3章　知識ベースのコンパイル

3．1　実用エキスパートシステム

3．1．1　実用システム

知識システムは，その知識ベースにある知識の量と質に関して次のように分類することができる（表3．1）[1]．このうち，テキストをコピーしただけのテキスト・システム，学生が自分自身をエキスパートと見立てたスチューデント・システム，実用的応用を考えないトイ・システム，知識が偏っているデモ・システムは，知識の量と偏りから実用システムからはほど遠いシステムである．

原理究明のための学問研究であるプロトタイプシステムには，implementationができ，アイデアが実際に期待されるように動作した段階，典型的なケースについて正常に動作したinformal evaluationの段階，大ケース，汚いケースについても正常に動作するformal evaluationの段階，設計者以外が実際に使用するfield testの段階があり，この延長に実用システムが存在する[1]．

実用システムに関する研究は，見通しのある問題の解決を目指す応用研究であり，実際に正常に機能し動作することが必要で，原則としてはできるというプロトタイプとは一線を画している．理論的に取り扱うことや，数学モデル化や制御フロー記述が可能なシステムでは，アルゴリズムが正しければ，プロトタイプを実用システムに拡張することは簡単であるが，知識システムが対象としている，科学的，理論的に取り扱うことが困難な問題

表3．1　知識システムのレベルによる分類

① テキスト・システム
② スチューデント・システム
③ トイ・システム
④ デモ・システム
⑤ プロトタイプ・システム
⑥ 実用システム

領域，数学モデル化や制御フロー記述が困難な問題領域，複雑，あいまい，不完全な開いた世界（ill structured open world）を扱う問題領域においては，両者の間には大きな溝があり，プロトタイプを実用システムに発展させるためには，解決しなければならない問題が数多く課題として残されている．

実用システムにおける問題点の一つは量の問題であり，もう一つの問題点は質の問題である．量の問題とは実用システムの規模の問題である．専門家の代役として非専門家の意志決定を支援することを可能とするためには，また専門家の知識を補うことを可能とするためには大きな知識ベースを必要とする．実用化されているシステムの大半は３００ルール以上の規模を持っており，１０００ルール以上のものも割合としては少なくない[2]（表３．２）．質の問題は機能の問題であり，推論に関する高次人工知能機能の問題や，ユーザインタフェースの問題がある．

表３．２　エキスパートシステムの規模[2]

開発段階 / ルール数	実用化したシステム	プロトタイプシステム		調査中	計
		現　在	最終ルール（見込み）		
1．50未満	1件	7件	2件	0件	10件
2．50～ 99	1件	4件	0	1件	6件
3．100～299	0	10件	7件	11件	28件
4．300～499	2件	5件	0	2件	9件
5．500～999	1件	4件	5件	6件	16件
6．1 000以上	3件	3件	9件	8件	23件
回答件数	8件	33件	23件	28件	92件
無　回　答	2件	—	10件	67件	—
計	10件	33件	33件	95件	171件
平均ルール数	646	460	1 420	1 171	1 197

3．1．2　規模の拡大に関する問題点

知識システムの規模が大きくなるに従い，知識の収集・整理方法，知識の管理方法，実行効率の向上に関する問題が出てくる．知識の収集・整理方法に関しては，知識の系統的収集方法，および知識を系統的に整理し，構造化，モジュール化して網羅的に安定に記述する方法を確立する問題がある．知識の整理に関する問題は，また対話量を増やし過ぎず，使い勝手を良くするための適正な知識の規模を設定する問題でもある．

知識システムの開発は，小規模の知識量から始めて，知識の追加，更新を行なってシステムを成長させながら行なうのが普通であるし，開発されたシステムも各自のノウハウを自由に入力できるよう自己成長能力を持ったシステムである必要があり，規模の拡大に伴う知識の追加，更新等のシステムの成長に関する問題もある．

大きな規模の知識システムにおいては，空間の拡大に伴う速度の低下とともに，競合の解消，無矛盾性の管理等のための速度の低下が問題となる．大規模データを高速処理するために，実行効率を向上させるためには，無矛盾性管理の高速技法の開発，推論の高速実行技法の開発，大規模データの処理，高速処理を可能とする知識表現の開発，メタ知識を含めた高速化を可能とする知識の抽出が問題となる．

3．1．3　高次人工知能機能と不完全な知識

現在，完全な知識を対象とした従来の演繹的推論を超える，次世代知識システムの基礎を提供すると考えられる，新しい高度な知識メカニズムとして，高次人工知能機能を実現し，利用することがＡＩ研究の大きな課題となっている[3]（図３．１）．高次人工知能機能としては，ある概念と他の概念との結合関係に関する連想（association），人間が日常的によく行なう推論の形式であり，学習と深い関わりがある類推（analogy），事実から規則を見つけだす帰納推論（induction），既に得られた概念から新しい概念を生成する発想（abduction）や，常識（common sense），直感（intuition）等がある．

知っていることには対応できるが，少しでも知らないとお手上げになるという現在の狭い知識ベースの幅を拡張するために，われわれは「完全な知識に加えて不完全な知識も含

て知識ベースを構成し，これを操作することにより，高次人工知能機能を実現する」という視点から，次世代知識ベースの構成を検討し，研究を進めている[4][5][6]．高次人工知能機能を実現する枠組みを図３．２に示す．

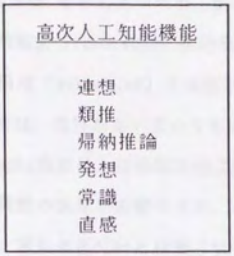


図３．１　高次人工知能機能の例

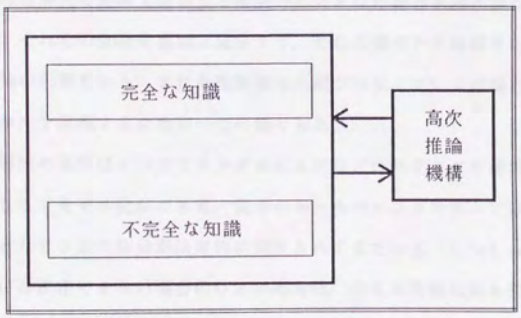


図３．２　高次人工知能機能を実現する枠組み

不完全な知識とは，常には真であるとは限らない知識で，不確実な知識，不備な知識，不明瞭な知識等である．不完全な知識を対象とする事により，知識の範囲が広がる．反面推論が非単調となるため，不完全な知識を操作・管理する必要があり，推論が複雑化する．この意味で，高速化技術が重要となる．

確率的な知識である不確実な知識に対しては，ＭＹＣＹＮ以来，確信度という考え方が使用されてきている．Bayes 確率を用いたり，Dempster－Shafer理論を用いたりしている[7]が，Bayes確率では無知量（ignorance）を効果的に扱えない問題や，信用の欠如（lack of belief）と不信用（disbelief）とを区別できない問題がある．

不明瞭な知識に対しては，境界があいまいなものの取り扱いが可能なFuzzy論理が用いられることが多い[8]．Fuzzy論理は，証拠間の独立性が疑がわしい場合に効果的であるが，合理的なメンバシップ関数の決定が必要であり，この決定が困難な場合が少なくない．

不備な知識としては，常には正しいとは限らない知識や欠落部分を含む知識があり，仮説推論は，この不備な知識に対処するための枠組みである[4]-[6]，[9]-[11]．

３．１．４　仮説推論

仮説推論は問題解決の過程で競合する知識や不完全な知識等を取り扱うときに必要となるものであり，それらの知識を仮説と見立てて，それに基づいた処理を行なうことにより進めてゆく推論の形態をいう．また仮説推論は人間が行なっている推論形態でもあり，高次推論機能の多くを実現するための一つの鍵でもある．

一般に問題解決の過程はインクリメンタルにＡＮＤ／ＯＲグラフを構築しながら同時に探索しているとしてモデル化ができる．従来のルールベースシステムでは強力なヒューリスティックスとしてＯＲの部分が決定的に選択されてきている．しかしながら知識が競合しており一意には決定できない場合のＯＲの処理は，全ての可能な解を探し出すことや，ある枝を仮に選んでみて推論を進め，もしうまくゆかない場合には，バックトラックして別の解を探すことが必要となる[12]．

仮説には，ユーザが与えるものと，システムが機能効率面で仮説として扱う，非決定性を有する選言的知識があり，これらの仮説を用いる推論のアーキテクチャとしては，エキスパートシステム，Truth Maintenance System，論理的推論，非単調推論等がある．

