人　工　知　能　104－6
（1996．5．14）

アクション言語 $\mathcal{A}$ を表現するオートマトンモデル

鍋島 英知，井上 克巳

豊橋技術科学大学 情報工学系
〒441 豊橋市天伯町雲雀ヶ丘 1-1
nabe@ai.tutics.tut.ac.jp
inoue@tutics.tut.ac.jp

本論文では，高級レベルのアクション言語 $\mathcal{A}$ と有限オートマトンの関係について議論する．最も重要な関係はこれらの等価性である．すなわち，$\mathcal{A}$ による任意の領域記述が決定性有限オートマトンに変換でき，逆に任意の有限オートマトンが $\mathcal{A}$ による記述に変換できる．このことから，あるアクション列が計画を遂行できるかどうかの判定が，対応するオートマトンがそのアクション列を受理するかどうかの問題に帰着され，さらにプランニング問題に対する最良の解も正規表現を用いて表現することができる．ここに，状態変化を伴う推論とオートマトン理論が深く関連付けられ，アクション言語に対する今後の見通しがよくなった．

Automaton model for action language $\mathcal{A}$

Hidetomo NABESHIMA, Katsumi INOUE

Department of Information and Computer Sciences
Toyohashi University of Technology
1-1 Hibarigaoka, Tempaku-cho, Toyohashi, Aichi 441, Japan

We investigate the relationship between the high-level action language $\mathcal{A}$ and finite automata (FA). We show $\mathcal{A}$ and FA are *equivalent*, that is, any domain description in $\mathcal{A}$ can be transformed to an FA, and conversely, any FA can be transformed to a description in $\mathcal{A}$. With these results, planning problems can be characterized by regular expressions, and vice versa. Here, reasoning about action meets the automata theory.

1. はじめに

最近，状態の変化やアクションを表現する研究において，高級なレベルのアクション言語が使用されている．アクション言語とは，アクションによる状態の変化（効果）を記述するために用いられる自然言語表現を採り入れた形式的モデルと考えられる．

アクション言語の中で最もシンプルなものの1つとして言語 $\mathcal{A}$ が Gelfond と Lifschitz[1] により提案されている．言語 $\mathcal{A}$ では，フレーム問題をその言語仕様のなかで解決しようとしている．しかし，言語 $\mathcal{A}$ そのものに対する処理系は開発されていない．

そこで，言語 $\mathcal{A}$ により記述された領域においてプランニングを行なうため，文献 1) では，拡張論理プログラムに変換する手法を提案している．しかしその手続きは完全なものではなく，しかも変換できない領域が存在する．

本論文では，言語 $\mathcal{A}$ による領域記述と等価な有限オートマトンが構成できること，逆に有限オートマトンと等価な言語 $\mathcal{A}$ による領域記述が構成できることを示す．言い換えると，言語 $\mathcal{A}$ は有限オートマトンと等価であることを示す．これにより，従来の手法では変換できなかった領域も含め，矛盾のない領域であればすべて正しく決定性有限オートマトンに変換できる．またプランニング問題も，オートマトンの受理言語として表すことができ，アクション列を正則表現を用いて簡潔に記述できるようになる．

まず言語 $\mathcal{A}$ を2章で紹介する．続く3章で，言語 $\mathcal{A}$ による領域記述を DFA の集合へ変換する手続きを定義し，その正当性を示す．4章では，逆に DFA を言語 $\mathcal{A}$ による領域記述に変換する手続きを定義し，その正当性を示す．5章が，本論文の考察である．

2. アクション言語 $\mathcal{A}$

アクション言語 $\mathcal{A}$[1] は，**value-命題**と **effect-命題**という2種類の命題からなる．value-命題は，“初期状態からアクション列 $A_1;\ldots;A_m$ を実行するとフルーエント F が成立する” ことを言明する：

$$F \textbf{ after } A_1;\ldots;A_m \quad (m \geq 1) \qquad (1)$$

$m=0$ のときは簡潔に：

initially F

と記述する．effect-命題は，“フルーエント $P_1,\ldots,P_n$ を含む状態でアクション A を実行するとフルーエント F が成立する” ことを言明する：

$$A \textbf{ causes } F \textbf{ if } P_1,\ldots,P_n \quad (n \geq 1) \qquad (2)$$

