アクション言語を用いた状態変化の表現と推論に関する研究

鍋島，英知

(Degree)
博士（工学）
(Date of Degree)
2001-03-31
(Date of Publication)
2008-04-09
(Resource Type)
doctoral thesis
(Report Number)
甲2239
(URL)
https://hdl.handle.net/20.500.14094/D1002239

博士論文

アクション言語を用いた状態変化の表現と推論に関する研究

平成 13 年 1 月

神戸大学大学院自然科学研究科

鍋島 英知

論文要旨

状態変化やアクションを記述するための知識表現に関する研究は，人工知能研究の初期の段階から議論されている分野の 1 つである．この分野において，最近活発に研究されているアプローチの 1 つが高級レベルのアクション言語である．アクション言語とは，自然言語表現を用いてアクションによる状態の変化を宣言的に記述する形式的モデルといえる．そのようなアクション言語の中で最初に提案されたのが，Gelfond と Lifschitz [9] による**言語** $\mathcal{A}$ である．言語 $\mathcal{A}$ では，決定性効果を持つアクションのみを記述することができる．以来，$\mathcal{A}$ を基として，$\mathcal{A}$ を拡張した様々なアクション言語が提案されている [10].

これらのアクション言語は，言語 $\mathcal{A}$ を拡張し，さらなる表現力を求めて提案されたものであるが，言語 $\mathcal{A}$ に比べ客観的にどの程度表現力が向上したのか分かっていない．記述可能な問題領域が広がったのかどうか，もし広がったのであれば何が記述可能になったのか，などの言語の表現力が明らかになっていない．また，これらの言語の多くは，その実現のための手法として実際には，拡張論理プログラムや一階述語論理，その他の非単調な枠組に変換している．これらは汎用の表現言語であるため，効率の良い推論システムを構築することは難しい.

そこで我々は，(i) アクション言語の表現能力の形式的解析手段を提供し，今後のアクション言語設計のための指標を与えること，(ii) その解析結果に基づいてより表現力豊かなアクション言語を提案すること，(iii) アクション言語のための効率の良い推論アルゴリズムを提案すること，そして，現実の問題では問題領域に対する完全な知識を記述することは難しく，むしろその領域における観測結果を記述することのほうが容易であることが多いという観点から，(iv) 環境から因果関係の知識を自動獲得する学習アルゴリズムを提案することを目的として研究を行った.

まず我々は**有限オートマトン** (FA) に着目し，言語 $\mathcal{A}$ で表現可能な領域のクラスと，有限オートマトンで表現できる言語のクラスとが等価であることを示した．言語 $\mathcal{A}$ と FA の等価性の結果は，形式言語論の観点からアクション言語の特性を解析するための基盤を提供する．言語 $\mathcal{A}$ におけるプランニング問題は，オートマトンの受理言語を求める問題に

帰着し，プランニングの解は，正則表現を用いることで簡潔かつ完全に表すことができる．

次に我々は，$\mathcal{A}$ と FA との等価性の結果に基づき，**非決定性有限オートマトン** (NFA) の観点から新しい**非決定性アクション言語** $\mathcal{NA}$ を提案した．言語 $\mathcal{NA}$ は，NFA と同じ表現力をもっており，$\mathcal{NA}$ と他のアクション言語との等価性を示すことで，その言語の適用範囲を形式的に議論できるようになる．実際に我々は，言語 $\mathcal{AR}$ [11] による記述を，それと等価な $\mathcal{NA}$ による記述に変換する手続きを与え，$\mathcal{NA}$ と $\mathcal{AR}$ が同じ表現能力を持つことを示す．

また，アクション言語のための効率的な推論アルゴリズムを提供するため，最近プランニング研究の分野において高速なプランニングアプローチとして注目されている SAT プランニングと呼ばれる手法に着目した．我々は，この SAT プランニングの技術が，アクション言語におけるプランニングだけでなく，モデル生成（初期状態の推定）に対しても適用できることを示し，アクション言語処理系 AMP を開発した．AMP を用いることで，実際に問題領域をアクション言語により記述し，プランニングやモデル生成，実行結果の予測などの各種推論を高速に行うことが可能になる．

次に我々は，因果関係の知識を自動獲得する学習アルゴリズムを提案するため，決定木を用いた因果関係の学習アルゴリズムを提案した．一般的に決定木は，ある命題が真であるか偽であるかという 2 値を出力するが，アクション言語においては，アクションの実行によりフルーエントが真になる，偽になるに加え，アクションが実行できない場合がある．そこで我々は，決定木が真・偽・実行不可能の 3 値を出力するように拡張し，そのような決定木を学習するアルゴリズムを提案した．この学習アルゴリズムを用いることで，不完全な知識しか持たない環境下においても，その環境における事象の観測結果から因果関係の知識を推定することができるようになる．

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景

本論文の主題は，**状態変化の表現と推論** (reasoning about action) である．これは，刻々と変化する現実の世界をどのようにモデル化し，その上でどのようにして推論を行うかという研究分野であり，人工知能研究の初期の段階から議論されている研究分野の１つである．状態変化の例として，コーヒーをいれる場合を考える．我々は，まずやかんに水を満たし，コンロにかけ火をつけてお湯を準備する．その一方で，コーヒーカップを取り出し，インスタントコーヒーであればカップにコーヒー粉を入れ，沸いたお湯を注ぐ．この例では，“やかんが空の状態” から “水が入った状態” へ変化し，そして “お湯になった状態” へと変化していく．本研究分野の目的は，このような状態の変化を上手く定式化するにはどうすれば良いのか，その上で効率よく推論するためにはどうすればよいのかを研究することにある．このことは，人間が普段の生活においてなにげなしに行っている常識的な推論を実現するためにも必要となる．

状態変化の表現と推論のために，これまでにも一階述語論理などの汎用の表現言語を用いた手法が研究されている．例えば**状況計算** (situation calculus) [26] は，一階述語論理を用いて状態の変化を記述する手法である．このため，プランニングなどの推論に，一階述語論理における各種の推論手続きを利用することができる．

常識的な推論では，しばしば例外やデフォルトといった概念が必要とされる．例えば，コーヒーカップにお湯を注げば，通常カップはお湯で満たされるが，例外的に，もしやかんの注ぎ口がごみで詰まっていたり，カップがひび割れていた場合，お湯を満たすことはできない．この種の推論は，新しい事実が知識に加えられると以前の推論結果が取り消されることがあるため，**非単調な** (nonmonotonic) 推論と呼ばれる．一方，一階述語論理は厳密な単調性を示す．そこで，**デフォルト論理** (default logic) [36] や**非単調論理** (nonmonotonic logic) [27] や**極小限定** (circumscription) [24] のような非単調な枠組みを用いて状況計算を拡張する手法も研究されている．

しかし，これらの手法では汎用の表現言語を用いているために，言語に精通していない初心者には記述しにくく，記述しなければならない知識の記述量も多い．また，汎用であるがために，効率の良い推論システムを構築することも難しい．

そこで，最近活発に研究されているのが高級レベルのアクション言語と呼ばれるものである．アクション言語とは，自然言語表現を用いてアクションによる状態の変化を宣言的に記述する形式的モデルといえる．構文に自然言語表現を用いているため，抽象度が高く，柔軟な記述が可能になる．本論文が対象とするのは，このアクション言語である．

アクション言語の特徴の１つとして，慣性の法則を言語の意味論によりサポートしていることが挙げられる．例えばコーヒー粉が入ったカップに“お湯を注ぐ”というアクションは，“カップがコーヒーで満たされる”という効果をもっている．お湯を注いでいる間に，カップが空を飛んだり，コーヒー粉がなくなったりはしない．人間は常識としてこのことを知っているが，このような問題を定式化する際には，“人間が常識として知っていること”も与えてやる必要がある．

アクションにより変化しない側面を特徴付ける問題は**フレーム問題** (frame problem) と呼ばれている．先に示した状況計算などの枠組みでは，何らかの方法で，アクション実行後も変化しない性質を指示する**フレーム公理** (frame axiom) を書く必要がある．もし書かなければ，カップにお湯を注いだ後，カップがどこにあるのか，またコーヒー粉が存在するかどうか推論することができない．

一方，アクション言語では，アクションが影響を及ぼさない性質は，**慣性の法則** により，アクション実行後も成立する．従ってフレーム公理を記述する必要がなく，知識の記述量を大幅に抑えることができ，問題領域の本質的な知識を記述することに注力することができる．

1.2 研究の目的

アクション言語は現在数多く提案されているが，その中でも最初に提案され，また最も基本的とされているのが，Gelfond と Lifschitz [9] による**言語** $\mathcal{A}$ である．言語 $\mathcal{A}$ では，決定性効果を持つアクションのみを記述することができる．以来，$\mathcal{A}$ を基として，$\mathcal{A}$ を拡張した様々なアクション言語が提案されている：非決定性効果をもつアクション [5,6,10,11,22,39]，同時発生アクション [3,5,10]，フルーエント間の制約 [11,17,22,39]，フルーエント間の依存関係 [10,12,22,39] などを表現できる言語が存在する．ここで**フルーエント** (fluent) と

は，状態に依存する属性のことで，例えばやかんのお湯が沸いているかどうか，などの性質を表す．

これらのアクション言語は，言語 $\mathcal{A}$ を拡張し，さらなる表現力を求めて提案されたものである．例えば，非決定性効果をもつアクションにより，ロボットなどのセンサーを定式化することができるようになる（センシングの結果は，障害物が検出される，もしくは検出されない，である）．また，フルーエント間の制約を用いることで，アクションの間接的な効果（カップにお湯を注げば，カップも温かくなるなど）も記述できるようになる．

しかしこれらのアクション言語が，言語 $\mathcal{A}$ に比べ客観的にどの程度表現力が向上したのかは分かっていない．記述可能な問題領域が広がったのかどうか，もし広がったのであれば何が記述可能になったのか，などの言語の表現力が明らかになっていない．また，これらの言語の多くは，その実現のための手法として実際には，拡張論理プログラム [2, 9] や一階述語論理，その他の非単調な枠組 [16, 17] に変換している．これらは汎用の表現言語であるため，効率の良い推論システムを構築することは難しい．

我々の研究の目的はここにある．第 1 に，アクション言語の表現能力の形式的解析手段を提供し，今後のアクション言語設計のための指標を与える．第 2 に，その解析結果に基づいてより表現力豊かなアクション言語を提案する．第 3 に，アクション言語処理系のための効率の良い推論アルゴリズムを考案し，定量的な評価を行える環境を実際に提供する．第 4 として，現実の問題では問題領域に対する完全な知識を記述することは難しく，むしろその領域における観測結果を記述することのほうが容易であることが多いという観点に立ち，問題領域から知識を自動的に獲得する機械学習のアルゴリズムを提案する．

1.3 研究の成果

アクション言語の表現能力の形式的な解析手段を提供するために，我々は**有限オートマトン** (FA) に着目し，言語 $\mathcal{A}$ で表現可能な領域のクラスと，有限オートマトンで表現できる言語のクラスとが等価であることを示した [28]．この等価性の証明のために，$\mathcal{A}$ による領域記述と FA とを相互に変換する 2 つのアルゴリズム，(i) FA を $\mathcal{A}$ に変換するアルゴリズム，(ii) $\mathcal{A}$ を FA に逆変換するアルゴリズムを定義した．前者のアルゴリズムは，$\mathcal{A}$ から FA への変換を行なうだけでなく，$\mathcal{A}$ のモデルを計算する手続きにもなっている．また後者のアルゴリズムも，FA を $\mathcal{A}$ における命題に逆変換するだけでなく，慣性の概念を用いて $\mathcal{A}$ による任意の領域記述を簡単化する機能をもつ．

言語 $\mathcal{A}$ と FA の等価性の結果は，形式言語論からアクション言語の特性を解析するための基盤を提供する．例えば，前者のアルゴリズムは，プランニング問題のすべての可能な解を**正則表現**により表すことを可能にする．後者のアルゴリズムは，アクション理論が有限オートマトンの様々な応用分野に適用することを可能にする．特に，言語 $\mathcal{A}$ の適用範囲をより形式的に議論できるようになる．

次に我々は，非決定性効果を持つアクションを記述できる言語（**非決定性アクション言語**）を対象とした．これまでにもいくつもの非決定性アクション言語が提案されているが，その非決定性の扱いは各言語ごとに異なっている．我々は，言語 $\mathcal{A}$ と FA の等価性の結果に基づき，**非決定性有限オートマトン** (NFA) の観点から新しい非決定性アクション言語 $\mathcal{NA}$ を提案した [31]．言語 $\mathcal{NA}$ における非決定性は，NFA に基づいた簡潔な定義となっている．

言語 $\mathcal{NA}$ は FA と同じ表現能力をもっているため，$\mathcal{NA}$ と他の非決定性アクション言語との等価性を示すことで，その言語の適用範囲が明らかになる．実際に我々は，極小変化の概念に基づく非決定性をもつ言語 $\mathcal{AR}$ [11] による記述を，それと等価な $\mathcal{NA}$ による記述に変換する手続きを与え，$\mathcal{NA}$ と $\mathcal{AR}$ が同じ表現能力を持つことを示した．

またアクション言語処理系のための効率の良い推論アルゴリズムを考案するために，我々は高速なプランニングアルゴリズムの 1 つとして最近注目されている**プランニンググラフ** [4] と呼ばれるデータ構造を用いた手法に着目した．それは，プラン探索空間をいったんプランニンググラフに符号化した後，充足可能性問題 (SAT) に変換し，高速な SAT ソルバにより解くという手法である．我々は，このプランニンググラフと SAT ソルバによるプラン抽出法を基に，**アクション言語処理系** AMP を実装した [32]．

アクション言語処理系 AMP を用いることで，実際に問題領域をアクション言語により記述し，プランニングやモデル生成（初期状態の推定），実行結果の予測などの各種推論を高速に行うことが可能になる．特に我々は，プランニンググラフと SAT ソルバによる手法が，プランニングだけでなく，モデル生成についても有効であることを示した．

現実の問題では，問題領域に対する完全な知識を記述することは難しく，むしろその領域における観測結果を記述することのほうが容易であることが多い．我々はこの観点から因果関係の知識を学習するアルゴリズムを提案した [30]．学習アルゴリズムへの入力は，事象の観測例であり，出力は，言語 $\mathcal{A}$ による因果関係の記述 — あるアクションを実行すると，それがどのような影響を及ぼすかという記述である．言語 $\mathcal{A}$ は命題的フルーエントのみを扱うので，学習アルゴリズムとして，決定木を学習するアルゴリズム [35] を採用

した．この学習アルゴリズムを用いることで，不完全な知識しか持たない環境下においても，その環境における事象の観測結果から因果関係の知識を推定することができる．

1.4 本論文の構成

本論文の構成は以下のようになっている．

まず第 2 章では，本論文に関連する研究を紹介する．まず最も基本的なアクション言語 $\mathcal{A}$ を紹介する．本研究の多くは，この言語 $\mathcal{A}$ を基礎としている．次に，非決定性アクション言語の 1 つである言語 $\mathcal{AR}$ を紹介する．言語 $\mathcal{AR}$ は，我々が提案する言語 $\mathcal{NA}$ と深い関係がある．また，最近大きな進展を見せているプランニングアルゴリズムの研究についても紹介する．そこでは，最速の**プラナ** (planner) の 1 つである `Blackbox` [19] のアルゴリズムを示す．我々が提案するアクション言語処理系 AMP の推論アルゴリズムは，このアプローチを基にしている．

第 3 章では，$\mathcal{A}$ の表現能力の形式的解析を行い，$\mathcal{A}$ と FA との等価性を示す．このために，$\mathcal{A}$ による記述と FA とを相互に変換する 2 つのアルゴリズムを定義し，その正当性を示す．そしてこの等価性の結果が，アクション言語とオートマトン理論にとってどのような利点をもたらすのかについて述べる．

第 4 章では，$\mathcal{A}$ と FA との等価性の結果に基き，非決定性有限オートマトン (NFA) の観点から新しい非決定性アクション言語 $\mathcal{NA}$ を提案し，これまでに提案されている非決定性アクション言語の表現能力を形式的に解析する手段を提供する．実際に我々は，言語 $\mathcal{AR}$ と言語 $\mathcal{NA}$ との等価性を証明する．

第 5 章では，SAT プランニングアプローチに基づくアクション言語処理系 AMP を紹介する．ここでは，アクション言語におけるプランニングアルゴリズムとモデル生成アルゴリズムを示し，その正当性を証明する．そして，SAT プランニングアプローチが，プランニングだけでなく，モデル生成に対しても有効であることを示す．

第 6 章では，言語 $\mathcal{A}$ における因果関係の学習アルゴリズムを提案する．我々が対象とする学習問題を定義し，それを解くための学習アルゴリズムを示す．

第 7 章では，まず，本論文の研究内容の成果をまとめる．そして，アクション言語の研究，オートマトン理論の研究，プランニング研究の各分野における本論文の位置付けを明確にする．最後に，今後の課題についてまとめる．

第2章　関連研究

本章では，本論文に関係する研究を紹介する．まず最も基本的なアクション言語 $\mathcal{A}$ を紹介する．本研究の多くは，この言語 $\mathcal{A}$ を基礎としている．次に，非決定性アクション言語の 1 つである言語 $\mathcal{AR}$ を紹介する．言語 $\mathcal{AR}$ は，第 4 章で我々が提案する言語 $\mathcal{NA}$ と深い関係がある．最後に，最近大きな進展を見せているプランニングアルゴリズムの研究について紹介し，現在最速の**プラナ** (planner) の 1 つである `Blackbox` [19] のアルゴリズムを示す．我々が提案するアクション言語処理系 AMP の推論アルゴリズムは，このアプローチを基にしている．

2.1　アクション言語 $\mathcal{A}$

状態の変化とアクションを表現する問題は，人工知能研究の初期の段階から議論されている問題の 1 つである．その最新の研究動向の中で，**アクション言語**の研究が急速に進んでいる．アクション言語とは，自然言語表現を用いてアクションによる状態の変化を宣言的に記述する形式的モデルといえる．そのようなアクション言語の中で最初に提案されたのが，本章で紹介する Gelfond と Lifschitz [9] による**言語 $\mathcal{A}$** である．

この言語 $\mathcal{A}$ の生い立ちについて簡単に説明する．言語 $\mathcal{A}$ はもともと，非単調推論の研究において “Yale Shooting 問題” とよばれる伝統的な例題を，簡単な構文論と意味論により解くことができる言語として提案された．この Yale Shooting 問題とは，非単調な推論を行うための手法として提案されていた**デフォルト論理** (default logic) [36] や**非単調論理** (nonmonotonic logic) [27] や**極小限定** (circumscription) [24] のような枠組みにおいて，我々の常識による推論の結果と一致するような知識表現ができないことを示した例題である [13]．この例題は，非単調推論を行う枠組みの妥当性の指標として現在でも利用されている．

言語 $\mathcal{A}$ は，その簡単な構文論・意味論にもかかわらず，Yale Shooting 問題を，自然言語表現を用いて直感的にも分かりやすく自然に記述することができる．本章では，この言

語 $\mathcal{A}$ の構文論と意味論を紹介する．

2.1.1 構文論

アクション言語 $\mathcal{A}$ [1]を 4 つ組 $\langle \mathbb{A}, \mathbb{F}, \mathbb{E}, \mathbb{V} \rangle$ で表す．ここで $\mathbb{A}$ は**アクション名**の集合，$\mathbb{F}$ は**フルーエント名**の集合，$\mathbb{E}$ は**効果命題** (effect proposition) の集合，$\mathbb{V}$ は**評価命題** (value proposition) の集合である．

フルーエント (fluent) とは，状態に依存する属性のことで，状態の変化とともにフルーエントのとる値も変化する．言語 $\mathcal{A}$ において，フルーエントは真または偽の 2 値をとる．フルーエント名 F の否定 $\neg F$ を**負のフルーエント**といい，F を**正のフルーエント**という．フルーエントを小文字の f で表し，フルーエント名を大文字の F で表す．

効果命題は，アクションとその効果の因果関係を記述する：

$$a \textbf{ causes } \phi \textbf{ if } \psi \tag{2.1}$$

ここで a はアクション名，ϕ はフルーエントの連言，ψ はフルーエントを論理演算子 $(\vee, \wedge, \neg, \supset, \equiv)$ で結合した式である．ϕ をアクションの**効果**と呼び，ψ をアクションの**前提条件**と呼ぶ．

評価命題は，観測した事実を記述する：

$$\phi \textbf{ after } a_1; \ldots; a_m \quad (m \geq 0) \tag{2.2}$$

ここで ϕ はフルーエントの連言，a_i はアクション名である．

いくつか有用な省略形を定義する．ここで $true$ は，恒真を意味する特別なフルーエントで，全ての状態で真となる．$false = \neg true$ である．効果命題 (2.1) に対し，前提条件 ψ が $true$ ならば，**if** 以下を省略する：

$$a \textbf{ causes } \phi$$

評価命題 (2.2) に対し，$m = 0$ であるならば，簡単に次のように記述する：

$$\textbf{initially } \phi \tag{2.3}$$

[1]本論文で定義する言語 $\mathcal{A}$ は，文献 [9] による定義に若干拡張を加えたものであるが，表現能力においては等価である．

2.1.2 意味論

言語 $\mathcal{A}$ による**領域記述**は，効果命題と評価命題の集合である．領域記述 D に含まれるすべてのアクション名の集合を $action(D)$ で表し，すべてのフルーエント名の集合を $\mathit{fluent}(D)$ で表す．**状態** σ を，全てのフルーエント名について，正または負のフルーエントのいずれかを含む集合と定義する：

$$\sigma = S \cup \{\neg F \mid F \in \mathit{fluent}(D) \setminus S\}.$$

ここで $S \subseteq \mathit{fluent}(D)$ である．任意の状態 σ と，フルーエント f のみからなる式 $\phi = f$ に対し，$f \in \sigma$ であるとき，かつそのときに限り，**状態** σ **で式** ϕ **が真**であると定義する．任意の式については，論理演算子の真偽値表に従って拡張する．

解釈 I は，σ_0 を初期状態，Φ を遷移関数とすると，その対 (Φ, σ_0) である．**遷移関数** Φ は，アクション名 a と状態 σ の対 (a, σ) を状態へ写像する．任意の解釈 I と，任意のアクション列 $a_1; \ldots; a_m$ に対して，$I^{a_1;\ldots;a_m}$ は次の状態を与える：

$$I^{a_1;\ldots;a_m} = (\Phi, \sigma_0)^{a_1;\ldots;a_m} = \Phi(a_m, \Phi(a_{m-1}, \ldots, \Phi(a_1, \sigma_0)\ldots)).$$

評価命題 (2.2) に対し，状態 $I^{a_1;\ldots;a_m}$ において ϕ が真であれば，評価命題 (2.2) が解釈 I において**真**であるという．

モデルを定義する前に，補助的な演算を定義する．任意のフルーエント名 F に対し，正のフルーエントを $|F|^+$ で表し，負のフルーエントを $|F|^-$ で表す．すなわち，

$$|F|^+ = |\neg F|^+ = F, \quad |F|^- = |\neg F|^- = \neg F$$

フルーエントの集合についても，同様に定義する：

$$\begin{aligned} |\{f_1, \cdots, f_n\}|^+ &= \{|f_1|^+, \cdots, |f_n|^+\}, \\ |\{f_1, \cdots, f_n\}|^- &= \{|f_1|^-, \cdots, |f_n|^-\} \end{aligned}$$

効果命題 $P = (a \textbf{ causes } f_1 \wedge \cdots \wedge f_n \textbf{ if } \psi)$ に対し，次の関数 effect を定義する：

$$\mathit{effect}(P) = \{f_1, \cdots, f_n\}$$

さらに，効果命題の集合 $\{P_1, \cdots, P_m\}$ に対して拡張する：

$$\mathit{effect}(\{P_1, \cdots, P_m\}) = \mathit{effect}(P_1) \cup \cdots \cup \mathit{effect}(P_m)$$

ここで $\mathit{effect}(\emptyset) = \emptyset$ とする．

解釈 I が領域記述 D の**モデル**であるのは，(i) D に含まれるすべての評価命題が I において真であり，(ii) すべてのアクション名 a とすべての状態 σ に対し，遷移関数が次式で定義される場合である：

$$\Phi(a, \sigma) = (\sigma \setminus (|e|^+ \cup |e|^-)) \cup e \tag{2.4}$$

ここで $e = \mathit{effet}(\mathcal{P})$，$\mathcal{P}$ は次式で与えられる：

$$\mathcal{P} = \{P \in D \mid P = (a \textbf{ causes } \phi \textbf{ if } \psi) \text{ かつ，状態 } \sigma \text{ で } \psi \text{ が真 }\}.$$

式 (2.4) は，アクション a の実行により影響を受けないフルーエントの値は，次の状態でも維持される定義となっている．これは**慣性の法則**を表している．式 (2.4) を満たす遷移関数は，多くとも 1 つしか存在しない．よって，同じ領域記述における異なったモデルは，そのモデルの初期状態によってのみ異なる．ある評価命題が領域記述 D のモデル M で真であるならば，

$$M \models (\phi \textbf{ after } a_1; \ldots; a_m).$$

と記述する．任意のモデルにおいて真であるならば，

$$D \models (\phi \textbf{ after } a_1; \ldots; a_m).$$

と記述する．

2.1.3 記述例

次に示すのは，Yale Shooting 問題 [13] を言語 $\mathcal{A}$ により記述した例である．初期状態で七面鳥は生きており (alive)，銃に弾は込められていない ($\neg \mathit{loaded}$)．銃に弾を込める (load) という動作が起こると銃に弾が込められた状態 (loaded) になる．銃に弾が込められた状態 (loaded) で銃を撃つと (shoot) その七面鳥は死ぬ ($\neg \mathit{alive}$)．

例題 1 Yale Shooting 領域

initially $\neg \mathit{loaded}$.

initially alive.

load **causes** loaded.

shoot **causes** $\neg \mathit{alive} \wedge \neg \mathit{loaded}$ **if** loaded.

この領域記述を D とする．D には，アクション名として，*load* と *shoot* の他に，何も影響を及ぼさないアクション *wait* を含むものとする．このとき，次の式が成立する：

$D \models \neg alive$ **after** $load; wait; shoot$.

次に示すのは，先の Yale Shooting 問題を少し変更した Murder Mystery 問題 [1] である．この問題では，初期状態で銃に弾が込められているかどうか分からないが，銃を撃った後 (shoot)，七面鳥が死んだ ($\neg alive$) ということが分かっている．

例題 2 Murder Mystery 領域

initially *alive*.

load **causes** *loaded*.

shoot **causes** $\neg alive \wedge \neg loaded$ **if** *loaded*.

$\neg alive$ **after** *shoot;wait*.

この領域記述を D' とする．D' は，初期状態で銃に弾が込められていた (loaded) ということを帰結する：

$D \models$ **initially** *loaded*.

言語 $\mathcal{A}$ では，このように過去について推論することもできる．

2.2 アクション言語 $\mathcal{AR}$

アクション言語 $\mathcal{AR}$ [11] は，$\mathcal{A}$ を基にして拡張された言語であり，非決定的効果をもつアクションや，間接的効果をもつアクション，状態に対する制約などを記述することができる．また言語 $\mathcal{AR}$ では，慣性に従うフルーエントと従わないフルーエント（**非慣性フルーエント**）の 2 種類を扱う．フルーエントは，真偽の 2 値だけでなく，複数の値をとることができる．

2.2.1 構文論

アクション言語 $\mathcal{AR}$ を 5 つ組 $\langle \mathbb{A}, \mathbb{F}, \mathbb{E}, \mathbb{C}, \mathbb{V} \rangle$ で表す．ここで $\mathbb{A}$ は**アクション名**の集合，$\mathbb{F}$ は**フルーエント名**の集合，$\mathbb{E}$ は**効果命題** (effect proposition) の集合，$\mathbb{C}$ は**制約** (constraint) の集合，$\mathbb{V}$ は**評価命題** (value proposition) の集合である．

言語 $\mathcal{AR}$ では，フルーエント名の集合 $\mathbb{F}$ は，慣性フルーエントの集合 $\mathbb{F}_I$ と非慣性フルーエントの集合 $\mathbb{F}_N$ の 2 つに分かれている．非慣性フルーエントの利点は，あるフルーエントが慣性に従うかどうかを，ユーザが状況に応じて自由に決定できるという点である [23]．フルーエント名 F に対し，その値域を集合 Rng_F で表す．もし F が命題的であるならば，$Rng_F = \{true, false\}$ である．フルーエント名 F が値 $V \in Rng_F$ を持つことを $(F \textbf{ is } V)$ と記述し，これを**原子式** と呼ぶ．原子式を論理演算子で結合したものを**式**と呼ぶ.

言語 $\mathcal{AR}$ では次の 4 種類の命題を記述できる：

$$a \textbf{ causes } \phi \textbf{ if } \psi \tag{2.5}$$

$$\textbf{always } \phi \tag{2.6}$$

$$a \textbf{ possibly changes } F \textbf{ if } \psi \tag{2.7}$$

$$\phi \textbf{ after } a_1; \ldots; a_n \tag{2.8}$$

ここで $a, a_1, \cdots, a_n$ はアクション名，ϕ, ψ は式，F はフルーエント名である．効果命題 (2.5) と評価命題 (2.8) は，ϕ が選言 ($\vee$) を含む一般的な論理式である点が，言語 $\mathcal{A}$ と異なる．命題 (2.6) は，状態に対する**制約**を表しており，あらゆる状態において，式 ϕ が常に成立することを言明する．命題 (2.7) は，前提条件 ψ の下でアクション a を実行すると，フルーエントの値が変化するかも知れないことを言明する．この命題が，アクションの**非決定性効果**を生む.

効果命題 (2.5) に対し，効果 ϕ が $false$ であるならば,

$$\textbf{impossible } a \textbf{ if } \psi$$

と記述する．これは前提条件 ψ の下でアクション a が実行不可能であることを表している．その他の省略形は，言語 $\mathcal{A}$ に従う.

2.2.2 意味論

言語 $\mathcal{AR}$ による領域記述を D とすると，その意味論は遷移関数 Res_D によって与えられる．Res_D は，入力としてアクション名と状態を受けとり，出力として状態の集合を返す関数である．直観的に $Res_D(a, \sigma)$ は，状態 σ でアクション a を実行した後の可能な状態の集合を表している.

まず，言語 $\mathcal{AR}$ における状態を定義する.