エキスパートシステムにおいては，ヒューリスティック・ルールを基にして中間仮説や最終結果を導く方法が採られており，この方法においてはDempster-Shafer理論や，Fuzzy理論により，仮説間に確からしさの数値を導入することにより重み付けをしている．しかしながら，確信度等の確からしさの数値の設定が困難なことと，それらの組み合わせ方が困難なことに問題がある．

Truth Maintenance Systemの代表的なものとしては，与えられた仮説を基にして，推論中におけるデータベースの一貫性を管理するDoyleのＴＭＳ（Truth Maintenance System）がある[13]が，このＴＭＳでは仮説の与え方が議論されていない．また，仮説の組み合わせに注目し，仮説間に数値による重み付けを行わないＡＴＭＳ（Assumption-based Truth Maintenance System）[14]や，仮説を発想するＣＭＳ（Clause Management System）[15]も，Truth Maintenance Systemのアーキテクチャに含まれる．DoyleのＴＭＳや，ＡＴＭＳにおいては，組み合わせ爆発の問題が生ずる．

論理的推論による仮説選定においては，与えられた既知の知識集合が無矛盾であるという前提の基で，無矛盾性のチェックを論理的に行う．複数の可能性の中から，観測や使用に適合する適当な仮説を見つける場合で，仮説はあらかじめ見えており，DoyleのＴＭＳや，ＡＴＭＳがこのアーキテクチャに含まれる．可能な仮説集合をあらかじめ与える必要があり，この点が問題となる．

非単調推論においては，新しい公理が加わることにより，成立しなくなる可能性がある定理や，ある定理を証明するために必要になる公理が仮説として取り扱われる．この場合仮説はあらかじめ見えない．ＣＭＳはこの場合に該当する．Non-monotonic Logic Ⅰ[16]，Default Logic[17]，Circumscription[18]等の手法が提案されており，これらにおいては信念の修正機構としてTruth Maintenance Systemが重要な位置を占めている．仮説推論においては新事実の追加・削除時に，データベースに及ぼす影響が非単調であり，このアーキテクチャに含まれる．

仮説推論は，不完全な知識を仮説として取り扱う，非単調推論を扱う最も簡単な形式であり，制約問題解決機構上の，および設計における高レベル問題表現の枠組みであり，実用的問題に比較的直接適用することが可能である．

3．1．5　高速化の必要性

知識ベースの規模の拡大に伴い，基本的な問題となる高速化の問題は，仮説推論のようにTruth Maintenance Systemの重要性が高い非単調推論においては，無矛盾性を管理するために重要であるし，また最悪の場合ＮＰとかＮＰ完全のクラスの問題となり，解空間の爆発が問題となるこの種の推論においては非常に重要となる．

仮説推論の高速化を実現するためには，別解の探索におけるバックトラックを避けること，無矛盾性の検査のコストを小さくすること，最悪の場合はＮＰ完全になるとしても，平均値として多項式オーダーの計算量となる手法を見い出すことが重要である．

高速化を可能とする技法のうち，ヒューリスティックスを利用する高速化技法としては，ネットワーク理論におけるDijkstra法のように最良優先探索を行うＡ＊の技法[19]や，タスクを書き込み実行順序を調整し冗長なパターンマッチングを少なくするアジェンダ（agenda）の技法[20]がある．問題領域の知識から独立した推論の高速化技法としては，ＲＥＴＥアルゴリズム等のように，ノードの共有や矛盾の早期発見による探索木の枝刈りにより高速化を図るネットワークを利用する技法[21][22]，ルールをコンパイルする技法[23][24][25]，重み付け等により注視点を制御する技法[26]等が現在開発研究されている．

3．1．6　知識ベースのコンパイル

知識の規模が大きくなると，知識の単なる足し算ではすまなくなり，限定された分野の問題では潜在していたものが顕在してくるため，知識の管理が必要となる．この意味で，高速化の実現のためには，知識の管理を高速化する必要がある．

知識をコンパイルすることにより，他の知識を包含する冗長な公理，定理や，重複している公理，定理を取り除くことができるため，知識の管理が簡単になる．また，深い知識をコンパイルして浅い知識を生成するなどによって，新しい定理の証明のための効率の良い推論を行うことも可能となる[27][28][29]．公理，定理等のコンパイルとともに，仮説のコンパイルも可能であり，仮説の圧縮を図ることが可能である．

反面，定理の証明の簡単化を図るためには，公理集合のコンパイルにより，全ての定理

をあらかじめ生成する必要があり，コンパイルにより知識集合が大きくなるという問題がある．また，定理の増加，仮説の取り込みにより，incrementalなコンパイルや，decrementalなコンパイルが必要となる場合があり，この処理が問題となる．

3．2　高速化の枠組

3．2．1　真理管理機構

不完全な知識を含む知識ベースを操作する場合，常には正しいとは限らない知識については，無矛盾性を確認する必要があり，また欠落部分を含む知識については，欠落した知識を発想する必要がある．無矛盾性を確認する計算量や，発想のために新しい組み合わせを生成する計算量は，問題空間の拡大に伴い指数的に増大することから，これらの機能を実現するためには，高速化技術の開発，研究が必要となる．

論理的推論の高速化技術としては，ＴＭＳ[13]，ＡＴＭＳ[14]，ＣＭＳ[15]，FreuderやMcWorthが研究を発表しているＣＳＰ（Constraint Satisfaction Problem）[30][31]等がある．

ＴＭＳは一度に一つの解しか求めることができず，仮設が文脈に依存するため，このデータ間のdependencyを記録する部分が大きく，大きい記憶領域を必要とする，またback trackを行ないながら矛盾を回避し，知識ベースの管理を行なうため，以前に利用したdataを繰り返し呼び出すなど，高速化の点からは問題のあるシステムであった．

ＡＴＭＳはＴＭＳに比較してback Trackを行なわない点が高速化の点では優れている．しかしながら，全解探索を行うシステムであり，仮説世界の爆発の可能性がある．ＡＴＭＳもＣＳＰも矛盾する空間の探索を避けることで推論の高速化をはかっているが，ＡＴＭＳとＣＳＰを比較した場合，ＡＴＭＳは命題で記述された問題を解くことが可能であり，ＣＳＰよりも一般的である．ＣＭＳにおいてシステムに送り込む節をホーン節に限定し，質問節をユニット節に限定したものがＡＴＭＳであり，ＣＭＳは論理基盤上でＡＴＭＳを一般化，拡張したものとして位置づけられる．ＣＭＳは，知識ベースのコンパイルにより，推論の高速化を図るとともに，無矛盾性の管理および欠落した知識の発想を可能とする枠組でもある．無矛盾性の管理および欠落した知識の発想を可能とする，Reiter他の示したＣＭＳという枠組は，仮説推論に有効であると思われるため，われわれは基本的なＣＭＳを実現し，有効性と実際の知識システムに応用するにあたっての問題点を検討した[6]．

ＣＭＳは，知識ベースのコンパイルにより，導出を行なうことなく，Minimal Support節をPrime Implicant節との包含関係から直接高速生成することを可能とする枠組ではあるが，

これを利用して実用的な仮説推論システムを実現するためには，Prime Implicant節をあらかじめ導出するという，知識ベースのコンパイルが重要となる．しかしながら，Reiter他はPrime Implicant節を用いた推論の高速化の可能性を示しているものの，そのために必要となるコンパイル技術に関しては検討を十分に行なっていなかった．そこで，仮説推論にReiter他の示したこの枠組を応用するための，知識ベースのコンパイル法の開発，およびコンパイルした知識を利用した，Minimal Support節の高速生成法の実現を検討することとする．

3．2．2　ＣＭＳの概要

ＣＭＳは一般の命題論理節を対象とし，公理節集合（Σ）と，問題解決器（Reasoner）から送り込まれた質問節（Ｃ）とから，Minimal Support節（Ｓ）を生成することにより，質問節の論理的帰結性や，無矛盾での充足可能性を検討する枠組である（図３．３）．

論理的帰結性を問う質問節の否定に対して，空節ではないMinimal Support節が生成された場合（$\Sigma \vdash \sim C \rightarrow S$，$S \neq \{\}$）が，質問節が公理節集合から論理的に帰結されない場合であり，Minimal Support節の否定は，証明するために欠けている知識に相当する（$\Sigma \vdash \sim S \rightarrow C$）．即ち欠けている知識は連言から成り立つ論理式で表現される．また，Minimal Support節が空節である場合（$S = \{\}$）は，ＣＭＳ中の公理節集合によって質問節を論理的に帰結することが可能で，欠けている知識が無い場合に相当する．