$n=0$ のときは **if** を落して：

$$A \textbf{ causes } F$$

と記述する．$P_1,\ldots,P_n$ を前提条件という．領域記述は，この2種類の命題の集合である．

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 $fluent(D)$ で表す．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．単に “フルーエント F” と記述した場合，F は正または負のフルーエントとする．**状態** σ を，すべてのフルーエントについて，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．つまり，

$$\sigma = S \cup \{\neg F \mid F \in fluent(D), F \notin S\} \qquad (3)$$

ここで $S \subseteq fluent(D)$ である☆．

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) からなる．**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：

$$I^{A_1;\ldots;A_m} = (\Phi,\sigma_0)^{A_1;\ldots;A_m} = \Phi(A_m,\Phi(A_{m-1},\ldots,\Phi(A_1,\sigma_0),\ldots). \qquad (4)$$

式 (1) のような value-命題に対して，

$$F \in I^{A_1;\ldots;A_m} \qquad (5)$$

であれば，式 (1) が解釈 I において真であるといい，その他の場合を偽であるという．

解釈 I が領域記述 D のモデルであるのは，D に含まれるすべての value-命題が I において真であり，すべてのアクション名，すべてのフルーエント F，すべての状態 σ に対し，以下の条件が満たされる場合である：

(1) もし，D が状態 σ で $P_1,\ldots,P_n$ を満たす式 (2) のような effect-命題を含めば，そのとき $F \in \Phi(A,\sigma)$ である，

(2) もし，D がそのような effect-命題を含まなければ，$F \in \sigma$ の場合に限り $F \in \Phi(A,\sigma)$ である☆☆．

上の条件を満たす遷移関数は，たかだか1つ存在する．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．領域記述はモデルを持てば無矛盾であり，モデルが1つしかなければ完全である．もし，value-命題が領域記述 D の任意のモデルで真であるならば，D はその value-命題

☆ 文献 1) では，負のフルーエントを含まないように状態を定義しているが，言語 $\mathcal{A}$ を定義する上で差異は生じない．

☆☆ この条件は慣性の法則を表している．

を帰結するといい，以下のように記述する：

$$D \models (F \textbf{ after } A_1;\ldots;A_m) \quad (m \geq 0). \qquad (6)$$

例題 1　Yale Shooting 領域

initially $\neg loaded$.
initially $alive$.
$load$ **causes** $loaded$.
$shoot$ **causes** $\neg alive$ **if** $loaded$.
$shoot$ **causes** $\neg loaded$.

例題 2　Murder Mystery 領域

initially $alive$.
$load$ **causes** $loaded$.
$shoot$ **causes** $\neg alive$ **if** $loaded$.
$shoot$ **causes** $\neg loaded$.
$\neg alive$ **after** $shoot;wait$.

3. 言語 $\mathcal{A}$ からオートマトンへの変換

3.1 諸　定　義

決定性有限オートマトン (DFA)　DFA $M = (Q, \Gamma, \delta, q_0, G)$ の定義を以下で与える：

Q : 状態の集合，
Γ : 入力アルファベット，
δ : 状態遷移関数 $(\delta : \Gamma \times Q \to Q)$，
q_0 : 初期状態 $(q_0 \in Q)$，
G : 最終状態の集合 $(G \subseteq Q)$.

