効率の良いモデル空間探査のための抽象化

大矢野　潤

1　はじめに

WWW（World Wide Web）の発達にともない，インターネット上で多様なサービスが提供されており，それらのうちいくつかのサービスは他のサービスと相互に依存している。例えば，インターネットを利用したショッピングモールで支払いをする場合には，そのサイトと提携している決裁機関を利用し，また商品の配送には物流システムへ依頼を転送する。さらに，商品自体の提供は多数の協力企業によるものである。このような，いくつかのサービスを複合してあるサービスを実現するという仕組みは，インターネット上での生涯学習支援においても例外ではない。

生涯学習におけるユーザの特徴としては年齢，居住地，目的，学習履歴，言語，学習時間帯などの“ばらつき”が大きいということが指摘される。これらのばらつきを吸収するために，e-Learning 教材を利用した学習支援が盛んに行われている。しかし，教材提供者が単一でないこと，さらに利用者それぞれの学習履歴のばらつきは，いわゆる単一のカリキュラムの提供を困難にしている。それゆえ，それぞれの学習者の目的や都合に応じて，妥当な順番でカリキュラムを提供し，同時に成績管理，単位貯蓄を行うシステムが必要となる[7]。

このようなシステムの構築には，分散データベースに対する時相論理を用いた問い合わせ処理を行う仕組みが必要である。筆者は，データベースに対して発行する問い合わせ文に時相演算子を持つ論理式を採用し，その解を得るための具体的な手続きとしてモデル検査技法を応用することを提案している[9，5]。

モデル検査（Model Checking)[2]は，ハードウェアやネットワークプロトコルの検証に対して有効であり，近年急速に発達している技術である。モデル検査機

(Model Checker) は，システムの仕様とコードの間の矛盾を見つけ，システムの仕様に矛盾した挙動を反例（counter example）としてプログラマに知らせる。通常，矛盾の証拠（witness）は一つあれば十分であり，モデル検査機が証拠を見つけた場合には，直ちに解析を中止する。さらに，モデル検査においては，数時間から数日間におよぶ検証は十分に実用的であるとされる。しかし，我々の目的は「データベースに対する問い合わせ」が主であり，解の候補は複数得られることが好ましい。また，ユーザへの応答時間は数十秒以内でなければならず，ユーザ自体も一般に複数存在する。これらの理由により，通常のモデル検査よりも極端な検索の効率化を行う必要がある。

筆者は，これらの理由から，モデル検査ではなくモデル空間探査（Model Exploring）という立場を主張し，次の戦略を提案してきた[10, 5]。

1．問い合わせ文の表現力に著しい制限を設け，解析アルゴリズムの計算量をおさえ，
2．複合プロセスを取り扱う場合に典型的に生じる状態数爆発を回避するために，抽象化を利用する。

本論文では，複合プロセスの挙動を効率良く探査するための抽象化と詳細化について議論し，その具体的なメカニズムを例を用いて説明する。

論文の構成は次のようになっている。次節で問題設定と準備を行う。第3節で時相論理を導入し，時相データベースの問い合わせ処理として採用するための根拠を述べる。第4節で具体的な検証のモデルとなるムーア機械と前順序関係を用いた抽象化と詳細化を導入する。さらに，この抽象化が状態数を押える仕組みと，詳細化がデータベースの問い合わせ処理になっていることを，例を用いて説明する。最後に，今後の課題について議論する。

2　準備

ここでは，生涯学習における問題点を形式化し，数学的議論を行うための準備をする。さらに，形式システムとデータベースの問い合わせとの関連を示す。

2.1 問題設定

インターネット上で提供されている多様な e-Learning 教材は，学習者の履修に大きな自由度をもたらす反面，それらの教材のなかから，自身の目的を達成するために学習履歴にあった履修順序を割出すことは初学者には困難である。複数の独立した団体が提供するコース間を組み合わせ，もしくは横断するようなコースの履修順序，いわゆるカリキュラムの提供があるとは限らない。

例えば，表1のようなコースが提供されている場合を考えてみよう。

表1：複数の団体から提供されるコース（例）

授業名	授業コード	提供者
会計コース	Accounting	会計大学院
簿記コース	Bookkeeping	商経学部
表計算ソフトウェアコース	Spread Sheet	政策情報学部

今，コース“Accounting”の修得には“Bookkeeping”の知識が，コース“Bookkeeping”の履修には“Spread Sheet”の知識が必要であるとする。さらに，ユーザ名“Alice”と“Bob”が共にコース“Accounting”の修得を目指しており，Alice は Spread Sheet は既習であるが，Bob はどのコースも学習したことがないとする。

この時の Alice と Bob の履修すべきカリキュラムは，表2のように，容易に割出すことができる。つまり，Alice は(1)Bookkeeping，(2)Accounting の順番で履修するのに対し，Bob は(1)Spread Sheet，(2)Bookkeeping，(3)Accountingの順番に履修する必要がある。

ここでは，理解を容易にするようにコース数3，ユーザ数2の単純な例を示した。現在，コースウェアをお互いに提供し合う試みが世界各地で行われている。これらの場合にはコースが数千から数万になることは珍しくなく，さらに，ユーザ数は，提供されているコースの数十から数百倍になるであろう。このような設定の下では，各々のユーザに個別のカリキュラムを自動的に作成し提供するシステムが，コース提供者にはカリキュラムの整合性を保証したり，個人情報が適切に保たれる“安全な”システムの運用を支援するシステムが必要となる。

表2：Alice と Bob の履修例

	Spread Sheet ⇒	Bookkeeping ⇒	Accounting
Alice		(1)	(2)
Bob	(1)	(2)	(3)