充足可能性を検討する場合，充足検討節に対して，Minimal Support節として空節が生成されない場合が，無矛盾で充足可能である場合に相当する．

発想とは，類推，普遍化，極限化，システム化等によって，既存知識ベースの問題解決能力とは無関係に，知識ベースの問題解決能力の外側にある問題の解決をめざす機能であり[32]，この意味では今回のＣＭＳによって生成される欠落している知識は，既存知識ベースの枠内にあり，厳密には発想ということはできない．しかしながら，この枠組は，常には正しいとは限らない知識や，欠落部分を含む知識等の，不完全な知識を含む知識ベースを操作することが可能であり，これらの操作を必要とする，仮説推論システムに適していると思われる．ここでMinimal Support節は，図３．４のように定義される．

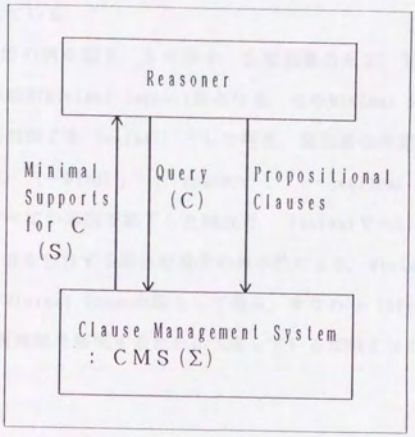


図3．3　CMSの枠組み

① $\Sigma \vdash S \cup C$

② $\Sigma \nvdash S$

③ Sの部分集合S'について，
$\Sigma \nvdash S'$，$\Sigma \nvdash S' \cup C$

図3．4　Minimal Support節の定義

図3．4において，①はMinimal Support節Sと質問節Cの和をΣから証明できることを意味しており，（Σ⊢～C→S）と変形できることから，質問節の否定を用いてMinimal Support節を生成することが可能である．②はSがΣから証明できないことを，そして③はSの最小性を意味している．

Minimal Support節の例を図3．5に示す．公理説集合をΣ，質問節をCとすると，{～bird} 他4つの節がMinimal Support節となる．このMinimal Support節の導出方法を図3．6に示す．質問節Cを {animal} とした場合，質問節の否定 {～animal} を支持節として導出が行われ，{～bird}，{～sparrow}，{～swallow}，{～taro} 等が次々に導出節となる．すべての導出が終了した時点で，{animal V ～bird} が {～bird} を包含するように，他の節を包含する節は定義③の最小性により，Minimal Support節の候補から外れ，残った節がMinimal Support節として残る．すなわち {bird} 等のこのMinimal Support節の否定が質問節を証明するために欠落している知識となる．

3．2．3　知識ベースのコンパイルによる高速化

Minimal Support節を生成する方法としては，節をそのまま登録して，質問に応じてMinimal Support節を計算するインタプリタ形式によるアプローチと，あらかじめPrime Implicant節を導出しておくコンパイル形式によるアプローチとがある．Minimal Support節は必ずしも空節でないため，空節が生成されない場合には，基本的には全部のMinimal Support節を導出する必要がある．文献[6]においては，インタプリタ形式を採用したが，この形式においては，公理節集合中の節数およびその節中のリテラル数が多い場合に，導出節が急激に多くなり，Minimal Support節を生成するための計算量が急激に大きくなる問題がある．

コンパイル形式は，公理節集合からあらかじめ導出したPrime Implicant節と，質問節のとの間で包含関係を検討し，集合計算により直接Minimal Support節を高速に生成する方法である．ここで，Prime Implicant節（II）は，図3．7のように定義される．

$\Sigma = \{\{\sim taro \vee swallow\},$
$\{\sim swallow \vee bird\},$
$\{\sim sparrow \vee bird\},$
$\{\sim bird \vee animal\}\}$ の時,

$C = \{animal\}$ の Minimal Support節

$\{\sim animal\}, \{\sim bird\}, \{\sim swallow\},$

$\{\sim sparrow\}, \{\sim taro\}$

図3．5　Minimal Support節の例

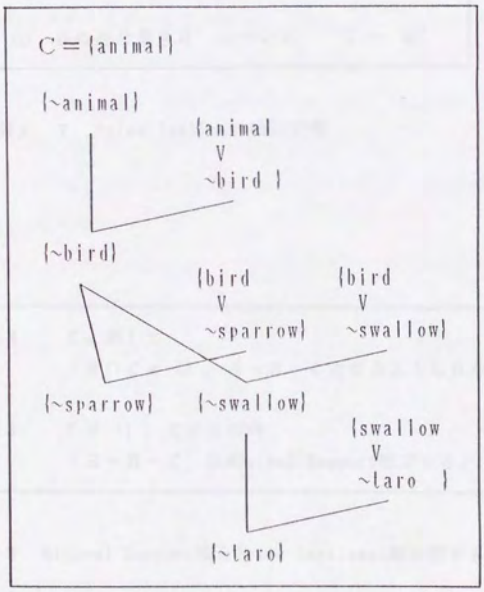


図3．6　Minimal Support節の導出例

Reiter他は，質問節（C）がPrime Implicant節（Π）に包含される場合，Minimal Support節（S）を簡単に生成できることを指摘している．定理2，3はSとΠとに関する定理である（図3．8）．Π－CによりSを求めるこの方法は，質問節がユニット節である場合，Minimal Support節の生成を高速化する有効な方法である．（ここでΣがホーン節のみで構成され，質問節Cがユニット節である場合が，ATMSに相当する．）CMSでは質問は節に限定されることから，Minimal Support節をprime Implicant節を利用して得ることができる．図3．9に，定理3によるMinimal Support節の生成例を示す．

① $\Sigma \vdash \Pi$
② Πの部分集合Π'について，　$\Sigma \nvdash \Pi'$

図3．7　Prime Implicant節の定義

定理2	Cに対して 「$\Pi \cap C \neq \{\}$，$S = \Pi - C$となるようなΠがある」
定理3	$C \neq \{\}$，$C \subseteq \Pi$の時 「$S = \Pi - C$　はMinimal Support節である」

図3．8　Minimal Support節とPrime Implicant節に関する定理

定理２の逆は必ずしも真ではなく，質問節がユニット節でない場合には，必ずしも単純な集合演算によりMinimal Support節が生成できるとは限らない．この場合，Minimal Support節の候補とprime Implicant節との間で包含関係を検討し，Minimal Support節の生成を行なう必要がある．この方法は質問節を構成するリテラルについて，その補リテラルを用いて導出を行なうことと同意であり，次のように拡張することが可能である．$C \not\subseteq \Pi$である場合には，$C' \subseteq \Pi$であるCの部分集合C'を求め，このC'に対して前述の方法を適用することにより，高速化を図ることが可能である．図３．１０に定理３によってはMinimal Support節を生成することができないため，C'を用いる必要がある例を示す．このようにPrime Implicant節があらかじめ導出されている場合，質問節の否定を用いて導出を行なうことなく，集合演算によりMinimal Support節の生成の高速化を図ることが可能である．

この方法は無矛盾での充足可能性を検討する場合にも，応用することが可能である．充足可能性を検討する節の否定が節である場合，すなわちユニット節である場合，このユニット節を用いて集合演算を行なうことにより，導出を行なうことなくPrime Implicant節を用いて，Minimal Support節を生成することが可能であり，無矛盾性の検討を高速で行なうことが可能となる．充足検討節を構成するリテラルについて，導出を行なったことと同じ意味であり，この節を追加充足する場合には，生成されたMinimal Support節が，新しいPrime Implicant節となる．

そこで，Minimal Support節の生成の高速化を図るための方法として，効果の大きいと思われる，コンパイル形式を対象として検討することにした．コンパイル形式で高速化を図るためには，Prime Implicant節の導出の高速化を図ることが必要であり，重要である．

$\Pi = \{\{p\}, \{q \vee s\}, \{q \vee r \vee \sim t\}\}$,
$C = \{q\}$ の時,

① $\sim C = \{\sim q\}$
② $\Pi - C = \{q \vee s\} - \{q\} = \{s\}$,
$\Pi - C = \{q \vee r \vee \sim t\} - \{q\} = \{r \vee \sim t\}$

$S = \{\{\sim q\}, \{s\}, \{r \vee \sim t\}\}$

図3．9　定理によるMinimal Support節生成例
（$C \neq \{\}$, $C \subseteq \Pi$の場合）

$\Pi = \{\{p\}, \{q \vee s\}, \{q \vee r \vee \sim t\}\}$,
$C = \{s \vee r\}$ の時,