状態

言語 $\mathcal{AR}$ ではフルーエントが複数の値を持つので，フルーエント名 F をその値 $V \in Rng_F$ に写像する関数 σ により状態を定義する．この関数を，**評価関数**と呼ぶ．評価関数を，さらに原子式に対する関数に拡張する：

$$\sigma(F \textbf{ is } V) = \begin{cases} \mathit{true}, & \text{if } \sigma(F) = V, \\ \mathit{false}, & \text{otherwise} \end{cases}$$

任意の式については，論理演算子の真理値表に従って拡張する．任意の制約 **always** ϕ に対し，評価関数 σ が式 ϕ を充足するとき，かつそのときに限り，**評価関数** σ **が制約 always** ϕ **を充足する**と定義する．

言語 $\mathcal{AR}$ による領域記述を D とする．**状態**とは，D に含まれるすべての制約 (2.6) を充足する評価関数である．

遷移関数

遷移関数 Res_D を定義するために，補助的な関数 Res^0_D と New^a_D を定義する．任意のアクション名 a と任意の状態 σ に対し，Res^0_D を次式で定義する：

$$Res^0_D(\sigma, a) = \left\{ \sigma' \;\middle|\; \begin{array}{l} \text{すべての } (a \textbf{ causes } \phi \textbf{ if } \psi) \in D \text{ に対し，} \sigma \text{ が } \psi \\ \text{を充足するならば，} \sigma' \text{ は } \phi \text{ を充足する状態である} \end{array} \right\}$$

この関数は，慣性の法則を持っていない遷移関数のようなものである．すなわち，アクションの効果は遷移後の状態において成立するが，アクションの影響を受けていないフルーエントは変化するかも知れない．そこで，慣性の法則を実現するために，関数 New^a_D を定義する．任意のアクション名 a と任意の状態 σ, σ' に対し，$New^a_D(\sigma, \sigma')$ は，次の条件のいずれかを満たす原子式 $(F \textbf{ is } \sigma'(F))$ の集合を返す：

条件 (1) F は慣性フルーエントであり，かつ $\sigma'(F) \neq \sigma(F)$ である．

条件 (2) 任意の $(a \textbf{ possibly changes } F \textbf{ if } \psi) \in D$ に対し，σ が ψ を充足する．

$New^a_D(\sigma, \sigma')$ の意図するところは，σ を遷移前の状態，σ' を遷移後の状態としたとき，遷移前と遷移後の状態間の変化分を集めることにある．そして遷移関数 Res_D の定義において，遷移前と遷移後の慣性フルーエントの状態変化が極小になるように定義する．これにより慣性の法則を実現する：

$$Res_D(a, \sigma) = \{\sigma' \in Res^0_D(a, \sigma) \mid \neg\exists \sigma'' \in Res^0_D(a, \sigma)\,(New^a_D(\sigma, \sigma'') \subset New^a_D(\sigma, \sigma'))\}$$

2.2.3 モデル

言語 $\mathcal{AR}$ による領域記述を D とすると，その**解釈**は遷移関数と初期状態の対 (Res_D, σ_0) で与えられる．任意の解釈に対し，評価命題の真偽値を定義する．解釈 $M = (Res_D, \sigma_0)$ と評価命題 $P = (\phi \textbf{ after } a_1; \cdots ; a_n)$ に対し，次のような列（履歴と呼ぶ）を考える．

$$\sigma_0, a_1, \sigma_1, \cdots, a_n, \sigma_n$$

ここで各 σ_k は状態であり，$\sigma_i \in Res_D(a_i, \sigma_{i-1})$ である．任意の履歴に対し，σ_n が式 ϕ を充足するならば，かつそのときに限り，**解釈 M において評価命題 P が真**である．ある解釈が，領域記述 D に含まれるすべての評価命題を充足するならば，その解釈を D の**モデル**と定義する．モデル M において評価命題 P が真であるとき，$M \models P$ と記述する．D の任意のモデルに対して真であるならば，$D \models P$ である．

2.2.4 記述例

以下に示すのは，帽子を被った Mary が湖に飛び込む例題である [11].

例題 3 Mary の帽子

$jump$ **possibly changes** hat **if** hat
$jump$ **causes** $lake$
$getout$ **causes** $\neg lake$
$puton$ **causes** hat
impossible $jump$ **if** $lake$
impossible $getout$ **if** $\neg lake$
impossible $puton$ **if** hat
always $lake \supset wet$
initially $\neg lake \wedge \neg wet \wedge hat$

ここで，$hat, lake, wet$ は，それぞれ，帽子をかぶっている，湖の中にいる，濡れているを表すフルーエントであり，すべて慣性フルーエントである．また $\neg Lake = (Lake \textbf{ is } false)$, $Lake = (Lake \textbf{ is } true)$ である．

アクション $jump$ は，帽子が飛んでいくかも知れないという非決定的効果と，湖に飛び込むと $(lake)$ 濡れる $(lake \supset wet)$ という間接的効果を持っている．アクション $getout$, $puton$ は，それぞれ，湖から出る，帽子をかぶるを表している．

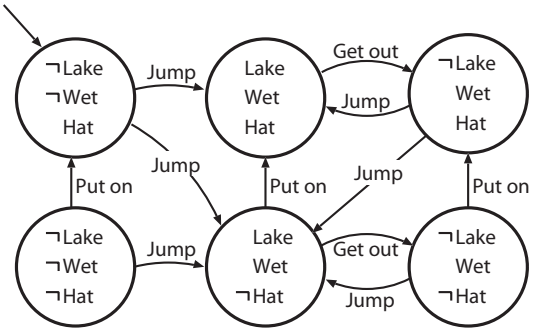

図 2.1: Mary が湖に飛び込む例題の状態遷移図

この問題の状態遷移図を図 2.1 に示す．ここでは，状態を表記するために，状態が充足するすべての原子式の集合により，状態を記述している．この問題は，初期状態が一意に定まるため，モデルを 1 つしか持たない．この問題では，次のような評価命題が帰結できる．

$$D \models \neg lake \wedge wet \ \textbf{after} \ jump; getout$$

2.3 SAT プランニング

ここ数年プランニングアルゴリズムの研究は，大きな進展をみせている．その発端となったのが Blum と Furst による Graphplan [4] である．

Graphplan は，STRIPS 形式のプランニング問題を入力とし，目標を達成する半順序プランを出力する**プラナ** (planner) である．Graphplan の特徴は，プラン探索空間を，いったん**プランニンググラフ**と呼ばれるデータ構造に符号化することにある．プランニンググラフでは，互いに相容れないアクション間・フルーエント間の関係（**相互排他関係**，mutual exclusion relation; mutex）がチェックしてある．次にプランニンググラフから体系的探索アルゴリズムによりプランを抽出する．このとき，先に調べておいた相互排他関係が探索空間を大きく絞り込むため，非常に効率よくプランを発見することできる．

しかし大規模な問題においては，相互排他関係を活用しても体系的探索アルゴリズムによるプラン抽出に時間がかかる．そこで Kautz と Selman は，プランニンググラフが**充足可能性問題** (Satisfiability; **SAT**) に自動変換できることを示し [18]，既存の高速な SAT ソルバによりプランを抽出するプラナ `Blackbox` を提案した [19]．現在のところ，`Blackbox` は最速のプラン生成器の 1 つである．このような SAT ソルバを用いたプランニングアプローチを **SAT プランニング**という．

ここではまず，STRIPS [7] について紹介する．STRIPS は，1970 年代に Fikes と Nilsson により開発されたプラナである．そのプランニング問題を記述するための STRIPS 言語は，今日に至るまで人工知能におけるプランニングの基本的枠組みとして広く研究されている．

2.3.1 STRIPS

STRIPS [7] は，一階述語論理の式の集合で表現された**世界モデル** (world model) を扱う問題解決器である．STRIPS システムは，世界の初期状態に関する記述（**初期世界モデル** (initial world model)）と，現在の状態を変化させるアクションに相当する**オペレータ** (operator) の集合により定義される．STRIPS は，初期世界モデルを与えられた**目標** (goal) を満たす世界モデルへと導くオペレータの列を**手段目標解析** (means-end analysis) により求める．

オペレータの記述は，**前提条件** (precondition)（一階述語論理の式），**追加リスト** (add list)（現在の世界モデルに追加される式のリスト），**削除リスト** (delete list)（現在の世界も出るから削除される式のリスト）からなる．

Fikes と Nilson によるオペレータ記述の意味の説明は非常に簡潔であり，先の括弧内のコメントがほとんどすべてである．Lifschitz は，文献 [21] で STRIPS の形式的説明を与えている．

例として，ロケット問題 [4] を STRIPS で記述する．これは，荷物をある目的地へロケットで運ぶ問題である．次の述語が存在する：

$rocket(X)$: X はロケットである
$place(X)$: X は場所である
$cargo(X)$: X は積荷である
$has\text{-}fuel(X)$: ロケット X は燃料を持っている
$at(X,Y)$: X は場所 Y にいる
$in(X,Y)$: 積荷 X は Y の中にある

この世界には，ロケット r と，荷物 a, b，出発点 l，目的地 p が存在する．初期世界モデルは，次のようになる：

$$rocket(r) \wedge cargo(a) \wedge cargo(b) \wedge place(l) \wedge place(p) \wedge$$
$$at(r,l) \wedge has\text{-}fuel(r) \wedge at(a,l) \wedge at(b,l).$$

目的は，荷物 a, b を場所 p に運ぶことであり，

$$at(a, p) \wedge at(b, p)$$

と表せる．オペレータは，荷物を積み込む $(load)$，荷物を降ろす $(unload)$，移動する $(move)$ の 3 種類が存在する：

- オペレータ $load(R, P, C)$
  - 前提条件　：$rocket(R) \wedge place(P) \wedge cargo(C) \wedge at(R, P) \wedge at(C, P)$
  - 追加リスト：$\{in(C, R)\}$
  - 削除リスト：$\{at(C, P)\}$
- オペレータ $unload(R, P, C)$
  - 前提条件　：$rocket(R) \wedge place(P) \wedge cargo(C) \wedge at(R, P) \wedge in(C, P)$
  - 追加リスト：$\{at(C, P)\}$
  - 削除リスト：$\{in(C, R)\}$
- オペレータ $move(R, F, T)$
  - 前提条件　：$rocket(R) \wedge place(F) \wedge place(T) \wedge (F \neq T) \wedge at(R, F) \wedge has\text{-}fuel(R)$
  - 追加リスト：$\{at(R, T)\}$
  - 削除リスト：$\{at(R, F), has\text{-}fuel(R)\}$

変数を含むオペレータは，**スキーマ** (schema) として扱われる．スキーマは，変数の異なる具体化それぞれに対応するオペレータの集合を表している．

STRIPS におけるプランニングは，手段目標解析と呼ばれる手法に基づいている．アルゴリズムの流れを簡単に説明する．STRIPS では，まず最初に，初期状態と目標状態の差異を取り出し，その差異を減少させるようなオペレータを選択する．そして次に，そのオペレータの前提条件と，そのオペレータにより達成できなかった目標の残りを次の**副目標** (sub-goal) にして，差異の検出とオペレータの選択を繰り返す．その結果，差異がなくなったときに，初期状態から目標状態へ至るオペレータの列，すなわちプランが得られる．

ただしこのプランニングアルゴリズムは完全ではないことが知られている．つまり，プランが存在したとしても，それを発見できない場合がある．その一方で，STRIPS は，初

期状態と目標状態の差を減少する方向の探索を優先的に進めるような探索手段をとることで，膨大な探索量を軽減している．

以下では，STRIPS におけるオペレータのことをアクションと呼び，追加リスト・削除リストのことをアクションの効果と呼ぶこととする．STRIPS では，世界の状態は述語の集合により表される．しかし，変数を含む述語はスキーマとして扱われ，すべての変数は具体化されるので，述語は命題とみなすことができる．その命題の真偽値は，アクションを実行することにより変化するので，以下では世界の状態を表す述語のことをフルーエントと呼ぶことにする．

2.3.2 Graphplan

Graphplan [4] は，STRIPS 形式のプランニング問題を受け取り，半順序プランを出力するプラナである．**半順序プラン** (partially ordered plan) とは，すべてのアクションの実行順序が定められていないプランである．順序付けされていないアクションは，どのような順番で実行しても目標を達成することができる．それに対し，すべてのアクションの実行順序が定められているプランを**全順序プラン** (totally ordered plan) という．言語 $\mathcal{A}$ においてアクション列に含まれるすべてのアクションは，演算子 ";" により全順序付けされている．

半順序プランは，順序付けされていないアクションに適当な順序を与えることで簡単に全順序プランに変換することができる．この操作はプランの**線形化** (linearization) と呼ばれる．例えば，半順序プラン

$$a_1; a_2; (a_{31}, a_{32}, a_{33}); a_4$$

は，アクション a_{31}, a_{32}, a_{33} をどのような順番で実行しても目標を達成できるので，以下のような全順序プランに線形化することができる：

$$a_1; a_2; a_{31}; a_{32}; a_{33}; a_4.$$

Graphplan は，プランの探索空間をプランニンググラフと呼ばれるデータ構造に符号化することで，非常に効率よくプランを探索することができる．Graphplan は，プランを発見するまで，次の 2 つのステップを繰り返しながら動作する：

1. プランニンググラフの展開

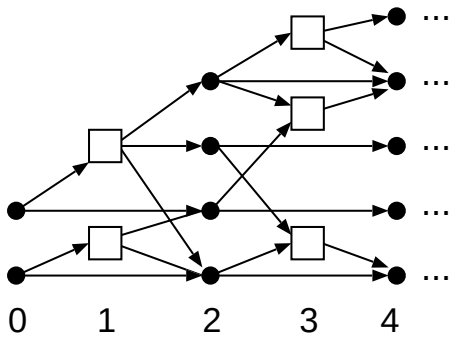

図 2.2: プランニンググラフ

2. プランニンググラフからプラン抽出

まずプランニンググラフの展開アルゴリズムから説明する．

プランニンググラフ展開

プランニンググラフ (planning graph) [4] とは，レベル付き有向グラフである[2]（図 2.2）．偶数レベルの頂点はフルーエントに対応し（図中の黒丸），奇数レベルの頂点はアクションに対応する（図中の四角）．レベル 0 は初期状態に対応し，最終レベルは目標状態に対応している．あるレベル i のフルーエント f は，f を前提条件に含むレベル $i+1$ のアクション a と辺で結ばれる．あるレベル j のアクション b は，b の効果に含まれるレベル $j+1$ の（複数の）フルーエントと辺で結ばれる．図中のフルーエント同士を結ぶ水平線は，レベル k のフルーエント f を次のレベル $k+1$ に伝播する特別なアクション “*no-op*(f)” を表している．レベル l のノードの集合を L_l で表す．

プランニンググラフは，あるアクションレベルにおいて複数のアクションを含んでいる．これらのアクションは並行して実行可能なアクションを表している．すなわち，どの順序で実行しても良いアクションを表している．もちろん，すべてのアクションを並行して実行することはできない．例えば，２つのアクションのうち，片方のアクションの効果が，他方のアクションの前提条件を否定する場合，それらのアクションの実行順序は制限を受ける．このことを厳密に定義するのが，次に示す相互排他関係である．

[2] グラフがレベル付きであるとは，グラフの頂点を，互いに素な集合 $L_1, L_2, \ldots, L_n$ に分割できるときをいう．ここで L_i は，隣り合うレベル (L_{i-1}, L_{i+1}) の頂点につながっている頂点の集合である．

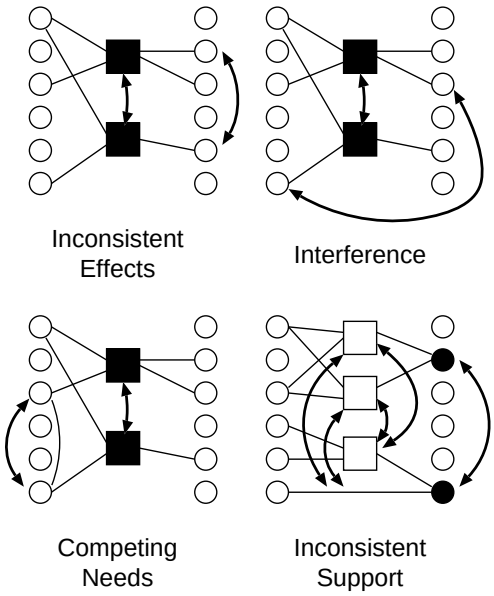

図 2.3: 相互排他関係 [40]

相互排他関係 (mutual exclusion relation; mutex) を次のように再帰的に定義する（図 2.3）[40]：

- **レベル i の 2 つのアクションが相互排他関係にある**のは，
  1. 片方の効果が他方の効果を否定する場合 (inconsistent effects)，または，
  2. 片方の効果が他方の前提条件を否定する場合 (interference)，または，
  3. 2 つのアクションの前提条件が，レベル $i-1$ で相互排他関係にある場合 (competing needs).
- **レベル i の 2 つのフルーエントが相互排他関係にある**のは，
  1. 片方が他方の否定である場合，または，
  2. 片方を導いたレベル $i-1$ のすべてのアクションが，他方を導いたレベル $i-1$ のすべてのアクションと相互排他関係にある場合 (inconsistent support).

あるレベル i における任意の 2 つのアクション（フルーエント）が相互排他関係にあるということは，レベル i においてその 2 つのアクション（フルーエント）が同時に成立することはない，ということを意味する．

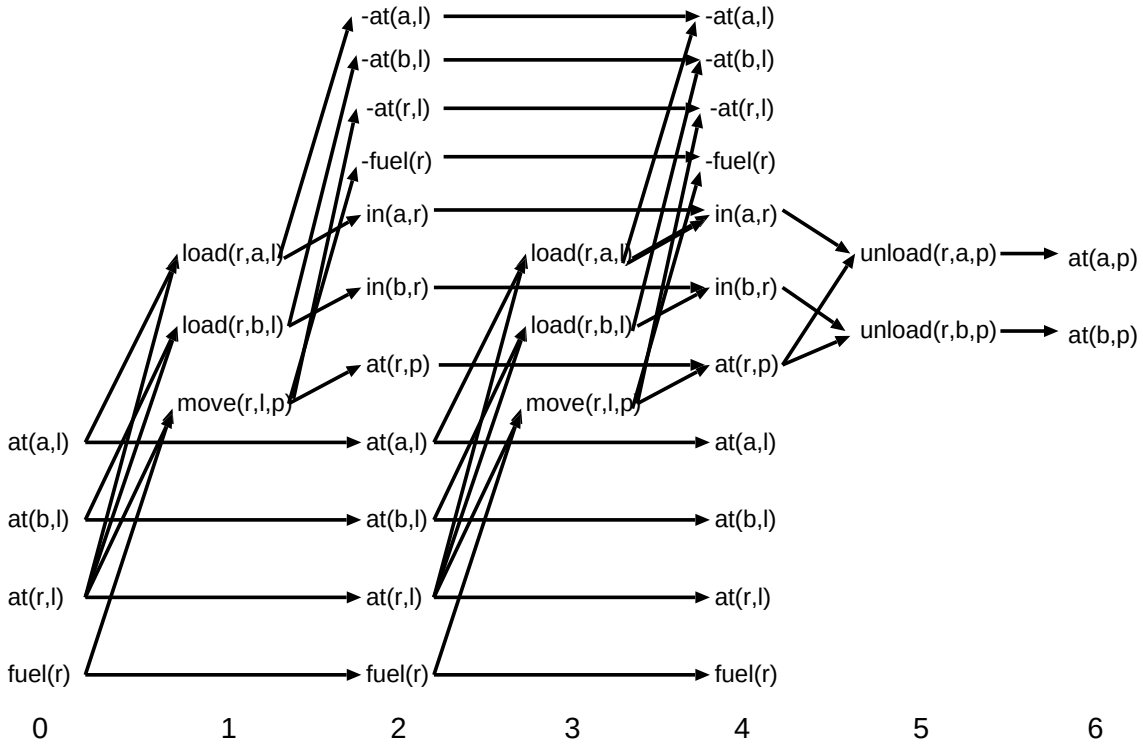

図 2.4: ロケット問題のプランニンググラフ

プランニンググラフの展開アルゴリズムは次のようになる．

初期レベル L_0 は，初期状態において真であるフルーエントからなる．奇数レベル i の頂点集合 L_i は，次の条件を満たすすべてのアクションから構成される：

> L_{i-1} で前提条件を満たすアクションであり，かつ，前提条件に含まれる任意の 2 つのフルーエント間に L_{i-1} において相互排他関係がないアクション

さらに，各フルーエント $f \in L_{i-1}$ に対し，f を伝播する $no\text{-}op(f)$ アクションを L_i に追加する．偶数レベルの頂点集合は L_{i+1} は，L_i に含まれるすべてのアクションの効果から構成される．

先のロケット問題のプランニンググラフをレベル 6 まで展開した結果を図 2.4 に示す．ただし実際には，レベル 3 と 5 には，もっと多くのアクションが存在する（例えば，$move(r, p, l)$ など）．同様にレベル 4 と 6 にも，もっと多くのフルーエントが存在する．

プラン抽出

プランニンググラフの最大レベル l の頂点集合 L_l に，目標に含まれるすべてのルーエントが含まれ，かつ，目標に含まれる任意の 2 つのフルーエント間に相互排他関係がない場合，Graphplan はプラン抽出アルゴリズムを適用する．プラン抽出アルゴリズムは，体系的な後向き探索アルゴリズムである．このアルゴリズムは，もし現在のプランニンググラフにプランが存在するならば，それを発見する．プランが存在しない場合，プラン抽出アルゴリズムは停止し，Graphplan は再びプランニンググラフの展開を行う．

プラン抽出アルゴリズムは次のようになる．$i = l$ とし，目標を $(g_1 \wedge \cdots \wedge g_n)$ とする．

レベル i における副目標を G^i とする（レベル l では，$G^l = (g_1 \wedge \cdots \wedge g_n)$ である）．アルゴリズムは，G^i を導くようなレベル $i-1$ におけるアクションの集合 A_1^{i-1} を求める．すなわち，A_1^{i-1} に含まれるすべてのアクションを実行すると，副目標 G^i が達成できるようなアクションの集合を求める．このアクションの集合は $no\text{-}op$ を含んでよい．また，A_1^{i-1} に含まれる任意の 2 つのアクションの間に相互排他関係があってはならない．

このようなアクションの集合は複数存在する．それらを $\{A_1^{i-1}, \cdots, A_m^{i-1}\}$ とする．このアクションの集合の集合から，1 つアクションの集合を選択する（バックトラックポイント）．ここでは添字順に選択するものとし，A_1^{i-1} を選ぶ．レベル $i-2$ における副目標は，A_1^{i-1} に含まれるすべてのアクションの前提条件となる．

プラン抽出アルゴリズムはこのようにして，(1) 副目標を達成するアクションの集合の選択，(2) アクションの集合から次の副目標の作成，を繰り返しながら動作する．そして，レベル 0 における副目標が初期状態を満たせば，アルゴリズムは停止し，そこに至るまでのグラフ上の経路をプランとして出力する．もし，レベル 0 における副目標が初期状態を満たさなかった場合，または，それ以前に副目標を満たすようなアクションの集合を作れなかった場合，アルゴリズムは，“アクション集合の選択” 時点までバックトラックし，別のアクション集合を選択し，再度プランの発見を試みる．

ロケット問題の例では，レベル 6 における副目標は

$$G^6 = at(a, p) \wedge at(b, p)$$

となる．この副目標を達成するようなレベル 5 のアクションの集合は，1 つしか存在しない：

$$\{unload(r, a, p), unload(r, b, p)\}$$

この集合に含まれるアクションの前提条件がレベル 4 における副目標 G^4 となる：

$$G^4 = at(r,p) \wedge in(a,r) \wedge in(b,r)$$

G^4 を達成するようなレベル 3 におけるアクションの集合は，複数存在する：

$$\{\{move(r,l,p), no\text{-}op(in(a,r)), no\text{-}op(in(b,r))\},$$
$$\{no\text{-}op(at(r,p)), no\text{-}op(in(a,r)), no\text{-}op(in(b,r))\},$$
$$\{no\text{-}op(at(r,p)), load(r,a,l), load(r,b,l))\},$$
$$\{no\text{-}op(at(r,p)), load(r,a,l), no\text{-}op(in(b,r)))\}, \cdots ,\}$$

ここで，例えば2番目のすべて $no\text{-}op$ アクションからなる集合を選んだとする．その場合，レベル 2 における副目標は先と同じである．この副目標を達成するためには，アクション $load(r,a,l)$ と $load(r,b,l)$ と $move(r,l,p)$ を同時に実行する必要があるが，$move(r,l.p)$ は残りの2つと相互排他関係にあるので（$move$ の効果が $load$ の前提条件を否定する），結局，副目標を達成するようなアクションの集合を作成できないため，バックトラックする必要がある．

Graphplan は，最終的に，図 2.4 のプランニンググラフから，次のプランを発見して停止する：

$$(load(r,a,l), load(r,b,l)) \ ; \ move(r,l,p) \ ; \ (unload(r,a,p), unload(r,b,p)) \qquad (2.9)$$

ここで，2つのアクション $load$ は，どちらを先に実行しても良いことを表している．アクション $unload$ に関しても同様である．

Graphplan では，複数の状態をひとまとめにして，１つのフルーエントレベルとして扱う．もし相互排他関係を無視したならば，Graphplan では，初期状態から任意のアクションを1つ実行して得られるすべての状態を足し合わせたものが，フルーエントレベル 2 となる．これにより，プランニンググラフの展開時には，状態空間の爆発を避けることができる．加えて，“明らかに同時に成立することがない2つのフルーエント（アクション）”を検査しておくことで（相互排他関係），余分なノードの追加を避けている．一方，プラン抽出ステップでは，ひとまとめにされた複数の状態をよりわけ，ゴールから到達可能な状態を求める必要があるが，ここでも相互排他関係が初期状態から到達不可能な副目標を削除する効果を持っている．

これらの優れた特徴のため，Graphplan は，それまでに研究されていたプランニングシステムよりも，多くの場合において桁数の異なる速さを示す [40]

2.3.3 Blackbox

Graphplan の弱点は，大規模な問題において，体系的探索アルゴリズムであるプラン抽出アルゴリズムが，プランの発見に非常に多くの時間を消費することである．これは，プランニンググラフのサイズが大きくなるにつれ，副目標の数が飛躍的に増大することが原因である．

そこで Kautz と Selman は，プランニンググラフが**充足可能性問題** (Satisfiability; **SAT**) に自動変換できることを示し [18]，既存の高速な SAT ソルバによりプランを抽出するプラナ Blackbox を提案した [19]．現在のところ，Blackbox は最速のプラン生成器の 1 つである．このような SAT ソルバを用いたプランニングアプローチを **SAT プランニング**という．プランニンググラフを SAT 問題に変換し，SAT ソルバでプランを抽出することで，プランニンググラフの構造にとらわれない効率の良いプラン探索を実現することができる．

Blackbox もまた Graphplan と同様に，プランを発見するまで，次の 2 つのステップを繰り返しながら動作する：

1. プランニンググラフの展開
2. プランニンググラフからプラン抽出

プランニンググラフの展開アルゴリズムは，Graphplan のそれに等しいので，以下では，Blackbox のプラン抽出アルゴリズムについて説明する．

プラン抽出

プランニンググラフの最大レベル l の頂点集合 L_l に，目標に含まれるすべてのルーエントが含まれ，かつ，目標に含まれる任意の 2 つのフルーエント間に相互排他関係がない場合，Blackbox はプラン抽出アルゴリズムを適用する．

プランニンググラフを以下の規則に従って SAT 問題に変換する：

(1) 初期レベル $L_0 = \{f_1, \cdots, f_n\}$ はレベル 0 において真であり，目標 $\{g_1, \cdots, g_m\}$ はレベル l において真である：

$$f_1^0 \wedge \cdots \wedge f_n^0 \wedge g_1^l \wedge \cdots \wedge g_m^l.$$

(2) レベル i において相互排他関係にある 2 つのアクション a, b は同時に成立しない：

$$\neg a^i \vee \neg b^i.$$

(3) アクションは，その前提条件を含意する．すなわち，レベル i のアクション a と辺で結ばれているレベル $i-1$ のフルーエント $p_1, \cdots, p_n$ に対し，以下の式を追加する：

$$a^i \supset p_1^{i-1} \wedge \cdots \wedge p_n^{i-1}.$$

(4) 偶数レベル i に含まれるフルーエントは，それらを効果として持つレベル $i-1$ のアクションの選言を含意する．すなわち，レベル i のフルーエント f と辺で結ばれているレベル $i-1$ のアクション $a_1, \cdots, a_n$ に対し，以下の式を追加する：

$$f^i \supset a_1^{i-1} \vee \cdots \vee a_n^{i-1}.$$

先のロケット問題において，レベル 6 まで展開したプランニンググラフ（図 2.4）を SAT 問題に変換すると以下のようになる：

- 変換規則 (1) より，

$$at^0(r,l) \wedge \mathit{has\text{-}fuel}^0(r) \wedge at^0(a,l) \wedge at^0(b,l) \wedge at^6(a,p) \wedge at^6(b,p).$$

- 変換規則 (2) より（一部のみ紹介），

$$\neg \mathit{move}^3(r,l,p) \vee \neg \mathit{load}^3(r,a,l)$$
$$\neg \mathit{move}^3(r,l,p) \vee \neg \mathit{load}^3(r,b,l)$$
$$\neg \mathit{move}^3(r,l,p) \vee \neg \mathit{no\text{-}op}^3(at(r,l))$$
$$\neg \mathit{move}^3(r,l,p) \vee \neg \mathit{no\text{-}op}^3(at(r,p))$$
$$\vdots$$

- 変換規則 (3) より（一部のみ紹介），

$$\mathit{unload}^5(r,a,p) \supset \mathit{in}^4(a,r) \wedge at^4(r,p)$$
$$\mathit{unload}^5(r,b,p) \supset \mathit{in}^4(b,r) \wedge at^4(r,p)$$
$$\mathit{move}^3(r,l,p) \supset at^2(r,l) \wedge \mathit{has\text{-}fuel}^2(r)$$
$$\mathit{no\text{-}op}^3(at(r,l)) \supset at^2(r,l)$$
$$\vdots$$

- 変換規則 (4) より（一部のみ紹介），

$$
\begin{aligned}
&at^6(a,p) \supset unload(r,a,p)^5 \vee no\text{-}op^5(at(a,p)) \\
&at^6(b,p) \supset unload(r,b,p)^5 \vee no\text{-}op^5(at(b,p)) \\
&at^4(at(r,p)) \supset move^3(r,l,p) \vee no\text{-}op^3(at(r,p)) \\
&\qquad\qquad\vdots
\end{aligned}
$$

このようにして得られた SAT 問題を，既存の高速な SAT ソルバにより解く．もしプランニンググラフにプランが存在するならば，SAT 問題の解が得られる．SAT 問題の解から，奇数レベルで真となるアクションを取り出し，冗長なアクションを削除し，レベルの順に並べれば（同じレベルに存在するアクションに順序関係はない），それが求めるプランになる[3]．ロケット問題の例では，Graphplan の場合と同じプラン (2.9) が得られる．プランが存在しない場合，SAT ソルバは，その SAT 問題が充足不可能であることを示すので，Blackbox は再びプランニンググラフの展開を行う．

以上が，Blackbox におけるプランニングアルゴリズムである．Blackbox は，Graphplan の弱点であった体系的プラン抽出アルゴリズムを，既存の高速な SAT ソルバによるプラン抽出アルゴリズムに置き換えることで，それまでのプランニングシステムでは解けなかった大規模な問題からもプランを発見することができる．Blackbox と Graphplan，その他のプランニングシステムとの性能比較については，文献 [19] で詳細な評価が行われている．

[3]アクション a が冗長であるとは，プランからアクション a を取り除いても目標を達成できる場合をいう．

第3章　言語 $\mathcal{A}$ とオートマトン

アクション言語 $\mathcal{A}$ の表現力には，その簡単な構文から，限界があることが分かる．このため $\mathcal{A}$ を拡張することに関心が集まり，さらなる表現力を求めた様々なアクション言語が提案されている [10]．しかしそれらの言語の多くは，言語 $\mathcal{A}$ に比べ客観的にどの程度表現力が向上したのか分かっていない．

我々は，言語 $\mathcal{A}$ の表現力を明らかにすることが新しいアクション言語の設計に有益であるという観点に立ち，この目的のために **有限オートマトン** [14](FA) に注目する．その結果，言語 $\mathcal{A}$ で表現可能な領域のクラスと，有限オートマトンで表現できる言語のクラスとが等価であることを示す [28]．この等価性の証明の過程において，$\mathcal{A}$ による領域記述と FA を相互に変換する 2 つのアルゴリズム，(i) FA を $\mathcal{A}$ に変換するアルゴリズム，(ii) $\mathcal{A}$ を FA に逆変換するアルゴリズム，を紹介する．そして，これらのアルゴリズムの健全性と完全性も証明する．

言語 $\mathcal{A}$ を FA に変換できることは，Gelfond と Lifschitz による $\mathcal{A}$ のモデル理論が遷移関数と状態の概念を採り入れたことからも予想されることである．我々が示す前者のアルゴリズムは，$\mathcal{A}$ から FA への変換を行なうだけでなく，$\mathcal{A}$ のモデルを計算する手続きにもなっている．また後者のアルゴリズムも，FA を $\mathcal{A}$ における命題に逆変換するだけでなく，**慣性**の概念を用いて $\mathcal{A}$ による任意の領域記述を簡単化する機能をもつ．

言語 $\mathcal{A}$ と FA の等価性の結果は，形式言語論からアクション言語の特性を解析するための基盤を提供する．例えば，前者のアルゴリズムは，プランニング問題のすべての可能な解を**正則表現**により表すことを可能にする．後者のアルゴリズムは，アクション理論が有限オートマトンの様々な応用分野に適用することを可能にする．特に，言語 $\mathcal{A}$ の適用範囲をより形式的に議論できるようになる．

3.1 領域記述の等価変換

本章では，言語 $\mathcal{A}$ の評価命題と効果命題をそれぞれ以下の形式に制限する：

$$f \textbf{ after } a_1; \ldots; a_m \tag{3.1}$$

$$a \textbf{ causes } e \textbf{ if } p_1 \wedge \ldots \wedge p_n \tag{3.2}$$

ここで a_i, a はアクション名，f, e, p_i はフルーエントである．このような制限を施しても言語 $\mathcal{A}$ の表現力が変わらないことを示す.