これらの議論から，筆者は生涯学習を支援するためのシステムは，

1．学習者へ，それぞれの目的達成に必要なカリキュラムを指示し，

2．コース提供者へ，コースの安全性に関する矛盾点をレポートする機能をもつ必要があると主張する。

2.2 データベース問い合わせ

関係データベースにおいて，テーブル $\mathcal{D}$ 中の属性 Φ をもつタプルの集合 $\{t \in \mathcal{D} \mid \Phi(t)\}$ を割出すための SQL 問い合わせ文は，次のようになる。

$$\text{select } * \text{ from } \mathcal{D} \text{ where } \Phi$$

ここで，検索中は関係の集合 $\mathcal{D}$ に変化がないと仮定されることが多い(1)。あるタプルに対して更新が行われた場合には，全体のドメインが変化することに注意しよう。すなわち，タプルの更新を $\xrightarrow{updateT}$ というイベント，ドメインを一つの状態とすると，データ更新処理は次のような状態遷移として表すことができる。

$$\mathcal{D} \xrightarrow{updateT} \mathcal{D}'$$

Alice と Bob がコースを履修する例を，状態遷移として解釈するための準備をしよう。

・T_{Bob}：Bob を表すタプル

・$\Phi_{AC}(x)$：xが“Accounting”を修了したことを表す

・SpreadSheet:コース“Spread Sheet”の履修を表す

・BookKeeping:コース“Bookkeeping”の履修を表す

・Accounting:コース“Accounting”の履修を表す

ここで，最初に Bob（T_{Bob}）はコース“Spread Sheet”を履修し，次にコース“Bookkeeping”，最後にコース“Accounting”を履修するというシナリオを考える。

(1) 検索中で関係の集合が変化する場合には汚れた検索（dirty read）とされる。

ここで初期状態を$\mathcal{D}_{\emptyset}$, Spread Sheetコース$\mathcal{S}$のみの履修が終った状態を$\mathcal{D}_{\mathcal{S}}$, つぎにBookKeepingが終った状態を$\mathcal{D}_{\mathcal{S},\mathcal{B}}$, 最終的にAccounting$\mathcal{A}$を習得した状態を$\mathcal{D}_{\mathcal{S},\mathcal{B},\mathcal{A}}$, とする。この時, Bobの履修に伴うデータベースの更新に対応する状態の遷移列πは次のようになるであろう。

$$\pi = \mathcal{D}_{\emptyset} \xrightarrow{SpreadSheet} \mathcal{D}_{\mathcal{S}} \xrightarrow{BookKeeping} \mathcal{D}_{\mathcal{S},\mathcal{B}} \xrightarrow{Accounting} \mathcal{D}_{\mathcal{S},\mathcal{B},\mathcal{A}} \quad (1)$$

T_{Bob}は初期ドメイン$\mathcal{D}_{\emptyset}$ではコース"Accounting"の履修者の集合のメンバではないが, 最終ドメイン$\mathcal{D}_{\mathcal{S},\mathcal{B},\mathcal{A}}$のメンバになっていることに注意する。

$$T_{Bob} \notin \{t \in \mathcal{D}_{\emptyset} \mid \Phi_{AC}(t)\} \text{ かつ } T_{Bob} \in \{t \in \mathcal{D}_{\mathcal{S},\mathcal{B},\mathcal{A}} \mid \Phi_{AC}(t)\} \quad (2)$$

つまり, Bobがコースを"SpreadSheet→BookKeeping→Accounting"の順番に履修していくことを表す状態遷移は, 図1のように, データベースのドメインに対応する状態の更新列として解釈することができる。以下, 後述するモデル検査の用語とデータベースの用語の対応のため, データベースのドメインと状態を, ドメインの遷移列と状態の遷移列を同一視する。さらに, sを状態, πを状態列$\pi = s_0, \cdots, s_n \cdots$, $\pi(i) = s_i$, $\pi^i = s_i, s_{i+1}, s_{i+2} \cdots$とする。

これまでの議論から, "私は会計のコース (Accounting) を履修したい"という問い合わせに対する, "あなたは, その前に表計算ソフトウェアのコース (Spread Sheet) を履修する必要があります"という結果を得るためには, 次の処理を行えばよい。

1．現在の状態から, 指定された, 未来の状態に到達する状態列をもとめ,

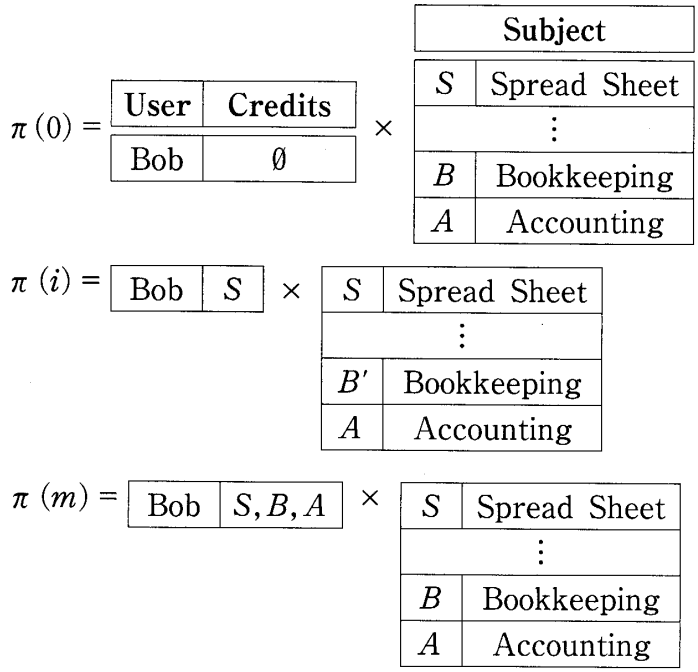
2．その状態列の性質を解析する。

本研究のゴールはこのような状態遷移列を自動的に獲得する方法論を構築することである。筆者は, モデル検査の技術を応用し, 時相論理の体系として様相μ計算を用い, その意味論を利用した状態遷移列を割出す方法については, 参考文献[10,5]で述べている。

2.3 モデル検査

モデル検査とは, システムの有限モデルMと論理的な性質Φが与えられた時に, その性質がモデル中で成立する ($M \models \Phi$) かどうかを体系的な方法で自動的に検査する技術である[Clarke & Emerson 1981]。