入力アルファベット　入力アルファベット Γ を，領域記述 D に含まれるすべてのアクション名の集合 $action(D)$ で定義する：

$$\Gamma = action(D).$$

状態　オートマトンの状態 q を，言語 $\mathcal{A}$ における状態の定義と同じく，領域記述 D に含まれるすべてのフルーエントについて，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：

$$q = S \cup \{\neg F \mid F \in fluent(D), F \notin S\} \qquad (7)$$

ここで $S \subseteq fluent(D)$ である．

目標状態　フルーエント F に関する目標状態の集合を $goal(F)$ で表す．$goal(F)$ は以下で定義される：

$$goal(F) = \{q \in Q \mid F \in q\}.$$

ここで Q は状態の集合である．F を目標状態とする DFA M を $M(F)$ で表し，M の最終状態の集合を $goal(F)$ で置き換えたものと定義する：

$$M(F) = (Q, action(D), \delta, q_0, goal(F)).$$

ここで $M = (Q, action(D), \delta, q_0, G)$ とする．F を目標状態とする DFA の集合 $\mathcal{M}$ を $\mathcal{M}(F)$ で表し，すべての $M \in \mathcal{M}$ を $M(F)$ で置き換えたものと定義する：

$$\mathcal{M}(F) = \{M(F) \mid M \in \mathcal{M}\}.$$

部分状態　複数の状態を効率良く表現するために，部分状態 q' を定義する．部分状態は，すべてのフルーエントを含むのではなく，一部のフルーエントを含む状態である：

$$q' = S' \cup \{\neg F \mid F \in S'', F \notin S'\} \qquad (8)$$

ここで $S', S'' \subseteq fluent(D)$, $S' \cap S'' = \phi$ である．部分状態は状態の集合と対応する．例えば，$fluent(D) = \{loaded, alive\}$ という領域における部分状態 $q' = \{loaded\}$ は，$\{loaded, alive\}$ という状態と $\{loaded, \neg alive\}$ という状態を表す．また，部分状態 $q'' = \{\phi\}$ はすべての状態を表す．

絶対値　フルーエント名 F に対して絶対値の演算を定義する：

$$|F| = F, \quad |\neg F| = F.$$

絶対値の演算を拡張し，状態についても定義する：

$$|q| = \{|F| \mid F \in q\}.$$

関連するフルーエントの集合　$rel(A)$ は，アクション A に関連するフルーエントを集める：

$$rel(A) = \bigcup \{|F|, |P_1|, \ldots, |P_n|\}$$
$$\text{for all } (A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n) \in D,$$
$$rel(\{A_1, \ldots, A_n\}) = rel(A_1) \cup \ldots \cup rel(A_n).$$

部分状態の分割　$div(Q, F)$ は，すべての部分状態 $q \in Q$ を，フルーエント F により，$q \cup \{F\}$, $q \cup \{\neg F\}$ という 2 つの部分状態に分割する：

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

$$div(Q, \{F_1; \ldots; F_n\}) =$$
$$div(div(\ldots div(div(Q, F_1), F_2) \ldots, F_{n-1}), F_n).$$

3.2 変換手続き Init, Trans

言語 $\mathcal{A}$ による領域記述は，2 つの手続き **Init**, **Trans** により DFA の集合に変換される．**Init** (図 1) は各 DFA の初期状態を求め，**Trans** (図 2) は遷移関数を求める．これらの手続きが生成する DFA は，領域記述のモデルとそれぞれ対応している (3.3節で証明する)．よって生成される各 DFA は，その初期状態によってのみ異なる．領域記述が完全であれば DFA は唯 1 つ生成される．

3.3 変換手続き Init, Trans の正当性

言語 $\mathcal{A}$ による領域記述を D とする．手続き **Init**,

Transは，D をDFAの集合 $\mathcal{M}$ に変換する．

定理1 モデルに関する手続きの正当性：D のモデル集合とDFA集合 $\mathcal{M} = \mathbf{Trans}(D, \mathbf{Init}(D))$ は，一対一対応する．

証明 D の初期状態の集合を Σ_0，遷移関数を Φ とする．すべてのオートマトン $M \in \mathcal{M}$ の初期状態の集合を Q_0 とし，任意のオートマトン $M \in \mathcal{M}$ の遷移関数を δ とする．$\Sigma_0 = Q_0$ かつ $\Phi = \delta$ が示せれば証明は完成する．図2より，D の遷移関数はオートマトンの遷移関数に等しい．