言語 $\mathcal{A}$ の一般的な評価命題は次の形式をしている：

$$f_1 \wedge \ldots \wedge f_k \textbf{ after } a_1; \ldots; a_m.$$

この評価命題は，式 (3.1) の形式の命題を組み合わせた次の式と等価である：

$$(f_1 \textbf{ after } a_1; \ldots; a_m) \wedge \ldots \wedge (f_k \textbf{ after } a_1; \ldots; a_m).$$

次に言語 $\mathcal{A}$ の一般的な効果命題

$$a \textbf{ causes } e_1 \wedge \ldots \wedge e_l \textbf{ if } \psi$$

が式 (3.2) の形式に変換できることを示す．まず前提条件 ψ を**選言標準形** (disjunctive normal form) に変換する：

$$a \textbf{ causes } e_1 \wedge \ldots \wedge e_l \textbf{ if } \rho_1 \vee \ldots \vee \rho_n$$

ここで ρ_i はフルーエントの連言である．遷移関数の定義より，上式は以下のように記述することができる.

$$a \textbf{ causes } e_1 \textbf{ if } \rho_1.$$
$$\vdots$$
$$a \textbf{ causes } e_l \textbf{ if } \rho_1.$$
$$\vdots$$
$$a \textbf{ causes } e_1 \textbf{ if } \rho_n.$$
$$\vdots$$
$$a \textbf{ causes } e_l \textbf{ if } \rho_n.$$

3.2 $\mathcal{A}$ から FA への変換

言語 $\mathcal{A}$ による領域記述を FA に変換するために，我々は，入力として無矛盾な領域記述を受けとり，出力として，**決定性有限オートマトン** (DFA) の集合を返すアルゴリズムを紹介する [28]．この DFA 集合と領域記述のモデル集合との間には，一対一対応が存在する．まず，このアルゴリズムを述べるための諸定義を行なう．

3.2.1 諸定義

決定性有限オートマトン (DFA) を 5 項組 $(Q, \Gamma, \delta, q_0, G)$ で定義する．ここで Q は状態の有限集合，Γ は有限の入力アルファベット，q_0 $(\in Q)$ は初期状態，G $(\subseteq Q)$ は最終状態の集合，δ は $\Gamma \times Q$ から Q への関数である．

$\mathcal{A}$ から FA への変換に際して，DFA の各項を次のように構成する．D を言語 $\mathcal{A}$ による領域記述とする．**オートマトンの状態** q を，$\mathcal{A}$ における状態と同様に定義する：

$$q = S \cup \{\neg F \mid F \in \mathit{fluent}(D) \setminus S\}. \tag{3.3}$$

ここで $S \subseteq \mathit{fluent}(D)$ である．**入力アルファベット** Γ を，$\mathit{action}(D)$ で定義する．遷移関数 δ と初期状態 q_0 は，第 3.2.2 節の関数により決定される．最終状態の集合 G は，$G = Q$ としておく．ただし，第 3.2.3 節においては Q の部分集合となる．

複数の状態を効率よく表現するために，**部分状態**の概念を導入する．部分状態 q' は，状態の不完全な表記で定義される：

$$q' = S' \cup \{\neg F \mid F \in S''\}.$$

ここで $S', S'' \subseteq \mathit{fluent}(D)$, $S' \cap S'' = \emptyset$ である．部分状態は状態の集合を表している．例えば $\mathit{fluent}(D) = \{\mathit{loaded},\ \mathit{alive}\}$ という領域において，部分状態 $q' = \{\mathit{loaded}\}$ は，$\{\mathit{loaded},\ \mathit{alive}\}$ と $\{\mathit{loaded}, \neg\mathit{alive}\}$ という状態を表す．また，部分状態 $\emptyset$ はすべての状態を表している．

任意のフルーエント名 F に対し，**絶対値**をとる操作を定義する：$|F| = F$, $|\neg F| = F$. 状態の絶対値についても同様に定義する：$|q| = \{|f| \mid f \in q\}$.

アクション a **に関連するフルーエントの集合** $rel(a)$ を以下で定義する：

$$rel(a) = \bigcup \{|e|, |p_1|, \ldots, |p_n|\} \qquad \text{for all } (a \textbf{ causes } e \textbf{ if } p_1 \wedge \ldots \wedge p_n) \in D.$$

すなわち，アクション a の効果を述べる命題に含まれるすべてのフルーエントの集合である．アクションの集合 $\{a_1, \ldots, a_n\}$ に対しても同様に定義する：

$$rel(\{a_1, \ldots, a_n\}) = rel(a_1) \cup \ldots \cup rel(a_n).$$

最後に，部分状態を**分割する**演算を定義する．任意の部分状態の集合 Q と任意のフルーエント f に対し，$div(Q, f)$ は，すべての部分状態 $q \in Q$ を，$q \cup \{f\},\ q \cup \{\neg f\}$ という 2 つの部分状態に分割する：

$$div(Q, f) = \begin{cases} \bigcup_{q \in Q} \{\ q \cup \{f\},\ q \cup \{\neg f\}\ \} & \text{if } |f| \notin |q| \text{ for any } q \in Q, \\ Q & \text{otherwise.} \end{cases}$$

$$div(Q, \{f_1, \ldots, f_n\}) = div(div(\ldots div(div(Q, f_1), f_2) \ldots, f_{n-1}), f_n).$$

3.2.2 $\mathcal{A}$ から DFA 集合への変換

言語 $\mathcal{A}$ による領域記述は，2 つの関数 **Init**（図 3.1），**Trans**（図 3.2）により，DFA の集合に変換される．**Init** は各 DFA の初期状態を求め，**Trans** はそれらの遷移関数を計算する．**Init** では，部分状態の概念を利用して，初期状態を計算する処理の効率化を図っている．つまり，あるアクション実行後の遷移先の複数の状態を，部分状態を使ってまとめて表現して処理している．

この 2 つの関数の正当性を示すために，次の定理を証明する．

定理 1 言語 $\mathcal{A}$ による無矛盾な領域記述を D，D のモデル集合を $\mathcal{I}$ とする．D を変換して得られた DFA 集合を $\mathcal{M} = \mathbf{Trans}(D, \mathbf{Init}(D))$ とする．このとき，$\mathcal{I}$ と $\mathcal{M}$ との間には一対一対応が存在する．しかも，モデル $I = (\Phi, \sigma_0)$ が $\mathcal{I}$ に存在するならば，かつ，そのときに限り，初期状態が σ_0 で，遷移関数が Φ である DFA $M \in \mathcal{M}$ が存在する．

証明 D のすべてのモデルの初期状態の集合を Σ_0，任意のモデルの遷移関数を Φ とする．$\mathcal{M}$ に含まれるすべての DFA の初期状態の集合を Q_0 とし，任意の $M \in \mathcal{M}$ の遷移関数を δ とする．図 3.1, 図 3.2 より，関数 **NextState** が定める δ が，Φ に等しいことは容易にわかる．よって，$\Sigma_0 = Q_0$ が示せれば証明は完結する．

このために，D に含まれる評価命題の数に関する帰納法を利用する．領域内に含まれる評価命題の数が k であるとき，その領域を D^k で表す．D^k のすべてのモデルの初期状

Init(D)
入力：k 個の評価命題を含む言語 $\mathcal{A}$ による領域記述 D
出力：初期状態の集合 Q_0
begin
  $Q^0 := \{\emptyset\}$;
  for $j := 1$ **to** k **do**
    j 番目の value 命題：$(f$ **after** $a_1; \ldots; a_m) \in D$;
    $T_0^j := div(Q^{j-1}, f)$;
    for $i := 1$ **to** m **do**
      $T_0^j := div(T_0^j, rel(a_i))$;
      $T_{i-1}^j := div(T_{i-1}^j, rel(a_i))$;
      $T_i^j := \{$**NextState**$(a_i, q, D) \mid q \in T_{i-1}^j\}$;
    $Q^j := \{q_0 \in T_0^j \mid f \in q_m,\ q_m \in T_m^j, \mathbf{NextState}^*(a_1; \ldots; a_m, q_0, D) = q_m\}$;
  $Q_0 := div(Q^k, fluent(D))$
end.

NextState(a, q, D)
入力：アクション a，状態 q，言語 $\mathcal{A}$ による領域記述 D
出力：q で a を実行した後の状態 q_1
begin
  $q' := \{f \mid (a$ **causes** f **if** $p_1 \wedge \ldots \wedge p_n) \in D$ and $\{p_1 \ldots p_n\} \subseteq q\}$;
  $q_1 := q' \cup \{f \in q \mid |f| \notin |q'|\}$
end.

NextState$^*(a_1; \ldots; a_m, q, D)$
入力：アクション列 $a_1; \ldots; a_m$，状態 q，言語 $\mathcal{A}$ による領域記述 D
出力：q で $a_1; \ldots; a_m$ を実行した後の状態 q_1
begin
  $q_1 := q$;
  for $i := 1$ **to** m **do** $q_1 :=$ **NextState**(a_i, q_1, D)
end.

図 3.1: 初期状態の集合を計算する関数 **Init**

Trans(D, Q_0)
入力：言語 $\mathcal{A}$ による領域記述 D，初期状態の集合 Q_0
出力：DFA の集合 $\mathcal{M}$
begin
  $Q := div(\{\emptyset\}, \mathit{fluent}(D))$;
  for all $q \in Q$ and all $a \in action(D)$ **do**
    $\delta(a, q) :=$ **NextState**(a, q, D);
  $\mathcal{M} := \{(Q, action(D), \delta, q_0, Q) \mid q_0 \in Q_0\}$
end.

図 3.2: 遷移関数を計算する関数 **Trans**

態の集合を Σ_0^k で表し，$\mathcal{M}^k = \mathbf{Trans}(D^k, \mathbf{Init}(D^k))$ に含まれるすべての DFA の初期状態の集合を Q_0^k で表す．ここで，任意の k に対して $\Sigma_0^k = Q_0^k$ が成立することを証明する．基底ステップは図 3.1 より明らかである．いま，k 以下のすべての数 l に対し，$\Sigma_0^l = Q_0^l$ を仮定する．$\Sigma_0^{k+1} = Q_0^{k+1}$ を証明するために，D^{k+1} を次式で定義する：

$$D^{k+1} = D^k \cup \{f \textbf{ after } a_1; \ldots; a_m\}. \quad (m \geq 0)$$

このとき D^{k+1} のすべてのモデルの初期状態の集合 Σ_0^{k+1} は,

$$\Sigma_0^{k+1} = \{q_0 \in \Sigma_0^k \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\}$$

と定義できる．ここで Φ は D^k の任意のモデルの遷移関数である．帰納法の仮定より,

$$\Sigma_0^{k+1} = \{q_0 \in Q_0^k \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\}$$

である．ところで関数 **Init** より，Q_0^{k+1} は,

$$Q_0^{k+1} = div(Q^{k+1}, \mathit{fluent}(D^{k+1}))$$

と表せる．ここで Q^{k+1} は,

$$Q^{k+1} = \{q_0 \in div(Q^k, S) \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\}.$$

となる．ここで $S = \{f\} \cup rel(\{a_1, \ldots, a_m\})$ である．部分状態の集合 $div(Q^k, S)$ が表す状態の集合と，$div(Q^k, \mathit{fluent}(D^{k+1}))$ が表す状態の集合は等しい．よって,

$$\begin{aligned} Q_0^{k+1} &= div(\{q_0 \in div(Q^k, S) \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\}, \mathit{fluent}(D^{k+1})) \\ &= \{q_0 \in div(Q^k, \mathit{fluent}(D^{k+1})) \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\} \end{aligned}$$

また，部分状態の集合 $div(Q^k, \mathit{fluent}(D^{k+1}))$ が表す状態の集合と，$div(Q^k, \mathit{fluent}(D^k))$ が表す状態の集合も等しい．従って，

$$\begin{aligned} Q_0^{k+1} &= \{q_0 \in div(Q^k, \mathit{fluent}(D^k)) \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\} \\ &= \{q_0 \in Q_0^k \mid f \in (\Phi, q_0)^{a_1;\ldots;a_m}\} \\ &= \Sigma_0^{k+1}. \end{aligned}$$ □

定理 1 より，関数 **Init** と **Trans** から求まる各オートマトンと，領域記述の各モデルとが，一対一に対応することがわかる．つまり，**Init** と **Trans** は，領域記述のモデルを計算する手続きでもある．

3.2.3 FA との対応

関数 **Init**,**Trans** から得られる DFA 集合を，1 つの FA に変換するために，次の定義を必要とする．いま，$\mathcal{M}$ を DFA の集合とする．$\mathcal{M}$ **により受理される言語** $L(\mathcal{M})$ を，次式により定義する：

$$L(\mathcal{M}) = \bigcap_{M \in \mathcal{M}} L(M).$$

ここで $L(M)$ は M の受理言語である．

ところで，正則集合の族は共通部分に関して閉じている [14] ので，$L(\mathcal{M})$ もまた正則集合である．よって，$L(\mathcal{M})$ を受理する FA を構成することができる．そこで，DFA の集合 $\mathcal{M}$ と $L(\mathcal{M})$ とを受理する FA を同一視し，$\mathcal{M}$ により両方の意味を適宜表すことにする．

2 つの関数 **Init**,**Trans** が，$\mathcal{A}$ による領域記述を DFA の集合に変換し，上の定義が，DFA の集合と同じ受理言語をもつ FA を構成する．この $\mathcal{A}$ から FA への変換アルゴリズムが，健全かつ完全であることを示すために，われわれは，さらに次の定義を必要とする．任意の DFA M に対し，$M(f)$ を，**フルーエント** f **を含む状態を最終状態とする DFA** として定義する．DFA の集合に対しても同様に定義する：$\mathcal{M}(f) = \{M(f) \mid M \in \mathcal{M}\}$.

この定義と定理 1 から，$\mathcal{A}$ から FA への変換手続きの正当性を明らかにする次の定理を導くことができる．

定理 2 $\mathcal{A}$ による無矛盾な領域記述を D とする．D を変換して得られる DFA 集合を $\mathcal{M} = \mathbf{Trans}(D, \mathbf{Init}(D))$ とし，フルーエント f を含む状態を最終状態とする FA $\mathcal{M}(f)$

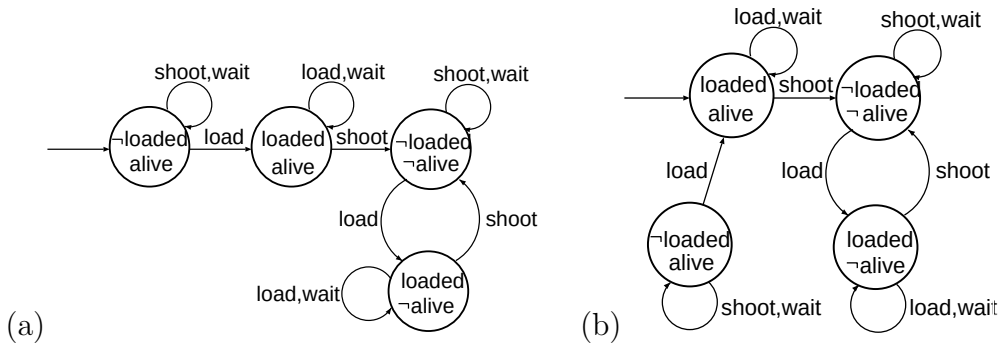

図 3.3: (a) Yale Shooting 領域と (b) Murder Mystery 領域の変換結果

を考える．このとき，$D \models (f \textbf{ after } a_1; \ldots; a_m)$ ならば，かつ，そのときに限り，$\mathcal{M}(f)$ は，アクション列 $a_1; \ldots; a_m$ を受理する．

系 3 $\mathcal{A}$ による無矛盾な領域記述を D とする．D を変換して得られる DFA 集合を $\mathcal{M} = \mathbf{Trans}(D, \mathbf{Init}(D))$ とする．このとき，$D \models (\textbf{initially } f)$ ならば，かつ，そのときに限り，任意の DFA $M \in \mathcal{M}$ の初期状態にはフルーエント f が含まれる．

3.2.4 変換例

図 3.3 (a),(b) は，例題 1,2 の領域記述を変換した結果である．これらの領域記述は完全であるので，唯 1 つの DFA に変換される．

この変換結果から，次のようにして，領域記述が帰結する評価命題を容易に知ることができる．図 3.3 (a) の DFA を M_a，(b) の DFA を M_b とする．(i) $M_a(\neg alive)$ は，アクション列 *load;wait;shoot* を受理する．したがって，Yale Shooting 領域において，$D \models (\neg alive \textbf{ after } load;wait;shoot)$ である．(ii) M_b の初期状態は *loaded* を含んでいる．つまり，Murder Mystery 領域において，$D \models (\textbf{initially } loaded)$ である．

3.3 FA から $\mathcal{A}$ への変換

FA を $\mathcal{A}$ による領域記述に逆変換するために，本章では，入力として DFA を受けとり，出力として $\mathcal{A}$ による領域記述を返すアルゴリズムを紹介する [28]．任意の FA は，それと等価な DFA に変換できるので [14]，FA を $\mathcal{A}$ に変換するには，このアルゴリズムで十分である．

RevTrans(M)
入力：DFA $M = (Q, \Gamma, \delta, q_0, G)$　出力：領域記述 D
begin
　$D := \{$**initially** $f \mid f \in q_0\}$;
　$D := D \cup \{a$ **causes** f **if** $p_1 \wedge \ldots \wedge p_n \mid q \in Q, a \in \Gamma, f \in \delta(a, q), q = \{p_1, \ldots, p_n\}\}$
end.

図 3.4: $\mathcal{A}$ による領域記述を生成する関数 **RevTrans**

3.3.1 DFA の状態

逆変換アルゴリズムでは，DFA の状態を，$\mathcal{A}$ における状態の定義と同様に扱う．よって，逆変換アルゴリズムに DFA を入力するには，DFA の状態を，フルーエントの集合として表さなければならない．DFA の状態数が N であれば，少なくとも $\lceil \log_2 N \rceil$ 個のフルーエントを用意する必要がある．DFA の状態を，前章と同様に式 (3.3) で定義する．

3.3.2 逆変換アルゴリズム

DFA は，2 つの関数 **RevTrans**（図 3.4），**Compress**（図 3.5）により，言語 $\mathcal{A}$ による領域記述に逆変換される．**RevTrans** は，任意の DFA を，$\mathcal{A}$ による領域記述に変換する：DFA の初期状態から評価命題を生成し，DFA の遷移関数から効果命題を生成する．**Compress** は，領域記述から冗長な効果命題を削除・簡単化する．**Compress** は，逆変換においてだけでなく，任意の領域記述を簡単化するために一般に利用できる．

3.3.3 逆変換アルゴリズムの正当性

RevTrans と **Compress** からなる逆変換アルゴリズムが，健全かつ完全であることを示すために，まず **Compress** の正当性を証明する．

定理 4 $\mathcal{A}$ による領域記述を D とする．このとき，D と **Compress**(D) は同じモデル集合をもつ．

Compress(D)
入力：領域記述 D　出力：D を簡単化した領域記述
begin
　while (冗長な効果命題が D より削除可能) **do**
　　$E_1, E_2, \ldots, E_6$ を D 中の効果命題とする;
　1) $E_1 =$ a **causes** f **if** $p_1 \wedge \ldots \wedge p_n \quad (n \geq 1)$;
　　if $f \in \{p_1, \ldots, p_n\}$ **then** $D := D \setminus \{E_1\}$;
　2) $E_2 =$ a **causes** f **if** $p_1 \wedge \ldots \wedge p_n \quad (n \geq 0)$;
　　$E_3 =$ a **causes** f **if** $q_1 \wedge \ldots \wedge q_m \quad (m \geq n)$;
　　if $\{p_1, \ldots, p_n\} \subseteq \{q_1, \ldots, q_m\}$
　　　then $D := D \setminus \{E_3\}$;
　3) $E_4 =$ a **causes** f **if** $p_1 \wedge \ldots \wedge p_n \quad (n \geq 0)$;
　　$E_5 =$ a **causes** f **if** $q_1 \wedge \ldots \wedge q_n$;
　　if $\neg p_i = q_j$ and $\{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\} = \{q_1, \ldots, q_{j-1}, q_{j+1}, \ldots, q_n\}$ **then**
　　　$D := (D \setminus \{E_4, E_5\}) \cup \{a$ **causes** f **if** $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$;
　4) $E_6 =$ a **causes** f **if** $p_1 \wedge \ldots \wedge p_n \quad (n \geq m \geq 1)$;
　　if $\neg f = p_i \ \ (1 \leq i \leq n)$ and $f \in \{q_1, \ldots, q_m\}$ かつ
　　　$(\{q_1, \ldots, q_m\} \setminus \{f\}) \subset (\{p_1, \ldots, p_n\} \setminus \{\neg f\})$ であるような効果命題
　　　　a **causes** $\neg f$ **if** $q_1 \wedge \ldots \wedge q_m$ が存在しない
　　　then
　　　　$D := (D \setminus \{E_6\}) \cup \{a$ **causes** f **if** $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$
end.

図 3.5: 領域記述を簡単化する関数 **Compress**

証明　まず，i) D の任意のモデルの遷移関数と **Compress**(D) の任意のモデルの遷移関数が等しいことを示し，その後，ii) D のすべてのモデルの初期状態の集合と **Compress**(D) のすべてのモデルの初期状態の集合が等しいことを示す.

i) **Compress** は，4 種類の冗長な効果命題を削除・簡単化する：1) **トートロジー消去**, 2) **包摂テスト**，3) ***div* の逆操作**，4) **前提条件の簡単化**．例えば，4) では，フルーエントの値を反転させる効果命題 a **causes** f **if** $\neg f, g$ に対し，a **causes** $\neg f$ **if** f, g という逆の効果を及ぼす命題が領域記述に存在しなければ，前提条件中の $\neg f$ を削除する．

これら 1) から 4) の操作を行なった後も，領域記述の任意のモデルの遷移関数が変化しないことを示す．ここでは，1) トートロジー消去についてのみ証明する．トートロジー消

去では，次のような効果命題を削除する：

$$\begin{aligned} &a \textbf{ causes } f \textbf{ if } p_1 \wedge \ldots \wedge p_n. \quad (n \geq 1), \\ &f \in \{p_1, \ldots, p_n\} \end{aligned} \tag{3.4}$$

Compress が命題 (3.4) を削除しても，任意のモデルの遷移関数が変化しないことを明らかにするために，すべての状態の集合 Σ を考え，Σ_1 と Σ_2 に分割する：

$$\begin{aligned} \Sigma_1 &= \{\sigma_1 \in \Sigma \mid \{p_1, \ldots, p_n\} \subseteq \sigma_1\}, \\ \Sigma_2 &= \{\sigma_2 \in \Sigma \mid \{p_1, \ldots, p_n\} \not\subseteq \sigma_2.\} \end{aligned}$$

任意の状態 $\sigma_2 \in \Sigma_2$ において，命題 (3.4) の前提条件は満たされないので，(3.4) を削除しても，Σ_2 からの遷移に対して影響を及ぼさない．一方，任意の状態 $\sigma_1 \in \Sigma_1$ において，命題 (3.4) は，$f \in \Phi(a, \sigma_1)$ であることを述べている．ところで，$f \in \{p_1, \ldots, p_n\}$ なので，$f \in \sigma_1$ である．よって (3.4) を削除しても，慣性の法則により，Σ_1 からの遷移に対して影響を及ぼさない．

ii) **Compress** は，評価命題に対し，何ら操作を加えない．よって，D と **Compress**(D) の任意のモデルの遷移関数が等しいことから，D と **Compress**(D) のすべてのモデルの初期状態の集合もまた，等しい． □

この定理を用いて，次の定理を証明する．

定理 5 任意の DFA を M とする．M を逆変換して得られた領域記述を $D =$ **Compress**(**RevTrans**(M)) とする．このとき，D は完全であり，その唯一のモデルは，M と同じ遷移関数と初期状態をもつ．

証明 図 3.4 より，**RevTrans**(M) が完全であり，唯一つのモデルをもつこと，そして，そのモデルが M と同じ初期状態と遷移関数をもつことが証明できる．**Compress** はモデルを変化させないので（定理 4），$D =$ **Compress**(**RevTrans**(M)) も完全であり，その唯一のモデルは，M と同じ遷移関数と初期状態をもつ． □

定理 5 より，逆変換手続きの正当性を明らかにする次の定理が導かれる：

定理 6 任意の DFA を M とする．M を逆変換して得られた領域記述を $D =$ **Compress**(**RevTrans**(M)) とする．このとき，フルーエント f を含む状態を最終状態とする DFA $M(f)$ がアクション列 $a_1; \ldots; a_m$ を受理するならば，かつ，そのときに限り，

$$D \models (f \textbf{ after } a_1; \ldots; a_m)$$

である．

3.3.4 逆変換例

次に示すのは，図 3.3 (b) のオートマトンを逆変換した結果である．

initially *alive*.

load **causes** *loaded*.

shoot **causes** ¬*alive* **if** *loaded*.

shoot **causes** ¬*loaded*.

initially *loaded*.