図1：Bobの履修とタプルの変化



モデル検査の技術はここ十数年のうちに劇的に進歩し，バスアーキテクチャ，ネットワークプロトコル，PKIなどに関するセキュリティプロトコルの安全性の検証に利用されている。そして，SPIN, SMV, Mochaなど多数の有用なツール（モデル検査機）が無償で提供されている。典型的なモデル検査機は，システムに対して安全性（Safety）や活性（Liveness），公平性（Fairness）に関する診断を行い，システムが検査に合格した場合にはそのことをプログラマに告げる。逆に，システムが検査に不合格であった場合（M, $\pi \not\models \Phi$），システムと仕様との矛盾πを報告する。このπを，仕様に対するシステムの反例（counter example）と呼ぶ。反例は，どのようにしてシステムの仕様が損なわれるのかを告げるため，システムプログラマに対し極めて重要な情報を提供する。

3　時相論理を用いた問い合わせ

本節では時相を用いた問い合わせ言語として時相論理 CTL*(Computational Tree Logic*）の意味論の一部について述べる。本論文の主張を理解するための最低限の意味であり，詳しい CTL や他の時相論理 LTL（Linear Time Logic）μ計算などの文法と意味論については[2]を参照されたい。

以下，sを状態，$\pi = s_0, \cdots, s_n \cdots$, を状態列とするとき，CTL* 式 Φ とその意味は次のように構成される。

$$M, s \models p \Leftrightarrow p \in \mathcal{L}(s)$$

$$M, s \models \neg\Phi \Leftrightarrow M, s \not\models \Phi$$

$$M, s \models \Phi_1 \vee \Phi_2 \Leftrightarrow M, s \models \Phi_1 \text{ or } M, s \models \Phi_2$$

$$M, s \models \Phi_1 \wedge \Phi_2 \Leftrightarrow M, s \models \Phi_1 \text{ and } M, s \models \Phi_2$$

$$M, s \models \mathrm{E}\Phi \Leftrightarrow \exists \pi \text{, s.t. } \pi(0) = s \text{ and } M, \pi \models \Phi$$

$$M, \pi \models \neg\Phi \Leftrightarrow M, \pi \not\models \Phi$$

$$M, \pi \models \Phi_1 \vee \Phi_2 \Leftrightarrow M, \pi \models \Phi_1 \text{ or } M, \pi \models \Phi_2$$

$$M, \pi \models \Phi_1 \wedge \Phi_2 \Leftrightarrow M, \pi \models \Phi_1 \text{ and } M, \pi \models \Phi_2$$

$$M, \pi \models \Phi_1 \mathrm{U} \Phi_2 \Leftrightarrow \exists k \geq 0 \text{ s.t. } M, \pi^k \models \Phi_2 \text{ and } 0 \leq j < k,\ M, \pi^i \models \Phi_1$$

$$M, \pi \models \mathrm{F}\Phi \Leftrightarrow \exists k \geq 0 \text{ s.t. } M, \pi^k \models \Phi$$

$$\mathrm{A}\Phi \equiv \neg\mathrm{E}\neg\Phi \qquad \mathrm{G}\Phi \equiv \neg\mathrm{F}\neg\Phi$$

本論文の範疇では，上記の論理式を以下のように直観的に解釈して構わない。

・AΦ：Φはすべてのパスで成立する

・EΦ：Φを充足するパスが存在する

・GΦ：Φはこのパス上でずっと成立する

・FΦ：Φはこのパス上でいつか成立する

ここで，論理式Φをシステム中でおきてはならないこと，例えばデッドロック（dead-lock）を表す論理式であるとすると，論理式AG¬Φは“これからどのような選択肢をとっても(A)”，“ずっと(G)”“デッドロックが起こらない（¬Φ）こと”を意味する。モデル検査機が システム Mについて$M \models \mathrm{AG}\neg\Phi$ を告げた場合は，そのシステムでは不都合が起きない，つまり安全であることの保証となる。

逆に，モデル検査が失敗した場合には，モデル検査機は反例をプログラマに告げる。この反例πは$M, \pi \not\models \mathrm{AG}\neg\Phi$という意味を持つ。これは，CTL* の意味論を用いると次のような性質を持つことがわかる。

$$\begin{aligned} M, \pi \not\models \mathrm{AG}\neg\Phi &\Leftrightarrow M, \pi \models \neg\mathrm{AG}\neg\Phi \\ &\Leftrightarrow M, \pi \models \mathrm{E}\neg\mathrm{G}\neg\Phi \\ &\Leftrightarrow M, \pi \models \mathrm{EF}\Phi \end{aligned}$$

つまり，モデル検査機が返した反例πは，あるパスπが存在し，そのパス上でいつか（eventually）Φが成り立つことを意味する。先の例で解釈すると，このパスπはシステムM上でデッドロックが起きることの証拠（witness）を意味する。プログラマはこの証拠πを解析することにより，システムがデッドロックの状態に到達するまでの具体的な経路を求めることができる。

4　モデル空間探査

これまで，データベースの更新列と状態遷移列を同一視することにより，データベースの問い合わせに時相論理を導入するアイデアを述べた。時相論理は数学モデルとしての状態遷移システムに対して，その意味が厳密に定義されており，その意味論に基づき具体的なモデル検査機を作ることができる。このため，データベースの更新列を数学的な状態遷移システムで再定義する必要がある。筆者は，これまでにXML（eXtensible Markup Language）を用い，RDF（Resource Description Framework）の枠組を利用して遷移システムの構造を記述し，様相μ計算の自然な（naive）意味づけのためのアルゴリズムを用いたプロトタイプシステムを作成した[8]。ここで，関係データベースの枠組ではなくXMLの記述を選択したのは，Web 上に展開するコースウェアの実現に有利であるという理由からである。

通常の時相論理は，そのモデル空間としてオートマトン，クリプキ構造（Kripke Structure），状態と遷移の双方をラベル付けした遷移システムDLTS（Doubly Labeled Transition System）を採用する。特に，モデル検査では状態の網羅的な検索を行うために，状態数は有限であることを仮定する場合が多い。ここで，モデルの大きさ$|M|$を$|M| = |S| + |\rightarrow|$と近似できるものと仮定する。ただし$|S|$を状態数，$|\rightarrow|$を状態遷移数とする。ここで仕様記述に$\mu$論理式を用いた場合，その論理式中の$\mu, \nu$演算子の入れ子の深さを$d$とすると，モデル検査に必要な計算量は$O(|M|^d)$となるため[2]，現実の検索システムに応用することはできない。