そこで $\Sigma_0 = Q_0$ を証明するために領域記述に含まれるvalue-命題の数に関する数学的帰納法を利用する．領域内に含まれるvalue-命題の数が k であるとき，その領域を D^k で表す．D^k の初期状態の集合を Σ^k で表し，D^k を変換して得られるオートマトンの集合の初期状態の集合を Q^k で表す．

基底 $\Sigma_0^0 = Q_0^0$ が成立することは，手続き **Init** より明らかである．

帰納法の仮定 $\Sigma_0^k = Q_0^k$.

帰納的ステップ $\Sigma_0^{k+1} = Q_0^{k+1}$ を証明する．領域記述 D^{k+1} を

$$D^{k+1} = D^k \cup \{F \textbf{ after } A_1; \ldots; A_m \quad (m \geq 0)\}$$

とすれば，

$$\Sigma_0^{k+1} = \{\sigma_0 \in \Sigma_0^k \mid F \in (\Phi, \sigma_0)^{A_1; \ldots; A_m}\}$$

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．ところで手続き **Init** より，

$$Q_0^i = div(Q^i, fluent(D))$$
$$Q^{k+1} = \{q_0 \in div(Q^k, S) \mid F \in (\Phi, q_0)^{A_1; \ldots; A_m}\}$$

ここで $S = \{F\} \cup rel(\{A_1, \ldots, A_m\})$ である．よって，

$$\begin{aligned} Q_0^{k+1} &= \{q_0 \in div(Q^k, fluent(D)) \mid F \in (\Phi, q_0)^{A_1; \ldots; A_m}\} \\ &= \{q_0 \in Q_0^k \mid F \in (\Phi, q_0)^{A_1; \ldots; A_m}\} \\ &= \Sigma_0^{k+1}. \quad \square \end{aligned}$$

定理1より，次の定理が導かれる：

定理2 論理的帰結に関する手続きの正当性：

$$D \models (F \textbf{ after } A_1; \ldots; A_m) \quad (m \geq 0)$$
$$\Updownarrow$$
DFA $M \in \mathcal{M}(F)$ は，アクション列 $A_1; \ldots; A_m$ を受理する．

ただし $\mathcal{M} = \mathbf{Trans}(D, \mathbf{Init}(D))$ とする．

3.4 変換例

例題1，例題2の各領域を変換した結果をそれぞれ図3，図4に示す．これらの領域は完全であるため，唯1つのオートマトンに変換される．

例題1において $D \models (\neg alive \textbf{ after } load; wait; shoot)$ であることが，定理2と図3のDFAが *load;wait;shoot* を受理することから分かる．

また例題2において $D \models (\textbf{initially } loaded)$ であることが，定理2と，図4において $loaded \in Q_0$ であることから分かる．

Init(D)
入力：言語 $\mathcal{A}$ による領域記述 D，
　　value-命題の数を k とする．
出力：初期状態の集合 Q_0
begin
　$Q^0 := \{\phi\}$;
　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**
　　$T_0^j := div(T_0^j, rel(A_i))$;
　　$T_i^j := \Phi(A_i, div(T_{i-1}^j, rel(A_i)))$;
　　$Q^j := \{q_0 \in T_0^j \mid F \in q_m,\ q_m \in T_m^j,$
　　　　q_mの初期状態は q_0である $\}$;
　$Q_0 := div(Q^k, fluent(D))$;
end.

図1　初期状態を求める手続き Init

Trans(D, Q_0)
入力：領域記述 D，初期状態の集合 Q_0
出力：DFAの集合 $\mathcal{M}$
begin
　$Q := div(\{\phi\}, fluent(D))$;
　for all $q \in Q$ **do**
　　for all $A \in action(D)$ **do**
　　$\delta(A, q) := \Phi(A, q)$;
　$\mathcal{M} := \bigcup_{q_0 \in Q_0} \{(Q, action(D), \delta, q_0, Q)\}$;
end.