① $\sim C = \{\sim s \wedge \sim r\}$
② $C' = \{\{s\}, \{r\}\}$　（但し　$C' \subseteq \Pi$）
$\Pi - C' = \{q \vee s\} - \{s\} = \{q\}$
$\Pi - C' = \{q \vee r \vee \sim t\} - \{r\} = \{q \vee \sim t\}$

$S = \{\{\sim s\}, \{\sim r\}, \{q\}\}$

図3．10　定理がそのまま適用できない場合
（$C \not\subseteq \Pi$の場合）

3．3　高速化アルゴリズム

3．3．1　導出戦略

節集合において，導出の高速化を図るためには，通常，表3．3に挙げるような意味導出，支持集合導出，線形導出等の戦略が用いられる．

順序付け戦略には，重み付け（weighting）や単位優先戦略（unit preference strategy）があり，制限戦略には，超導出（hyper-resolution）や負の超導出（negative hyper-resolution）のように，T値を持つものとF値を持つものに分類する意味導出（semantic resolution）があり，横型優先戦略には，入力導出（input resolution）や順序付け線形導出（OL導出：order linear resolution）のような線形導出（linear resolution）がある．これ以外には図3．6で用いた支持集合戦略（set-of-support resolution），冗長な知識や重複している節を前向き包摂（forward subsumption）や，後向き包摂（backward subsumption）により捨てる刈り込み戦略，デモジュレーション（demoduration）のように単位等価節によって書換を行う正準化戦略等がある[33][34][35].

表3．3　導出戦略

導出戦略
①　順序付け戦略
②　制限戦略
③　支持集合戦略
④　横型優先戦略
⑤　刈り込み戦略
⑥　正準化戦略

3．3．2　高速化アルゴリズム

（1）高速化アルゴリズムの概要

導出の高速化のために通常用いられる，意味導出，支持集合導出，線形導出等の戦略は，空節の導出を目的としているために，今回のように Minimal Support節が必ずしも空節でない場合には，高速化の戦略としてそのまま利用することはできない．幅優先戦略等による全解探索は，効率が悪く，特に公理節が多くなった場合には実用的ではない．

節中のリテラルは，双形リテラル（biform literal）と単形リテラル（monoform literal）に分類することが可能である．双形リテラルとは，節集合中に正リテラルを含む節およびその負リテラルを含む節が存在する，すなわち相補リテラル（complementary pair）が存在するリテラルをいう．単形リテラルとは，補リテラル（complementary literal）の存在しない，すなわち相補リテラルが存在しないリテラルをいう．

そこで，公理節集合中のリテラルのうち，双形リテラルに注目し，この双形リテラルを利用した Prime Implicant節導出の高速化の方法（順序づけ節コンセンサス法：Ordered Clause Consensus Method：ＯＣＣ法）を開発した．アルゴリズムを図３．１１に示す．

① ユニット節が存在する場合，ユニット導出を行なう．
② すべての双形リテラルについて，正リテラルを含む節およびその補リテラルを含む節の間の組み合わせ数を算出する．
③ 導出に用いられていない双形リテラルのうち，②で計算した組み合わせ数の最も少ないリテラルを選ぶ．
④ 選択されたリテラルについて導出を行なう．
⑤ 他の節を包含する節を削除する．
⑥ すべての双形リテラルについて導出が終了していない場合，③に戻る．

図３．１１　順序づけ節コンセンサス法（ＯＣＣ法）

導出順序の決定方法を図3．１２に例を用いて示す．図3．１２において，Σは公理節集合であり，ＣＭＳの提案があった文献[15]において用いられた節集合である．この例においては，ユニット節が無いため，①の手順は実行されない．次いで手順②により，組み合わせ数を計算している．Ｌの数とはリテラルＬの正リテラルの数を，～Ｌとは負リテラルの数を示している．この例においては組み合わせ数の少ない順は，｛ｓ｝，｛ｔ｝，｛ｒ｝，｛ｐ｝，｛ｑ｝の順になる．

同じ公理節集合を用いた，アルゴリズムの③～⑥の適用例を図3．１３に示す．手順に③により，双形リテラルの内，組み合わせ数のもっとも少ないリテラル｛ｒ｝が選択される．手順に④により，このｒを用いて導出が行なわれ，｛ｐＶｑ｝，｛ｐＶｑＶ～ｔ｝の二つの節が導出される．手順⑤により，この時点で包含関係を検討し冗長となる節の消去を行なう．この例では一本下線で示した二つの節が，｛ｐＶｑ｝を包含しており消去される．この例にもあるように消去の対象となるのは公理節だけではなく導出節も対象となる．同様に｛ｐ｝，｛ｑ｝についても導出を行ない，Prime Implicant節として，｛ｐ｝，｛ｑＶｓ｝，｛ｑＶｒＶ～ｔ｝の節が導出され終了する．図中”Ｘ”で示した箇所は排他的関係にあるため，導出ができなかったところである．

この方法では，公理節集合に含まれる双形リテラルについて，それぞれただ１回導出を行なえばよいため，導出の終了が明らかである．また他の節を包含する節については毎回消去を行ない，対象としている節の減少を図るとともに，一般の導出に先だって組み合わせ数を計算し，導出を行なう双形リテラルの順序づけを行なうことにより，導出回数の減少を図り，導出の高速化を図っている．組み合わせ数の少ない順に導出を行なったこの例では，導出節の生成回数は７回，排他的関係にあるため生成を中止した回数は１回であった．同じ例について，逆に組み合わせ数の多い順に導出を行なうと，生成等の回数は１４回となり，約２倍に回数が増える．

（２）順序付けの効果

あるリテラルとその否定のリテラルの数，および各リテラルに関する，有りうる最大の導出回数を，双補リテラルの可能な最大の組み合わせ数により定義する（図3．１４）．図3．１４において，$m_1 = a_1 \cdot b_1$ であるが，m_2 以降についてはこのように単純ではない．L_n以前に導出された節の数をｎとし，一つの節の中には同一のリテラルが複数存在しないとすると，正負のリテラルの数が等しくなる，$\{(a_n + b_n + n)/2\}^2$ の値

$\Sigma = \{p \vee q \vee r,\ p \vee \sim q,\ p \vee \sim r,\ \sim p \vee q \vee s,\ q \vee r \vee \sim t\}$

L	Lの数 (a)	~Lの数 (b)	a*b
p	3	1	3
q	3	1	3
r	2	1	2
s	1	0	0
t	0	1	0

s, t, r, p, qの順に導出を行なう

図3．１２　順序づけの方法

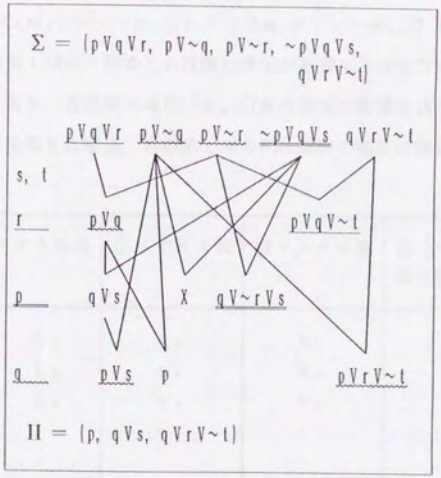


図3．１３　順序づけ節コンセンサス法の適用例

が最大組み合わせ数となり，最大導出回数となる．

L_1の段階での導出節のすべてにリテラルL_2が入っている場合が，L_2に関する組み合わせ数が最大になるときであり，m_2は次のように表現することができる．

$$m_2 \fallingdotseq \{(a_2+b_2+m_1)/2\}^2$$

同様にm_3以降についても，当初のリテラル数と，新たに導出された節の数により，最大組み合わせ数，すなわち最大導出回数を表現することができる．

$$m_n \fallingdotseq \{(a_n+b_n+m_1+m_2+\cdots+m_{n-1})/2\}^2$$

最初の設定により$a_{n+1}\cdot b_{n+1} \geqq a_n\cdot b_n$であり，$a\cdot b$の可能性としての最大値は$\{(a+b)/2\}^2$であることから，$a\cdot b=\{(a+b)/2\}^2$，とおき，

$$m_n' \fallingdotseq \{(a_{n+1}+b_{n+1}+m_1+m_2+\cdots+m_{n-1})/2\}^2$$

とすると，$m_n' \geqq m_n$となり，組み合わせ数の小さい方が，各段階の導出回数は小さくなる．

また，

$$m_n \fallingdotseq \{(a_n+b_n)/2\}^2 + (m_1+\cdots+m_{n-1})\cdot\{(m_1+\cdots+m_{n-1})+2\cdot(a_2+b_2)\}/4$$