このため，本研究では次の戦略をとる[10, 5]。

1．時相論理式を$d \leq 2$のμ計算式と等価な表現力を持つものに制限し，

2．モデル空間は適当な方法で抽象化を行う。

実際の検索では，交替数，つまり複雑さが2を越える時相論理式を用いることは

まれであり，通常の検索の場合は$d=1$の論理式を組み合わせることで十分である。ここで，論理式が$d=1$の複雑さを持つ場合に必要な計算量は$O(|M|)$となり，効率のよい検索が期待できる。

このため，現実問題としては，モデルの大きさを小さくするための工夫が必要である。例えば，ROBDD（Reduced Ordered Binary Decision Diagram）等を利用し，シンボリックに記述され実際の状態数を減らしたものに対してモデル検査を行う方法（Symbolic Model Checking）や，実際のモデルが必要となるまでモデルの構築を遅らせる（on-the-fly）方法，全体のモデルを構築するのではなく，実際に調べようとする状態の近傍から必要な状態のみ探査する方法（Local Model Checking）などが知られている[2]。

また，具体的なモデルを検査の対象とするのではなく，検証の結果に影響のない部分を捨象し，抽象化されたモデルに対して検索を行う抽象化技法と多数のツールが提案されている[11]。抽象モデルに対する探査は，特に安全性の検証に有効であることが知られている。さらに，本研究のコース探査においても安全性に対する反例が，実際の目的を達成する上での証拠として利用できることはすでに述べた。ここでは，前順序を用いた抽象化と，それに基づくいくつかのAG解析（Assume Guarantee Reasoning）を用いる。AG解析において，当該プロセスは“仮定した環境の下で，ある性質を保証する”ことを主張する。このため，プロセスは例外的な環境に対する言明を避けることができ，抽象化の処理を効率的に行うことができる。反面，AG解析において計算の進行を示す尺度である位相が粗い場合には，当該プロセスと環境プロセスの間での悪しき循環（circularity）が発生することがあり，健全でないルールが混入する可能性がある[4]。このため，本研究ではモデル空間をムーア機械（Moore Machine）に限定する。ムーア機械においてはAG解析における循環を避けることが可能である[3]。

4.1 ムーア機械

ムーア機械とは，単純な制限つきラベルつき遷移システムと解釈してよい。ムーア機械の合成，ムーア機械間の模倣関係，そして模倣関係に基づく前順序は，プロセスの合成やプロセスの仕様と実装の関係を表現することができる。ここで用いる

ムーア機械の定義は最低限にとどめることとする。なお，定義の大部分は文献[3]から得ている。

定義4.1 ムーア機械Mは次のようなタプル$\langle S, s_0, I, O, L, R\rangle$で定義される。

・S - 状態の集合，$s_0 \in$ S：初期状態，

・I - 入力命題の集合，O：出力命題の集合（ただし$I \cap O = \emptyset$）

・L：$S \rightarrow 2^o$ - それぞれの状態にその状態で真になる出力命題の集合を対応付ける関数

・R $\subseteq$ S $\times 2^I \times$ S - 状態遷移関係

さらに，本研究ではムーア機械に Non-blocking であるという条件をつける。Non-blocking とは，"$\forall s \in S$と$\forall i \in I$に対し，$R(s, i, \tilde{s})$ となる$\tilde{s}$が存在する"ことである。つまり，すべての状態において，環境がどのように変化してもプロセスがデッドロックを起こすことはない。Non-blocking は AG 解析の重要な条件であるとともに，分散環境において，環境とプロセスが独立に動作することを表している。

次にムーア機械の合成を定義する。

定義4.2 ムーア機械$P = \langle S^P, s_0^P, I^P, O^P, L^P, R^P\rangle$，$Q = \langle S^Q, s_0^Q, I^Q, O^Q, L^Q, R^Q\rangle$ は$O^P \cap O^Q = \emptyset$の時に合成可能であるといい，その条件の下で，P||Q = $\langle S, s_0, I, O, L, R\rangle$を次のように定義する。

・$S = S^P \times S^Q$

・$s_0 = <s_0^P, s_0^Q>$

・$I = (I^P \cup I^Q) \setminus (O^P \cup O^Q)$

・$O = O^P \cup O^Q$,

・$L(<p, q>) = L^P(p) \cup L^Q(q)$, for all states $<p, q> \in S$

・R(<p,q>,i,<$\tilde{p}$,$\tilde{q}$>$\Leftrightarrow$

$R^P(p, (i \cup L^Q(q)) \cap I^P, \tilde{p})$ and $R^Q(q, (i \cup L^P(p)) \cap I^Q, \tilde{q})$,

for all states $<p,q>, <\tilde{p},\tilde{q}> \in S$ and inputs $i \subseteq I$

最後に模倣関係を定義する。

まず，ムーア機械 P, Q が$O^Q \subseteq O^P$かつ$I^Q \subseteq I^P \cup O^P$と条件付けられているもの

とする。

定義4.3 二項関係 $\Theta_{P\preceq Q}\subseteq S^P\times S^Q$ を Pから Qへの模倣関係（*simulation relation*）とする。ただし，

1．$(s_0^P, s_0^Q)\in\Theta_{P\preceq Q}$

2．$\forall(p,q)\in\Theta_{P\preceq Q}$ に対して $L^P(p)\cap O^Q=L^Q(q)$

3．$\forall(p,q)\in\Theta_{P\preceq Q}$ と $\forall i\subseteq I$ かつ $\tilde{p}\in S^P$ が $R^P(p,i,\tilde{p})$ となっている。このとき，ある状態 $\tilde{q}\in S^Q$ が存在して，$R^Q(q,(i\cup L^P(p))\cap I^Q,\tilde{q})$ かつ $(\tilde{p},\tilde{q})\in\Theta_{P\preceq Q}$ となっている。