図2　遷移関数を求める手続き Trans

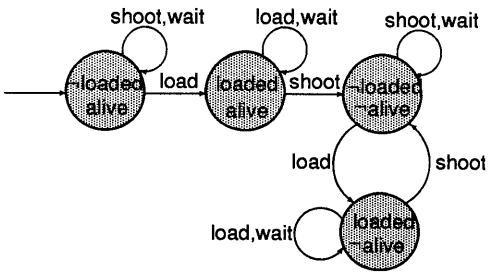

図3　Yale Shooting 領域の変換結果

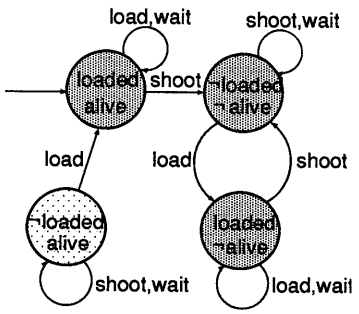

図4　Murder Mystery 領域の変換結果

4. オートマトンから言語 $\mathcal{A}$ への逆変換

4.1 オートマトンの状態

オートマトンを言語 $\mathcal{A}$ による領域記述へ逆変換するには，オートマトンの状態がフルーエントの集合で表されている必要がある．オートマトンの状態数を $|Q|$ とすれば，$|Q|$ 個の状態を表現するためには $\lceil \log_2 |Q| \rceil$ 個のフルーエントを用意すれば良い．オートマトンの状態を，前章と同様に式 (7) で定義する．

4.2 逆変換手続き $\mathbf{Initially}^{-1}$, $\mathbf{Effect}^{-1}$, Compress

決定性有限オートマトンは，3つの手続きにより言語 $\mathcal{A}$ による領域記述に逆変換される．$\mathbf{Initially}^{-1}$ (図5) は，オートマトンの初期状態から value-命題を生成する．$\mathbf{Effect}^{-1}$ (図6) は，オートマトンの遷移関数から effect-命題を生成する．Compress (図7) は，冗長な effect-命題を削除する．手続き Compress は，言語 $\mathcal{A}$ への逆変換のためのステップというより，領域記述の簡単化のための手続きとして一般に利用できる．

4.3 逆変換手続き $\mathbf{Initially}^{-1}$, $\mathbf{Effect}^{-1}$, Compress の正当性

決定性有限オートマトンを M とする．手続き

$\mathbf{Initially}^{-1}(M)$
入力：DFA M
出力：領域記述 D_1
begin
$D_1 := \bigcup_{F \in q_0} \{\textbf{initially } F\};$
end.

図5　value-命題を求める手続き $\mathbf{Initially}^{-1}$

$\mathbf{Effect}^{-1}(M)$
入力：DFA M
出力：領域記述 D_2
begin
$D_2 := \bigcup_{\substack{F \in \delta(A,q) \\ A \in action(D) \\ q \in Q}} \{A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n\},$
ただし $q = \{P_1, \ldots, P_n\}$ とする;
end.

図6　初期状態候補を求める手続き $\mathbf{Effect}^{-1}$

$\mathbf{Initially}^{-1}$, $\mathbf{Effect}^{-1}$, Compress は，M を言語 $\mathcal{A}$ による領域記述 D に変換する．

定理3　モデルに関する手続きの正当性：M を逆変換して得られる D は完全である．ただし $D = \mathbf{Compress}(\mathbf{Initially}^{-1}(M), \mathbf{Effect}^{-1}(M))$ とする．

証明 図5よりオートマトンの初期状態 q_0 は，領域記述 D の初期状態 σ_0 に等しい．また図6より，オートマトンの遷移関数は，手続き $\mathbf{Effect}^{-1}$ より得られる領域記述 D_2 の遷移関数に等しい．後述の定理4より，手続き Compress により領域記述 D_2 から冗長な effect-命題を削除しても遷移関数は変化しない．よって M を逆変換して得られる D は完全である．　□