最後の評価命題のみが，例題 2 の領域記述と異なっている．この理由は，我々の逆変換アルゴリズムが，**initially** のみを使って初期状態を完全に記述するからである．しかし，これら 2 つの領域記述は，同じ完全なモデルをもっている．このことは，変換／逆変換アルゴリズムの正当性の一面を示している．

例題 4 人間と狼と山羊とキャベツの問題

図 3.6 は，人間と狼と山羊とキャベツの問題の状態遷移図である [14]．M,W,G,C は，それぞれ男と狼と山羊とキャベツを表している．最初，男と狼と山羊とキャベツは川の左岸にいる．男は小舟を使って，狼と山羊とキャベツを右岸へ運ぶことを目標にしている．小舟には男のほかに荷物が一つしか乗らない．また，狼と山羊を岸に残していくと狼が山羊を食べてしまい，山羊とキャベツを岸に残していくと山羊がキャベツを食べてしまう．図中の正のフルーエントは左岸にいることを表し，負のフルーエントは右岸にいることを表している．各アクション m,w,g,c は，M,W,G,C の対岸への移動を表す．

図 3.6 のオートマトンを DFA に変換するために，次のような仮定を設けた：遷移先の定義されていないアクションは，遷移元に回帰するものとする．この仮定より得られる DFA を M とする．以下に示すのは，M を言語 $\mathcal{A}$ による領域記述に逆変換した結果である．

initially M. **initially** W.
initially G. **initially** C.

m **causes** M **if** $\neg M, W, \neg G, C$
m **causes** $\neg M$ **if** $M, W, \neg G, C$
m **causes** M **if** $\neg M, \neg W, G, \neg C$
m **causes** $\neg M$ **if** $M, \neg W, G, \neg C$

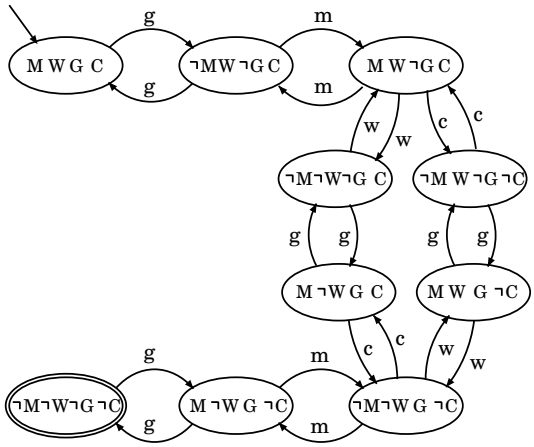

図 3.6: 人間と狼と山羊とキャベツの問題

g **causes** M **if** $\neg M, \neg G$

g **causes** G **if** $\neg M, \neg G$

g **causes** $\neg M$ **if** M, G

g **causes** $\neg G$ **if** M, G

w **causes** M **if** $\neg W, \neg G, C$

w **causes** W **if** $\neg M, \neg G, C$

w **causes** $\neg M$ **if** $W, \neg G, C$

w **causes** $\neg W$ **if** $M, \neg G, C$

w **causes** M **if** $\neg W, G, \neg C$

w **causes** W **if** $\neg M, G, \neg C$

w **causes** $\neg M$ **if** $W, G, \neg C$

w **causes** $\neg W$ **if** $M, G, \neg C$

c **causes** M **if** $\neg C, W, \neg G$

c **causes** C **if** $\neg M, W, \neg G$

c **causes** $\neg M$ **if** $C, W, \neg G$

c **causes** $\neg C$ **if** $M, W, \neg G$

c **causes** M **if** $\neg C, \neg W, G$

c **causes** C **if** $\neg M, \neg W, G$

c **causes** $\neg M$ **if** $C, \neg W, G$

c **causes** $\neg C$ **if** $M, \neg W, G$

RevTrans 終了時には 160 個あった効果命題が，**Compress** 実行後，24 個にまで減少した．また，これらの効果命題の前提条件は，問題中の制約（山羊とキャベツを岸に残してはいけない，など）を表している．

上の領域記述を D とすると，M の受理言語に対応するアクション列の集合 S は，次式で与えられる：

$$S = \{\ s \mid D \models (\neg M\ \textbf{after}\ s) \wedge (\neg W\ \textbf{after}\ s) \wedge (\neg G\ \textbf{after}\ s) \wedge (\neg C\ \textbf{after}\ s)\ \}$$

ここで s はアクション列である．

3.4 考察

本節では，$\mathcal{A}$ と FA の表現力の等価性と，その応用，そして関連研究について述べる．

3.4.1 $\mathcal{A}$ と FA の等価性

$\mathcal{A}$ による領域記述を D とし，$f \in \mathit{fluent}(D)$ とする．D **において** f **を実現する言語**を，$L(D, f) \stackrel{\text{def}}{=} \{s \mid D \models f\ \textbf{after}\ s\}$ で定義する．ここで s はアクション列である．

この定義と，定理 2，定理 6 から，次の 2 つの定理を示すことができる．

定理 7 $\mathcal{A}$ による無矛盾な領域記述を D，$\mathcal{M} = \mathbf{Trans}(D, \mathbf{Init}(D))$ とする．このとき，$L(D, f) = L(\mathcal{M}(f))$ である．

定理 8 任意の DFA を M，$D = \mathbf{Compress}(\ \mathbf{RevTrans}(M))$ とする．G を M の最終状態の集合とすれば，

$$L(M) = \bigcup_{q \in G} \bigcap_{f \in q} L(D, f).$$

この 2 つの定理から，言語 $\mathcal{A}$ で表現可能な領域のクラスと，FA で表現できる言語とクラスとが等価であることがわかる．

ところで，FA と正則表現，正則文法の間には，図 3.7 のような関係がある [14]．図中の A から B へ向かう辺は，A のタイプの記述が与えられたとき，それと等価な B のタイプの記述を構成できることを示している．すなわち，$\mathcal{A}$ による領域記述 D が，式 (2.2) のような評価命題を帰結するとき，そのアクション列 $a_1; \ldots; a_m$ を正則表現／正則文法で表すことができる．このことは，次のプランニング問題に対して，効力を発揮する．

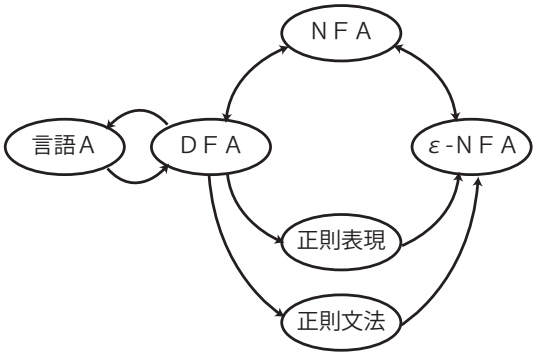

図 3.7: 言語 $\mathcal{A}$ と FA のクラス

3.4.2 プランニング

プランニング問題を解くために，次の定義を必要とする．領域記述 D に含まれるアクションのすべての列からなる集合を $action^*(D)$ で定義する．$L \subseteq action^*(D)$ に対して，$w \in L$ が L の**極小列**であるとは，w の真部分列が L に含まれないときをいう．**フルーエント f を実現するプランニングの解**を，$L(D, f)$ に含まれる極小列として定義する．

プランニング問題として，例題 1 の Yale Shooting 領域において，七面鳥を殺すための（$\neg alive$ を実現するための）アクション列を求めてみよう．まず，Yale Shooting 領域を DFA M に変換する（図 3.3 (a)）．つぎに，$M(\neg alive)$ が受理する言語を計算し，それを正則表現で表す：

$$
\begin{aligned}
&L(\mathcal{M}(\neg alive)) = \\
&\quad (shoot + wait)^*; load; (load + wait)^*; shoot; \\
&\quad (shoot + wait + load; (load + wait)^*; shoot)^* \\
&+ (shoot + wait)^*; load; (load + wait)^*; shoot; \\
&\quad (shoot + wait + load; (load + wait)^*; shoot)^*; \\
&\quad load; (load + wait)^*.
\end{aligned}
$$

この式は，$\neg alive$ を実現する**すべてのアクション列**を表している．この結果より，$\neg alive$ を実現するプランニングの解は $load; shoot$ と求まる．このように，$\mathcal{A}$ と FA の等価性の結果を用いて，$\mathcal{A}$ におけるプランニング問題を，FA の受理言語を求める問題に置き換えることができる．

3.4.3 健全性と完全性

Gelfond と Lifschitz は，文献 [9] で，$\mathcal{A}$ による領域記述を拡張論理プログラム [8] に変換する手続き π を提案している．π は，領域記述が類似した命題を含むとき，健全ではない．ここで 2 つの効果命題が**類似している** (similar) とは，それらの前提条件のみが異なるときをいう．我々の $\mathcal{A}$ から FA への変換手続きは，領域記述が類似した命題を含んでいても健全である．

また，手続き π は帰結に関して完全でない．このことは，次の反例により確かめることができる．

f **after** a.

a **causes** f **if** f.

これら 2 つの命題からなる領域記述 D が，**initially** f を帰結することは明らかである．しかし，拡張論理プログラム $\pi(D)$ は，この命題の変換結果 $holds(f, s0)$ を帰結しない．一方，$\mathcal{A}$ から FA への我々の変換手続きは，定理 2 より，このような領域記述に対しても完全である．したがって，D を変換して得られる FA は，その初期状態に f を含む．

ところで D を **Compress** により簡単化すると，2 番目の命題がトートロジー消去により削除できる．この領域記述（すなわち，**Compress**(D)）においては，π もまた完全である．

3.4.4 関連研究

Kartha は，言語 $\mathcal{A}$ による領域記述を，Pednault の ADL に基づいた手法 [34] と Reiter の形式化 [37] に変換する正当な手続きを提案している [16]．Kartha の手続きと，我々の逆変換手続きを結び付けることで，FA を，これらの形式でも模倣できるようになる．しかし，我々の結果とは違って，Kartha は $\mathcal{A}$ へ逆変換する手法を示していないので，これらの言語間の等価性は明らかではない．

3.4.5 表現の問題

有限状態系を用いて，アクションの効果を表現することは，特に新しいアイデアではない．例えば，先の “男と狼と山羊とキャベツの問題” は，オートマトン理論の教科書 [14]

で紹介されている．世界をオートマトンで表現することの利点と欠点は，John McCarthy が AI に論理を用いて以来，彼により議論されている．状態グラフは，問題に対する最終的な仕様書としての表現形式に適している．その主な欠点は，**詳細化に対する頑健性** [25] (elaboration tolerance) を欠いていることである．一方，論理による表現は，局所的な変化に対し，状態グラフよりも適している．我々の提案した FA を $\mathcal{A}$ に逆変換する手続きの利点の 1 つはここにある．現在，オートマトンの応用分野は，テキスト編集，語彙解析，並行プロセス，プログラム検証など幅広く存在する．このような動的な領域の設計を，$\mathcal{A}$ やその他のアクション言語で行なえるのならば，それを修正することは容易になると考えられる．

言語 $\mathcal{A}$ には，慣性の法則による記述面の利点がある．アクション a と状態 σ に対し，$\delta(a, \sigma) = \sigma$ であれば，この遷移関係の記述は，$\mathcal{A}$ において削除することができる．これは，まさしく **Compress** （図 3.5）が実行していることである．事実，**RevTrans** （図 3.4）は，慣性の法則を利用していない．**Compress** は，逆変換手続きの中で使われるが，もっと一般に，$\mathcal{A}$ による任意の領域記述を簡単化・最適化するのに有用である．アクション言語の実用化に際して，このような簡単化の手続きは，さらに重要なものとなると考えられる．

3.5 まとめ

本章では，アクション言語 $\mathcal{A}$ の表現力を形式的に解析し，$\mathcal{A}$ と FA が同じ表現力を持つことを示した．その証明の過程において (i) FA を $\mathcal{A}$ に変換するアルゴリズム，(ii) $\mathcal{A}$ を FA に逆変換するアルゴリズムとを定義した．前者のアルゴリズムは $\mathcal{A}$ から FA への変換を行なうだけでなく，$\mathcal{A}$ のモデルを計算する手続きにもなっている．また後者のアルゴリズムは FA を $\mathcal{A}$ における命題に逆変換するだけでなく，**慣性**の概念を用いて $\mathcal{A}$ による任意の領域記述を簡単化する機能をもつ．

言語 $\mathcal{A}$ と FA の等価性の結果は，形式言語論からアクション言語の特性を解析するための基盤を提供する．例えば，前者のアルゴリズムは，プランニング問題のすべての可能な解を**正則表現**により表すことを可能にする．後者のアルゴリズムは，アクション理論が有限オートマトンの様々な応用分野に適用することを可能にする．特に，言語 $\mathcal{A}$ の適用範囲をより形式的に議論できるようになる．

第4章　非決定性アクション言語 $\mathcal{NA}$

言語 $\mathcal{A}$ が提案されて以来，$\mathcal{A}$ を基として，$\mathcal{A}$ を拡張し，さらなる表現力を求めた様々なアクション言語が提案されている：非決定性効果をもつアクション [5,6,10,11,22,39]，同時発生アクション [3,5,10]，フルーエント間の制約 [11,17,22,39]，フルーエント間の依存関係 [10,12,22,39] などを表現できる言語が存在する．

ここでは非決定性効果をもつアクションを記述できる言語（**非決定性アクション言語**）を対象とし，1) **非決定性有限オートマトン** (NFA) の観点から新しい非決定性アクション言語 $\mathcal{NA}$ を提案し，2) すでに提案されている非決定性アクション言語の表現力を形式的に解析する手段を提供する [31]．

言語 $\mathcal{A}$ で表現可能な領域のクラスと，有限オートマトン (FA) で表現できる言語のクラスとが等しいことからも，非決定性アクション言語を NFA の観点から設計することは，自然な拡張であると考えられる．

我々が提案する非決定性アクション言語 $\mathcal{NA}$ は，NFA と同じ表現力をもっており，例えば $\mathcal{NA}$ と他のアクション言語との等価性を示すことで，その言語の適用範囲を形式的に議論できるようになる．実際に我々は，言語 $\mathcal{AR}$ [11] による記述を，それと等価な $\mathcal{NA}$ による記述に変換する手続きを与え，$\mathcal{NA}$ と $\mathcal{AR}$ が同じ表現力を持つことを示す．

4.1　構文論

アクション言語 $\mathcal{NA}$ では，非決定性効果を持つアクションや，間接的効果を持つアクション，実行不可能なアクション，状態に対する制約を記述することができる．

4.1.1　構成要素

アクション言語 $\mathcal{NA}$ を5つ組 $\langle \mathbb{A}, \mathbb{F}, \mathbb{E}, \mathbb{C}, \mathbb{V} \rangle$ で表す．ここで $\mathbb{A}$ は**アクション名**の集合，$\mathbb{F}$ は**フルーエント名**の集合，$\mathbb{E}$ は**効果命題** (effect proposition) の集合，$\mathbb{C}$ は**制約** (constraint) の集合，$\mathbb{V}$ は**評価命題** (value proposition) の集合である．

言語 $\mathcal{NA}$ において，フルーエントは真または偽の 2 値をとる．フルーエント名 F の否定 $\neg F$ を**負のフルーエント**といい，F を**正のフルーエント**という．フルーエントを小文字の f で表し，フルーエント名を大文字の F で表す．フルーエントを，論理演算子 $(\vee, \wedge, \neg, \supset, \equiv)$ で結合したものを**式**と呼ぶ．

言語 $\mathcal{NA}$ では，アクション列の集合を簡単に表すために，正則表現が利用できる．r と s がそれぞれ言語 R と S を表す正則表現のとき，$(r+s), (r;s), (r^*)$ は，それぞれ集合 $R \cup S, RS, R^*$ を表す． ここで $RS = \{rs \mid r \in R, s \in S\}$ であり，R^* は R の Kleene 閉包である [14]．正則表現 r に対し，それが表す集合を $L(r)$ と記述する．

4.1.2 命題

言語 $\mathcal{NA}$ では 3 種類の命題を記述できる．まず，アクションとその効果（非決定的効果かも知れない）を記述する**効果命題**がある：

$$a \textbf{ causes } \phi_1 \mid \cdots \mid \phi_n \textbf{ if } \psi \tag{4.1}$$

ここで a はアクション名，各 ϕ_i はフルーエントの連言肢，ψ は式である．各 ϕ_i をアクションの**効果**と呼び，ψ をアクションの**前提条件**と呼ぶ．$n = 1$ のときアクションの効果は**決定的**であり，$n \geq 2$ のとき**非決定的**である．この命題は，前提条件 ψ が成立する状態においてアクション a を実行したならば，$\phi_1, \cdots, \phi_n$ のうち，いずれか 1 つがアクション実行後の状態で成立することを言明する．縦棒演算子 $(|)$ は 2 項演算子であり，その優先順位は $\wedge$ や $\vee$ よりも低い．この縦棒演算子の厳密な意味は第 4.2 節で与える．縦棒演算子は，論理式における論理和とも排他的論理和とも異なる．縦棒演算子では $f \mid \neg f$ という記述が可能であるが，これを論理和で記述すると $f \vee \neg f$ となり恒真となる．排他的論理和で記述した場合も $(f \wedge \neg f) \vee (\neg f \wedge f)$ となり恒真となる．

次に，状態に対する**制約**が記述できる：

$$\textbf{always } \phi \tag{4.2}$$

ここで ϕ は式である．この命題は，あらゆる状態において，式 ϕ が常に成立することを言明する．

最後に，観測した事実を記述する**評価命題**がある：

$$\phi \textbf{ after } r \tag{4.3}$$

ここで ϕ は式，r はアクション名から構成される正則表現である．この命題は，$L(r)$ に含まれる任意のアクション列を実行した後，ϕ が成立することを言明する．

4.1.3 省略形

いくつか有用な省略形を定義する．ここで *true* は，恒真を意味する特別なフルーエントで，全ての状態で真となる．$\mathit{false} = \neg \mathit{true}$ である．

効果命題 (4.1) に対し，前提条件 ψ が *true* であるならば，**if** 以下を省略する：

$$a \textbf{ causes } \phi_1 \mid \cdots \mid \phi_n$$

また，a **causes** *False* **if** ψ の形をした効果命題を次のように記述する：

$$\textbf{impossible } a \textbf{ if } \psi$$

評価命題 (4.3) に対し，$r = \varepsilon$ (空列) であるならば，簡単に次のように記述する：

$$\textbf{initially } \phi$$

変数を含む命題は，**スキーマ** (schema) として扱われる（例題 6 参照）．スキーマは，変数の異なる具体化それぞれに対応する命題の集合を表している．

4.2 意味論

言語 $\mathcal{NA}$ では，効果命題・制約・評価命題を使って，状態変化領域を記述する．その命題の集合を**領域記述**と呼ぶ．我々は，その**解釈**を**非決定性状態遷移システム** $\langle Q, \mathcal{A}, \delta, q_0 \rangle$ で与える．ここで Q は状態の集合，$\mathcal{A}$ はアクション名の集合，q_0 は初期状態，δ は $Q \times \mathcal{A}$ から 2^Q への関数であり**遷移関数**と呼ぶ[1]．

まず，状態と式の真偽値を定義する．

[1] アクション名の集合 $\mathcal{A}$ は，任意の解釈において同じである．従って，本来ならば解釈の定義に含める必要はない．ここでは，有限オートマトンとの対応のために解釈に加えている．

4.2.1 状態と式

状態 q を，全てのフルーエント名について，正または負のフルーエントのいずれかを含む集合と定義する：

$$\sigma = S \cup \{\neg F \mid F \in \mathit{fluent}(D) \setminus S\}.$$

ここで $S \subseteq \mathit{fluent}(D)$ である．

任意の状態 q と，フルーエント f のみからなる式 $\phi = f$ に対し，$f \in q$ であるとき，かつそのときに限り，**状態** q **で式** ϕ **が真**であると定義する．任意の式 ϕ' については，論理演算子の真偽値表に従って拡張する．状態 q で式 ϕ' が真であるならば，$q \models \phi'$ と記述する．

任意の状態 q と制約 **always** ϕ に対し，$q \models \phi$ であるとき，かつそのときに限り，**状態** q **で制約 always** ϕ **が真**であると定義し，$q \models$ **always** ϕ と記述する．状態 q が，領域記述に含まれる全ての制約を充足するならば，それを**正当な状態**と呼ぶ．

4.2.2 正当な遷移関数

正当な遷移関数を定義する前に，補助的な演算を定義する．

集合族 $A_1, \cdots, A_n$ に対し，次の直積演算を定義する：

$$A_1 \times \cdots \times A_n = \{a_1 \cup \cdots \cup a_n \mid a_1 \in A_1, \cdots, a_n \in A_n\}$$

効果命題 $P = (a$ **causes** $\phi_1 \mid \cdots \mid \phi_n$ **if** $\psi)$ に対し，次の関数 *effect* を定義する：

$$\mathit{effect}(P) = \{c(\phi_1), \cdots, c(\phi_n)\}$$

ただし $\phi_i = f_1 \wedge \cdots \wedge f_m$ に対して，$c(\phi_i) = \{f_1, \cdots, f_m\}$ とする．さらに，効果命題の集合 $\{P_1, \cdots, P_m\}$ に対して拡張する：

$$\mathit{effect}(\{P_1, \cdots, P_m\}) = \mathit{effect}(P_1) \times \cdots \times \mathit{effect}(P_m)$$

ここで $\mathit{effect}(\emptyset) = \emptyset$ とする．

言語 $\mathcal{NA}$ による領域記述を D とする．任意の状態 q と任意のアクション a に対し，遷移関数 δ が次式で定義されるとき，δ を**正当な遷移関数**と呼ぶ．

$$\delta(q, a) = \{q' \in Q \mid q' = (q \setminus (|e|^+ \cup |e|^-)) \cup e,\ e \in \mathit{effect}(\mathcal{P})\}$$

ここで Q は全ての状態の集合であり，$\mathcal{P}$ は次式で与えられる：

$$\mathcal{P} = \{P \in D \mid P = (a \textbf{ causes } \phi_1 \mid \cdots \mid \phi_n \textbf{ if } \psi) \text{ かつ } q \models \psi\}$$

$\mathit{effect}(\mathcal{P})$ は，状態 q においてアクション a を実行すると発生する全ての非決定的効果を表している．正当な遷移関数において，アクションの効果 $e \in \mathit{effect}(\mathcal{P})$ に含まれないフルーエントは，遷移前の値を遷移後も持続する定義となっている．これは**慣性の法則**を表している．

最後に遷移関数 δ を，状態とアクション列の組に対する関数 $\hat{\delta} : Q \times \mathcal{A}^* \to 2^Q$ に拡張する：

- $\hat{\delta}(q, \varepsilon) = \{q\}$
- $\hat{\delta}(q, wa) = \bigcup_{r \in \delta(q,w)} \delta(r, a)$

ここで w はアクション名の列を表し，a はアクション名を表す．$\hat{\delta}(q, a) = \delta(q, a)$ であるので，両者をともに δ で表すことにする．

4.2.3 モデル

モデルを定義するための補助的な定義を与える．

任意の解釈 $M = \langle Q, \mathcal{A}, \delta, q_0 \rangle$ と，任意の式 ϕ に対し，$M(\phi) = \langle Q, \mathcal{A}, \delta, q_0, Q(\phi) \rangle$ と定義する．ここで $Q(\phi) = \{q \in Q \mid q \models \phi\}$ である．もし状態数が有限であるならば，$M(\phi)$ は，まさに，**非決定性有限オートマトン** (NFA) である．すなわち $Q(\phi)$ は，目標状態の集合を表している．

オートマトンと同様に，$M(\phi)$ **により受理される言語** $\mathcal{L}(M(\phi))$ を以下で定義する：

$$\mathcal{L}(M(\phi)) = \{w \in \mathcal{A}^* \mid \delta(q_0, w) \cap Q(\phi) \neq \emptyset\}$$

さらに，$M(\phi)$ **により確実に受理される言語** $L(M(\phi))$ を以下で定義する：

$$L(M(\phi)) = \{w \in \mathcal{A}^* \mid \delta(q_0, w) \subseteq Q(\phi)\}$$

$\mathcal{L}(M(\phi))$ は，目標状態に到達する可能性を持っている全てのアクション列の集合を表している．一方 $L(M(\phi))$ は，必ず目標状態に到達するような全てのアクション列の集合を

表している．$L(M(\phi)) \subseteq \mathcal{L}(M(\phi))$ である．もし $M(\phi)$ が**決定性有限オートマトン** (DFA) であるならば，両者は等しい．

次に，評価命題の真偽値を定義する．任意の解釈 $M = \langle Q, \mathcal{A}, \delta, q_0 \rangle$ と，任意の評価命題 $P = (\phi \textbf{ after } r)$ に対し，$L(r) \subseteq L(M(\phi))$ であるとき，かつそのときに限り，M において P が**真である**と定義し，$M \models P$ と記述する．同様に，もし $L(r) \subseteq \mathcal{L}(M(\phi))$ であるならば，かつそのときに限り，M において P が**軽信的 (credulous) に真である**と定義し，$M \mathrel{|\!\sim} P$ と記述する．特に比較するときに，$M \models P$ を，M において P が**懐疑的 (skeptical) に真である**ということもある．

言語 $\mathcal{NA}$ による領域記述を D とする．解釈 $M = \langle Q, \mathcal{A}, \delta, q_0 \rangle$ が D の**モデル**であるのは，

i) Q が，全ての正当な状態の集合であり，

ii) δ が，Q 上に定義された正当な遷移関数であり，

iii) D に含まれる全ての評価命題が M において真であるとき，

かつそのときに限る．条件 i) を満たす状態集合は 1 つに決定され，条件 ii) を満たす遷移関数も，高々 1 つしか存在しない．従って，同じ領域記述における異なったモデルは，その初期状態によってのみ異なる．領域記述は，モデルを持てば**無矛盾**である．評価命題 P が D の任意のモデルに対して真であるならば，$D \models P$ と書く．

同様に，解釈 $M = \langle Q, \mathcal{A}, \delta, q_0 \rangle$ が D の**軽信的なモデル**であるのは，i) と ii) に加え，

iii') D に含まれる全ての評価命題が M において軽信的に真であるとき，

かつそのときに限る．評価命題 P が D の任意の軽信的モデルに対して軽信的に真であるならば，$D \mathrel{|\!\sim} P$ と書く．

4.2.4 記述例

以下に示すのは，第 2.2.4 節で示した帽子を被った Mary が湖に飛び込む問題を言語 $\mathcal{NA}$ で記述した例である．先頭の命題のみが，言語 $\mathcal{AR}$ による記述と異なっている．言語 $\mathcal{NA}$ では，Mary がジャンプすると帽子が飛んでいくかもしれないという非決定的効果を $hat \mid \neg hat$ と表現することができる．

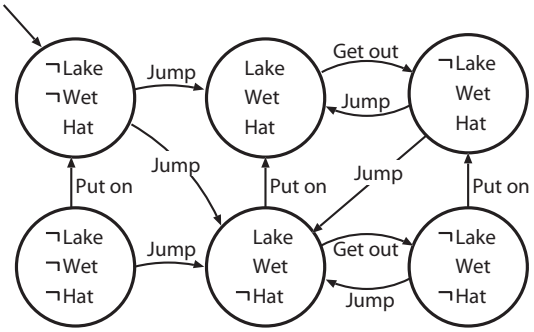

図 4.1: Mary が湖に飛び込む例題のモデル

例題 5 Mary の帽子 [11]

$jump$ **causes** $hat \mid \neg hat$ **if** hat

$jump$ **causes** $lake$

$getout$ **causes** $\neg lake$

$puton$ **causes** hat

impossible $jump$ **if** $lake$

impossible $getout$ **if** $\neg lake$

impossible $puton$ **if** hat

always $lake \supset wet$

initially $\neg lake \wedge \neg wet \wedge hat$

この例題のモデルを図 4.1 に示す．同図において遷移先の定義されていないアクションは，状態 $\{false\}$ に遷移するものとする．この問題は，初期状態が一意に定まるため，モデルを 1 つしか持たない．この問題では，次のような評価命題が帰結できる．

$D \models \neg lake \wedge wet$ **after** $jump; getout$

$D \not\models hat$ **after** $jump; getout$

$D \mid\sim hat$ **after** $jump; getout$

4.3 $\mathcal{NA}$ と有限オートマトン

本節では，$\mathcal{NA}$ と有限オートマトン (FA) の表現力が等しいことを示す．ここで領域記述に含まれるフルーエント名とアクション名の数は有限であることを仮定する．

最初に，軽信的な帰結関係に基づけば，$\mathcal{NA}$ と FA の等価性は明らかである．すなわち，

定理 9 言語 $\mathcal{NA}$ による領域記述を D とする．D の軽信的モデル全体の集合を $\{M_1, \cdots, M_n\}$ とする．任意の式 ϕ に対し，$\mathcal{L}(M_1(\phi)) \cap \cdots \cap \mathcal{L}(M_n(\phi))$ を受理する FA $M'(\phi)$ を構成できる．このとき，

$$D \mid\!\sim \phi \textbf{ after } r \quad \Leftrightarrow \quad L(r) \subseteq \mathcal{L}(M'(\phi))$$

証明 $\mathcal{L}(M_i(\phi))$ が正則集合であることは，その定義より明らかである．正則集合の族は共通部分に関して閉じているので [14]，$\mathcal{L}(M_1(\phi)) \cap \cdots \cap \mathcal{L}(M_n(\phi))$ もまた正則集合であり，それを受理する FA を構成することができる．□

従って，言語 $\mathcal{NA}$ による任意の領域記述から，軽信的帰結関係において等価な FA を構成できることが分かる．この逆の関係についても同様のことがいえる：第 3.3 節において，任意の FA を言語 $\mathcal{A}$ による領域記述に変換する正当な手続きを定義した．言語 $\mathcal{A}$ は $\mathcal{NA}$ のサブセットであり，$\mathcal{A}$ による記述は $\mathcal{NA}$ による記述でもある．

よって以下では，懐疑的な帰結関係についても等価となることを示す．まず，補助的な結果を示す．

補題 1 $M = \langle Q, \Gamma, \delta, q_0, G \rangle$ を任意の NFA とする．M により確実に受理される言語

$$L(M) = \{w \in \Gamma^* \mid \delta(q_0, w) \subseteq G\}$$

は，DFA $M' = \langle 2^Q, \Gamma, \delta', \{q_0\}, 2^G \rangle$ の受理言語

$$L(M') = \{w \in \Gamma^* \mid \delta'(\{q_0\}, w) \in 2^G\}$$

に等しい．ここで δ' は以下で与えられる：

$$\delta(q_1, a) \cup \cdots \cup \delta(q_n, a) = \{p_1, \cdots, p_m\}$$

のとき，かつそのときに限り，

$$\delta'(\{q_1, \cdots, q_n\}, a) = \{p_1, \cdots, p_m\}$$

ここで $q_i, p_j \in Q, a \in \Gamma$ である．

証明 最初に，入力列 $w \in \Gamma^*$ の長さに関する数学的帰納法を利用して，$\delta(q_0, w) = \delta'(\{q_0\}, w)$ を示すことができる．次に，M と M' の受理言語が等しいことを示すために，$L(M) = L(M')$ でないと仮定する．すなわち，次式を満たす w が存在すると仮定する：

$$\exists w((w \notin L(M)) \wedge w \in L(M')) \vee \\ ((w \in L(M)) \wedge w \notin L(M'))$$

まず，左辺 $w \notin L(M) \wedge w \in (M')$ を満たす w が存在しないことを示す．受理言語の定義より，この式は，次のように変換できる：

$$(\delta(q_0, w) \not\subseteq G) \wedge (\delta'(\{q_0\}, w) \in 2^G)$$

ここで $\delta(q_0, w) = \delta'(\{q_0\}, w)$ であるので，上式は矛盾する．同様に右辺 $w \in L(M) \wedge w \notin (M')$ の矛盾も証明できる．従って $L(M) = L(M')$ である．□

この結果は，“確実に受理される言語” もまた正則集合であり，それを受理するオートマトンを構成できることを示している．この結果より，次の定理を示すことができる．

定理 10 言語 $\mathcal{NA}$ による領域記述を D とする．D のモデル全体の集合を $\{M_1, \cdots, M_n\}$ とする．任意の式 ϕ に対し，$L(M_1(\phi)) \cap \cdots \cap L(M_n(\phi))$ を受理する FA $M'(\phi)$ を構成できる．このとき，

$$D \models \phi \ \textbf{after}\ r \quad \Leftrightarrow \quad L(r) \subseteq L(M'(\phi))$$

証明 補題 1 より $L(M_i(\phi))$ は正則集合である．したがって，定理 9 と同様に，$L(M_1(\phi)) \cap \cdots \cap L(M_n(\phi))$ もまた正則集合であり，それを受理する FA を構成することができる．□

以上のことから，$\mathcal{NA}$ と FA で表現可能な領域のクラスが等しいことが分かる．

4.4 例題

以下に示すのは，金庫の中にある宝石を狙う泥棒について記述したものである．

例題 6 宝石を狙う泥棒

$$
\begin{array}{rl}
\mathit{open} & \textbf{causes} \quad \mathit{opened} \\
\mathit{open} & \textbf{causes} \quad \mathit{true} \mid \mathit{ringing} \quad \textbf{if} \quad \mathit{on(alarm1)} \vee \mathit{on(alarm2)} \\
\mathit{get} & \textbf{causes} \quad \mathit{jewel} \quad \textbf{if} \quad \mathit{opened} \\
\mathit{cut(X)} & \textbf{causes} \quad \neg \mathit{on(X)} \\
\textbf{impossible} & \mathit{open} \quad \textbf{if} \quad \mathit{opened} \\
\textbf{impossible} & \mathit{get} \quad \textbf{if} \quad \neg \mathit{opened} \vee \mathit{jewel} \\
\textbf{impossible} & \mathit{cut(X)} \quad \textbf{if} \quad \neg \mathit{on(X)} \\
\textbf{initially} & \neg \mathit{open} \wedge \neg \mathit{ringing} \wedge \neg \mathit{jewel} \\
\textbf{initially} & \mathit{on(alarm1)} \wedge \mathit{on(alarm2)}
\end{array}
$$

屋敷には警報器が２つある．泥棒は，警報を鳴らさずに，金庫の扉をうまく開けることができるかも知れない．この領域記述を D として，ここでは，警報を鳴らさずに宝石を手に入れるためのアクション列 r_1, r_2 を求めてみる：

$$
\begin{array}{rcl}
D & \mid\!\sim & \mathit{jewel} \wedge \neg \mathit{ringing} \ \textbf{after} \ r_1 \\
D & \models & \mathit{jewel} \wedge \neg \mathit{ringing} \ \textbf{after} \ r_2
\end{array}
$$

r_1 は目標状態に到達する可能性を持っているアクション列であり，r_2 は必ず目標状態に到達するようなアクション列である．

このプランニング問題を解くために，まず D のモデル M を求める (図 4.2)．M は D の唯一のモデルであり，M に目標状態を与えた $M(\mathit{jewel} \wedge \neg \mathit{ringing})$ は NFA である．定理 9 より，この NFA の受理言語 $\mathcal{L}(M(\mathit{jewel} \wedge \neg \mathit{ringing}))$ が，r_1 を満たすアクション列である．