Pから Qへの模倣関係 $\Theta_{P\preceq Q}$ が存在する時，“Qは Pを模倣する”，“QはPを抽象化する”，もしくは“Pは Qを詳細化する”といい，$P\preceq Q$と記述する。

ここで，$P\preceq Q$が成り立つならば，Q中には Pのパスに対応するすべてのパスが存在することに注意する。

この模倣関係から得られる前順序 $\preceq$ について，いくつか重要な結果を得ることができる[3,5]。ここでは結果のみを証明抜きで紹介する。

定理4.1 $P\preceq P'\Leftrightarrow T$が Pのトレース木であるとき，その木の P'上への射影$[\mathrm{T}]_{P'}$ は P'のトレース木となる。

つまり，模倣前順序関係にあるプロセスの計算に対応する木はそれぞれ包含関係にあることがわかる。この定理から，次のいくつかの有用な補題が得られる。

補題4.1 $P\preceq P$。

補題4.2 $P||Q\preceq P$。

補題4.3 $P\preceq P'$とするとき $P||Q\preceq P'||Q$。

補題4.4 $P\preceq P'$かつ $Q\preceq Q'$とするとき $P||Q\preceq P'||Q'$。**(AG解析 1)**

補題4.5 $P||Q'\preceq P'$と $P'||Q\preceq Q'$が成り立っているとするとき $P||Q\preceq P'||Q'$。**(AG解析 2)**

4.2 抽象領域上のモデル検査

定理4.1は次のことを述べていた。"$P \leq P'$とするとき，Pのトレースに対応する計算がP'上に存在する。このことは，P'上での性質 AG¬Φ，すなわち，"すべてのパス上でずっと¬Φが成り立つ"という主張が，Pの計算をすべて含むP'上で成立するのであれば，同様の性質はモデルP上でも成立しても良いことを意味する。つまり$P' \models$ AG¬Φの結果が肯定的であれば，その結果を用いて$P \models$ AG¬Φを導き出すというルールは健全（sound）である。

反面，モデル検査機が反例πとともに否定的な応答をした場合，つまり$P', \pi \models \neg$AG Φという判定結果を出力した場合には，この結果を用いて$P, \pi \models$AG¬Φという結論を導き出すことは一般に不健全（unsound）である。意味論の議論から，$P', \pi \models$ EF Φという結論を用いて$P, \pi \models$ EF Φを帰結しているのと同値となってしまうが，モデルPより真に多くの計算を含むモデルP'上で得られた反例πに対応する計算列が，モデルP上に真に存在するかの確認が必要である。

もし，P'上の反例πが$P \leq P'$となるPの反例になっている場合には，πを真の反例であるという。逆に，P'上の反例πに対応する反例がPに存在しない場合にはπをまがいもの（spurious）の反例であるという。この確認は，模倣前順序$P \leq P'$を導出する時に構築した模倣関係$\Theta_{P \leq P'}$を利用して，反例πを詳細化することで可能である。

以下，模倣関係を用いた抽象化と，そこで得られる反例を詳細化していく手続きを例を用いて説明していく。

4.3 抽象化と詳細化

ここでは，ムーア機械の模倣関係に関して成り立つ性質を利用することにより，探査するモデルの状態数を減らすことを試みる。効率の良い抽象化を行うことにより，モデル検査技術をデータベース検索へ適応することを現実的なものにしていく。

4.4 不要なプロセスの捨象による抽象化

まず，今までに用いたコース履修の例を使って，学生を表すエージェントプロセスと，コース教材を表すプロセスを抽象化し，状態数を減らしていく方法について述べる。

日常生活で何らかの事柄について調べる場合に，注目する対象に関連のない事象は無視して考えるであろう。この単純なアイデアは，調べたいプロセスを P，関連のないプロセスを Qとし，補題4.2の $P||Q \leq P$を利用することで実現できる。

コースを利用する学生を Alice，Bob，Charlie，David とし，それぞれに対応するプロセスを A_A，A_B，A_C，A_D とする。これらの学生が同時に授業を履修することは，それぞれに対応するプロセスを合成したプロセス $\mathrm{Agents} \equiv A_A||A_B||A_C||A_D$ を用いて表現することができる。また，Bobにのみ注目し，他の学生を無視したプロセスは単に $\mathrm{Agents}' \equiv A_B$と定義することができる。この時，補題4.2により

$$\mathrm{Agents} \leq \mathrm{Agents}' \tag{3}$$

が成り立つ。

さらに，学生は表計算ソフトウェア（Spread Sheet），簿記（Bookkeeping），会計（Accounting），データベース（Database），ネットワーク（Network）のコースが履修可能であるとし，それぞれのコースに対応するプロセスを C_{SS}，C_{BK}，C_{AC}，C_{DB}，C_{NW} とする。さらに，これらのコースが同時に開講されている状況を $\mathrm{Courses} \equiv C_{SS}||C_{BK}||C_{AC}||C_{DB}||C_{NW}$ を用いて表すものとする。ここでそれぞれのコース Bがコース Aに依存することを $A \rightarrow B$と図示するとしたとき，Courses に現れるコースの依存関係が図2のようになっているものとする。このとき，$\mathrm{Courses}' \equiv C_{SS}||C_{BK}||C_{AC}$とすると，学生のプロセスの場合と同様に，補題4.2により

$$\mathrm{Courses} \leq \mathrm{Courses}' \tag{4}$$

が成り立つ。

さらに，補題4.4を使うと

$$\mathrm{Agents}||\mathrm{Courses} \leq \mathrm{Agents}'||\mathrm{Courses}' \tag{5}$$