定理3より，DFA M を逆変換して得られる領域記述 D は完全であるので，D は唯1つのモデルをもつ．

定理4　Compress による遷移関数の不変性：手続き Compress により領域記述を簡単化しても，遷移関数は変化しない．

証明 手続き Compress は，3種類の冗長な effect-命題を取り除く．それぞれトートロジー消去，包摂テスト，div の逆操作に対応している．ここでは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 effect-命題を削除する：

Compress(D_1, D_2)
入力：$D_1 = \mathbf{Initially}^{-1}(M)$,
　　$D_2 = \mathbf{Effect}^{-1}(M, D)$
出力：領域記述 D
begin
　$D := D_1 \cup D_2$;
　while (effect-命題が D より削除可能) **do**
　　E, E_1, E_2, E_a, E_b を D の要素とする;
　　$E =$ A **causes** F **if** $P_1, \ldots, P_n$;　$(n \geq 1)$
　　if $F \in \{P_1, \ldots, P_n\}$ **then**
　　　$D := D - \{E\}$;
　　$E_1 =$ B **causes** G **if** $Q_1, \ldots, Q_l$;　$(l \geq 0)$
　　$E_2 =$ B **causes** G **if** $R_1, \ldots, R_m$;　$(m \geq l)$;
　　if $\{Q_1, \ldots, Q_l\} \subseteq \{R_1, \ldots, R_m\}$ **then**
　　　$D := D - \{E_2\}$;
　　$E_a =$ C **causes** H **if** $S_1, \ldots, S_k$;
　　$E_b =$ C **causes** H **if** $T_1, \ldots, T_k$;　$(k \geq 0)$
　　if $\neg S_i = T_j$ and $\{S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n\}$
　　　$= \{T_1, \ldots, T_{j-1}, T_{j+1}, \ldots, T_m\}$ **then**
　　　$D := (D - \{E_a, E_b\}) \cup$
　　　　$\{C$ **causes** H
　　　　　if $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n\}$;
end.

図 7　effect-命題を簡単化する手続き Compress

$$A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n \quad (n \geq 1), \qquad (9)$$
$$\exists\, i\ (P_i = F)\ \ (1 \leq i \leq n).$$

このような命題を削除しても遷移関数が変化しないことを示すために，すべての状態の集合 Σ を考える．Σ を，次を満たす 2 つの集合 Σ_1, Σ_2 に分割する：

$\{P_1, \ldots, P_n\} \subseteq \sigma_1$　　for any $\sigma_1 \in \Sigma_1$,
$\{P_1, \ldots, P_n\} \not\subseteq \sigma_2$　　for any $\sigma_2 \in \Sigma_2$.

Σ_2 に含まれる任意の状態においてアクション A を実行しても，式 (9) は遷移先に影響を及ぼさない．よって Σ_2 のみを対象にした場合，式 (9) を削除しても遷移関数に変化はない．また Σ_1 に含まれる任意の状態に対して，式 (9) は：

$F \in \Phi(A, \sigma_1)$　for any $\sigma_1 \in \Sigma_1$

であることを述べている．ところで $F = P_i$ なので $F \in \sigma_1$ である．よって Σ_1 のみを対象にした場合，式 (9) を削除しても慣性の法則により遷移関数に変化はない．　□

定理 3 より，次の定理が導かれる：

定理 5　論理的帰結に関する逆変換手続きの正当性

DFA $M(F)$ は，アクション列
$A_1; \ldots; A_m$ を受理する．
　　⇕
$D \models (F$ **after** $A_1; \ldots; A_m)$　$(m \geq 0)$

ただし $D =$ **Compress**$(\mathbf{Initially}^{-1}(M), \mathbf{Effect}^{-1}(M))$ とする．

4.4 変 換 例

図 4 のオートマトンを言語 $\mathcal{A}$ による領域記述に逆変換した結果を以下に示す．

initially *alive*.
load **causes** *loaded*.
shoot **causes** $\neg$*alive* **if** *loaded*.
shoot **causes** $\neg$*loaded*.
initially *loaded*.