であり，右辺第1項は，他のLの段階の導出が影響を及ぼさない場合の，最大の組み合わせ数を表しており，右辺第2項は，L_n以前の導出の影響を表している．全体の合計としての導出回数を考えた場合，右辺第1項の和は導出の順序に関係が無く一定である．

リテラル名	正リテラル数	負リテラル数	各リテラルに関する最大導出回数
L_1	a_1	b_1	m_1
L_2	a_2	b_2	m_2
L_3	a_3	b_3	m_3
・	・	・	・
・	・	・	・
L_n	a_n	b_n	m_n

図3．14　組み合わせ数の計算方法

これに対し

$$\Sigma（右辺第2項）=（n-1）\cdot m_1{}^2/4+\cdots+m_{n-1}{}^2/4+\cdots$$

であり，この式において影響の大きい2次の項のみを考えると，早い段階の導出回数の影響が大きく，組み合わせ数の小さい方を先に導出を行なった方が，全体の合計としての導出回数を減少させるためには効果がある．

結果として，組み合わせ数の小さい方から行なう方が，各段階の導出回数も，全体の合計としての導出回数も少なくなることになる．

3．3．3　論理関数との関係

（1）論理関数の主項（prime implicant）

論理関数を扱うブール代数は，命題論理および述語論理とともに，2値問題を扱う記号論理の分野に含まれる．この意味で，論理関数における主項の生成方法は，今回のPrime Implicant 節の導出方法と密接な関係があるものと思われる．論理関数の主項（素項）は，prime implicantともいうが，ここでは節を対象としたprime implicantと区別するため項を対象としたものを主項といい区別することにする．主項とは，論理関数を積項の形で表現する時，この関数を表現した項，もしくは項の組み合わせから生成される（implied）共有項（内項ともいう）であって，他の項に包含されない（not absorbed）ものをいう．

今回の知識ベースは節を対象としており，節集合は和積形で表現される．和積形は積和形に変換することが可能であり，論理関数においては，主項は変数の積の形で表現されるため，論理関数の主項の生成方法の双対な方法が，節を対象とした Prime Implicant節の導出方法となりうる．双対な方法による Prime Implicant節の導出方法では，公理が節集合で示された場合，これを積和形に変換した関数が元の関数の偽に対応するとし，この偽の関数の主項を求め，これを元の節集合のPrime Implicant節に変換することになる．

たとえば，関数Fが和積形で次のように与えられたとすると，

$$F=\{(C,D),\ (A,B,C'),\ (A',C',D'),\ (A',B,C',D),\ (A,B',C,D)\}$$

（ここでは変数Vの否定の表現としてV'を用いる）

ドモルガンの法則を用いて，この和積形を積和形に変換できる．

$$F'=\{C'D',\ A'B'C,\ ACD,\ AB'CD',\ A'BC'D'\}$$

この関数F'は，表3．4の真理値表に対応していることから，この真理値表より，関数Fを積和形で次のように表現することができる．

F＝{A'B'C'D, A'BC'D, A'BCD', A'BCD, AB'C'D, ABC'D, ABCD'}

論理関数の主項に関しては，主項の生成方法と同時に，論理式の簡単化が大きな問題となっている．簡単化については，項数２０程度以下でしか最小性の保証が得られていないが，ＣＭＳの枠組みを利用するにあたっては，最小性は必ずしも必要でないため，ここでは主項の生成方法のみを検討対象とする．

（２）各種の主項の生成方法

主項の生成方法としては，図表による方法[36][37][38]，和積形の展開に基づく方法[39][40]，共有展開に基づく方法[41][42]等がある．

① 図表による方法

（イ）Karnaughの方法[36]（「mapによる方法」とも呼ばれる）

図3．１５のようなKarnaugh図を用いて，論理関数を簡略化し主項を生成方法する方法である．９変数以上の関数の最小論理和形のKarnaugh図による生成は急激に複雑になる．

（ロ）Quine-McCluskeyの方法[37][38]（「table法」とも呼ばれる）

関数を最小項の和で与え，表を用いて主項を生成する方法である．Karnaugh図による方法と同様に，多変数関数の扱いが困難である．

表3．4　関数Fの真理値表

A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
F'	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1
F	0	1	0	0	0	1	1	1	0	1	0	0	0	1	1	0

② 和積形の展開に基づく手法（「木探索による方法」とも呼ばれる）

関数を和積形で表現し，木探索により主項を生成する方法である．

（イ）Slagle-Chang-Leeの方法[39]（図３．１６）

与えられた論理関数を和積形で表現し，木の根から枝出しを行ないながら，その枝にラベルを付け，得られた木の各道筋についてのラベルの積により，主項を生成する方法である．アルゴリズムを図３．１７に示す．

（ロ）Slagle-Chang-Leeの方法の改良法[40]

図３．１７に示したアルゴリズム②において，頻度順に順序付けを行なう．その他は同じ．

（ハ）節展開法[40]

Slagle-Chang-Leeの方法の改良法である．アルゴリズム②において，Ｓから要素数最小の節Ｃoを選ぶ．複数存在する時はその中で頻度和最大のものをＣoとする．その他は同じ．

③ 共有展開に基づく手法

（イ）Tisonの方法[41]

与えられた論理関数を積和形で表現し，その各変数について，幅優先で１回づつ簡略化を行なうことにより，新しい共有項を生成し，包含関係から非主項を取り除いて，主項を生成する方法である．

（ロ）Morrealeの方法[42]

与えられた論理関数を，その各変数について，その肯定リテラルを含む項，否定リテラルを含む項，およびいずれのリテラルも含まない項の，３種類に展開し，tagging functionを作ることによって，非主項の生成を防止しながら，主項を生成する方法である．

Z \ XY	00	01	11	10
0	0	0	0	1
1	1	0	1	1

図３．１５　Karnaugh図の例[36]

（ＸＹＺ＋Ｘ’Ｙ’Ｚ＋ＸＹ’）

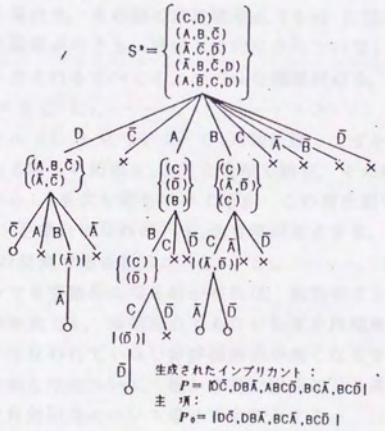


図３．１６　Slangle-Chang-Leeの方法の例[40]

3．3．4　双対手法

（1）双対手法の候補

ＣＭＳの提案者であるReiter他は，和積形の展開に基づく方法の１種であるSlagle-Chang-Leeの方法をその候補として示唆しているが，和積形に展開するためには，偽である積和形から真である積和形を求める必要が有り，公理節集合の偽集合は，元の集合よりも大きい場合が少なくないこと，また節集合に無い節が真か偽か不明であることにより，この方法の応用は難しい．

共有展開に基づく方法（コンセンサス法）の１種であるTisonの方法は，積項を対象としているため，偽である積和形から真である積和形を求める必要が無く，Prime Implicant節を導出するための双対な方法として利用することが可能である．

① 与えられた論理関数Ｆを和積形で表現し，その各和項に含まれるリテラルによる節の集合を，木の根の非終端節点（Ｓ*）に対応づける．

② 木の非終端節点のうち，枝出しの行なわれていない節点（Ｓ）を一つ選び，この各節に含まれるすべてのリテラルを順序付ける．

$O_s = \langle L_1, L_2, \cdots, L_i, \cdots \rangle$

③ 各リテラル（L_i）について，O_sの順に従ってＳから枝出しを行なう．次のように定める新しい節点S_iとＳとを枝で結び，その枝にラベルL_iを付ける．

（イ）ＳからL_iを含む節をすべて除き，この残余節をS_iの要素とする．
　　S_iが空集合になれば，S_iを有効節点とする．

（ロ）S_iの要素である各節からL_1，L_2，・・・，L_{i-1}およびL_i'を除く．
　　一つでも空集合になる節が有れば，無効節点とする

（ハ）有効節点でも，無効節点でもなければ非終端節点とする．

④ 枝出しの行なわれていない非終端節点が無くなるまで，②③を繰り返す．

⑤（イ）得られた木について，根から有効節点に至る道上のラベルの積を，すべての有効節点について求め集合Ｐとする．

（ロ）Ｐから主項P_sを生成する．

図3．17　Slangle-Chang-Leeの方法のアルゴリズム

図3．１８に，前に挙げた（表3．４，図3．１６）積和形の関数Ｆを例にとって，Tison の方法により主項を生成する方法を示す．Ａ，Ｂ，Ｃ，Ｄに関して順に新しい項の生成が行われ，各段階において，他の項を包含する項の消去が行われている．この場合，｛ＢＣＤ'，Ｃ'Ｄ，Ａ'ＢＤ，Ａ'ＢＣ｝が主項Ｐｏとなる．