が成り立つことがわかる。

いよいよ，モデル空間の探索を行う。

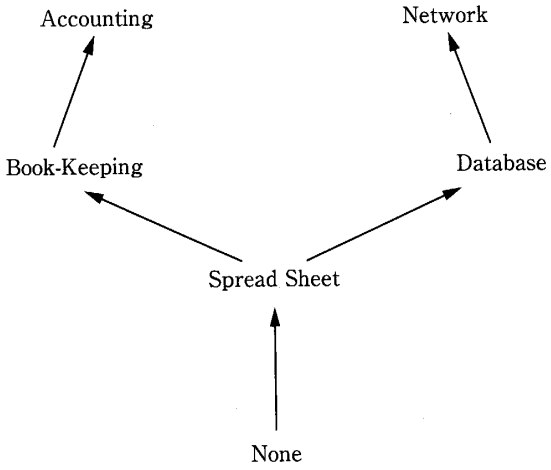
コース会計学を履修したことを論理式 Φ_{AC}で表すものとする。このとき，現在の状況から会計学を履修するまでの履修過程 π は次の論理式を満たす。

$$\mathrm{Agents}||\mathrm{Courses},\ \pi \models \mathbf{EF}\,\Phi_{AC} \tag{6}$$

これは

$$\mathrm{Agents}||\mathrm{Courses},\ \pi \models \mathbf{AG}\neg\Phi_{AC} \tag{7}$$

図2：コースの依存関係



と等価であったことを思い出そう。

今，Bobと他の学生との競合を考える必要がなく，また，コース同士のI/O依存関係（履修順序）がSS→BK，SS→AC，BK→AC，SS→DB，DB→NWのように与えられている場合には，式(5)，および(6)により，式(7)の代わりに

$$\mathrm{Agents}' || \mathrm{Courses}',\ \pi \models \mathbf{AG}\neg\Phi_{AC} \tag{8}$$

の反例πを求めれば十分であることがわかる。すなわち，

$$\mathrm{Agents}' || \mathrm{Courses}',\ \pi \models \mathbf{EF}\,\Phi_{AC} \tag{9}$$

の探査を行えばよい。このとき，それぞれの学生を表現するプロセスの状態数を$|A|$，コースを表すプロセスの状態数を$|C|$とするとき，全体の探査に必要な状態数を$|A|^4 \times |C|^5$から$|A| \times |C|^3$に減らすことができる。

4.5 状態の商集合による抽象化

前の議論について，関心のないプロセスを無視する方法について議論した。一般に現実のプロセスは余分な状態を多数含むため，いくつかの状態を同一視した，いわゆる状態の商集合を作ることにより状態数の少ない抽象化されたプロセスを作ることを考える。

まず，前例の学生Bobに対応する簡単なプロセスを，ムーア機械を使って定義してみる。

ムーア機械：学生 Bob

・I ={CourseID, CertID}

ただし，CourseID は履修しているコースの ID を，CertID は履修終了時に得られる修了証の ID を表す。

・O = {AgentID, Tkt[]}

ただし，AgentID は自分の ID（この場合"Bob"）を，Tkt[]は取得している修了証の状態を示している。

・S = {A_s, A_m, Tkt[]}のすべての解釈の集合

・S_t = (A_s, AgentID = Bob, Tkt[]= ⊥)

・p:$p(s)$ = s|o

$$A_s \xrightarrow{\{CourseID=BK\}} A_m \xrightarrow{\{CertID=BK\}} A_e$$

	AgentID = Bob	AgentID = Bob	AgentID = Bob
・δ ：	Tkt[BK] = Not	Tkt[BK] = Taking	Tkt[BK] = Fin
	Tkt[SS] = Fin	Tkt[SS] = Fin	Tkt[SS] = Fin
	Tkt[-{BK,SS}]= ⊥	Tkt[-{BK,SS}]= ⊥	Tkt[-{BK,SS}]= ⊥

| Tkt[] | を履修可能なコースの数とする時，学生プロセス A の状態数 | A | は次のようになる。

$$|A| = |\{A_s, A_m\}| \times |\{\text{Not, Taking, Fin}\}|^{|\text{Tkt[]}|} \quad (10)$$

同じく，簿記の教材に対応するコース BookKeeping に対応するプロセスをムーア機械を用いて定義しよう。ここで簿記のコース BookKeeping は表計算ソフトウェアのコース SpreadSheet に依存していることに注意する。

ムーア機械:コース（BookKeeping）

・I={AgentID, Tkt[SS]=FinTkt[BK]=Not}

ここで，AgentID はそのコースを現在履修しているエージェントの ID を，Tkt[SS]=Fin はコース表計算（SS）が終了していること，Tkt[BK]=Not は本コース（BK）が未修得であることを示している。

・O={CourseID, StudentID, CertID},

CourseID は自身のコースを表す ID，この場合は BK を，StudentID は履修し

ている学生のID，CertIDは修了証のIDを表している。

・$S=\{C_s, C_m\}\times\{\text{StudentID}\}$ のすべての解釈の集合。

・$S_t=(C_s, \text{StudentID}=\perp, \text{CourseID}=BK)$

$p:p:p(s)=s\mid o$

$$C_s \xrightarrow{\{\text{AID,Tkt[SS]=Fin,Tkt[BK]=Not}\}} C_m \xrightarrow{\{\text{AID}\}} C_s$$

δ :	CourseID=BK	CourseID=BK	CourseID=BK
	StudentID= $\perp$	StudentID=AID	StudentID= $\perp$

上の定義によるコースCの状態数$|C|$は次の式で表すことができる。

$$|C| = |\{C_s, C_m\}| \times |\text{StudentID}| \tag{11}$$

次に，これまでの具体的なプロセスから抽象化されたプロセスを定義する。ここでは，単純に同様の状態を同一視し，また外部のプロセスの挙動に影響を与えない内部変数（StudentID，Ticket status,…）を捨象して得られる，状態の商集合を状態とするプロセスを考える。簡略化された抽象プロセスは図3を参照すること。

A_Bを抽象化して得られるプロセスBobは模倣順序$A_B \leq$ Bobを満たす。この時，プロセスBobは状態を3つしか持たないことに注意する。同様に，$C_{ss} \leq$ SSであるプロセスSS，$C_{BK} \leq$ BKであるプロセスBK，$C_{AC} \leq$ ACであるプロセスACの状態数はそれぞれ2である。