最後の value-命題が，例題 2 のものと異なっている．しかし領域の初期状態は等しく，本質的に等価である．

図 8 は，$0; (1; 0)^*$ を受理する DFA M である☆．M を言語 $\mathcal{A}$ による領域記述に逆変換した結果を以下に示す．

initially x.
initially y.
0 **causes** $\neg y$ **if** y.
0 **causes** $\neg x$ **if** $x, \neg y$.
0 **causes** x **if** $\neg x, y$.
1 **causes** $\neg x$ **if** x.
1 **causes** $\neg y$ **if** y.
1 **causes** y **if** $x, \neg y$.

この領域記述において，M の受理言語 $0; (1; 0)^*$ に対応するアクション列 S は，次を満たす 2 つのアクション列 S_1 と S_2 より，$S = S_1 \cap S_2$ と表せる☆☆．

x **after** S_1.
$\neg y$ **after** S_2.

5. 考　　察

5.1 言語 $\mathcal{A}$ と DFA の等価性

定義 1　フルーエント F を実現するアクション列：言語 $\mathcal{A}$ による領域記述を D とする．D が，式 (1) のような value-命題を帰結するとき：

$D \models (F$ **after** $A_1; \ldots; A_m)$　$(m \geq 0)$

アクション列 $A_1; \ldots; A_m$ は，フルーエント F を実現するという．

定義 2　オートマトン集合の受理言語：オートマト

☆ A^* は A の Kleene-閉包を表す
☆☆ このような表現となるのは，言語 $\mathcal{A}$ では **after** の前に 1 つ以上のフルーエントを記述できないためである．

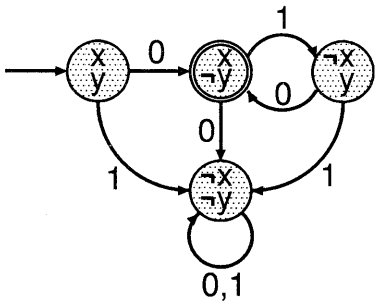

図8　$0;(1;0)^*$ を受理する DFA M

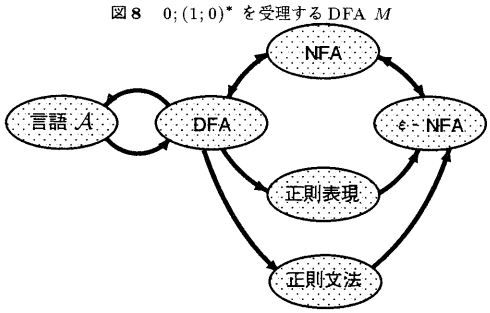

図9　言語 $\mathcal{A}$ と有限オートマトン

ンの集合を $\mathcal{M}$ とする．$\mathcal{M}$ の受理言語 $L(\mathcal{M})$ を，すべてのオートマトン $M \in \mathcal{M}$ の受理言語 $L(M)$ の和集合と定義する：

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

変換手続きと逆変換手続きの正当性，そして上の定義より次の2つの定理が成り立つ．

定理6　領域記述 D とそれを変換して得られる DFA の集合 $\mathcal{M}$ を考える．フルーエント F を実現するアクション列の集合は，F を目標状態とするオートマトンの集合 $\mathcal{M}(F)$ の受理言語に等しい．

定理7　DFA M とそれを変換して得られる領域記述 D を考える．F を目標状態とするオートマトン $M(F)$ の受理言語は，フルーエント F を実現するアクション列の集合に等しい．

この2つの定理より，言語 $\mathcal{A}$ と有限オートマトンが等価であることが示せた．また有限オートマトンで受理される言語と，正則表現で表される集合は一致する[2]（図9）．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表す．よって，あるフルーエントを実現するアクションの列を正則表現で表すことが可能である．