Tisonの方法により双対問題としてPrime Implicant節を生成する場合，双対問題への変換と，これから求めた主項を元の問題の Prime Implicant節に逆変換する必要がある．図3．１９にごく簡単な例について双対な方法による主項の生成方法を示す．双対な方法では，節集合Σを積和形で表現し，この積和形の集合Σ'の主項Ｐｏ'を求め，さらにその結果を和積形に再変換し，元の節集合の主項Ｐｏを求めることになる．

全公理節に関して幅優先で導出を行なう方法も考えられるが，空節が導出されない場合には終了が保証されていないため，主項が導出される保証はない．

（2）双対手法との比較

節を対象とした今回の導出方法は，積項と節との違いは有るものの，Tisonの方法によく似ており，コンセンサス法の１種であるということができる．順序づけ節コンセンサス法は，出発点の違いにより，双対問題に変換すること無く，節に対して直接導出を行なっている点，および順序づけを行なっている点が，Tisonの方法と異なっている点であり，より速くPrime Implicant節を導出することを可能としている．

論理関数においては，変数の数が多くなった場合，これに比例する形で，１つの項の中の変数の数が多くなるのが，普通のケースであると思われるが，実際の知識ベースにおいては，公理節集合中のリテラルの数が多くなった場合でも，リテラルの数の多い節は非常に稀な例であり，たとえば船舶の航行に関するプロダクションルールの本体部の条件の個数は約３であり[43][44]，１つの節の中のリテラルの数はあまり多くならないと思われる．この意味で，実際の知識ベースを対象とする場合，節集合中のリテラルの数が多くなった場合でも，計算速度の低下は論理関数におけるほど顕著ではないと思われ，コンセンサス法の効果がより期待できる．

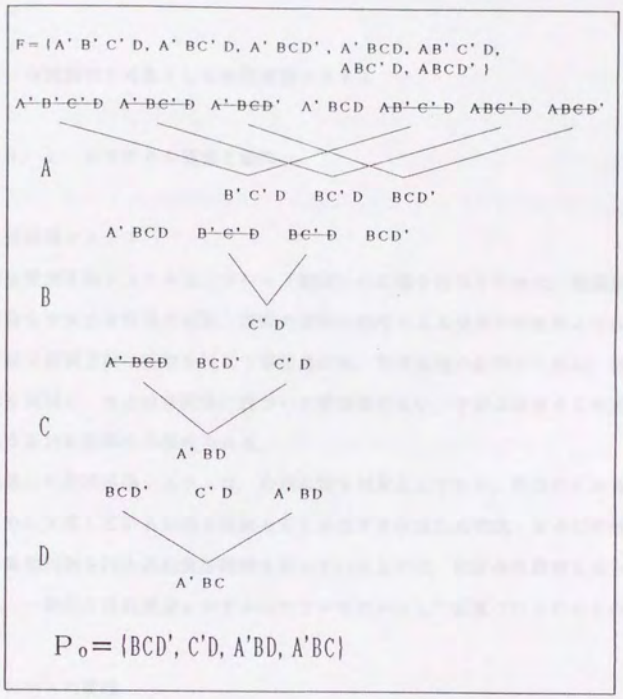


図3．１８　Tisonの方法による主項の生成方法

節集合Σ

$$\Sigma = \{(a \vee b \vee c),\ (a \vee b \vee \sim c)\}$$
$$= \{(a, b, c),\ (a, b, c')\}$$

積和形の表現

$$\Sigma' = \{a'b'c',\ a'b'c\}$$

積和形の主項

$$P_0' = \{a'b'\}$$

節集合Σの主項

$$P_0 = \{(a, b)\} = \{a \vee b\}$$

図3．１９　双対な方法によるPrime Implicant節の生成方法

3．4　命題論理を対象とした仮説推論システム

3．4．1　システムの構成と動作

（1）仮説推論システム

一般的な仮説推論システムは，ゴール（観測）の証明を行なうために，仮説空間を決定し仮説集合を生成する仮説生成部，仮説の展開の順序による探索の制御および仮説候補の選択と不要な仮説選択の回避を行なう仮説選択部，無矛盾性の証明のために，推論を展開し正当性を判別し，また依存関係に基づいた管理を行ない，矛盾に関係する仮説の棄却制御を行なう仮説検証部から構成される．

今回構築した仮説推論システムは，命題論理を対象としており，発想的にある定理を証明するために欠落している知識を仮説として生成する仮説生成機能，および矛盾に関係する仮説の棄却制御を行う仮説検証機能を持っているものの，仮説選択機能を持っていないことから，一般的な仮説推論システムのサブシステムとして位置づけられるものである．

（2）システムの構成

構築した仮説推論システムは，管理部，データ変換部，データベースおよび導出部から構成される．（図3．20）

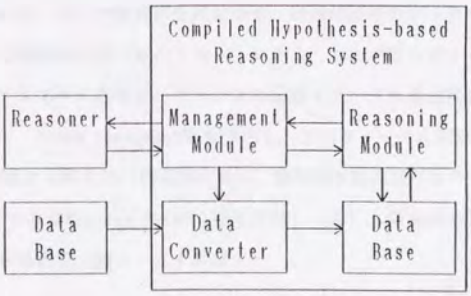


図3．20　仮説推論システムの構成

① 管理部（Management Module）

問題解決器（Reasoner）から質問節を受け取り，欠落した知識の仮説として，もしくは無矛盾での充足可能性の判定結果として，Minimal Support節を問題解決器に戻す．また，データ変換部に対して問題解決器のデータベースからのデータの取り込みの指示，導出部に対してPrime Implicant節の導出，もしくはMinimal Support節の生成指示を行なう．

② データ変換部（Data Converter）

問題解決器のデータベースから公理節集合を取り込み，データの表現形式をシステム用に，公理集合名，節等の項よりなる複合項の形に変換する．

③ データベース（Data Base）

データベースの中の各節は，導出部から必要に応じて呼び出され，作業記憶に複写される．また導出されたPrime Implicant節はデータベースの中に記録される．

④ 導出部（Reasoning Module）

双形リテラルについて，導出可能な節の組み合わせ数を計算し，順序づけを行なってPrime Implicant節を導出するとともに，管理部より指示を受け，Prime Implicant節との包含関係から，Minimal Support節の高速生成を行なう．

（3）システムの動作

本研究においては，システムはSONY NEWS-841上にインタプリタ形式のK-Prologを用いて構築した．システムの起動は問題解決器の中で行なう．

問題解決器においては，節は公理節集合名と節のリストとを項とする複合項の形で表現される（①）．システムは，データ変換部を起動させ，問題解決器のデータベースから公理節集合を取り込み，公理節集合名，節のリスト，クラス，および節のリスト長を項とする複合項の形に変換し，システム内のデータベースに記録する．また導出部を起動させ，この公理節集合についてPrime Implicant節を導出し，データベースに記録する（②）．Minimal Support節を生成する時には，問題解決器は，質問節を複合項の形でシステムに対して送り込む(③)．システムはMinimal Support節を生成し（④），問題解決器に戻す．

データの表現形式とその例を，図3．21に示す．

3．4．2　システムの構築にあたって

（1）Prime Implicant節の導出

Prime Implicant節の導出に当たっては，順序づけ節コンセンサス法（ＯＣＣ法）を用いた．ＯＣＣ法を用いるに当たっては，導出に用いるリテラルを含む節の探索，冗長な節の削除が重要である．

節リスト中のリテラルの探索を容易にするため，節リストのソートを行なった．また冗長な節の削除の効率化を図るため，節リストの長さをあらかじめ，複合項の中に記録し，長さの小さい節から包含関係のチェックを行なった．また，新しい節が導出される毎に全節リストについて包含関係を検討すると，これに要する時間が多大になるため，節の削除はあるリテラルについての導出が終了する毎に行なうことにした．