一方 r_2 を求めるためには，定理 10 より，M を決定性状態遷移システム M' に変換すれば良い．M' を図 4.3 に示す．M から M' への変換は補題 1 を利用する．M' に目標状態を与えた $M'(\mathit{jewel} \wedge \neg \mathit{ringing})$ は DFA であり，その受理言語 $L(M'(\mathit{jewel} \wedge \neg \mathit{ringing}))$ は，必ず目標状態に到達するアクション列 r_2 を表している．

このようにして，アクション言語におけるプランニング問題を，オートマトン理論における受理言語を求める問題に帰着できる．同様にして，

- ある状態においてあるフルーエントが成立するかしないか（予測）
- ある状態でどういうフルーエントが成立しているか（推定）

などを問い合わせることも可能である．

図 4.2: 宝石を盗め

4.5 非決定性によるセンサーの定式化

非決定性を用いることで，障害物を検出するセンサーなどを定式化することができるようになる．以下に示すのは，障害物を検出するセンサーを装備した迷路探索ロボットの知識表現である：

$$forward \quad \textbf{causes} \quad p(X, Y-1) \quad \textbf{if} \quad p(X,Y) \wedge n \wedge \neg obstacle(X, Y-1)$$
$$forward \quad \textbf{causes} \quad p(X+1, Y) \quad \textbf{if} \quad p(X,Y) \wedge e \wedge \neg obstacle(X+1, Y)$$
$$forward \quad \textbf{causes} \quad p(X-1, Y) \quad \textbf{if} \quad p(X,Y) \wedge w \wedge \neg obstacle(X-1, Y)$$
$$forward \quad \textbf{causes} \quad p(X, Y+1) \quad \textbf{if} \quad p(X,Y) \wedge s \wedge \neg obstacle(X, Y+1)$$

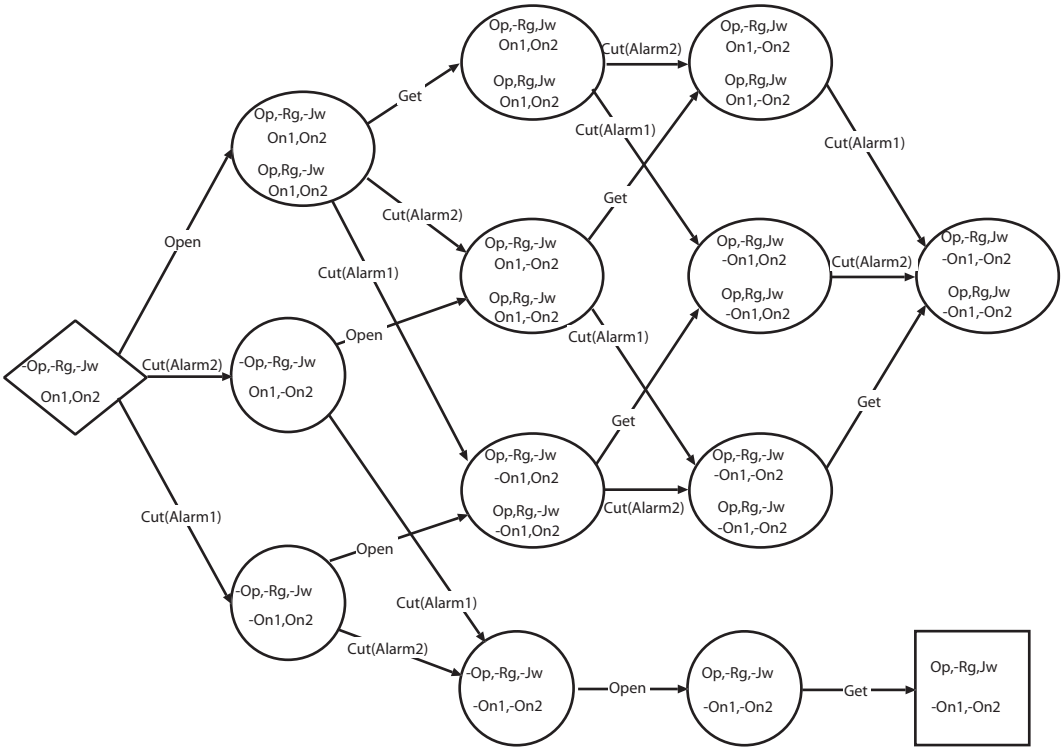

図 4.3: 慎重に宝石を盗め

$$sense \quad \textbf{causes} \quad obstacle(X, Y-1) \mid \neg obstacle(X, Y-1) \quad \textbf{if} \quad p(X,Y) \wedge n$$
$$sense \quad \textbf{causes} \quad obstacle(X+1, Y) \mid \neg obstacle(X+1, Y) \quad \textbf{if} \quad p(X,Y) \wedge e$$
$$sense \quad \textbf{causes} \quad obstacle(X-1, Y) \mid \neg obstacle(X-1, Y) \quad \textbf{if} \quad p(X,Y) \wedge w$$
$$sense \quad \textbf{causes} \quad obstacle(X, Y+1) \mid \neg obstacle(X, Y+1) \quad \textbf{if} \quad p(X,Y) \wedge s$$
$$left \quad \textbf{causes} \quad w \quad \textbf{if} \quad n$$
$$left \quad \textbf{causes} \quad s \quad \textbf{if} \quad w$$
$$left \quad \textbf{causes} \quad e \quad \textbf{if} \quad s$$
$$left \quad \textbf{causes} \quad n \quad \textbf{if} \quad e$$
$$right \quad \textbf{causes} \quad e \quad \textbf{if} \quad n$$
$$right \quad \textbf{causes} \quad s \quad \textbf{if} \quad e$$
$$right \quad \textbf{causes} \quad w \quad \textbf{if} \quad s$$
$$right \quad \textbf{causes} \quad n \quad \textbf{if} \quad w$$
$$\textbf{always} \quad n \mathbin{\dot{\vee}} e \mathbin{\dot{\vee}} w \mathbin{\dot{\vee}} s$$
$$\textbf{always} \quad \dot{\bigvee} p(X,Y)$$

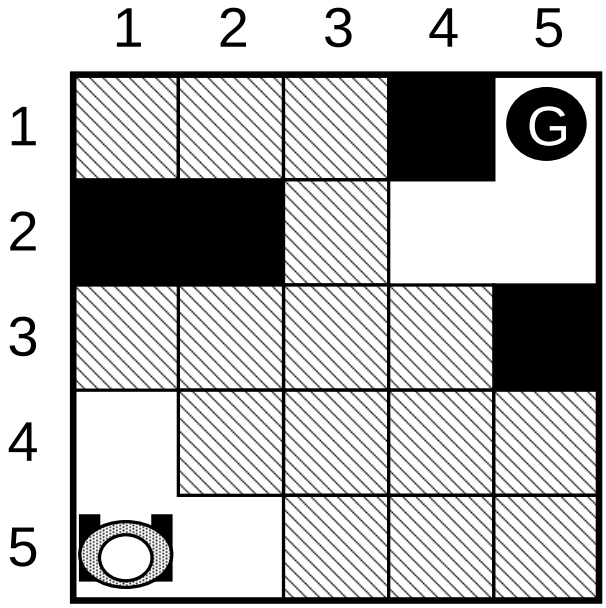

図 4.4: 迷路を探索するロボット

ここで，演算子 $\dot{\vee}$ は排他性を表す演算子で，次式で定義される：

$$\phi_1 \dot{\vee} \cdots \dot{\vee} \phi_k \equiv (\phi_1 \vee \cdots \vee \phi_k) \wedge \bigwedge_{1 \leq i \neq j \leq k} \neg(\phi_i \wedge \phi_j)$$

例えば，制約 **always**$\dot{\bigvee} p(X, Y)$ は，ロボットが常に 1 つの場所にしか存在しないことを宣言している．

迷路（図 4.4）に関する知識は，次のような記述になる．

initially $p(1, 5)$

initially $\neg obstacle(1, 4) \wedge \neg obstacle(2, 5) \wedge \neg obstacle(5, 2) \wedge \neg obstacle(4, 2)$

initially $obstacle(1, 2) \wedge obstacle(2, 2) \wedge obstacle(4, 1) \wedge obstacle(5, 1)$

これは，探索開始以前にすでに分かっている迷路の形状に関する知識である．これ以外の部分に関する知識（図中の灰色の部分）は，まだ分かっていない．

アクション $sense$ により，前方に障害物があるかどうかを検出することができる．検出した結果は，障害物が存在する，または存在しない $(obstacle \mid \neg obstacle)$ である．もし障害物が存在しなければ，前進 $(forward)$ することができ，その時検出した結果は，知識

ベースに蓄えられていく．もし障害物にまったく出会わなければ，最短のプランとして

$$
\begin{array}{ll}
forward; sense; forward; right; & \cdots \quad p(1,3) \text{ まで移動} \\
sense; forward; sense; forward; sense; forward; & \cdots \quad p(4,3) \text{ まで移動} \\
left; forward; right; forward; left; forward & \cdots \quad p(5,1) \text{ まで移動}
\end{array}
$$

が得られる．

このように非決定性を用いることでロボットのセンサーなどの興味深い分野にアクション言語を適用することができるようになる．またこの例で示したように，環境に関する知識が不完全な状況であったとしても，非決定性を利用することで，プランニングなどの推論を行うことが可能になる．

4.6 $\mathcal{AR}$ と $\mathcal{NA}$

アクション言語 $\mathcal{AR}$ [11] は，$\mathcal{A}$ を基にして拡張された言語であり，非決定的効果をもつアクションや間接的効果をもつアクションを記述できる．言語 $\mathcal{NA}$ における間接的効果の記述方法は，$\mathcal{AR}$ のものを参考にしている．言語 $\mathcal{AR}$ の特徴を以下に示す．これらは $\mathcal{NA}$ にはない特徴である．

- 慣性に従うフルーエントと従わないフルーエント（**非慣性フルーエント**）がある．また，フルーエントは，真偽の 2 値だけでなく，複数の値をとることができる．
- アクションの非決定的効果は，命題 **possibly changes** により記述する．遷移後の状態は，極小変化の概念により定義される．

逆に，$\mathcal{AR}$ にはないが，$\mathcal{NA}$ が持っている特徴として，

- アクションの非決定的効果は，効果命題において縦棒演算子 (||) により分けて記述する．
- $\mathcal{NA}$ では記述できるが，$\mathcal{AR}$ では直接記述できない領域が存在する[2]．

本節では，言語 $\mathcal{AR}$ がこれらの特徴を持つのにもかかわらず，$\mathcal{NA}$ による等価な領域記述に変換できることを示す．言語 $\mathcal{AR}$ については，第 2.2 節を参照のこと．

[2]ある領域を直接記述できるとは，その領域で使用されているフルーエント名を変更せずに記述できることをいうものとする．

4.6.1 $\mathcal{AR}$ から $\mathcal{NA}$ へ

言語 $\mathcal{AR}$ による領域記述を $D_{\mathcal{AR}}$ で表し，$\mathcal{NA}$ による領域記述を $D_{\mathcal{NA}}$ で表す．$D_{\mathcal{AR}}$ から $D_{\mathcal{NA}}$ への変換は，次の４つの段階を経て行なわれる．ここで，領域記述に含まれるフルーエント名，アクション名，命題の数，フルーエントの値域は有限であると仮定する．

(1) **非慣性フルーエントの処理**

$D_{\mathcal{AR}}$ に含まれる全ての非慣性フルーエントの集合を $\mathcal{F}_N$，全てのアクション名の集合を $\mathcal{A}$ とする．全ての $F \in \mathcal{F}_N, a \in \mathcal{A}$ に対し，次の命題

a **possibly changes** F

を $D_{\mathcal{AR}}$ に加える．これ以後，$D_{\mathcal{AR}}$ に含まれるフルーエントは，すべて慣性フルーエントとして扱う．

(2) **多値フルーエントの処理**

$D_{\mathcal{AR}}$ に含まれる全てのフルーエント名 F に対し，F が真／偽以外の値を持つフルーエントならば，次の操作を行なう．ここで $Rng_F = \{V_1, V_2, \cdots, V_n\}$ とする．$D_{\mathcal{AR}}$ に含まれるすべての原子式 F **is** V_i を，真／偽の値を持つフルーエント F_{V_i} で置き換える．さらに $D_{\mathcal{AR}}$ に次の制約を加える：

always $F_{V_1} \dot{\vee} F_{V_2} \dot{\vee} \cdots \dot{\vee} F_{V_n}$

(3) **possibly changes 命題と効果命題の処理**

$D_{\mathcal{AR}}$ に含まれる全てのアクション名 a について，次の処理を施す．まず，a に関する効果命題と **possibly changes** 命題を集める：

a **causes** ϕ_1 **if** ψ_1

$\vdots$

a **causes** ϕ_n **if** ψ_n

a **possibly changes** F_1 **if** θ_1

$\vdots$

a **possibly changes** F_m **if** θ_m

これらの命題から構成される領域記述を D とする．D の全ての状態 s に対し，

$$Res_D(a, s) = \{s_1, \cdots, s_k\}$$

を求め，次の命題を D_{NA} に加える：

$$a \textbf{ causes } c(s_1 \setminus s) \mid \cdots \mid c(s_k \setminus s) \textbf{ if } c(s) \tag{4.4}$$

ただし，$t = \{F_1, \cdots, F_l\}$ に対し，$c(t) = F_1 \wedge \cdots \wedge F_l$ とする．

(4) **評価命題と制約の処理**

$D_{\mathcal{AR}}$ に含まれる評価命題と制約は，$D_{\mathcal{NA}}$ でもそのまま利用できるので，すべて $D_{\mathcal{NA}}$ にコピーする．

この手続きの正当性を示す．

定理 11 言語 $\mathcal{AR}$ による領域記述を $D_{\mathcal{AR}}$ とする．$D_{\mathcal{AR}}$ を，上の変換手続きにより，言語 $\mathcal{NA}$ による領域記述 $D_{\mathcal{NA}}$ に変換する．このとき，

$$D_{\mathcal{AR}} \models \phi \textbf{ after } a_1; \cdots; a_n \quad \Leftrightarrow \quad D_{\mathcal{NA}} \models \phi \textbf{ after } a_1; \cdots; a_n$$

証明 各ステップについて証明の概略を示す．

(1) 言語 $\mathcal{AR}$ による領域記述を D とし，ステップ 1 の処理を施した後の領域記述を D' とする．このステップの正当性の証明は，任意のアクション名 a と任意の状態 σ に対し，$Res_D(a, \sigma) = Res_{D'}(a, \sigma)$ を示すことである．

遷移関数 Res_D は，遷移前と遷移後の慣性フルーエントの変化を極小にするように定義されている．従って，遷移前の状態 σ に含まれる任意の非慣性フルーエント F は，遷移後の状態 $\sigma' \in Res_D(a, \sigma)$ において任意の値 $V \in Rng_F$ をとることができる．

D' において F は，**possibly changes** 命題に変換され，さらに慣性フルーエントとして扱われる．ここで，(a) $Res^0_D(a, \sigma) = Res^0_{D'}(a, \sigma)$ であることと，(b) New^a_D の定義の条件 (2) より， F は，遷移関数 $Res_{D'}$ においても，極小化の対象にはならない．従って F は，遷移後の状態 $\sigma' \in Res_{D'}(a, \sigma)$ において，任意の値 $V \in Rng_F$ をとることができる．

(2) $\mathcal{AR}$ におけるフルーエントは，複数の値を持つことができるが，特に変数が利用できるわけではなく，それらは命題的に振舞っている．従って，このような書き換えを行なっても，モデルが本質的に変化することはない．

(3) 言語 $\mathcal{AR}$ による領域記述を D とし，ステップ 3 の処理を施して得られる言語 $\mathcal{NA}$ による領域記述を D' とする．この時点で D に含まれるフルーエントは，すべて慣性フルーエントであり，それらは真／偽の値を持つ命題的フルーエントである．従って D における状態を原子式の集合ではなく，$\mathcal{NA}$ のように，全てのフルーエント名について，正または負のフルーエントのいずれかを含む集合と考えることができる．ここでの正当性の証明は，任意のアクション名 a と任意の状態 σ に対し，$Res_D(a,\sigma) = \delta(\sigma,a)$ を示すことである．ここで δ は D' の正当な遷移関数である．

$Res_D(a,\sigma) = \{\sigma_1,\cdots,\sigma_n\}$ とすれば，式 (4.4) と $\mathcal{NA}$ の遷移関数の定義より，$\delta(\sigma,a) = \{\sigma_1,\cdots,\sigma_n\}$ を示すことができる．

(4) 言語 $\mathcal{AR}$ においても，言語 $\mathcal{NA}$ においても，制約は，状態を同様に制限するだけである．また，これまでのステップで両領域の遷移関数が等しいことが分かっている．従って，同じ評価命題をもつ両領域のモデルは等しい．□

定理 11 より，言語 $\mathcal{AR}$ による任意の領域記述は，あらゆる評価命題の帰結関係において等価な言語 $\mathcal{NA}$ による記述に変換できることが分かる

4.6.2 変換例

言語 $\mathcal{AR}$ で記述した例題 3 を，前節の変換手続きを用いて，言語 $\mathcal{NA}$ による記述に戻してみる．

$Jump$ **causes** $Lake \wedge Hat \mid Lake \wedge \neg Hat$ **if** $\neg Lake \wedge Hat$
$Jump$ **causes** $Lake$ **if** $\neg Lake \wedge \neg Hat$
$GetOut$ **causes** $\neg Lake$ **if** $Lake$
$PutOn$ **causes** Hat **if** $\neg Hat$
impossible $Jump$ **if** $Lake \wedge Hat$
impossible $Jump$ **if** $Lake \wedge \neg Hat$
impossible $GetOut$ **if** $\neg Lake$
impossible $PutOn$ **if** Hat
always $Lake \supset Wet$
initially $\neg Lake \wedge \neg Wet \wedge Hat$

元々の領域記述と比べると多少冗長な記述になっているが，この領域記述は図 4.1 と同じモデルをもつ．

4.6.3 $\mathcal{NA}$ から $\mathcal{AR}$ へ

言語 $\mathcal{NA}$ による任意の領域記述は，以下のようにして，それと等価な言語 $\mathcal{AR}$ による記述に変換することができる．

言語 $\mathcal{NA}$ による領域記述を $D_{\mathcal{NA}}$ とする．定理 10 による変換を用いて，$D_{\mathcal{NA}}$ をそれと等価な FA M に変換し，さらに M を DFA M' に等価変換する [14]．第 3.3 節の手続きにより，M' を 言語 $\mathcal{A}$ による記述に変換したものを $D_{\mathcal{A}}$ とする．このとき，

定理 12 任意の評価命題 ϕ **after** $a_1; \cdots ; a_n$ に対して，

$$D_{\mathcal{NA}} \models \phi \textbf{ after } a_1; \cdots ; a_n \quad \Leftrightarrow \quad D_{\mathcal{A}} \models \phi \textbf{ after } a_1; \cdots ; a_n$$

証明 定理 10 および定理 6 を用いて証明できる．□

言語 $\mathcal{A}$ は言語 $\mathcal{AR}$ のサブセットであるので，$\mathcal{A}$ による記述は，$\mathcal{AR}$ による記述でもある．従って，$\mathcal{NA}$ による任意の領域記述は，あらゆる評価命題の帰結関係において等価な言語 $\mathcal{AR}$ による記述に変換できる．

定理 11 と定理 12 より，$\mathcal{NA}$ で表現できる領域のクラスと，$\mathcal{AR}$ で表現できる領域のクラスとは等しいことが分かる．このように，$\mathcal{NA}$ は，他のアクション言語の適用範囲を形式的に議論するのに利用できる．

ところで，言語 $\mathcal{NA}$ では直接記述できるが，言語 $\mathcal{AR}$ では直接記述できない領域が存在する[3]．以下でそのような領域が存在することを示す．

言語 $\mathcal{AR}$ による領域記述を D とする．ここで，次のような遷移を考える：

$$Res_D(\sigma_1, a) = \{\sigma_1, \sigma_2\} \tag{4.5}$$

$f, g \in \sigma_1,\ \neg f, \neg g \in \sigma_2$ とする．もし D が a に関する **possibly changes** 命題を含んでいないと仮定すれば，極小変化の定義より，遷移先に σ_2 が含まれることはない．よって仮定は矛盾する．従って D は，次のような命題を含んでいる必要がある：

$$\begin{array}{l} a \textbf{ possibly changes } f \textbf{ if } \psi \\ \textbf{always } \neg f \Rightarrow \neg g \end{array} \tag{4.6}$$

[3]ある領域を直接記述できるとは，その領域で使用されているフルーエント名を変更せずに記述できることをいうものとする．

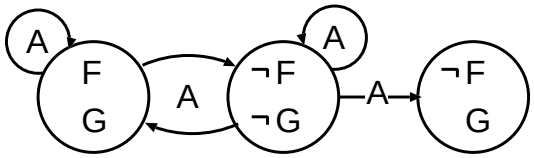

図 4.5: $\mathcal{AR}$ では直接記述できない例

または

$$\begin{aligned} &a \textbf{ possibly changes } f \textbf{ if } \psi \\ &a \textbf{ possibly changes } g \textbf{ if } \psi \\ &\textbf{always } (f \wedge g) \vee (\neg f \wedge \neg g) \end{aligned} \tag{4.7}$$

ここで ψ は状態 σ_1 で真となる前提条件である．ここでさらに次のような遷移を考える：

$$Res_D(\sigma_2, a) = \{\sigma_1, \sigma_2, \sigma_3\}$$

$\neg f, g \in \sigma_3$ とする．しかし，D は (4.6) または (4.7) 式の命題を含んでいるので，σ_2 から σ_3 へ遷移することはできない．逆に，D が (4.6) または (4.7) 式の命題を含んでいない場合，(4.5) 式の遷移を実現できない．すなわち，言語 $\mathcal{AR}$ では直接記述できない領域が存在する．ところで，言語 $\mathcal{NA}$ では次のように直接記述できる：

$$\begin{aligned} &a \textbf{ causes } (f \wedge g) \mid (\neg f \wedge \neg g) \textbf{ if } f \wedge g \wedge \psi \\ &a \textbf{ causes } (f \wedge g) \mid (\neg f \wedge \neg g) \mid (\neg f \wedge g) \textbf{ if } \neg f \wedge \neg g \wedge \psi' \end{aligned}$$

ここで ψ' は状態 σ_2 で真となる前提条件である．

4.7 関連研究

言語 $\mathcal{AR}$ と同じ極小変化の概念を持つアクション言語に，Lifschitz の DL_{if} [22] や Turner の $\mathcal{AC}$ [39] がある．$\mathcal{AC}$ は $\mathcal{AR}$ の拡張版として提案されている．DL_{if} も $\mathcal{AC}$ も，非決定性アクションや制約を記述することができる．さらに，これらの言語では，状態間の依存関係を表現することができる：

lake **suffice for** *wet*

これは状態に対する規則として解釈される．制約 **always** $lake \supset wet$ とは異なり，対遇をとることはできない．

アクションの非決定的効果の取扱い方には，いくつかのアプローチがある．上に挙げた極小変化の概念に基づくアプローチもその１つであり，言語 $\mathcal{NA}$ のように排他的に効果を記述するアプローチもその１つである．Boutilier と Friedman による言語 $\mathcal{A}^{\mathrm{ND}}$ も $\mathcal{NA}$ と同様の縦棒演算子を用いて非決定的効果を排他的に記述している [6]．ただし $\mathcal{A}^{\mathrm{ND}}$ では，その解釈をプロセス論理の一種で与えており，また効果命題に対し，アクションの前提条件が相互に充足してはならないという強い制限を課している．Borncheuer と Thielscher による $\mathcal{A}_N$ も，非決定的効果を複数の効果命題を排他的に用いて記述する [5]：

$$jump \quad \textbf{alternatively caused} \quad lake \wedge hat$$
$$jump \quad \textbf{alternatively caused} \quad lake \wedge \neg hat$$

さらに彼らは，$\mathcal{A}_N$ を同時発生アクションを扱えるように拡張し ($\mathcal{A}_{NCC}$)，それを論理プログラムに変換する完全で健全な手続きを示している．$\mathcal{A}^{\mathrm{ND}}$ や $\mathcal{A}_N$ による記述を $\mathcal{NA}$ による記述に変換することは，非決定性の取扱い方が似ていることから，比較的容易であると考えられる．

Baral と Gelfond による $\mathcal{A}_C$ もまた同時発生アクションを扱うことができる [3]．彼らは，$\mathcal{A}_C$ による記述を拡張論理プログラムに変換する完全で健全な手続きを示している．同時発生アクションを表現できるよう $\mathcal{NA}$ を拡張することも，今後の重要な課題の１つである．

4.8 まとめ

本章では，NFA に基づく新しい非決定性アクション言語 $\mathcal{NA}$ を提案した．

$\mathcal{NA}$ は FA と同じ表現力を持つ．$\mathcal{NA}$ における２種類のプランニング（必ず目標を達成するようなプランを求める場合と，目標を達成する可能性のあるプランを求める場合）は，$\mathcal{NA}$ と FA との等価性を利用することで，前者の場合は NFA の受理言語を求める問題に帰着し，後者の場合は DFA の受理言語を求める問題に帰着することを示した．

また非決定性を用いて障害物などを検出するセンサーを定式化できることを示し，環境に関する知識が不完全な状況であったとしても，プランニングなどの推論を行うことが可能になることを示した．

さらに言語 $\mathcal{NA}$ を利用することで，他のアクション言語の適用範囲を形式的に議論することが可能になる．実際に我々は，極小変化の概念に基づく非決定性をもつ言語 $\mathcal{AR}$ が，言語 $\mathcal{NA}$ と等価な表現力を持つことを示した．

第5章　アクション言語処理系 AMP

ここ数年プランニングアルゴリズムの研究は大きな進展をみせている．第 2.3 節で紹介したように，特に SAT プランニングと呼ばれる手法が注目を集め，現在多くの研究が行われている．

その一方で，状態変化やアクションを記述するための知識表現の研究も活発に行われている．最近用いられているアプローチの１つが高級レベルのアクション言語である．Gelfond と Lifschitz がアクション言語 $\mathcal{A}$ を提案して以来，$\mathcal{A}$ を拡張し，さらなる表現力を求めた様々なアクション言語が提案されている [10]．アクション言語は状態変化領域を記述するための言語であるが，その記述に対してプランやモデルを問い合わせるための言語でもある．しかしながら，アクション言語の処理系に関する研究はそれほど進んでいるとは言えない．

本章では，これら２つのアプローチを統合させる．すなわち， SAT プランニングの技術が，プランニングだけでなく，モデル生成などのアクション言語の他の特徴にも有効であることを示す．この目的のために我々は，プランニンググラフと SAT ソルバによる高速プランニングアプローチに基づいた**アクション言語処理系** AMP を実装した [33]．AMP は入力として 言語 $\mathcal{A}$ による記述を受け取り，プランニングやモデル生成（初期状態の推定），実行結果の予測などの問い合わせに高速に応答する．

アクション言語におけるプランニングは，Graphplan や `Blackbox` における STRIPS 形式のプランニング問題よりも困難である．なぜならアクション言語による領域記述には評価命題が含まれているからである．言語 $\mathcal{A}$ の評価命題は，ある初期状態においてアクション列を実行した後の状態について述べる “観測された事実” の宣言である．$\mathcal{A}$ におけるプランニングは，初期状態の完全な記述が与えられる代わりに，しばしばそのような評価命題と目標条件により定義される．従ってそのような場合にプランを求めるためには，まず初期状態を推定する必要がある．この初期状態の推定は AMP の特徴の 1 つである．我々はこの初期状態の推定を**モデル生成**と呼ぶ．

AMP は Java 言語で実装されている．`Blackbox` など他の高速なプラン生成器の多くは

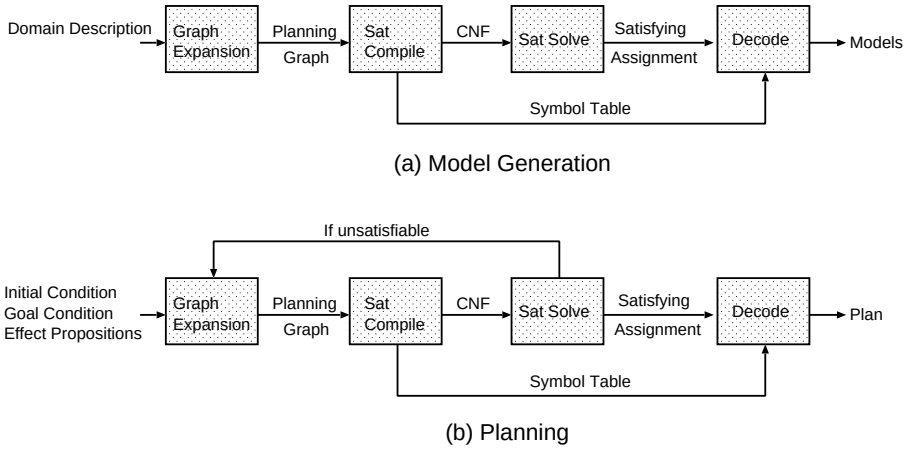

図 5.1: AMP のアーキテクチャ

C 言語で実装されているため，AMP は速度面で少し劣る．しかしながら Java 言語は，高い拡張性・再利用性，プラットフォーム独立などの特徴をもっており，AMP の各モジュールは別のプログラムに容易に組み込むことができる．また AMP は，プランニングやモデル生成で用いられるプランニンググラフを表示する GUI (graphical user interface) をもっている．

5.1 AMP

アクション言語処理系 AMP のアーキテクチャを図 (5.1) に示す．領域記述 D が与えられると，AMP はまず D のモデル集合を求める（図 5.1 (a)）．モデル集合生成後，AMP は，**シミュレーション・プランニング・実行結果の予測**という 3 種類の問い合わせに答えることができる：

$$D \models \phi \textbf{ after } a_1; \ldots; a_m \tag{5.1}$$

$$D \models \psi \textbf{ after } X \tag{5.2}$$

$$D \models X \textbf{ after } a_1; \ldots; a_m \tag{5.3}$$

式 (5.1) は，アクション列 $a_1; \ldots; a_n$ を実行後，ϕ が成立するかどうかという問い合わせである．この形式の問い合わせを**シミュレーション**と呼ぶ．式 (5.2) は，目標条件 ψ を達成するようなアクション列 X の問い合わせ，すなわち**プランニング**である．式 (5.3) は，アクション列 $a_1; \ldots; a_n$ を実行後，どのようなフルーエントが成立するかという問い合わ

せである．この形式の問い合わせを**実行結果の予測**と呼ぶ．プランニングの場合，AMP は図 5.1 (b) の処理を実行する．その他の問い合わせの場合はモデルを用いることで簡単に答えることができるので省略する．