これらの抽象プロセスを複合して得られるプロセスと具体的なプロセスの合成は，補題4.4により以下の模倣関係が成立することがわかる。

$$A_B||C_{ss}||C_{BK}||C_{AC} \leq \text{Bob}||\text{SS}||\text{BK}||\text{AC} \tag{12}$$

ここで，抽象化された合成プロセスは高々$3\times2\times2\times2=24$の状態数に押えられることがわかる。

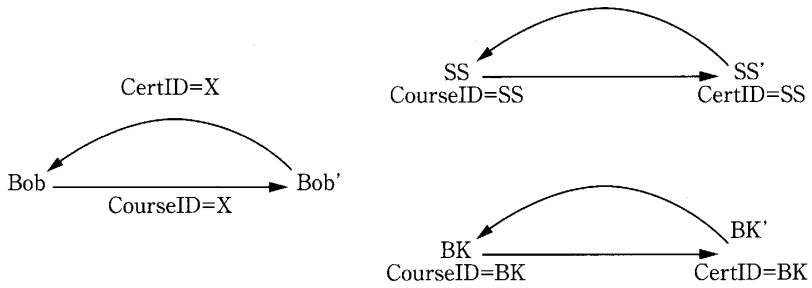
この簡略化された合成プロセスに対してモデル探査を行うことにより，BobがC_{AC}を履修し終るまでの過程πを得ることができる。

$$\text{BK}||\text{SS}||\text{AC}||\text{Bob}, \pi \models \mathbf{EF}\,\Phi_{AC} \tag{13}$$

今後，例を簡単にするために，最終目標を会計コースの修了（Φ_{AC}）ではなく，簿記コースの修了（Φ_{BK}）とする。つまり，

$$\text{BK}||\text{SS}||\text{Bob}, \pi \models \mathbf{EF}\,\Phi_{BK} \tag{14}$$

図3：抽象化されたコース



のトレースを求めていく。

それでは，実際に抽象化されたプロセスを合成してみよう。

まず，SS||BK を先に合成したプロセス(SS||BK)||Bob は図4のようになる。

次に，SS||BKとBobを合成したプロセスは図5のようになる。

図5のプロセス SS||BK||Bobから，次の4種類の状態列を接頭語として持つパス π_0，π_1，π_2，π_3 が得られる。

・π_0=（CID=SS）（CertID=SS）（CID=SS）（CertID=SS）…

・π_1=（CID=SS）（CertID=SS）（CID=BK）（CertID=BK）…

・π_2=（CID=BK）（CertID=BK）（CID=SS）（CertID=SS）…

・π_3=（CID=BK）（CertID=BK）（CID=BK）（CertID=BK）…

これらのパスはいずれも Bob がコース簿記（BK）を修了できていること，つまり論理式 EF Φ_{BK} を充足していることに注意しよう。

ここで，抽象領域で得られたパスは，必ずしも具体領域上の実際のパスになっているとは限らないことを思い出そう。抽象領域で得られたそれぞれのパスは具体領域上で確認していく必要がある。次の例では，π_0，π_1，π_2，π_3 が具体プロセス A_Bの実際の挙動になっているかを確認している。

$$\pi_0=\begin{pmatrix}\text{CID=SS}\\ \text{Tkt[SS]=Not}\end{pmatrix}\begin{pmatrix}\text{CertID=SS}\\ \text{Tkt[SS]=Taking}\end{pmatrix}\begin{pmatrix}\text{CID=SS}\\ \text{Tkt[SS]=Fin}\end{pmatrix}\begin{pmatrix}\text{CertID=SS}\\ \text{Tkt[SS]=Taking}\end{pmatrix}\cdots$$

$$\pi_1=\begin{pmatrix}\text{CID=SS}\\ \text{Tkt[SS]=Not}\\ \text{Tkt[BK]=Not}\end{pmatrix}\begin{pmatrix}\text{CertID=SS}\\ \text{Tkt[SS]Taking}\\ \text{Tkt[BK]=Not}\end{pmatrix}\begin{pmatrix}\text{CID=BK}\\ \text{Tkt[SS]=Fin}\\ \text{Tkt[BK]=Not}\end{pmatrix}\begin{pmatrix}\text{CertID=BK}\\ \text{Tkt[SS]=Fin}\\ \text{Tkt[BK]=Taking}\end{pmatrix}\cdots$$

$$\pi_2=\begin{pmatrix}\text{CID=BK}\\ \text{Tkt[SS]=Not}\\ \text{Tkt[BK]=Not}\end{pmatrix}\begin{pmatrix}\text{CertID=BK}\\ \text{Tkt[SS]=Not}\\ \text{Tkt[BK]=Taking}\end{pmatrix}\begin{pmatrix}\text{CID=SS}\\ \text{Tkt[SS]=Not}\\ \text{Tkt[BK]=Fin}\end{pmatrix}\begin{pmatrix}\text{CertID=SS}\\ \text{Tkt[SS]=Taking}\\ \text{Tkt[BK]=Fin}\end{pmatrix}\cdots$$

図4 ：(SS||BK)||Bob

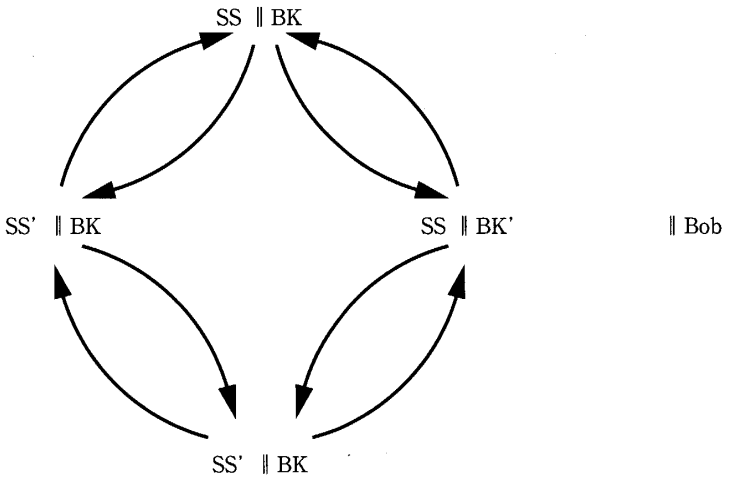
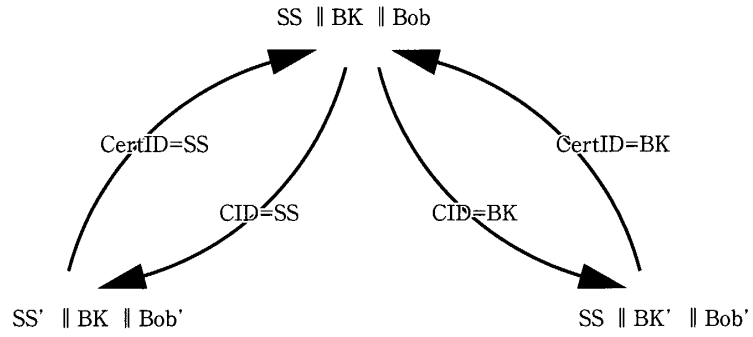