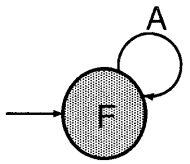

図10　反例のオートマトンへの変換結果

5.2 プランニング

定義3　極小列：$action^*(D)$ を，領域 D に含まれるすべてのアクションの列からなる集合とする．$L \subseteq action^*(D)$ に対して，$w \in L$ が L の極小列であるとは，w と異なる任意のアクション列 $w' \in L$ に対して，w' が w の部分列とならないことをいう．

定義4　F に関するプランニングの解：F を実現するアクション列の極小列を，F に関するプランニングの解という．

例題1の Yale Shooting 領域について，フルーエント $\neg alive$ を成立させる（被害者を殺害する）ためのプランニングを行なう．まず Yale Shooting 領域をオートマトンに変換し（図4），受理言語 $L(\mathcal{M}(\neg alive))$ を求める：

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

この結果より，$\neg alive$ に関するプランニングの解は $load;shoot$ と求まる．

5.3 完全性

文献1)では，言語 $\mathcal{A}$ による領域記述を拡張論理プログラムへ変換する手続き π が提案されている．しかし，この変換手続き π は，健全ではあるが完全ではない．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.

このような2つの命題からなる領域 D が，**initially** F を帰結することは明らかである．しかし，この命題の π による変換結果—$holds(F, s0)$.—は，拡張論理プログラム $\pi(D)$ により含意されない．

オートマトンへの変換手続きは，その正当性より完全かつ健全である．よって，領域 D を変換したオートマトン（図10）の初期状態には F が含まれる．

5.4 類似した命題

領域記述を拡張論理プログラムへ変換する前述の

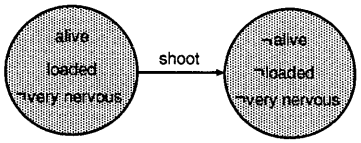

図 11　類似した命題の変換結果の一部

手続き π は，類似した命題を含む領域を変換できない．ここで類似した命題とは，前提条件のみが異なる effect-命題をいう：

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

shoot **causes** $\neg$*alive* **if** *very nervous*.

類似した命題が領域記述のなかに存在すると，π による変換結果は健全でなくなる．

オートマトンへの変換手続きでは，領域記述のなかに類似した命題が存在しても健全性は損なわれない（図 11）．

6. おわりに

本論文では，言語 $\mathcal{A}$ による領域記述と有限オートマトンを，それぞれ他方に変換する手続きを提案し，その正当性を示した．言い換えると，言語 $\mathcal{A}$ と有限オートマトンが等価であることを示した．

これにより，プランニング問題をオートマトンの受理言語を求める問題として表せる．従来の手法では扱えなかった領域も，オートマトンに変換することにより利用できるようになった．プランニングの解であるアクション列は，正則表現を用いて簡潔に記述できる．

本研究の最終的な目的は，さまざまなオートマトンのモデルを用いた言語 $\mathcal{A}$ の拡張にある．すでに提案されている言語 $\mathcal{A}$ を拡張したクラスのアクション言語のモデル☆ と，各種のオートマトンとの間にどのような関係があるのか研究することが今後の課題である．

参考文献

1) 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).

2) Hopcroft, J. E. and Ullman, J. D.: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, MA. (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))

3) Baral, C. and Gelfond, M.: Representing Concurrent Actions in Extended Logic Programming, *Proceedings of IJCAI-93*, vol.2, pp.866–871 (1993).

4) Kartha, G. N. and Lifschitz, V.: Actions with indirect effects (preliminary report), *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pp.341–350 (1994).

5) Giunchiglia, E. and Lifschitz, V.: Dependent fluents, *Proceedings of IJCAI-95*, vol.2, pp.1964–1969 (1995).

☆ 非決定性アクションを入れた $\mathcal{A}$ の拡張[1]，同時実行アクションを入れた $\mathcal{A}_c$[3]，フルーエント間の制約を入れた $\mathcal{AR}$[4]，フルーエント間の依存関係を入れた $\mathcal{ARD}$[5] などが提案されている．