まずモデル生成アルゴリズムを述べるための諸定義を行なう．

5.2 諸定義

部分解釈の概念を導入する．**部分解釈** I' は，遷移関数 Φ と部分状態 σ'（第 3.2.1 節参照）で定義される：

$$I' = (\Phi, \sigma')$$

部分解釈は解釈の集合を表している．例えば部分解釈 $(\Phi, \{loaded\})$ は，$(\Phi, \{loaded,\ alive\})$ と $(\Phi, \{loaded, \neg alive\})$ という解釈を表している．ある部分解釈 I' が表すすべての解釈が D のモデルとなるとき，I' を**部分モデル**と呼ぶ．AMP は部分モデルを出力する．

本章では，言語 $\mathcal{A}$ の効果命題を以下の形式に制限する：

$$a \textbf{ causes } \vec{e} \textbf{ if } \vec{p} \tag{5.4}$$

ここで a はアクション名，$\vec{e}, \vec{p}$ はフルーエントの連言である．このような制限を施しても言語 $\mathcal{A}$ の表現力が変わらないことは，第 3.1 節で示した[1]．

言語 $\mathcal{A}$ では，あるアクション名 a に関する効果命題を複数記述することができる：

$$\begin{aligned} &a \textbf{ causes } \vec{e}_1 \textbf{ if } \vec{p}_1, \\ &\qquad \vdots \\ &a \textbf{ causes } \vec{e}_n \textbf{ if } \vec{p}_n. \end{aligned}$$

これらの効果命題を区別するためにアクション名を変更し，あるアクション名に関する効果命題が唯 1 つになるようにする：

$$\begin{aligned} &a_1 \textbf{ causes } \vec{e}_1 \textbf{ if } \vec{p}_1, \\ &\qquad \vdots \\ &a_n \textbf{ causes } \vec{e}_n \textbf{ if } \vec{p}_n. \end{aligned} \tag{5.5}$$

[1]式 (5.4) ではアクションの効果はフルーエントの連言で表されている．しかし表現力が変わらないことは同様に示すことがきる．

AMP が出力するプランは**半順序プラン** (partially ordered plan) であるため，このような変換を施しても，簡単に言語 $\mathcal{A}$ のアクション列に戻すことができる（第 5.4 節参照）．式 (5.5) の効果命題を，**アクション名 a に関する効果命題**と呼ぶ．

AMP におけるプランニンググラフの定義は，第 2.3.2 節で示した Graphplan における定義に等しい．ただし，プランニンググラフにおける相互排他関係の定義が若干異なり，例外的な条件が追加される．このことを厳密に示すために，以下で AMP における相互排他関係を再定義する．

相互排他関係 (mutual exclusion relation; mutex) を次のように再帰的に定義する：

- **レベル i の 2 つのアクションが相互排他関係にある**のは，

  1. 片方の効果が他方の効果を否定する場合 (inconsistent effects)，または，
  2. 片方の効果が他方の前提条件を否定する場合 (interference)，または，
  3. 2 つのアクションの前提条件が，レベル $i-1$ で相互排他関係にある場合 (competing needs)．

  ここで，アクション名 a に関する効果命題が複数存在した場合，それらについては上記 2 の規則を適用しない．これは，言語 $\mathcal{A}$ ではそのような効果命題について，片方の効果が他方の前提条件を否定することを認めているからである．例えば，次の領域記述において，

  $$\mathit{shoot} \textbf{ causes } \neg \mathit{alive} \textbf{ if } \mathit{loaded}$$
  $$\mathit{shoot} \textbf{ causes } \neg \mathit{loaded}$$

  言語 $\mathcal{A}$ は $\Phi(\mathit{shoot}, \{\mathit{loaded}, \mathit{alive}\}) = \{\neg \mathit{loaded}, \neg \mathit{alive}\}$ という遷移を認める．

- **レベル i の 2 つのフルーエントが相互排他関係にある**のは，

  1. 片方が他方の否定である場合，または，
  2. 片方を導いたレベル $i-1$ のすべてのアクションが，他方を導いたレベル $i-1$ のすべてのアクションと相互排他関係にある場合 (inconsistent support)．

5.3 モデル生成

モデル生成アルゴリズムは，入力として言語 $\mathcal{A}$ による領域記述 D を受け取り，そのモデル集合 $\mathbb{I}$ を出力する．もし D が次の形式の評価命題を**含まない場合**，

$$\phi \textbf{ after } a_1;\ldots;a_m.\ \ (m \geq 1) \tag{5.6}$$

D のモデル集合 $\mathbb{I}$ を求めることは容易である．すなわち，

$$\mathbb{I} = \{(\Phi, \sigma_0')\}, \qquad \text{where } \ \sigma_0' = \bigcup_{(\textbf{initially } f_1 \wedge \cdots \wedge f_l) \in D} \{f_1, \ldots, f_l\}. \tag{5.7}$$

ここで遷移関数 Φ は，D に含まれる効果命題から簡単に求めることができる．(Φ, σ_0') は，D のすべてのモデルを表す部分モデルである．

もし D が式 (5.6) の形式の評価命題を**含む場合**は，モデル生成アルゴリズムの核である以下の手続きを適用する．

以下に示す手続きの流れを簡単に説明する．この手続きは，まず，式 (5.6) の評価命題を，初期状態から実行したときの実行経過を表すプランニンググラフを構成する．もちろんこの時点では，初期状態は明らかではない．従って，真偽値が不明なフルーエント（f とする）を前提条件に持つアクションをグラフに追加する際は，すべての可能性を考慮するため，グラフに f と $\neg f$ の両方を追加する．プランニンググラフでは，あるフルーエントレベルに f と $\neg f$ の両方が存在してもよい．なぜなら，f と $\neg f$ との間には相互排他関係があり，その相互排他関係により，“同時に成立することがないフルーエント”として識別できるからである．

このようにして作成されたプランニンググラフは，任意の初期状態からアクション列 $a_1;\ldots;a_m$ を実行した場合のあらゆる状態遷移の経過を内包している．式 (5.6) は，“アクション列 $a_1;\ldots;a_m$ を実行後，ϕ が成立した”ということを主張しているので，ϕ を成立させるような状態遷移のみを抽出する必要がある．我々は，プランニンググラフを SAT 問題に変換し，SAT ソルバを用いてそのような状態遷移を抽出する．

モデル生成アルゴリズムの手続きを厳密に説明する．D に含まれる式 (5.6) の形式の評価命題の集合を $\mathcal{V}$ とし，$D' = D \setminus \mathcal{V}$ とする．まず，すべての評価命題 $V \in \mathcal{V}$ に対し，V を充足するモデル集合 $\mathbb{I}_V$ を，以下の 4 つのアルゴリズムを用いて求める：

$$\mathbb{I}_V = \textbf{ModelExtract}(D', \textbf{SatSolveAll}(\textbf{SatCompile}^M(\textbf{Expand}^M(D', V)))).$$

ここで，それぞれのアルゴリズムは次の働きをもつ：

(1) D' と V からプランニンググラフ G_V を作成し（図 5.2），

$$G_V = \mathbf{Expand}^M(D', V)$$

(2) G_V を SAT 問題 P_V に変換し（図 5.5），

$$P_V = \mathbf{SatCompile}^M(G_V)$$

(3) P_V のモデル集合 $\mathbb{J}_V$ を求め，

$$\mathbb{J}_V = \mathbf{SatSolveAll}(P_V)$$

(4) $\mathbb{J}_V$ から V を充足するモデル集合 $\mathbb{I}_V$ を抽出する（図 5.6）．

$$\mathbb{I}_V = \mathbf{ModelExtract}(D', \mathbb{J}_V)$$

アルゴリズム **SatSolveAll** は，命題論理の充足可能性問題のモデル集合を求めることのできる SAT ソルバであれば何でもよい．現在 AMP では，体系的 SAT ソルバでは最速のソルバの 1 つである Satz [20] を使用している．

各評価命題を充足するモデル集合の共通部分をとれば，それが求める D のモデル集合 $\mathbb{I}$ である．すなわち，

$$\mathbb{I} = \bigcap_{V \in \mathcal{V}} \mathbb{I}_V.$$

である．

Expand$^M(D, V)$
入力：領域記述 D，評価命題 $V = (\phi$ **after** $a_1; \ldots; a_n)$
出力：V に対応するプランニンググラフ G
begin
  初期条件を $\{f_1, \ldots, f_n\}$ とし，目標条件を $\phi = \{g_1, \ldots, g_m\}$ とする
  G を空のプランニンググラフとする
  $G =$ **AddFluents**$(G, \{f_1, \ldots, f_n, \neg f_1, \ldots, \neg f_n\}, 0)$
  $G =$ **AddFluents**$(G, \{g_1, \ldots, g_m, \neg g_1, \ldots, \neg g_m\}, n)$
  for $i = 1$ **to** n
    $\vec{p} = a_i$ の前提条件
    /* グラフ中に存在しないフルーエントを追加 */
    for all $p \in \vec{p}$
      if $p \notin L_{2i-2}$ **and** $\neg p \notin L_{2i-2}$ **then**
        $G =$ **AddFluents**$(G, \{p, \neg p\}, 0)$
        for $j = 1$ **to** $i - 1$
          $G =$ **AddAction**$(G, no\text{-}op(p), 2j - 1)$
          $G =$ **AddAction**$(G, no\text{-}op(\neg p), 2j - 1)$
        end
      end
    end
    /* アクション a_i を追加 */
    $G =$ **AddAction**$(G, a_i, 2i - 1)$
    /* *no-op* アクションを追加 */
    for all $f \in L_{2i-2}$
      $G =$ **AddAction**$(G, no\text{-}op(f), 2i - 1)$
    end
  end
end.

図 5.2: モデル生成用プランニンググラフ展開アルゴリズム **Expand**^{M}

AddFluent(G, f, i)
入力：プランニンググラフ G，フルーエント f，追加するレベル i
出力：フルーエントを追加したグラフ G
begin
  $L_i = L_i \cup \{f\}$
  for all $f' \in L_i$
    if $f' \neq f$ **then**
      f と f' との相互排他関係をチェック
  end
end.

AddFluents$(G, \mathcal{F}, i)$
入力：プランニンググラフ G，フルーエント集合 $\mathcal{F}$，追加するレベル i
出力：フルーエント集合を追加したグラフ G
begin
  for all $f \in \mathcal{F}$
    $G =$ **AddFluent**(G, f, i)
  end
end.

図 5.3: プランニンググラフにフルーエントを追加するアルゴリズム **AddFluent**

AddAction(G, a, i)
入力：プランニンググラフ G，アクション a，追加するレベル i
出力：アクションを追加したグラフ G
begin
  $\vec{p} = a$ の前提条件
  $\vec{e} = a$ の効果
  if $\vec{p} \subseteq L_{i-1}$ **and** L_{i-1} において $\vec{p}$ の任意の 2 要素間に相互排他関係がない **then**
    $L_i = L_i \cup \{a\}$
    for all $a' \in L_i$
      if $a' \neq a$ **then**
        a と a' との相互排他関係をチェック
    end
    AddFluents$(G, \vec{e}, i+1)$
  end
end.

図 5.4: プランニンググラフにアクションを追加するアルゴリズム **AddAction**

SatCompile$^M(G)$

入力：プランニンググラフ G

出力：SAT 問題

begin

初期条件を $\{f_1, \ldots, f_n\}$ とし，目標条件を $\{g_1, \ldots, g_m\}$ とする

$l = G$ の最大レベル

G を以下の規則に従って SAT 問題に変換する

(1) 初期条件はレベル 0 において真であり，目標条件はレベル l において真である

$f_1^0 \wedge \ldots \wedge f_n^0 \wedge g_1^l \wedge \ldots \wedge g_m^l$

(2) レベル i において相互排他関係にあるアクション a, b は同時に成立しない

$\neg a^i \vee \neg b^i$

(3) レベル i のアクション a は，その前提条件 $p_1, \ldots, p_k$ を含意する

$a^i \supset p_1^{i-1} \wedge \ldots \wedge p_k^{i-1}$

(4) レベル i のフルーエント f は，f を効果として持つレベル $i-1$ のすべての
アクション $a_1, \ldots, a_k$ を含意する

$f^i \supset a_1^{i-1} \vee \ldots \vee a_k^{i-1}$

(5) *no-op* 以外のレベル i のアクション a は，前提条件 $p_1, \ldots, p_k$ が真であれば
実行されなければならない

$p_1^{i-1} \wedge \ldots \wedge p_k^{i-1} \supset a^i$

end.

図 5.5: モデル生成用 SAT 変換アルゴリズム **SatCompile**^{M}

ModelExtract(D, J)
入力：領域記述 D，SAT 問題のモデル J
出力：領域記述のモデル (Φ, σ_0)
begin
　領域記述に含まれる効果命題から遷移関数 Φ を求める
　$\sigma_0 = \{p^x \in J \mid x = 0\}$
　return (Φ, σ_0)
end.

ModelExtract$(D, \mathbb{J})$
入力：領域記述 D，SAT 問題のモデル集合 $\mathbb{J}$
出力：領域記述のモデル集合 $\mathbb{I}$
begin
　領域記述に含まれる効果命題から遷移関数 Φ を求める
　for all $J \in \mathbb{J}$
　　$\mathbb{I} = \mathbb{I} \cup$ **ModelExtract**(J)
　end
end.

図 5.6: モデル抽出アルゴリズム **ModelExtract**

5.3.1 モデル生成アルゴリズムの正当性

モデル生成アルゴリズムの正当性は，次の定理により示される．

定理 13 任意の数の効果命題と次の評価命題

$$\textbf{initially}\ \phi_0$$

を含む領域記述を D とする．次の評価命題を V とする．

$$V = (\phi_n\ \textbf{after}\ a_1; \ldots; a_n)$$

$D \cup \{V\}$ のモデル集合を $\mathbb{I}$ とする．モデル生成アルゴリズムより求まる SAT 問題を

$$P = \textbf{SatCompile}^M(\textbf{Expand}^M(D, V))$$

とし，P のモデル集合を $\mathbb{J}$ とする．このとき,

$$\textbf{Extract}(\mathbb{J}) = \mathbb{I}$$

である．

証明 付録 A.1 参照.

5.3.2 例題

次の例題のモデル集合を求めてみる．

例題 7 Yale Shooting 問題の亜種

$$load\ \textbf{causes}\ loaded.$$

$$shoot\ \textbf{causes}\ \neg alive \wedge \neg loaded\ \textbf{if}\ loaded.$$

$$\neg alive\ \textbf{after}\ load; shoot.$$

$$\textbf{initially}\ \neg loaded.$$

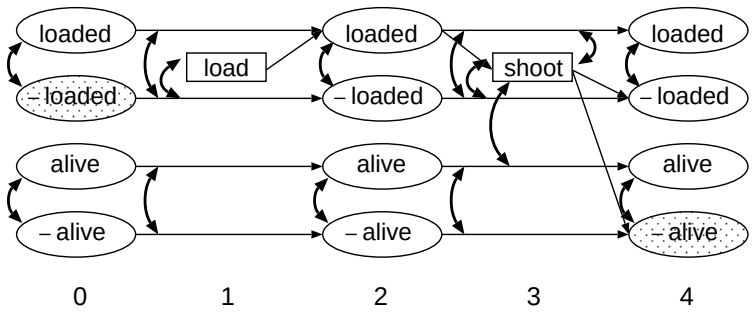

図 5.7: Yale Shooting 問題の亜種のプランニンググラフ

アルゴリズム $\mathbf{Expand}^M$ が出力したプランニンググラフを図 5.7 に示す．相互排他関係にある 2 つの頂点は，図中で弧で結ばれている．このプランニンググラフを SAT 問題に変換すると次式になる：

$$
\begin{aligned}
&\neg loaded^0 \wedge \neg alive^4 \\
&\wedge (\neg load^1 \vee \neg no\text{-}op(\neg loaded^1)) \\
&\wedge (\neg no\text{-}op(loaded)^1 \vee \neg no\text{-}op(\neg loaded)^1) \\
&\wedge (\neg no\text{-}op(alive)^1 \vee \neg no\text{-}op(\neg alive)^1) \\
&\wedge (\neg shoot^3 \vee \neg no\text{-}op(loaded)^3) \\
&\wedge (\neg shoot^3 \vee \neg no\text{-}op(\neg loaded)^3) \\
&\wedge (\neg shoot^3 \vee \neg no\text{-}op(alive)^3) \\
&\wedge (\neg no\text{-}op(loaded)^3 \vee \neg no\text{-}op(\neg loaded)^3) \\
&\wedge (\neg no\text{-}op(alive)^3 \vee \neg no\text{-}op(\neg alive)^3) \\
&\wedge load^1 \\
&\wedge (no\text{-}op(loaded)^1 \supset loaded^0) \\
&\wedge (no\text{-}op(\neg loaded)^1 \supset \neg loaded^0) \\
&\wedge (no\text{-}op(alive)^1 \supset alive^0) \\
&\wedge (no\text{-}op(\neg alive)^1 \supset \neg alive^0) \\
&\wedge (shoot^3 \supset loaded^2) \\
&\wedge (loaded^2 \supset shoot^3)
\end{aligned}
$$

$$\wedge\,(\mathit{no\text{-}op}(\mathit{loaded})^3 \supset \mathit{loaded}^2)$$
$$\wedge\,(\mathit{no\text{-}op}(\neg\mathit{loaded})^3 \supset \neg\mathit{loaded}^2)$$
$$\wedge\,(\mathit{no\text{-}op}(\mathit{alive})^3 \supset \mathit{alive}^2)$$
$$\wedge\,(\mathit{no\text{-}op}(\neg\mathit{alive})^3 \supset \neg\mathit{alive}^2)$$
$$\wedge\,(\mathit{loaded}^2 \supset \mathit{load}^1 \vee \mathit{no\text{-}op}(\mathit{loaded})^1)$$
$$\wedge\,(\neg\mathit{loaded}^2 \supset \mathit{no\text{-}op}(\neg\mathit{loaded})^1)$$
$$\wedge\,(\mathit{alive}^2 \supset \mathit{no\text{-}op}(\mathit{alive})^1)$$
$$\wedge\,(\neg\mathit{alive}^2 \supset \mathit{no\text{-}op}(\neg\mathit{alive})^1)$$
$$\wedge\,(\mathit{alive}^4 \supset \mathit{no\text{-}op}(\mathit{alive})^3)$$
$$\wedge\,(\neg\mathit{alive}^4 \supset \mathit{shoot}^3 \vee \mathit{no\text{-}op}(\neg\mathit{alive})^3)$$
$$\wedge\,(\mathit{loaded}^4 \supset \mathit{no\text{-}op}(\mathit{loaded})^3)$$
$$\wedge\,(\neg\mathit{loaded}^4 \supset \mathit{shoot}^3 \vee \mathit{no\text{-}op}(\neg\mathit{loaded})^3)$$

これを解くと，モデル集合

$$\mathbb{I} = \{(\Phi, \{\neg\mathit{loaded}, \mathit{alive}\}), (\Phi, \{\neg\mathit{loaded}, \neg\mathit{alive}\})\}$$

を得る．

5.4 プランニング

AMP のプランニングアルゴリズムは，Graphplan のグラフ展開アルゴリズムと，Blackbox のプラン抽出アルゴリズムを基にしている．ただし，次の点が異なる：

- 相互排他関係の定義が言語 $\mathcal{A}$ 特有の例外を含んでいる（第 5.2 節参照）
- プラン抽出における余分な探索を減らすため，プラン抽出前に，目標条件から後向きに到達可能な頂点のみを SAT 変換の対象にしている（図 5.9 参照）

AMP のプランニングアルゴリズム **SearchPlan** を図 5.8 に示す．ここでアルゴリズム **SatSolve** は，命題論理の充足可能性問題のモデルを少なくとも 1 つ求めることのできる SAT ソルバである．

アルゴリズム **SearchPlan** は，入力として言語 $\mathcal{A}$ による領域記述 D と，初期状態 ϕ_0，目標条件 ψ，探索するプランの最大長 l を受け取り，ψ を達成する長さ l 未満のプランが存在するならば，ψ を達成する最短のプランを出力する．

ここで示したアルゴリズム **SearchPlan** は，入力として初期状態を 1 つのみ受け取る．もし領域記述が複数のモデルを持つ場合，すなわち複数の初期状態が存在する場合は，次のようにして，“すべての初期状態から目標条件を達成可能なプラン” を求めることができる．

領域記述 D のすべてのモデルの初期状態の集合を $\Sigma_0 = \{\sigma_{0,1}, \cdots, \sigma_{0,n}\}$ とする．

(1) 任意に初期状態を 1 つ選ぶ．それを $\sigma_{0,i} \in \Sigma_0$ とする．残りの初期状態の集合を Σ_0' とする．

(2) アルゴリズム **SearchPlan** を用いてプラン X を求める：

$$X = \mathbf{SearchPlan}(D, \sigma_{0,i}, \psi, l)$$

プランが見つからなければ，明らかに，すべての初期状態から目標条件を達成可能なプランは存在しない．

(3) 残りのすべての初期状態 $\sigma_{0,j} \in \Sigma_0'$ に対し，プラン X が，$\sigma_{0,j}$ から実行を開始して，目標を達成できるかどうか調べる．

(4) もしすべての初期状態において目標を達成できたならば，それが求めるプランである．

(5) 目標を達成できない場合は，(2) に戻り，別のプランを探索する．

- このとき，まず SAT ソルバに別の SAT 問題の解を探索させる（SAT 問題の解は唯 1 つとは限らない）．
- それでもプランが見つからなければ，プランニンググラフをさらに展開する．

上記手続きを用いることで，領域記述の任意のモデルにおいて目標を達成できるプランを求めることができる．

SearchPlan(D, ϕ_0, ψ, l)
入力：領域記述 D，初期条件 ϕ_0，目標条件 ψ, 探索する最大プラン長 l
出力：プラン
begin
  G を空のプランニンググラフとする
  $G =$ **AddFluents**$(G, \phi_0, 0)$　　/* 初期レベルの作成 */
  $i = 0$
  forever do
    /* プラン抽出ステップ */
    if $\psi \subseteq L_{2i}$ **and** L_{2i} において ψ の任意の 2 要素間に相互排他関係がない **then**
      $P =$ **SatCompile**$^P(G)$
      $J =$ **SatSolve**(P)
      if $J \neq \emptyset$ **then**
        return **PlanExtract**(J)
      end
    end
    if $i \geq l$ **then**　　/* 停止条件 1 */
      return $\emptyset$
    end
    /* プランニンググラフ展開ステップ */
    for all $a \in action(D)$
      $\vec{p} = a$ の前提条件
      if $\vec{p} \subseteq L_{2i}$ **and** L_{2i} において $\vec{p}i$ の任意の 2 要素間に相互排他関係がない **then**
        $G =$ **AddAction**$(G, a, 2i + 1)$
      end
    end
    if アクションが 1 つも追加できなかった **then**　　/* 停止条件 2 */
      return $\emptyset$
    end
    for all $f \in L_{2i}$
      $G =$ **AddAction**$(G, no\text{-}op(f), 2i + 1)$
    end
    $i = i + 1$
  end
end.

図 5.8: プランニングアルゴリズム **SearchPlan**

SatCompile$^P(G)$
入力：プランニンググラフ G
出力：SAT 問題
begin
初期条件を $\{f_1, \ldots, f_n\}$ とし，目標条件を $\{g_1, \ldots, g_m\}$ とする
$\{g_1, \ldots, g_m\}$ から後向きに到達可能な頂点からなる G の部分グラフを G' とする
$l = G'$の最大レベル
G' を以下の規則に従って SAT 問題に変換する
(1) 初期条件はレベル 0 において真であり，目標条件はレベル l において真である
$f_1^0 \wedge \ldots \wedge f_n^0 \wedge g_1^l \wedge \ldots \wedge g_m^l$
(2) レベル i において相互排他関係にあるアクション a, b は同時に成立しない
$\neg a^i \vee \neg b^i$
(3) レベル i のアクション a は，その前提条件 $p_1, \ldots, p_k$ を含意する
$a^i \supset p_1^{i-1} \wedge \ldots \wedge p_k^{i-1}$
(4) レベル i のフルーエント f は，f を効果として持つレベル $i-1$ のすべての
アクション $a_1, \ldots, a_k$ を含意する
$f^i \supset a_1^{i-1} \vee \ldots \vee a_k^{i-1}$
end.

図 5.9: プランニング用 SAT 変換アルゴリズム **SatCompile**^{P}

5.4.1 例題

Yale Shooting 問題（例題 1）において，次式を満たすアクション列 X を求めてみる．

$$D \models \neg alive \textbf{ after } X$$

Yale Shooting 問題ではモデルは唯 1 つであり，その初期状態は $\{\neg loaded, alive\}$ である．アルゴリズム **SearchPlan** がレベル 4 までプランニンググラフを展開した結果を図 5.11 に示す．レベル 4 まで展開すると，目標条件が最終レベルに含まれる（$\neg alive \in L_4$）ので，プランニンググラフが SAT 問題に変換される．目標条件から後向きに到達可能な頂点 $\neg alive^4, shoot^3, loaded^2, load^1$ のみが変換の対象となる．変換結果を以下に示す：

$$\neg alive^4$$
$$\wedge (\neg alive^4 \supset shoot^3)$$
$$\wedge (shoot^3 \supset loaded^2)$$
$$\wedge (loaded^2 \supset load^1)$$

PlanExtract(D, J)
入力：領域記述 D，SAT 問題のモデル J
出力：プラン
begin
$\mathbb{A} = \{p^i \in J \mid p$ はアクション，かつ, p は正のリテラル $\}$
$\mathbb{A}$ に含まれるアクションを，添え字を基準に昇順に並び替えたものを $a_1^0, \ldots, a_m^l$ とする．
アクション列 $a_1^0, \ldots, a_m^l$ から冗長なアクションを取り除いたものを $b_1^0, \ldots, b_n^l$ とする †.
return $b_1^0, \ldots, b_n^l$
end.

図 5.10: プラン抽出アルゴリズム **PlanExtract**

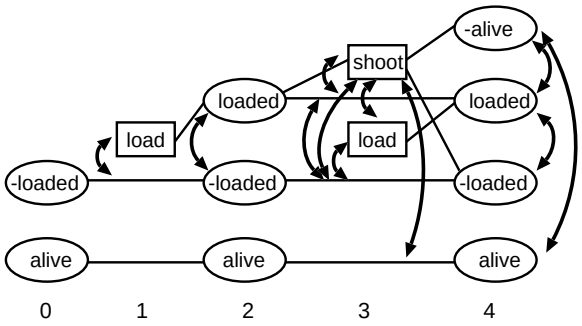

図 5.11: Yale Shooting 問題のプランニンググラフ

これを解くと，プラン $X = load^1; shoot^3$ を得る．

5.4.2 初期状態の仮定

AMP では，初期状態で真偽値が不明なフルーエントに対し，**閉世界仮説** (CWA) を宣言することができる．CWA は，以下のようにしてを効率良く実装されている [40].

ある奇数レベル i において，追加したいアクションの前提条件に $\neg F$ が現れるのに，L_{i-1} に $F, \neg F$ が含まれない場合，レベル 0 からレベル $i-1$ までの各偶数レベルに $\neg F$ を追加し，各奇数レベルに アクション $no\text{-}op(\neg F)$ を追加する[2].

同様にして，初期状態で真偽値が不明なフルーエントを，真または偽と仮定してプラン

[2] F は，レベル i においてアクションを追加するために初めて必要とされたフルーエント名であり，それまでのレベルの相互排他関係に影響を及ぼさない．

を生成することができる．これは次のように処理される．

ある奇数レベル i において，追加したいアクションの前提条件に f が含まれるのに，L_{i-1} に $f, \neg f$ が含まれない場合，レベル 0 からレベル $i-1$ までの各偶数レベルに $f, \neg f$ を追加し，各奇数レベルにアクション $no\text{-}op(f)$ と $no\text{-}op(\neg f)$ とを追加する．ここで，各レベルに追加した 2 つのノードは相互排他関係にある．この仮定の下で生成されたプランは，領域記述のあるモデルにより帰結されるプランとなる．

5.5 変数スキーマ

言語 $\mathcal{A}$ では変数を扱うことができないが，AMP では変数を含む命題をスキーマとして扱うことができる．例えば，次のような記述が可能である：

$$shoot(X) \textbf{ causes } \neg alive(Y) \wedge \neg loaded(X) \textbf{ if } turkey(Y) \wedge loaded(X). \tag{5.8}$$

$$\textbf{initially } \neg loaded(gun1) \wedge \neg loaded(gun2).$$

$$\textbf{initially } turkey(1) \wedge turkey(2).$$

式 (5.8) の効果命題は，以下の効果命題の省略形として扱われる：

$$shoot(gun1) \textbf{ causes } \neg alive(1) \wedge \neg loaded(gun1) \textbf{ if } turkey(1) \wedge loaded(gun1).$$

$$shoot(gun1) \textbf{ causes } \neg alive(2) \wedge \neg loaded(gun1) \textbf{ if } turkey(2) \wedge loaded(gun1).$$

$$shoot(gun2) \textbf{ causes } \neg alive(1) \wedge \neg loaded(gun2) \textbf{ if } turkey(1) \wedge loaded(gun2).$$

$$shoot(gun2) \textbf{ causes } \neg alive(2) \wedge \neg loaded(gun2) \textbf{ if } turkey(2) \wedge loaded(gun2).$$

AMP は Java 言語で実装されている．Java 言語は，高い拡張性・再利用性，プラットフォーム独立などの特徴をもっており，AMP の各モジュールは別のプログラムに容易に組み込むことができる．例えば AMP では，フルーエントを 1 階述語論理における述語と同等に扱っており（単一化の際には変数の出現検査も行っている），他の AI プログラミングに簡単に利用することができる．

5.6 AMP のインターフェイス

AMP は図 5.12 に示す GUI (graphical user interface) をもっている．このインターフェイスは以下のような機能を提供する：

- プランニンググラフの表示
- 領域記述の編集
- AMP の各モジュールの実行時間の測定
- ベンチマークテスト

5.7 実験結果

本節では，AMP のモデル生成とプランニングの実験結果を示す．ベンチマーク問題は，Blackbox に添付されている問題集から選んだ[3]．

まずモデル生成とプランニングの性能比較を行った（図 5.1）．Block-12 は 7 個のブロックを指定の形に積み上げる問題で，12 ステップのプランを必要とする．Block-large-a は 9 個のブロックを積み上げる問題であり，これも 12 ステップのプランを必要とする．モデル生成では目標条件とプランとを与え，モデルを 1 つ生成させた．プランニングでは目標条件とモデルを 1 つ与え，プランを 1 つ求めた．

表 5.1 より，モデル生成では，プランニングに比べ，グラフのサイズ・SAT 問題の節数ともに小さく，実行時間も短いことがわかる．これは，モデル生成アルゴリズムが，プランニングアルゴリズムを強く制限したアルゴリズムとなっているためと考えられる．グラフ展開時に使用するアクションは評価命題により与えられたアクションのみであり，また SAT 変換では $\mathbf{SatCompile}^M$ の規則 (5) が探索空間を絞り込んでいる．

次に，現在最速のプラン生成器の 1 つである Blackbox (version 3.6b) と AMP のプランニング速度の比較を行った（図 5.2）．これらの問題では初期状態が与えられているので，AMP はモデル生成を行っていない．AMP は Java 言語で実装され，Blackbox は C 言語で実装されている．括弧内の数字は，プラン抽出にかかった時間を表している．Blackbox では，オプションに `-solver satz` を指定した．

表 5.2 より，規模の小さな問題では約 3 倍程度の速度差があるが，問題の規模が大きくなるにつれその差は縮まっていることが分かる．Block-large-a では，AMP が Blackbox に優っているが，これは Blackbox の Satz モジュールが非常に多くの時間を消費したためである．

[3] `http://www.research.att.com/~kautz/blackbox/`

図 5.12: AMP のインターフェース

	Problem	Nodes	Clauses	Time [msec]
Model	block-12	1212	1572	1590
generation	block-large-a	1512	1940	1640
Planning	block-12	2375	26381	6650
	block-large-a	4227	117108	19170

表 5.1: モデル生成とプランニングの性能比較

Problem	Plan length	Nodes	Clauses	AMP [msec]	Blackbox [msec]
tire-world.04	12	1036	2798	3290	270
rocket-a	7	2039	22814	10770	3630
block-12	12	2375	26381	6650	2040
block-large-a	12	4227	117108	19170	19500
logistics-log-b	13	4466	25609	29770	15650
logistics-log10	10	8068	109044	31360	11970
logistics-log11	11	6775	70171	62280	37400

表 5.2: AMP と Blackbox のプランニング性能比較

(実験環境：366MHz PentiumII and 128MB memory.)