<u>問題解決器からの公理節の表現</u>

```
pro(group, clause_list)
  ex.   pro(gl, [a, ~b, c])
        pro(gl, [a, b, c])
```

<u>仮説推論システムのデータベースにおける節の表現</u>

```
pro_list(group, clause_list, class, list_length)
  ex.   pro_list(gl, [a, ~b, c], original, 3)
        pro_list(gl, [a, b, c], original, 3)
        pro_list(gl, [a, c], prime_i, 2)
```

<u>質問節の表現</u>

```
cms(group, query, Minimal_Supports)
  ex.   cms(gl, ms([a]), MS)
```

<u>Minimal Support節の表現</u>

```
pro_list(group, clause_list, class, list_length)
  ex.   pro_list(gl, [~a], min_sup, 1)
        pro_list(gl, [c], min_sup, 1)
```

図3．21　データの表現形式

メモリーの節約に当たっては，working memoryを利用し，導出の終了した Prime Implicant節等，最小限必要な節だけをデータベース中に残し，他はメモリーから消去するとともに，知識集合にはグループ名をつけ，ある知識集合に関するPrime Implicant節の導出に関係無い知識に関する導出を抑制した．また，メモリーの節約，及び不要なバックトラックを避けるため，可能な範囲でcut（！）を用いた．

（2）Minimal Support節の生成

Minimal Support節の生成については，Reiter他が指摘した高速生成方法を用いることとした．この場合，質問節が非ユニット節の時には，必ずしも集合の引き算によってMinimal Support節を生成することができないので，質問節を分解して集合の引き算を行なうとともに，生成されたMinimal Support節の候補の間で，Prime Implicant節導出において行なったように，節リストのソート，節リストの長さを利用した効率化を図った．

質問節がユニット節の時には，包含関係だけを検討することにより，Minimal Support節を生成することが可能であるので，この場合不必要となる処理を省略し，生成の高速化を図った．

（3）仮説の生成棄却

このシステムにおいては，質問節を証明するために，欠けている知識を仮説として生成するが，コンパイルを有効に利用するため，採用した仮説によるPrime Implicant節の変更は行なわず，仮説の生成棄却は，一度生成されたPrime Implicant節には影響を与えないようにした．

また，質問節の無矛盾性を検討する場合にも，質問節は仮説に相当するため，コンパイルの結果を有効に利用するため，基本的にはPrime Implicant節の変更を行なわない．質問節を定理として，知識集合に追加することも可能であるが，古いPrime Implicant節から，新しいPrime Implicant節を導出することになり，高速化の妨げとなるからである．

３．４．３　他の真理管理機構との関係

（１）ＡＴＭＳとの関係

ＡＴＭＳ問題はすべての最小のnogoodを見いだし，すべてのリテラルに対し次の性質を持つ環境の集合を同定することであるということができる．

① Soundness　各環境でリテラルが成立する

② Cosistency　環境はnogoodではない

③ Completeness　あるリテラルが成立する環境は同定されたある環境の上位集合に相当する

④ Minimality　同定されたどの環境も他の環境の部分集合ではない

ＡＴＭＳにおいてはデータは，＜datum, label, justification＞の形で表現される．このデータ表現において，labelはデータの成立する環境の集合を表し，命題リテラルの中のあるリテラルで，あるデータが成立する仮説に相当する．またjustificationはデータの理由付けとなる下位のデータの集合を表している．

図３．２２に，ＡＴＭＳにおける，ＡＮＤ集合で表現される無矛盾な，Minimal Labelを生成する方法を示す．図においては，データ＜da_1, {{A, B}, {B, C, D}}, {...}＞＜da_2, {{A, C}, {D, E}}, {...}＞に対し，{A, B, C}, {A, B, C, D}, {B, C, D, E}という Minimal Labelが生成されている．

ＡＴＭＳを一般化したものとしてＣＭＳが位置づけられるように，同じ問題をＣＭＳを用いて解くことが可能である．ＣＭＳを適用した例を図３．２３に示す．da_3を質問ユニット節とし，{da_3Ｖ～da_1Ｖ～da_2} 等のホーン節からなる集合を公理節集合として，Minimal Support節を生成すると，このMinimal Support節を構成するリテラルの否定のリテラルの連言からなる，無矛盾なMinimal labelを求めることができる．

（２）ＣＳＰとの関係

ＣＳＰ(Constraint Satisfaction Problem)[45]は，ＡＴＭＳと同様に inconsistentと既に分かった空間の探索を，いかに避けるかを達成するアプローチである．ＣＳＰにおいては制約を満たす，命題論理でいえばリテラルに相当する変数の値を決めることを行っており，このことは制約ネットワークで表現することが可能である．より簡単な制約ネット

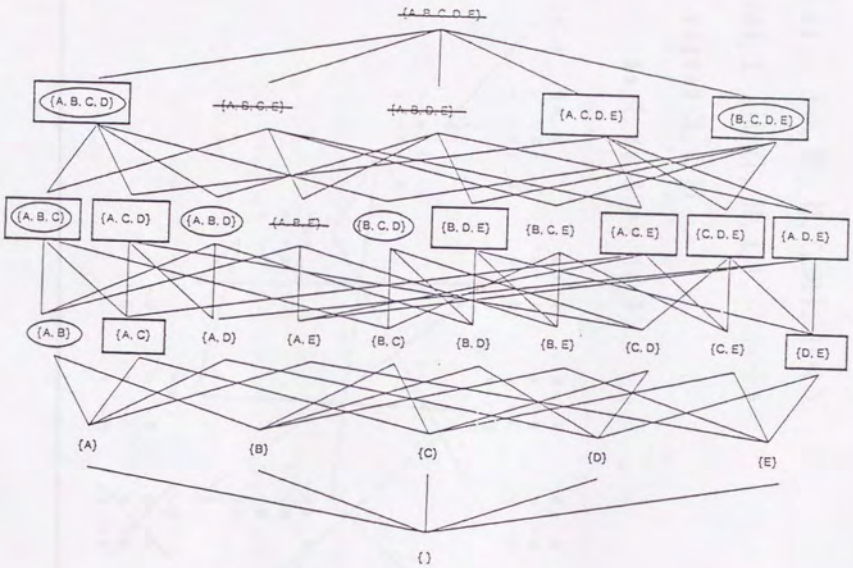


図3．２２　ＡＴＭＳにおけるMinimal Labelの生成方法[14]

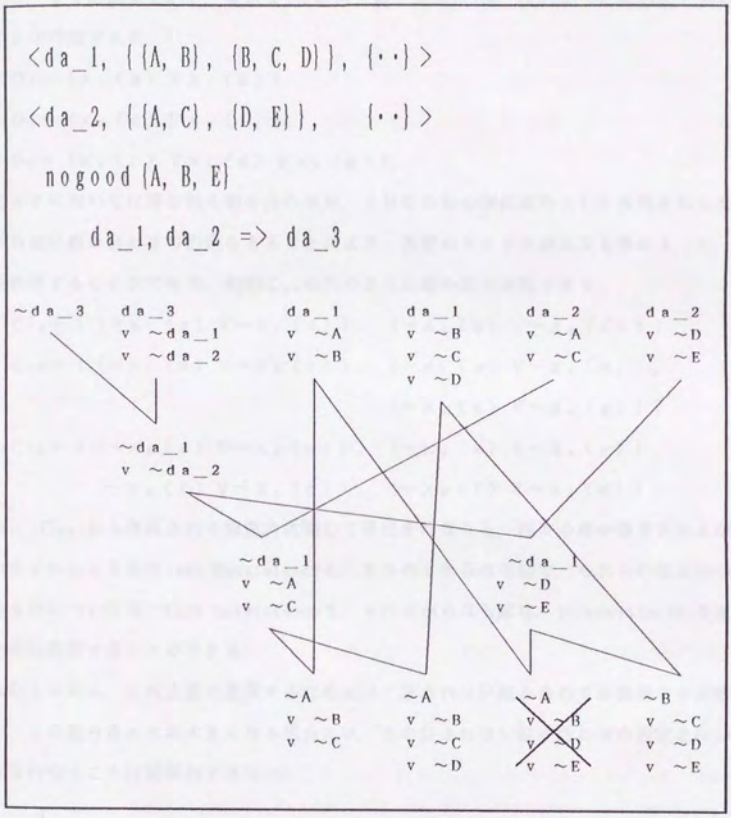


図3．２３　ＣＭＳによるMinimal Labelの生成方法

ワークへの変換を行なう，Local Consistencyアルゴリズムは，無効な状態を前処理で除くことを行なっており，これはＡＴＭＳの機能と類似している．

図３．２４の例において，変数X_iの範囲を表す変域D_iは，次のように節形式で表現することが可能である．

$D_1 = \{x_1(a) \vee x_1(b)\}$

$D_2 = \{x_2(e) \vee x_2(f)\}$

$D_3 = \{x_3(c) \vee x_3(d) \vee x_3(g)\}$

ＣＳＰにおいては許される組み合わせが，ＡＮＤの形の制約条件として表現されるため，許されない組み合わせの否定をとることにより，否定のリテラルからなる節によって，制約を表現することができる．制約C_{ij}は次のように節の形で表現できる．

