Michael P. Georgeff. Decision Procedures for BDI Logics. *Journal of Logic and Computation*, Vol. 8, No. 3, pp. 292–343, 1998.

[25] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002. (古川康一 (監訳). エージェントアプローチ 人工知能, 共立出版, 2008).

[26] Sebastian Sardina, Lavindra de Silva, and Lin Padgham. Hierachical Planning in BDI Agent Programming Languages: A Formal Approach. *The Fifth International Joint Conference on Autonomous Agents and Multiagent Systems 2006*, pp. 1001–1008, 2006.

[27] Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff. Formal method in DAI: Logic-Based Representation and Reasoning. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pp. 331–376. The MIT Press, 1999.

[28] Telecom Italia. JADE: Java Agent Development Framework. `http://jade.tilab.com`, 2010.

[29] Masanao Toda. *Man, robot, and society: models and speculations*. M. Nijhoff Pub., 1982.

[30] A. Walczak, L. Braubach, A. Pokahr, and W. Lamesdorf. Augmenting BDI Agents with Deliberative Planning Techniques. *The 5th International Workshop on Programming Multiagent Systems (PROMAS-2006)*, pp. 113–127, 2006.

[31] 坂上義秋. 生活を楽にするロボット : 役に立つロボットの出現はいつ?(テーマ関連/オーガナイズドセッション (3)). 情報処理学会研究報告. CVIM, [コンピュータビジョンとイメージメディア], Vol. 2004, No. 113, pp. 61–68, 2004-11-11.

[32] 林久志. 動的環境における階層型マルチエージェントプランニング. エージェント合同シンポジウム (JAWS2006) 講演論文集, 2006.

[33] 高田司郎, 新出尚之, 藤田恵. 拡張 BDI 論理 tomatoes を用いた強化学習のモデル化について. 人工知能学会論文誌, Vol. 26, No. 1, pp. 156–165, 2011.

[34] 新出尚之, 藤田恵, 高田司郎. 動的プランナと BDI エージェントの結合の形式化について. In *Proc. of JAWS2010*, 2010. `T-1.pdf`.

[35] 新出尚之, 高田司郎, 藤田恵. 拡張 BDI 論理 tomatoes による協調行為のモデル化と応用. 人工知能学会論文誌, Vol. 26, No. 1, pp. 13–24, 2011.

[36] 新出尚之, 高田司郎, 藤田恵. 拡張 BDI 論理 tomato を用いた確率的状態遷移のモデル化とその応用. 情報処理学会論文誌 数理モデル化と応用, Vol. 4, No. 3, pp. 59–72, 2011.

[37] 中岡慎一郎, 三浦郁奈子, 森澤光晴, 金広文男, 金子健二, 梶田秀司, 横井一仁. ヒューマノイドロボットのコンテンツ技術化に向けて — クリエイターによる多様な表現の創出が可能な二足歩行ヒューマノイドロボットの実現 —. *Synthesiology*, Vol. 4, No. 2, pp. 80–91, 2011.

[38] 中川博之, 大須賀昭彦, 本位田真一. ゴール指向要求分析を用いた self-adaptive システムの構築. 情報処理学会論文誌, Vol. 50, No. 10, pp. 2500–2513, 2009.

[39] 中川博之, 大須賀昭彦, 本位田真一. ビヘイビア記述に基づく自己適応システム実装 フレームワークの提案. 人工知能学会論文誌, Vol. 26, No. 1, pp. 1–12, 2011.

[40] 五十棲隆勝, 赤地一彦, 平田勝, 金子健二, 梶田秀司, 比留川博久. ヒューマノイドロボット HRP-2 の開発. 日本ロボット学会誌 = Journal of Robotics Society of Japan, Vol. 22, No. 8, pp. 1004–1012, 2004-11-15.