5.8 まとめ

我々は，SAT プランニングのアプローチが，プランニングだけでなく，モデル生成などのアクション言語の他の特徴にも有効であることを示した．我々が提案するアクション言語処理系 AMP は，入力として 言語 $\mathcal{A}$ による記述を受け取り，プランニングやモデル生成（初期状態の推定），実行結果の予測などの問い合わせに高速に応答する．

実験結果からも分かるように，Java 言語による実装では，C 言語による実装と比べ速度面で不利である．しかし，より抽象度の高いプログラミングをすることが可能であること，メモリ管理に悩まされることがないこと等の理由から，複雑なアルゴリズムであっても比較的容易に実装することができる．この利点を生かし，今後，さらなる処理速度の向上や，より表現力豊かなアクション言語に対応させていく予定である．

プランニンググラフのサイズは，グラフ展開・プラン抽出アルゴリズムの双方に大きな

影響を及ぼすため，グラフのサイズを抑えることは速度向上のための重要な要因である．AMP や Blackbox で用いられているプランニンググラフの展開アルゴリズムは，初期状態からグラフを順次展開していくが，特に目標条件を指向しているわけではない．なぜなら各偶数レベルにおいて実行可能なすべてのアクションを次の奇数レベルに加えているからである．そこで，初期状態から目標条件に向かうグラフ展開に加え，目標条件から初期状態へ向かうグラフ展開を行うことで，プランニンググラフのサイズを抑える研究がなされている [15]．我々もこれまでに，目標条件から初期状態へ向かうプラン生成システムを実装しているので [29]，今後これらを組み合わせてプランニンググラフのサイズの抑制を図る予定である．

第6章　因果関係の学習

現実の問題では，問題領域に対する完全な知識を記述することは難しく，むしろその領域における観測結果を記述することのほうが容易であることが多い．我々はこの観点から因果関係の知識を学習するアルゴリズムを提案する [30]．学習アルゴリズムへの入力は，事象の観測例であり，出力は，言語 $\mathcal{A}$ による因果関係の記述 — あるアクションを実行すると，それがどのような影響を及ぼすかという記述である．言語 $\mathcal{A}$ は命題的フルーエントのみを扱うので，学習アルゴリズムとして，決定木を学習するアルゴリズム [35] を採用した．

6.1　学習アルゴリズム

我々は，言語 $\mathcal{A}$ による因果関係の学習に決定木の学習アルゴリズム [35] を用いる．まず，学習する決定木を定義する．

決定木は，任意のアクション名 a と任意のフルーエント名 F の対 $\langle a, F\rangle$ に対して定義される．

定義 1 決定木 $\langle a, F\rangle$ の節点と枝を以下で定義する：

- 各非終端節点は，フルーエント名をもつ．その節点から出る枝には，そのフルーエントの真偽値がラベル付けされる．
- 各終端節点は，“T” または “F” または “×” というラベルをもつ．

直感的には，終端節点に到達する経路が，アクション a の前提条件を表しており，終端節点が，その前提条件のもとでアクション a を実行した場合，フルーエント名 F にどのような影響を及ぼすのかを表している：F が真になる (true) ／偽になる (false) ／ a は実行できない（×）．

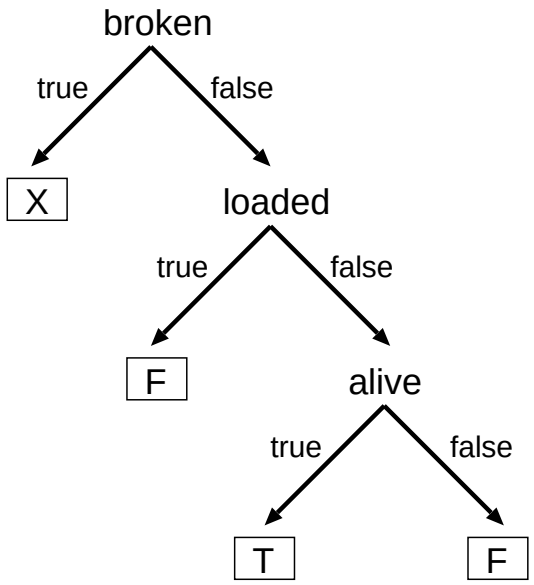

図 6.1: 決定木 $\langle shoot, alive \rangle$

図 6.1 に，決定木 $\langle shoot, alive \rangle$ の例を示す．アクション $shoot$ の実行により，フルーエント $alive$ が真になるのか (true)，偽になるのか (false)，それとも実行できないのか（×）が表現されている．この決定木を言語 $\mathcal{A}$ による記述に直訳すると次のようになる：

$$shoot \textbf{ causes } \neg alive \textbf{ if } \neg broken \wedge loaded$$

$$\textbf{impossible } shoot \textbf{ if } \neg broken$$

ここで **impossible** a **if** ϕ は，a **causes** $false$ **if** ϕ の省略形であり，前提条件 ϕ が成立する状態では，アクション a が実行できないことを表している．$shoot$ が $alive$ に影響を及ぼさない場合は，慣性の法則により記述する必要がない．また，銃が壊れている $(broken)$ ならば，$shoot$ を実行できないので，前者の命題をさらに簡単化することができる：

$$shoot \textbf{ causes } \neg alive \textbf{ if } loaded$$

学習アルゴリズムへの入力は，観測された事例の集合である．この集合のことを**訓練集合** (training set) と呼ぶ．“例” を以下で定義する．

定義 2 例は，次の形式をしている：

$$(f \textbf{ after } a) \wedge (\textbf{initially } p_1 \wedge \cdots \wedge p_m) \quad (m \geq 0)$$

これは，フルーエント $p_1, \cdots, p_m$ が観測された状態でアクション a を実行したところ，フルーエント f が観測されたことを表している．これを簡潔に次のように記述する：

$$f \textbf{ after } a \textbf{ if } p_1 \wedge \cdots \wedge p_m$$

$p_1 \wedge \cdots \wedge p_m$ を**前提条件**，f を**結果**と呼ぶ．また，各 p_i の絶対値を取ったもの $|p_i|$ をアクション a の**属性**と呼ぶ．アクションが実行できなかった場合は，結果を $false$ と記述する．さらに，f が正のフルーエントであるならば，この例を**正例**と呼び，f が負のフルーエントであるならば，この例を**負例**と呼ぶ．f が $false$ の場合，この例を**実行不能例**と呼ぶ．

上の定義において，我々は，システムがすべてのフルーエントを観測できるとは仮定しない．観測結果には欠損データが存在するかも知れないし，ノイズが含まれるかも知れない．またアクションの長さを 1 に限定している．これらについては第 6.4 節で触れる．

定義 3 アクション名 a とフルーエント名 F に関する訓練集合を $\mathcal{I}$ とする．$\mathcal{I}$ に含まれる正例の数を p，負例の数を n，実行不能例の数を k とする．このとき，最も数の多い結果を返す関数を定義する：

$$Majority(\mathcal{I}) = \begin{cases} F & \textbf{if } \ p \geq n \textbf{ and } p \geq k \\ \neg F & \textbf{if } \ n > p \textbf{ and } n \geq k \\ \times & \textbf{if } \ k > p \textbf{ and } k > n \end{cases}$$

定義 4 訓練集合 $\mathcal{I}$ とフルーエント名 F に対し，結果に F を含む例を取り出す演算を定義する：

$$\mathcal{I}_F = \{\ (g \textbf{ after } a \textbf{ if } \vec{p}) \in \mathcal{I} \ \mid \ |g| = F \text{ または } g = false\ \}$$

同様に，アクション名 a を含む例を取り出す演算を定義する：

$$\mathcal{I}^a = \{\ (f \textbf{ after } X \textbf{ if } \vec{p}) \in \mathcal{I} \ \mid \ X = a\}$$

定義 5 任意のフルーエント p に対し，p **の値**を以下で定義する：

$$Value(p) = \begin{cases} \mathrm{T} & \textbf{if } \ p \text{ が正のフルーエント} \\ \mathrm{F} & \textbf{if } \ p \text{ が負のフルーエント} \\ \times & \textbf{if } \ p \text{ が } false \end{cases}$$

我々が提案する決定木の学習アルゴリズムは，ID3 [35] を基にしている．ID3 とは次の点が異なる：ID3 が学習する決定木は，ある命題が真になるか偽になるかを決定するための条件を表している．一方，我々が学習する決定木 $\langle a, F\rangle$ は，アクション a を実行すると F が真になるのか偽になるのか，それとも a が実行不能であるのかを決定するための条件を表しており，“真”, “偽”, “×” の 3 値を扱う．

決定木の学習アルゴリズムでは，なるべく簡潔な決定木を構成するために，**情報量**の概念に基づいて，決定木の各ノードを配置していく．

定義 6 離散型確率変数 X の定義域を $\{v_1, \cdots, v_n\}$ とし，各事象 v_i の生起確率を $P(v_i)$ で表す．このとき，任意の一つの事象が生起することによって伝えられる**平均情報量** $I(P(v_1), \cdots, P(v_n))$ を

$$I(P(v_1), \cdots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i)$$

で定義する．

アクション名 a とフルーエント名 F に関する訓練集合を $\mathcal{I}$ とする．$\mathcal{I}$ に含まれる正例の数を p，負例の数を n，実行不能例の数を k とする．正例または負例または実行不能例が観測される確率を，$\mathcal{I}$ における例の比率で与える．このとき，例が観測されることによって伝えられる情報量は，

$$I\left(\frac{p}{|\mathcal{I}|}, \frac{n}{|\mathcal{I}|}, \frac{k}{|\mathcal{I}|}\right) = -\frac{p}{|\mathcal{I}|} \log_2 \frac{p}{|\mathcal{I}|} - \frac{n}{|\mathcal{I}|} \log_2 \frac{n}{|\mathcal{I}|} - \frac{k}{|\mathcal{I}|} \log_2 \frac{k}{|\mathcal{I}|}$$

で与えられる．ここで $|\mathcal{I}|$ は集合 $\mathcal{I}$ の要素数を表す．この情報量は，3 種類の例が等しい生起確率をもつとき，最大値である 1.585 ビットとなる（正例，負例，実行不能例のうち，どれが観測されるか予想できないので，それを判断するために 1.585 ビットの情報量が必要となる）．

ここでフルーエント名 G を決定木の根に置いたとする．決定木は，訓練集合 $\mathcal{I}$ を，2 つの集合 $\mathcal{I}_1, \mathcal{I}_2$ に分割する[1]：

$$\mathcal{I}_1 = \{\ (f \textbf{ after } a \textbf{ if } \vec{p}) \in \mathcal{I} \ \mid\ \vec{p} \text{ に } G \text{ が含まれる}\ \}$$
$$\mathcal{I}_2 = \{\ (f \textbf{ after } a \textbf{ if } \vec{p}) \in \mathcal{I} \ \mid\ \vec{p} \text{ に } \neg G \text{ が含まれる}\ \}$$

[1]すべての例の前提条件には G が含まれるものと仮定する．すなわち，欠損データは存在しないと仮定する．

$\mathcal{I}_i$ に含まれる正例の数を p_i，負例の数を n_i，実行不能例の数を k_i とする．もし我々が G が真となる枝を進むと，我々は質問に答えるために，さらに $I(p_1/|\mathcal{I}_1|, n_1/|\mathcal{I}_1|, k_1/|\mathcal{I}_1|)$ ビットの情報量を必要とする．従って，フルーエント G のテスト後，例を分類するために，平均すれば

$$Remainder(G) = \frac{p_1 + n_1 + k_1}{|\mathcal{I}|} I\left(\frac{p_1}{|\mathcal{I}_1|}, \frac{n_1}{|\mathcal{I}_1|}, \frac{k_1}{|\mathcal{I}_1|}\right) + \frac{p_2 + n_2 + k_2}{|\mathcal{I}|} I\left(\frac{p_2}{|\mathcal{I}_2|}, \frac{n_2}{|\mathcal{I}_2|}, \frac{k_2}{|\mathcal{I}_2|}\right)$$

ビットの情報量を必要とする．従ってフルーエント G により，訓練集合を分類することで獲得される情報量は,

$$Gain(G) = I\left(\frac{p}{|\mathcal{I}|}, \frac{n}{|\mathcal{I}|}, \frac{k}{|\mathcal{I}|}\right) - Remainder(G)$$

となる．決定木の学習アルゴリズムは，この情報利得が最大となるフルーエントを，決定木の根に配置する．

学習アルゴリズムを図 6.2 に示す．アルゴリズム **CausalRelationLearning** は，入力として訓練集合 $\mathcal{I}$ を受け取り，$\mathcal{I}$ に含まれるすべてのアクション名 A とフルーエント名 F に対し，決定木 $\langle a, F\rangle$ を学習する．サブルーチン **DecisionTreeLearning** が，個々の決定木を学習するアルゴリズムである．ここでは簡単のため，欠損データやノイズを考慮していない．

6.2 実行例

人間と狼と山羊とキャベツの問題（例題 4）に対し，学習アルゴリズムを適用した結果を示す．ここでは，図 3.6 において遷移先のないアクションは実行不可能なアクションであると仮定した．入力として，アクション g (山羊を連れて対岸にわたる) に関する観測例を 5 つ与えた：

$$\begin{array}{rlll}
\overline{M} & \textbf{after} & g & \textbf{if} \quad M \wedge W \wedge G \wedge C \\
M & \textbf{after} & g & \textbf{if} \quad \overline{M} \wedge \overline{W} \wedge \overline{G} \wedge C \\
\overline{M} & \textbf{after} & g & \textbf{if} \quad M \wedge \overline{W} \wedge G \wedge C \\
\mathit{false} & \textbf{after} & g & \textbf{if} \quad \overline{M} \wedge \overline{W} \wedge G \wedge \overline{C} \\
\mathit{false} & \textbf{after} & g & \textbf{if} \quad M \wedge W \wedge \overline{G} \wedge C
\end{array}$$

CausalRelationLearning($\mathcal{I}$)
入力：訓練集合 $\mathcal{I}$
出力：決定木の集合 $\mathbb{D}$
begin
  $\mathbb{D} = \emptyset$
  $\mathbb{A} = \mathcal{I}$ に含まれるアクション名の集合
  $\mathbb{F} = \mathcal{I}$ に含まれるフルーエント名の集合
  for each $a \in \mathbb{A}$
    for each $F \in \mathbb{F}$
      $\mathbb{P} = \mathcal{I}_F^a$ に含まれるすべての属性の集合
      $\mathbb{D} = \mathbb{D} \cup$ **DecisionTreeLearning**(a, F, $\mathcal{I}_F^A$, $\mathbb{P}$, $Value(Majority(\mathcal{I}_F^a))$)
end.

DecisionTreeLearning (a, F, $\mathcal{I}$, $\mathbb{P}$, V)
入力：アクション名 a, フルーエント名 F，例集合 $\mathcal{I}$，属性の集合 $\mathbb{P}$, デフォルトの値 V
出力：決定木 $D = \langle a, F \rangle$
begin
  if $\mathcal{I} = \emptyset$ **then**
    return V
  if すべての例の結論が X である **then**
    return $Value(\mathrm{X})$
  if $\mathbb{P} = \emptyset$ **then**
    return $Value(Majority(\mathrm{I}))$
  $Gain(P)$ が最大となるようなフルーエント $P \in \mathbb{P}$ を選ぶ
  $\mathbb{P}' = \mathbb{P} \setminus \{P\}$
  $\mathcal{I}_1 = \{\ (f$ **after** a **if** $\vec{p}) \in \mathcal{I}\ \mid\ \vec{p}$ に G が含まれる $\}$
  $\mathcal{I}_2 = \{\ (f$ **after** a **if** $\vec{p}) \in \mathcal{I}\ \mid\ \vec{p}$ に $\neg G$ が含まれる $\}$
  決定木 D の根を P とする
  根 P の正の子ノードを **DecisionTreeLearning**(A, F, $\mathcal{I}_1$, $\mathbb{P}'$, $Value(Majority(\mathcal{I}_1))$) とする
  根 P の負の子ノードを **DecisionTreeLearning**(A, F, $\mathcal{I}_2$, $\mathbb{P}'$, $Value(Majority(\mathcal{I}_2))$) とする
end.

図 6.2: 因果関係の学習アルゴリズム

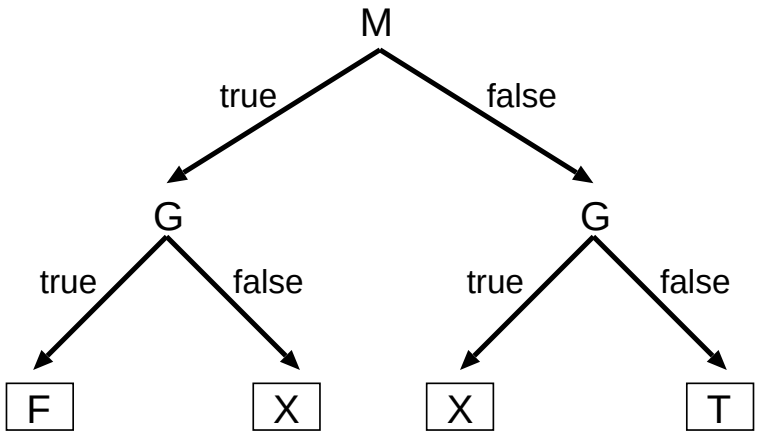

図 6.3: 学習された決定木 $\langle g, M \rangle$

このとき学習された決定木 $\langle g, M \rangle$ を図 6.3 に示す．この決定木を言語 $\mathcal{A}$ における命題に変換すると，次のようなる：

g **causes** $\overline{M}$ **if** $M \wedge G$
g **causes** M **if** $\overline{M} \wedge \overline{G}$
impossible g **if** $M \wedge \overline{G}$
impossible g **if** $\overline{M} \wedge G$

上記命題は，すべての状態において正しい遷移を与える．2 つの **impossible** 命題は，もともとの領域記述には含まれていないという点では冗長であるが，この問題の制約を良く表している（山羊を連れて移動するためには，山羊と人間が同じ岸にいなければならない）．

次に，アクション w (狼を連れて対岸にわたる) に関する全ての観測例を入力として与えたところ，次のような学習結果が得られた (**impossible** 命題は省略)：

w **causes** $M \wedge W$ **if** $\overline{M} \wedge \overline{W} \wedge \overline{G} \wedge C$
w **causes** $\overline{M} \wedge \overline{W}$ **if** $M \wedge W \wedge G \wedge \overline{C}$
w **causes** $\overline{M} \wedge \overline{W}$ **if** $M \wedge \overline{G}$
w **causes** $M \wedge W$ **if** $\overline{M} \wedge G$

この問題には制約を満たさない状態が 6 つ存在するが，そのような状態における観測結果は与えられないため，上記学習結果は元々の記述より多少簡単になっている．我々が文献 [28] で提案した言語 $\mathcal{A}$ の簡単化手続きでは，この種の簡単化は達成できない．

この実験では，システムが全てのフルーエントを正しく観測できることを仮定していた．例題 4 は 4 つのフルーエントしか含まない小規模なものなので，この仮定は妥当なものか

も知れないが，一般に，観測結果はノイズや欠損データを含んでいる．そのようなデータに対し，ID3 では，統計的手法を用いて，仮説を生成するアルゴリズムを提案している．本研究でも，そのような手法を取り入れ，より大きな規模の問題で実験し，定量的な評価を行なう必要がある．

6.3 関連研究

本研究では入力例のアクション列の長さを 1 に制限していたが，より一般的にするためには，複数アクションの列も入力できるよう拡張する必要がある．それにより，単一のアクションの因果関係を学習するだけでなく，複数アクションの列をマクロのような形で学習することを考えている．

単一のアクション，複数のアクションの区別なく，因果関係を学習する枠組として，文献 [41] で紹介されている行動ネットワーク (behavior network) の学習や，文献 [38] で紹介されている信念ネットワーク (belief network) の学習などがある．前者は，アクション間の因果関係 – あるアクションが実行された後にどのアクションが実行可能になるのかという関係を学習する．後者は，より一般的に，事象間の依存関係を学習する．

6.4 まとめ

本章では，決定木の学習アルゴリズムを利用した因果関係の学習するアルゴリズムを提案した．この学習アルゴリズムは，(1) アクションを 1 つ実行 (2) 環境の変化を観測というステップを繰り返し動作し観測例を蓄積するシステムを対象としている．この学習アルゴリズムを用いることで，不完全な知識しか持たない環境下においても，その環境における事象の観測結果から因果関係の知識を推定することができる．

第7章　結論

本章では，本研究の内容を要約し，その成果をまとめ，今後の研究課題について述べる．

7.1　研究内容の要約

状態変化やアクションを記述するための知識表現に関する研究において最近活発に研究されているアプローチの 1 つが高級レベルのアクション言語である．最も基本的なアクション言語 $\mathcal{A}$ が提案されて以来，$\mathcal{A}$ を拡張し，さらなる表現力を求めたさまざまなアクション言語が提案されている．しかし，それらの言語は，言語 $\mathcal{A}$ に比べ客観的にどの程度表現力が向上したのか分かっていない．記述可能な問題領域が広がったのかどうか，もし広がったのであれば何が記述可能になったのか，などの言語の表現力が明らかになっていない．また，これらの言語の多くは，その実現のための手法として実際には，拡張論理プログラムや一階述語論理，その他の非単調な枠組に変換している．これらは汎用の表現言語であるため，効率の良い推論システムを構築することは難しい．

そこで我々は，(i) アクション言語の表現能力の形式的解析手段を提供し，今後のアクション言語設計のための指標を与えること，(ii) その解析結果に基づいてより表現力豊かなアクション言語を提案すること，(iii) アクション言語のための効率の良い推論アルゴリズムを提案すること，そして，現実の問題では，問題領域に対する完全な知識を記述することは難しく，むしろその領域における観測結果を記述することのほうが容易であることが多いという観点から，(iv) 環境から因果関係の知識を自動獲得する学習アルゴリズムを提案することを目的として研究を行った．

それぞれの結果について以下に述べる．

第 3 章において我々は，アクション言語の表現能力の形式的な解析手段を提供するため，有限オートマトン (FA) に着目し，言語 $\mathcal{A}$ で表現可能な領域のクラスと，FA で表現できる言語のクラスとが等価であることを示した．言語 $\mathcal{A}$ と FA の等価性の結果は，形式言語論の観点からアクション言語の特性を解析するための基盤を提供する．言語 $\mathcal{A}$ におけ

るプランニング問題は，オートマトンの受理言語を求める問題に帰着し，プランニングの解は，正則表現を用いることで簡潔かつ完全に表すことができる．

また $\mathcal{A}$ と FA の等価性の結果は，アクション言語をオートマトンのさまざまな応用分野に適用することを可能にする．オートマトンなどの状態グラフは，問題に対する最終的な仕様書としての表現形式に適しているが，その主な欠点は，詳細化に対する頑健性に欠いていることである．一方，論理による表現は，局所的な変化に対し，状態グラフよりも適している．現在，オートマトンの応用分野は，テキスト編集，語彙解析，並行プロセス，プログラム検証など幅広く存在する．このような動的な領域の設計を，$\mathcal{A}$ やその他のアクション言語で行なえるのならば，それを修正することは容易になると考えられる．またアクション言語には，慣性の法則による記述面の利点がある．言語 $\mathcal{A}$ では，N 個の状態は，少なくとも $\lceil \log_2 N \rceil$ 個のフルーエントにより表現することができ，さらにアクションがフルーエントに影響を及ぼさないのであれば，その旨を明言する必要はない．

第 4 章では，$\mathcal{A}$ と FA との等価性の結果に基づき，非決定性有限オートマトン (NFA) の観点から新しい非決定性アクション言語 $\mathcal{NA}$ を提案した．言語 $\mathcal{NA}$ は，NFA と同じ表現力をもっており，$\mathcal{NA}$ と他のアクション言語との等価性を示すことで，その言語の適用範囲を形式的に議論できるようになる．実際に我々は，極小変化の概念に基づく非決定性をもつ言語 $\mathcal{AR}$ が，言語 $\mathcal{NA}$ と等価な表現力を持つことを示した．言語 $\mathcal{NA}$ における非決定性の表現は，NFA における表現に基づいた簡潔な表現となっており，ユーザは，問題に対し，状態グラフを記述する要領で，言語 $\mathcal{NA}$ により記述することができる．また非決定性を用いて障害物などを検出するセンサーを定式化できることを示し，環境に関する知識が不完全な状況であったとしても，プランニングなどの推論を行うことが可能になることを示した．

第 5 章では，アクション言語のための効率的な推論アルゴリズムを提供するため，最近プランニング研究の分野において高速なプランニングアプローチとして注目されている SAT プランニングと呼ばれる手法に着目した．我々は，この SAT プランニングの技術が，アクション言語におけるプランニングだけでなく，モデル生成（初期状態の推定）に対しても適用できることを示し，アクション言語処理系 AMP を開発した．AMP は Java 言語により実装されており，さまざまなプラットフォームで実行することができる．AMP を用いることで，実際に問題領域をアクション言語により記述し，プランニングやモデル生成，実行結果の予測などの各種推論を高速に行うことが可能になる．

第 6 章では，因果関係の知識を自動獲得する学習アルゴリズムを提案するため，決定木

の学習アルゴリズムをアクション言語における因果関係の学習に適用した．一般的に決定木は，真または偽の 2 値を決定するが，アクション言語においては，アクションの実行によりフルーエントが真になる，偽になるに加え，アクションが実行できない場合がある．そこで我々は，決定木が真・偽・実行不可能の 3 値を出力するように拡張し，そのような決定木を学習するアルゴリズムを提案した．この学習アルゴリズムは，(1) アクションを 1 つ実行 (2) 環境の変化を観測というステップを繰り返し動作し観測例を蓄積するシステムを対象としている．この学習アルゴリズムを用いることで，不完全な知識しか持たない環境下においても，その環境における事象の観測結果から因果関係の知識を推定することができる．

本研究を，動的な世界で推論し行動する知的主体を構成するための研究として捉えれば，我々は，知的主体の知識表現言語に関する研究と，その言語における推論アルゴリズムに関する研究と，知識を自動獲得する学習アルゴリズムに関する研究を行ったといえる．我々が提案した言語 $\mathcal{NA}$ は，非決定性効果をもつアクション，間接的効果をもつアクション，状態に対する制約が記述できる．特にその簡潔な非決定性の表現により，ユーザは状態遷移グラフを記述する要領で，問題を言語 $\mathcal{NA}$ により記述することができ，また状態遷移グラフで記述するよりも記述量を抑えることが可能である．また，アクション言語処理系 AMP は，最新のプランニング技術を基に，アクション言語による知識記述に対し，プランニングやモデル生成などの推論を高速に行うことができる．最後に我々が提案した学習アルゴリズムは，未知の環境においても，試行を繰り返すことにより，その環境における因果関係を獲得する能力を提供する．

7.2 今後の課題

我々は，言語 $\mathcal{A}$ と $\mathcal{NA}$ で表現可能な領域のクラスが，有限オートマトンで表現できる言語のクラスと等価であることを示した．これらの言語は表現力において等価ではあるが，ある問題を記述するときに必要となる記述量において大きく異なるものと考えられる．これらの言語における記述量の形式的解析を行うことは今後の課題の 1 つである．特にある問題を，アクション言語により記述する場合とオートマトンにより表現する場合とでは，どのような記述量の違いがあるのか興味深い．

アクション言語には，状態遷移システムを記述する言語としての側面と，プランニングなどの問い合わせを行う言語としての側面がある [22]．アクション言語を状態遷移システ

ムを記述する言語として捉えた場合に，アクション言語が，形式言語論における文法やプッシュダウンオートマトン，チューリング機械，ペトリネットなどの FA のより一般的なクラスとどのような関係にあるのか探求することも今後の課題の 1 つである．

我々は言語 $\mathcal{NA}$ において，非決定性効果をもつアクション，間接的効果をもつアクション，状態に対する制約が記述できるような言語仕様を与えた．これらに加え，同時発生アクションや静的な因果関係を記述できるアクション言語がすでに提案されている．同時発生アクションは，複数のアクションを同時に実行することにより，あるタスクを遂行することを表現する（例えば，重たい荷物を運ぶために複数の搬送機を使用する必要がある場合など）．静的な因果関係は，フルーエントが他のフルーエントに影響を及ぼすことを表現する．これは状態に対する制約によっても表現することが可能だが，制約式は対偶をとることができるのに対し，静的な因果関係は規則であるため対偶をとることができない点が異なる．

同時発生アクションや状態に対する制約を表現できるように言語 $\mathcal{NA}$ を拡張していくことは，今後の重要な研究課題である．また，これらの特徴とオートマトン理論との関係も探求する．

本論文で紹介したアクション言語処理系 AMP は，言語 $\mathcal{A}$ による領域記述のみを扱うことができる．我々は，現在，SAT プランニングアプローチに基づいた言語 $\mathcal{NA}$ のための効率の良い推論アルゴリズムに関する研究を進めている．このアルゴリズムを完成させ，AMP を，非決定性アクションや制約を含む領域を扱えるように拡張することも今後の大きな課題である．

こうした AMP がより表現力の高い言語を扱えるようにする研究とともに，現在の AMP のプランニング・モデル生成アルゴリズムの効率の向上にも取り組む．第 5.8 節でも触れたが，プランニンググラフのサイズはグラフ展開・プラン抽出アルゴリズムの速度に大きな影響を及ぼしており，グラフのサイズを抑えることは速度向上のための重要な要因である．我々は現在，目標条件から初期状態へ向かうグラフ展開アルゴリズムに関する研究も進めており，こうした後向きの展開アルゴリズムと，従来の前向きの展開アルゴリズムを組み合わせることでグラフのサイズの抑制し，速度向上を図る予定である．