図5 ：SS||BK||Bob



$$\pi_3 = \begin{pmatrix} \text{CID=BK} \\ \text{Tkt[BK]=Not} \end{pmatrix} \begin{pmatrix} \text{CertID=BK} \\ \text{Tkt[BK]=Taking} \end{pmatrix} \begin{pmatrix} \text{CID=BK} \\ \text{Tkt[SS]=Fin} \end{pmatrix} \begin{pmatrix} \text{CertID=BK} \\ \text{Tkt[SS]=Taking} \end{pmatrix} \cdots$$

これらの操作を続けていくことにより最終的に次のことがわかる。

・パスπ_0, π_3は“Tkt[SS(BK)]=Fin”から“Tkt[SS(BK)]=Taking”に遷移する動作を含むため，プロセスA_B上では実現しない。

・パスπ_2は“Tkt[SS]=Not”から“Tkt[BK]=Taking”に遷移する動作を含むため，プロセスC_{BK}上では実現しない。

これらのことから抽象領域上で得られたパスπ_0，π_2，π_3は具体領域上ではまがいものであることがわかり，唯一π_1が正しい証拠であることがわかった。

$$C_{BK}||C_{SS}||A_B, \pi_1 \models \mathbf{EF}\,\Phi_{BK} \tag{15}$$

5　結論

本研究は，インターネット上で生涯学習を支援するための環境の構築をめざしている。インターネット上のサービスはそれぞれが自律分散的に提供されている。また，学習者の学習状況に一定の仮定をおくことができないため，標準的なカリキュラムを提供することが困難である。しかし，初学者が自力で自身の目標を達成するためのカリキュラムを構築することは難しい。

これらの理由から学習者の目標，学習履歴，制約条件をクリアしたカリキュラムを自動的に生成し，提供するシステムが必要となる。さらに，コースの提供者にとっても，学費の納入の確認，矛盾のないカリキュラムの提供，他大学との単位互換等，整合性のあるサービスの提供を保証するシステムが必要である。

これらの問題に答えるためには，次の能力を持つ制約解消系が必要である。

・制約条件を満たしながら，未来の目標を達成するために必要な手続きを，順序立てて例示し，

・構築したシステムが常に安全なサービスを提供する。

筆者は，これらの問い合わせは本質的に時間に関する性質を含んでいると指摘する。しかし，通常，データベースに対する問い合わせは，過去，もしくは現在に対するデータに対してのみ行われるため，この未来を含めた時間の性質をもつ問に対して解を与えることができない。このため，本研究ではモデル検査の技術と，時相論理を問い合わせ言語とするモデル探索により，未来の世界に対しての検索を可能にすることを提案している。しかし，過去のデータは通常一意であるのに対し，未来に対する検索は，可能な限りの到達可能世界を探査する必要がある。これは，モデル検査における計算量と状態数の爆発という現象を引き起こす。

この計算量と状態数の爆発を防ぐために，筆者は次の方策を提案した。

・利用する時相論理の表現力を制限し，計算量を押え，

・抽象化技法の適切な利用により，状態数爆発を防ぐ。

本論文では，特にムーア機械を利用した抽象化技法と，まがいものの解答を防ぐための詳細化技法を，具体例を用いて提案した。ここで使用した抽象化技法は，コースウェアなどコース相互の依存性が少ない場合には，プロセス数を極端に制限できるためにその有効性が期待できる。

今後の課題としては，厳密な状態探査アルゴリズムとその計算量の提示を行う必要がある。また，実際のコースウェアに応用していくことにより，現実の生涯教育における有効性と問題点を確認していきたい。

参考文献

[1] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Form. Methods Syst. Des.*, Vol. 15, No. 1, pp. 7－48, 1999.

[2] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 1999.

[3] Thomas A. Henzinger, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. An assume-guarantee rule for checking simulation. In *Formal Methods in Computer-Aided Design*, pp. 421－432, 1998.

[4] Patrick Maier. A set-theoretic framework for assume-guarantee reasoning. In *ICALP*, pp. 821－834, 2001.

[5] Jun Oyano and Nobuo Saito. Temporal queries over world wide web. In *SAINT Workshops 2005*, pp. 418－421, 2005.

[6] Y.S.Ramakrishna and Scott A.Smolka. Partial-order reduction in the weak modal mu-calculus. In *CONCOR'97:Proceedings of the 8th International Conference on Concurrency Theory*, pp.5－24, London, UK, 1997. Springer-Verlag.

[7] 大矢野潤。生涯学習のための計算モデル。千葉商大紀要　第40巻第4号，P.127，2003。

[8] 大矢野潤。分散環境にある文書からのコース抽出と時制を用いた問い合わせについて。第2回科学技術フォーラム FIT2003 D－016, 2003。

[9] 大矢野順。Applying model checking techniques to temporal queries over World Wide Web.千葉商大紀要　第42巻第2号，P.97，2004。

[10] 大矢野潤。効率の良いコース探査のための抽象化。第3回科学技術フォーラム FIT2004 B－038, 2004。

[11] 高橋孝一　田辺良則。抽象化を用いた検証ツールの調査。産業総合研究所算譜科学グループ研究速報，2003。