我々が提案した因果関係の学習アルゴリズムは，入力として受け取ることができる観測例のアクション列の長さを 1 に制限している（第 6.3 節参照）．より一般的で強力な学習アルゴリズムにするためには，複数のアクションを連続して実行した結果からも因果関係の規則を学習できるように拡張する必要がある．現実の問題では，環境に関する完全な知

識を記述することは困難であり，不完全な知識しか与えられない場合が多い．その点からも，知的主体が自ら学習することで知識を補っていく能力は非常に重要である．より強力な学習アルゴリズムの枠組みについて研究を進めることも，今後の重要な課題である．

謝辞

本論文を作成するにあたり，指導教官である羽根田博正教授に貴重なご助言と親切なご指導をいただきまた．ここに記し，深く御礼申し上げます．研究以外の面に関しても物事に関する考え方や姿勢について多くのことを学ぶことができました．本当にありがとうございました．

本論文をまとめるにあたり，有益なコメントをいただきました，瀧和男教授と阿部重夫教授に厚く御礼申し上げます．

また，豊橋技術科学大学の学部生時代から，多くのご助言とご指導を頂きました井上克已助教授に深く感謝いたします．本研究分野に対する深い知識には，とても多くのことを教えられました．本当にありがとうございました．

なお，本研究の一部は，文部省科学研究費補助金（特別研究員奨励費）によるものである．

付 録A

A.1 AMPのモデル生成アルゴリズムの正当性の証明

定理 13 任意の数の効果命題と次の評価命題

$$\textbf{initially } \phi_0$$

を含む領域記述を D とする．次の評価命題を V とする．

$$V = (\phi_n \textbf{ after } a_1; \ldots; a_n)$$

$D \cup \{V\}$ のモデル集合を $\mathbb{I}$ とする．モデル生成アルゴリズムより求まる SAT 問題を

$$P = \mathbf{SatCompile}^M(\mathbf{Expand}^M(D, V))$$

とし，P のモデル集合を $\mathbb{J}$ とする．このとき，

$$\mathbf{Extract}(\mathbb{J}) = \mathbb{I}$$

である．

証明 アクション列の長さに関する数学的帰納法を用いる．アクション列の長さが k であるとき，その評価命題を V_k と表す．すなわち，

$$V_k = (\phi_k \textbf{ after } a_1; \ldots; a_k)$$

$D \cup \{V_k\}$ のモデル集合を $\mathbb{I}_k$ で表す．D と V_k から作成した SAT 問題を

$$P_k = \mathbf{SatCompile}^M(\mathbf{Expand}^M(D, V_k))$$

とし，P_k のモデル集合を $\mathbb{J}_k$ で表す．このとき任意の k に対し，

$$\mathbf{Extract}(\mathbb{J}_k) = \mathbb{I}_k \tag{A.1}$$

であることを証明する．

基底ステップ $k = 0$ のとき，すなわち $V_k = \textbf{initially}\ \phi_k$ のとき，アルゴリズムが出力するモデル集合は，

$$\mathbb{J}_0 = \{(\Phi, \{f_1, \ldots, f_n, g_1, \ldots, g_m\})\}$$

となる．ここで $\phi_0 = (f_1 \wedge \ldots \wedge f_n)$, $\phi_k = (g_1 \wedge \ldots \wedge g_m)$ である．$\mathbb{J}_0$ が V_k を充足するすべてのモデルの集合であることは明らかである．

いま k 未満のすべての数 l に対し $\textbf{Extract}(\mathbb{J}_l) = \mathbb{I}_l$ を仮定し，式 (A.1) を証明する．そのために，次の 2 つのことを証明する：

(a) 任意の $I_k \in \mathbb{I}_k$ に対し，次式を満たす $J_k \in \mathbb{J}_k$ が存在する（完全性）．

$$I_k = \textbf{Extract}(J_k) \tag{A.2}$$

(b) 任意の $J_k \in \mathbb{J}_k$ に対し，次式を満たす $I_k \in \mathbb{I}_k$ が存在する（健全性）．

$$\textbf{Extract}(J_k) = I_k \tag{A.3}$$

まず (a) を証明する．評価命題 V_{k-1} を以下で定義する：

$$V_{k-1} = (\psi\ \textbf{after}\ a_1; \ldots; a_{k-1})$$

ここで ψ は，ある $I_k \in \mathbb{I}_k$ に対し，

$$I_k \models V_{k-1}$$

を満たすフルーエントの連言であり，かつ，プランニンググラフ $G_k = \textbf{Expand}^M(D, V_k)$ に含まれるすべてのフルーエント名を含むものとする．

D と V_{k-1} から作成したプランニンググラフを

$$G_{k-1} = \textbf{Expand}^M(D, V_{k-1})$$

とする．ここで G_k と G_{k-1} との違いは，“G_k には，アクションレベル L_{2k-1} とフルーエントレベル L_{2k} が存在するが，G_{k-1} には存在しない” ことにある（図 A.1）．G_{k-1} を SAT 変換したものを $P_{k-1} = \textbf{SatCompile}^M(G_{k-1})$ とすると，P_k は，P_{k-1} を用いて次のように表すことができる：

$$P_k = (P_{k-1} \setminus \{\psi^{2k-2}\}) \cup Q$$

Q は，L_{2k-1}, L_{2k} の SAT 変換結果であり，次の要素から構成される：

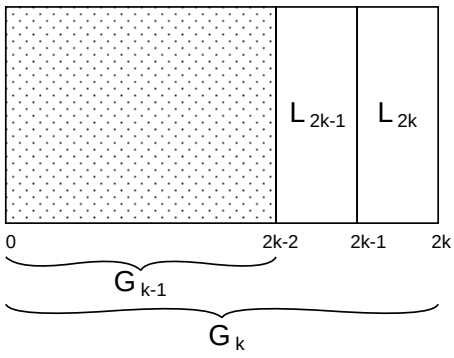

図 A.1: G_{k-1} と G_k との違い

- **SatCompile**^{M} の規則 (1) より,

$$\phi_k^{2k} \tag{A.4}$$

ここで初期条件 ϕ_0^0 はすでに P_{k-1} が含んでいる.

- **SatCompile**^{M} の規則 (2) より, 相互排他関係にある 2 つのアクション $b, c \in L_{2k-1}$ について,

$$\neg b^{2k-1} \vee \neg c^{2k-1} \tag{A.5}$$

- **SatCompile**^{M} の規則 (3) より L_{2k-1} に含まれるアクション名 a_k に関する効果命題を

$$a_{k,1} \textbf{ causes } \vec{e}_1 \textbf{ if } \vec{p}_1$$
$$\vdots$$
$$a_{k,s} \textbf{ causes } \vec{e}_s \textbf{ if } \vec{p}_s$$

とすると,

$$\begin{aligned} &a_{k,1}^{2k-1} \supset \vec{p}_1^{2k-2} \\ &\quad\vdots \\ &a_{k,s}^{2k-1} \supset \vec{p}_s^{2k-2} \end{aligned} \tag{A.6}$$

$$\forall f \in L_{2k_2}(\textit{no-op}(f)^{2k-1} \supset f^{2k-2}) \tag{A.7}$$

- **SatCompile**^{M} の規則 (4) より，L_{2k-1} に含まれるアクション名のうち，フルーエント f を効果に含むアクションの選言を $\mathit{haveEffect}(f)$ で表すと，

$$\forall f \in L_{2k}(f^{2k} \supset \mathit{haveEffect}(f)^{2k-1}) \tag{A.8}$$

- **SatCompile**^{M} の規則 (5) より，式 (A.6) の対偶が存在する：

$$\begin{aligned} &\bar{p}_1^{2k-2} \supset a_{k,1}^{2k-1} \\ &\quad\vdots \\ &\bar{p}_s^{2k-2} \supset a_{k,s}^{2k-1} \end{aligned} \tag{A.9}$$

式 (A.2) を満たす J_k が存在することを示すために，実際に J_k を求めてみる．SAT 問題 P_{k-1} のモデル集合を $\mathbb{J}_{k-1}$ とする．帰納法の仮定と ψ の定義より，

$$I_k = \mathbf{Extract}(J_{k-1})$$

となるモデル $J_{k-1} \in \mathbb{J}_{k-1}$ が存在する．そのようなモデル J_{k-1} に対し，

$$J_k = J_{k-1} \cup J_Q$$

となる Q のモデル J_Q を次のようにして作成する．式 (A.4) より，ϕ_k^{2k} は明らかに真でなければならない．従って式 (A.8) より，

$$\forall f \in \phi_k(\mathit{haveEffect}(f)^{2k-1})$$

も真でなければならない．任意のフルーエント $f \in \phi_k$ に対し，

$$\mathit{haveEffect}(f) = a_{k,t} \vee \ldots \vee a_{k,u} \vee \mathit{no\text{-}op}(f) \qquad (1 \leq t \leq u \leq s)$$

とすると，これらのうち少なくとも 1 つのアクションを真にできることを示す．

まず，$a_{k,t}, \ldots, a_{k,u}, \mathit{no\text{-}op}(f)$ の前提条件 $\bar{p}_t^{2k-2}, \ldots, \bar{p}_u^{2k-2}, f^{2k-2}$ がすべて偽の場合を考える．これは

$$\psi \models \neg\bar{p}_t^{2k-2} \wedge \ldots \wedge \neg\bar{p}_u^{2k-2} \wedge \neg f^{2k-2}$$

と表せる．ψ は，モデル I_k においてアクション列 $a_1; \ldots; a_{k-1}$ を実行した後の状態で成立するフルーエントの連言である．その状態で a_k を実行しても f が成立することはな

い．つまり $I_k \not\models V_k$ となり矛盾する．従って少なくとも 1 つのアクションの前提条件が真である．

次に $\vec{p}_t^{2k-2}, \ldots, \vec{p}_u^{2k-2}$ が偽，f^{2k-2} が真である場合を考える．このときは，$no\text{-}op(f)^{2k-1}$ を真とすればよい．ここで $no\text{-}op(f)^{2k-1}$ と相互排他関係にあるアクション b が真であったとする．b が $no\text{-}op(f)^{2k-1}$ と相互排他関係にあるということは，相互排他関係の定義より，以下の条件のうち少なくとも 1 つを満たす必要がある：

(i) b の前提条件が $\neg f$ を含む場合

$J_{k-1} \models f^{2k-2}$ という仮定より，b の前提条件が満たされることはなく，b は偽である．

(ii) b の効果が $\neg f$ を含む場合．この場合を，さらに次の 2 つの場合に分ける

- $b = no\text{-}op(\neg f)$ の場合

  $\neg f^{2k}$ は偽であるので，b を真にする必然性はない．

- $b = (a_{k,i}$ **causes** $\vec{e}_i$ **if** $\vec{p}_i)$ の場合

  $\vec{e}_i$ が $\neg f$ を含むならば，$\vec{p}_i^{2k-2}$ は J_{k-1} において偽でなければならない．もし $\vec{p}_i^{2k-2}$ が真であったとすると，それは $\psi \models \vec{p}_i$ ということである．しかし ψ は，モデル I_k においてアクション列 $a_1; \ldots; a_{k-1}$ を実行した後の状態で成立するフルーエントの連言なので，$\psi \models \vec{p}_i$ であるとすると，$I_k \not\models V_k$ となり矛盾する．

(iii) b の前提条件に含まれるあるフルーエント g が，f とレベル $2k-2$ において相互排他関係にある場合

f と g がレベル $2k-2$ で相互排他関係にあるということは，f と g を導いたレベル $2k-3$ のすべてのアクションが相互排他関係にあるということである．ここで，

$$f^{2k-2} \supset c_1^{2k-3} \vee \ldots \vee c_x^{2k-3}$$
$$g^{2k-2} \supset d_1^{2k-3} \vee \ldots \vee d_y^{2k-3}$$

とする．$\mathbf{SatCompile}^M$ の規則 (2) より，相互排他関係にある 2 つのアクションが同時に真となることはない．従って f^{2k-2} が真であるならば，

ある c_i^{2k-3} $(1 \leq i \leq x)$ が真でなければならないが，相互排他関係により $d_1^{2k-3}, \ldots, d_y^{2k-3}$ はすべて偽となるので，g^{2k-2} も偽となる．従ってアクション b の前提条件が満たされることはないので，b は偽である．

従って $\bar{p}_t^{2k-2}, \ldots, \bar{p}_u^{2k-2}$ がすべて偽の場合，$no\text{-}op(f)^{2k-1}$ を真とすればよい．それにより矛盾が生じることはない．

次にあるアクション $a_{k,i}$ $(t \leq i \leq u)$ の前提条件 p_i が真である場合を考える．式 (A.6)(A.9) より，アクション $a_{k,i}$ も真となる．ここで $a_{k,i}$ と相互排他関係にあるアクション b が真であったと仮定する．b が $a_{k,i}$ と相互排他関係にあるということは，その定義より，以下の条件のうち少なくとも 1 つを満たす必要がある：

(I) b の効果と $a_{k,i}$ の前提条件とが矛盾する場合

レベル $2k-1$ にはアクション $a_{k,i}, \ldots, a_{k,s}$ と $no\text{-}op$ アクションしか存在しないので，相互排他関係の定義より，b は $no\text{-}op$ アクションであり，明らかに b の前提条件は満たされない．従って b は偽である．

(II) b の前提条件と $a_{k,i}$ の効果が矛盾する場合

上の場合と同様に b は $no\text{-}op$ アクションである．$b = no\text{-}op(g)$ とする．g に関する式 (A.8) の形式の規則を

$$g^{2k} \supset a_{k,v}^{2k-1} \vee \ldots \vee a_{k,w}^{2k-1} \vee no\text{-}op(g)^{2k-1}$$

とする $(1 \leq v \leq w \leq s)$．g^{2k} が目標条件 ϕ_k に含まれるフルーエントであるか，もしくは ϕ_{k-1} で前提条件を満たすアクションの効果に含まれるフルーエントであるとする．この場合，$a_{k,i}$ が真であることと矛盾する．なぜなら $a_{k,i}$ は，モデル I_k の下でアクション列 $a_1; \ldots; a_{k-1}$ を実行した後の状態で実行可能であり，$I_k \not\models V_k$ となるからである．

従って g は，目標条件に含まれず，かつ，ϕ_{k-1} で前提条件を満たすアクションの効果にも含まれない．よって $g^{2k}, no\text{-}op(g)^{2k-1}$ をともに偽としても矛盾は生じない．

(III) b の効果と $a_{k,i}$ の効果が矛盾する場合．この場合を，さらに次の 2 つの場合に分ける

- b が $no\text{-}op$ の場合

これは上の (II) の場合と同様である．

- b が $a_{k,j}$ $(1 \leq j \leq s)$ の場合

  この場合，$a_{k,i}$ が真であることと矛盾する．なぜなら $a_{k,i}$ は，モデル I_k の下でアクション列 $a_1; \ldots; a_{k-1}$ を実行した後の状態で実行可能であり，$I_k \not\models V_k$ となるからである．

(IV) b の前提条件に含まれるフルーエント g_1 が，$a_{k,i}$ の前提条件に含まれるフルーエント g_2 とレベル $2k-2$ において相互排他関係にある場合

これは先の (iii) の場合と同様である．

従って p_i が真であるならば $a_{k,i}$ を真とすればよい．それにより矛盾が生じることはない．

以上により，Q を充足し，J_{k-1} と矛盾しないモデル J_Q を構成することができる．$J_k = J_{k-1} \cup J_Q$ とすれば，J_k は P_k を充足するモデルである．従って，

$$I_k = \mathbf{Extract}(J_{k-1}) = \mathbf{Extract}(J_k)$$

である．

次に (b) を証明する．評価命題 V_{k-1} を以下で定義する：

$$V_{k-1} = (\psi \textbf{ after } a_1; \ldots; a_{k-1})$$

ここで ψ は，プランニンググラフ $G_k = \mathbf{Expand}^M(D, V_k)$ に含まれるすべてのフルーエント名を含むものとする．その符号は任意である．D と V_{k-1} から作成したプランニンググラフを

$$G_{k-1} = \mathbf{Expand}^M(D, V_{k-1})$$

とする．G_{k-1} を SAT 変換したものを $P_{k-1} = \mathbf{SatCompile}^M(G_{k-1})$ とする．SAT 問題 P_{k-1} から目標条件を取り除いたものを，

$$P'_{k-1} = P_{k-1} \setminus \{\psi^{2k-2}\}$$

とする．P_k は P'_{k-1} を用いて次のように表すことができる：

$$P_k = P'_{k-1} \cup Q$$

ここで Q は，L_{2k-1}, L_{2k} の SAT 変換結果であり，(a) の場合と同様に式 (A.4) 〜 (A.9) で構成される．

P'_{k-1} のモデル集合を $\mathbb{J}'_{k-1}$ とする．このとき $\mathbb{J}_k$ は $\mathbb{J}'_{k-1}$ を用いて

$$\mathbb{J}_k = \{J'_{k-1} \cup J_Q \mid J'_{k-1} \in \mathbb{J}'_{k-1},\quad J'_{k-1} \cup J_Q \models P_k\} \tag{A.10}$$

と表せる（J_Q は Q のモデル）．ここで $\mathbb{J}'_{k-1}$ を次の 2 つの集合に分割する：

- $\mathbb{K}_1 = \{J'_{k-1} \in \mathbb{J}'_{k-1} \mid \mathbf{Extract}(J'_{k-1}) \models V_k\}$
- $\mathbb{K}_2 = \mathbb{J}'_{k-1} \setminus \mathbb{K}_1$

前者は，ある $J' \in \mathbb{J}'_{k-1}$ について，$\mathbf{Extract}(J') = I_k$ となる $I_k \in \mathbb{I}_k$ が存在するようなモデルの集合である．後者は，そうでないモデルの集合である．式 (A.10) を $\mathbb{K}_1, \mathbb{K}_2$ を用いて書き換える：

$$\begin{aligned}\mathbb{J}_k = &\{K_1 \cup J_Q \mid K_1 \in \mathbb{K}_1,\quad K_1 \cup J_Q \models P_k\} \\ &\cup\ \{K_2 \cup J_Q \mid K_2 \in \mathbb{K}_2,\quad K_2 \cup J_Q \models P_k\}\end{aligned} \tag{A.11}$$

いま示すべきことは，任意の $K_2 \in \mathbb{K}_2$ に対し，

$$K_2 \cup J_Q \models P_k \tag{A.12}$$

を満たす J_Q が存在しないことの証明である．もしそのような J_Q が存在すれば，モデル生成アルゴリズムは間違ったモデル $J_k = K_2 \cup J_Q$ を出力することになるからである．

$\mathbb{K}_2$ の定義より，$K_2 \in \mathbb{K}_2$ には次の 2 つの場合が考えられる．

- $K_2 \not\models \phi_k^{2k-2}$ であり，かつ，ϕ_k^{2k} を成立させるアクションの前提条件が K_2 において充足されない場合．
- $K_2 \models \phi_k^{2k-2}$ であるが，$\neg\phi_k^{2k}$ を成立させるアクションの前提条件が K_2 において充足される場合．

まず前者において，式 (A.12) を満たす J_Q が存在しないことを証明する．前者の場合，$K_2 \not\models f^{2k-2}$ となるフルーエント $f \in \phi_k$ と，f を効果に含むが前提条件がレベル $2k-2$ で満たされないアクション $\{a_{k,t}, \ldots, a_{k,u}\}$ が存在する．これらのアクションの前提条件は偽であるので，式 (A.6) (A.9) より，$a_{k,t}, \ldots, a_{k,u}$ は偽である．式 (A.8) より

$$f^{2k} \supset a_{k,t}^{2k-1} \vee \ldots \vee a_{k,u}^{2k-1} \vee \mathit{no\text{-}op}(f)^{2k-1} \tag{A.13}$$

であるので，$no\text{-}op(f)^{2k-1}$ を真としなくてはならないが，$no\text{-}op(f)^{2k-1}$ の前提条件 f^{2k-2} は偽であるので，式 (A.7) で矛盾が生じる．

次に後者において，式 (A.12) を満たす J_Q が存在しないことを証明する．後者の場合，フルーエント $f \in \phi_k$ と，$\neg f$ を効果に含み前提条件がレベル $2k-2$ で満たされるアクション $\{a_{k,v}, \ldots, a_{k,w}\}$ が存在する．式 (A.6) (A.9) より，$a_{k,v}, \ldots, a_{k,w}$ は真である．これらのアクションは，フルーエント f を効果に含むアクション $\{a_{k,t}, \ldots, a_{k,u}, no\text{-}op(f)\}$ とそれぞれ相互排他関係にある．従ってアクション $a_{k,t}, \ldots, a_{k,u}, no\text{-}op(f)$ は偽である．しかし式 (A.13) において f^{2k} が真であるので，矛盾する．

以上より，任意の $K_2 \in \mathbb{K}$ に対し，式 (A.12) を満たす J_Q が存在しないことが証明された．従って，式 (A.10) は次のように書き直すことができる：

$$\mathbb{J}_k = \{K_1 \cup J_Q \mid K_1 \in \mathbb{K}_1 \text{ に対し}, K_1 \cup J_Q \models P_k \text{ となる } J_Q \text{ が存在する}\}$$

このとき任意の $J_k \in \mathbb{J}_k$ に対し，

$$\mathbf{Extract}(J_k) = I_k$$

となる $I_k \in \mathbb{I}_k$ が存在することは明らかである． □

参考文献

[1] Baker, A. B.: Nonmonotonic Reasoning in the Framework of the Situation Calculus, *Artificial Intelligence*, Vol. 49, No. 1–3, pp. 5–23 (1991).

[2] Baral, C. and Gelfond, M.: Representing Concurrent Actions in Extended Logic Programming, *Proceedings of IJCAI-93*, pp. 866–871 (1993).

[3] Baral, C. and Gelfond, M.: Reasoning about Effects of Concurrent Actions, *Journal of Logic Programming*, Vol. 31, No. 1–3, pp. 85–117 (1997).

[4] Blum, A. L. and Furst, M. L.: Fast Planning through Planning Graph Analysis, *Artificial Intelligence*, Vol. 90, No. 1–2, pp. 279–298 (1997).

[5] Bornscheuer, S.-E. and Thielscher, M.: Explicit and Implicit Indeterminism: Reasoning about Uncertain and Contradictory Specifications of Dynamic Systems, *Journal of Logic Programming*, Vol. 31, No. 1–3, pp. 119–155 (1997).

[6] Boutilier, C. and Friedman, N.: Nondeterministic Actions and the Frame Problem, *AAAI Spring Symposium Series* (1995).

[7] Fikes, R. E. and Nilsson, N. J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, Vol. 2, No. 3–4, pp. 189–208 (1971).

[8] Gelfond, M. and Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing*, Vol. 9, No. 3–4, pp. 365–385 (1991).

[9] Gelfond, M. and Lifschitz, V.: Representing Action and Change by Logic Programs, *Journal of Logic Programming*, Vol. 17, No. 2–4, pp. 301–321 (1993).

[10] Gelfond, M. and Lifschitz, V.: Action Languages, *Electronic Transactions on AI*, Vol. 3, No. 16 (1998).

[11] Giunchiglia, E., Kartha, G. N. and Lifschitz, V.: Representing Action: Indeterminacy and Ramifications, *Artificial Intelligence*, Vol. 95, pp. 409–443 (1997).

[12] Giunchiglia, E. and Lifschitz, V.: Dependent Fluents, *Proceedings of IJCAI-95*, pp. 1964–1969 (1995).

[13] Hanks, S. and McDermott, D.: Nonmonotonic Logic and Temporal Projection, *Artificial Intelligence*, Vol. 33, No. 3, pp. 379–412 (1987).

[14] Hopcroft, J. E. and Ullman, J. D.: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Massachusetts (1979). (野崎 昭弘ほか 訳：オートマトン 言語理論 計算論 (I, II)，サイエンス社 (1984)).

[15] Kambhampati, S., Parker, E. and Lambrecht, E.: Understanding and Extending Graphplan, *Proceedings of ECP-97: Recent Advances in AI Planning* (Steel, S. and Alami, R.(eds.)), LNAI, Vol. 1348, Berlin, Springer, pp. 260–272 (1997).

[16] Kartha, G. N.: Soundness and Completeness Theorems fo Three Formalizations of Action, *Proceedings of IJCAI-93* (1993).

[17] Kartha, G. N. and Lifschitz, V.: Actions with Indirect Effects (Preliminary Report), *Proceedings of KR-94*, pp. 341–350 (1994).

[18] Kautz, H. and Selman, B.: Pushing the Envelope: Planning, Propositional Logic and Stochastic Search, *Proceedings of AAAI-96*, pp. 1194–1201 (1996).

[19] Kautz, H. and Selman, B.: Unifying SAT-based and Graph-based Planning, *Proceedings of IJCAI-99*, pp. 318–325 (1999).

[20] Li, C. M. and Anbulagan: Heuristics Based on Unit Propagation for Satisfiability Problems, *Proceedings of IJCAI-97*, pp. 366–371 (1997).

[21] Lifschitz, V.: On the Semantics of STRIPS, *Reasoning about Actions and Plans* (Georgeff, M. P. and Lansky, A. L.(eds.)), Kaufmann, Los Altos, CA, pp. 1–9 (1987).

[22] Lifschitz, V.: Two Components of an Action Language, *Annals of Mathematics and Artificial Intelligence*, Vol. 21, pp. 305–320 (1997).

[23] Lifschitz, V.: Action languages, Answer Sets, and Planning, *The Logic Programming Paradigm: A 25-Year Perspective*, Spring-Verlag, pp. 357–373 (1999).

[24] McCarthy, J.: Circumscription—A Form of Nonmonotonic Reasoning, *Artificial Intelligence*, Vol. 13, No. 1–2, pp. 27–39 (1980).

[25] McCarthy, J.: Mathematical Logic in Artificial Intelligence, *Dædalus*, Vol. 117, No. 1, pp. 297–311 (1988).

[26] McCarthy, J. and Hayes, P. J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence, *Machine Intelligence*, Vol. 4, pp. 463–502 (1969).

[27] McDermott, D. and Doyle, J.: Nonmonotonic Logic I, *Artificial Intelligence*, Vol. 13, No. 1–2, pp. 41–72 (1980).

[28] 鍋島英知, 井上克巳: アクション言語 $\mathcal{A}$ のためのオートマトン理論, 情報処理学会論文誌, Vol. 38, No. 3, pp. 462–471 (1997).

[29] 鍋島英知, 井上克巳, 羽根田博正: Java 言語によるアクション言語の処理系の実装, 第 13 回人工知能学会全国大会論文集, pp. 26–29 (1999).

[30] 鍋島英知, 井上克巳, 羽根田博正: アクション言語 $\mathcal{A}$ における因果関係の学習, 情報処理学会 第 115 回 知能と複雑系研究会, pp. 41–46 (1999).

[31] 鍋島英知, 井上克巳, 羽根田博正: 有限オートマトンに基づく非決定性アクション言語, 情報処理学会論文誌, Vol. 40, No. 10, pp. 3661–3671 (1999).

[32] Nabeshima, H., Inoue, K. and Haneda, H.: Implementing an Action Language Using a SAT Solver, *Proceedings of ICTAI-2000*, pp. 96–103 (2000).

[33] 鍋島英知, 井上克巳, 羽根田博正: SAT ソルバによるアクション言語処理系の実装, 電子情報通信学会技術研究報告, Vol. 99, No. 534, pp. 53–60 (2000).

[34] Pednault, E. P. D.: ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus, *Proceedings of KR-89* (Brachman, R. J., Levesque, H. J. and Reiter, R.(eds.)), San Mateo, California, Morgan Kaufmann, pp. 324–332 (1989).

[35] Quinlan, J. R.: Induction of Decision Trees, *Readings in Machine Learning* (Shavlik, J. W. and Dietterich, T. G.(eds.)), Morgan Kaufmann (1990). Originally published in *Machine Learning* 1:81–106, 1986.

[36] Reiter, R.: A Logic for Default Reasoning, *Artificial Intelligence*, Vol. 13, No. 1–2, pp. 81–132 (1980).

[37] Reiter, R.: The Frame Problem in the Situation Calculus: a Simple Solution (Sometimes) and a Completeness Result for Goal Regression, *Artificial Intelligence and the Mathematical Theory of Computation* (Lifschitz, V.(ed.)), Academic Press, San Diego, pp. 418–420 (1991).

[38] Russell, S. J. and Norvig, P.: *Artificial Intelligence : A Modern Approach*, Prentice-Hall Intl Edns. (1995). (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 (1997)).

[39] Turner, H.: Representing Actions in Logic Programs and Default Theories: A Situation Calculus Approach, *Journal of Logic Programming*, Vol. 31, No. 1–3, pp. 245–298 (1997).

[40] Weld, D. S.: Recent Advances in AI Planning, *AI Magazine*, Vol. 20, No. 2, pp. 93–123 (1999).

[41] 山田誠二: リアクティブプランニングにおける学習, 日本ロボット学会誌, Vol. 13, No. 1, pp. 38–43 (1995).

執筆論文一覧

学術論文誌

- 鍋島 英知，井上 克巳，羽根田 博正．有限オートマトンに基づく非決定性アクション言語．情報処理学会論文誌，40(10):3661–3671, 1999.
- 鍋島 英知，井上 克巳．アクション言語 $\mathcal{A}$ のためのオートマトン理論．情報処理学会論文誌，38(3):462–471, 1997.

国際会議論文

- Hidetomo Nabeshima, Katsumi Inoue and Hiromasa Haneda．Implementing an Action Language Using a SAT Solver．*ICTAI 2000 (The Twelfth IEEE International Conference on Tools with Artificial Intelligence)*, Vancouver, British Columbia, Canada, 96–103, Nov. 2000.

研究報告

- 鍋島 英知，井上 克巳，羽根田 博正．SAT ソルバによるアクション言語処理系の実装．情報処理学会 第 119 回 知能と複雑系研究会，2000.
- 鍋島 英知，井上 克巳，羽根田 博正．アクション言語 $\mathcal{A}$ における因果関係の学習．情報処理学会 第 115 回 知能と複雑系研究会，1999.
- 鍋島 英知，井上 克巳．アクション言語における非決定性アクションの表現と推論に関する考察．人工知能学会 第 32 回 人工知能基礎論研究会，1998.

- 鍋島 英知，井上 克已．アクション言語 $\mathcal{A}$ のためのオートマトン理論．平成 8 年度「人工知能のための高次推論原理と知能プログラミング言語に関する研究」 第 1 回研究会，1996.
- 鍋島 英知，井上 克已．アクション言語 $\mathcal{A}$ を表現するオートマトンモデル．第 104 回 情報処理学会人工知能研究会，1996.

口頭発表

- 宇治田 正和，鍋島 英知，井上 克已，羽根田 博正．アクション言語処理系評価環境の GUI による構築．平成 12 年 電気関係学会関西支部連合大会, 2000. （発表予定）
- 鍋島 英知，坂田 史郎，井上 克已，羽根田 博正．SAT ソルバと後向き推論によるアクション言語 $\mathcal{A}$ の実装．第 14 回 人工知能学会全国大会，2000.
- 鍋島 英知，井上 克已，羽根田 博正．Java 言語によるアクション言語処理系の実装．第 13 回 人工知能学会全国大会，1999.
- 井上 克已，山本 友和，鍋島 英知．アクション言語のためのボトムアップ処理系．第 11 回 人工知能学会全国大会，1997.