$C_{12} = \{\{\sim x_1(a) \vee \sim x_2(e)\},\ \{\sim x_1(a) \vee \sim x_2(f)\}\}$

$C_{13} = \{\{\sim x_1(a) \vee \sim x_3(c)\},\ \{\sim x_1(a) \vee \sim x_3(d)\},$
$\{\sim x_1(a) \vee \sim x_3(g)\}\}$

$C_{23} = \{\{\sim x_2(e) \vee \sim x_3(c)\},\ \{\sim x_2(e) \vee \sim x_3(g)\},$
$\{\sim x_2(f) \vee \sim x_3(c)\},\ \{\sim x_2(f) \vee \sim x_3(d)\}\}$

D_i，C_{ij}から構成される節集合に関して導出を行なうと，種々の節が導出されるが，２つのｘからなる節は Arc Consistencyを，３つのｘからなる節で，それらの間にpathのあるものについては，Path Consistencyを，ｋのｘからなる節は，k-Consistency を求めるために利用することができる．

しかしながら，この方法で表現するためには，許されない組み合わせを表現する必要があり，この組み合わせが大きくなる場合には，この許されない組み合わせの否定を用いて導出を行なうことは効果的ではない．

例	変数	X_1, X_2, X_3
	変域	$D_1 = \{a, b\}$, $D_2 = \{e, f\}$, $D_3 = \{c, d, g\}$
	制約	$C_{12} = \{be, bf\}$, $C_{13} = \{bc, bd, bg\}$,
		$C_{23} = \{ed, fg\}$

制約を満たす解

$\{X_1, X_2, X_3\} = \{b, e, d\}, \{b, f, g\}$

図３．２４　ＣＳＰの問題の例[45]

3．4．4　命題論理を対象とした仮説推論システムの評価

（1）評価方法

今回開発した順序づけ節コンセンサス法の評価を，次の３種類の方法を比較することにより行なった．また，コンパイル形式によるMinimal Support節生成の高速化の程度を評価するため，インタプリタ形式との比較を行なった．

① 幅優先導出方法
(Breadth-first Method)

② 非順序づけ節コンセンサス法
(Non-ordered Clause-Consensus Method)

③ 順序づけ節コンセンサス法
(Ordered Clause-Consensus Method：ＯＣＣ法)

評価に用いた公理集合としては，過去にプロダクションルールによりエキスパートシステムを構築した[46][47]，船舶の衝突予防に関する，実際の知識集合の中心的な部分と同じ内容をもつ命題論理節集合，および乱数により発生させた命題論理節集合とを用いた．前者の知識集合の節の数は最大６０，この時のリテラル数は３６，一つの節中の平均リテラル数は２．８であった．乱数により節を生成する場合，これを参考として一つの節中の平均リテラル数を３とし，また４，８，１６のリテラルについて正負リテラルを発生させ，それぞれのケースについて１６の節を生成した．表３．５にPrime Implicant節の導出時間を，表３．６，３．７に，Minimal Support節の生成時間を示す．ＣＰＵ時間は，各ケースとも，あらかじめ設定したそれぞれ３つの質問節に対する時間を平均したものである．使用計算機は演算速度２．４MIPSの SONY-NEWS-841 である．なお，時間計測はＣＰＵ時間50.000秒で打ち切り，明かにこれを超えると思われるものについては計測を省略した．

（2）Prime Implicant節の導出時間（表３．５）

幅優先法と比較した場合，節コンセンサス法を用いることにより約５～９０倍以上高速でPrime Implicant節を導出することができた．

ごく小規模ではあるが実際の知識集合を対象としたケースについては，順序づけをしない場合と比較して，順序づけを行なった場合の方がケースによっては１０倍以上の高速化を図ることができた．これは節集合中の双形リテラルの組み合わせ数が平均していないことにより，順序づけの効果が大きく現われたものである．両者の比が約２程度の場合もあるが，これはたまたま，良い順番で導出が行なわれたためであると思われる．順序づけをリテラル毎に行なう方法も実行してみたが，最初に一度だけ行なう方法との差は，今回検討した範囲ではほとんどなかった．

公理節集合を乱数により発生させた場合が，最も効果が無い場合であり，当然のことながら両者にはほとんど差は無かった．実際の知識集合を対象としたケースの方が，節数，リテラル数は多いのにもかかわらず，乱数を用いて生成したケースよりも，節数の増加に伴うＣＰＵ時間の増加は顕著ではなく，実際の知識集合を対象としたケースの方が，順序づけ節コンセンサス法を用いることにより，より高速化を期待できる結果となった．

表３．５　命題論理におけるPrime Implicant節の生成時間

	Number of Clauses	Number of Literals	Number of Prime Implicants	CPU Time (Unit : sec)		
				Ordered Consensus Method	Non-ordered Consensus Method	Breadth -first Method
Propositions Generated at Random	16	4	4	6.0	5.5	97.0
	16	8	33	66.3	67.5	3019.2
	16	16	141	477.0	520.9	4864.0
Propositions of a Knowledge Set	16	18	11	7.6	22.7	33.9
	31	25	27	53.3	109.8	210.1
	31	26	67	36.2	443.4	1476.9
	60	36	102	171.1	978.6	15626.6

（３）Minimal Support節の生成時間（表３．６，３．７）

ＯＣＣ法を用い，コンパイル形式とインタプリタ形式との比較を行なった．空節が生成されない場合，質問節がユニット節のケースでは，コンパイル形式の方が１０～１００倍程度高速であった．また，質問節が２つのリテラルから構成されるケースでも，１０～２０倍程度の高速化を達成することができた．Minimal Support節を生成する時間と，Prime Implicant節を導出した時間とを合計しても，コンパイル形式の方が２倍以内ではあるが，インタプリタ形式よりも速い結果となっている．空節が生成される場合には，空節が生成される時点に大きく依存しており，全解探索をしないため差が小さくなっているが，２～２０倍程度高速である．Minimal Support節の生成に関しても，実際の知識集合を対象としたケースの方が，ＯＣＣ法の高速化の効果が，より期待できる結果となった．

幅優先法（インタプリタ形式）と，ＯＣＣ法（コンパイル形式）とを比較した場合，空節が生成されない場合，コンパイル形式の方が約６０～１００００倍以上高速でMinimal Support節を生成することができた．しかしながら節の数が少なく空節が生成される場合にには，必ずしもコンパイル形式の方が速いとはいえなかった．これはコンパイル形式では，Minimal Support節を生成するためには，リテラルの照合，集合演算および包含関係の検討を行なう必要があるためであると思われる．１６のリテラルを用いて，乱数により１６の命題論理節を発生させたケースにおいて，１４１のPrime Implicant節が生成されたように，このPrime Implicant節の数の増加に伴い，集合演算等の時間が大きくなるため，今回のシステムでは，この計算に時間を要することとなる．また，空節が生成されない場合の計算量を少なくすることを第一に考え，導出順序を決めたため，質問節に含まれるリテラルについて先に導出を行なった方が空節が速く生成される場合に，幅優先法より遅くなったものと思われる．空節の生成を目的とする場合には，検討すべき事項となる．

表3．6　命題論理におけるMinimal Suppor節の生成時間

（乱数により発生させた公理節集合の場合）

Number of Clauses	Number of Literals	Literals in Query Clause	Minimal Support Clauses	CPU Time (unit:sec)		
				Ordered Consensus Method		Breadth-first Method
				Compiled	Interpreted	
16	4	1	non-empty	0．4	7．5	102．4
16	8	1	non-empty	1．2	118．1	above 50000
		2	non-empty	6．9	146．5	above 50000
		3	empty	4．3	92．4	26．9
16	16	1	non-empty	9．0	822．0	–
		2	non-empty	121．3	1122．7	–
		3	empty	55．1	120．4	17．9
16	18	1	non-empty	1．1	10．3	65．0
		2	empty	1．2	9．7	39．8

表3．7　命題論理におけるMinimal Suppor節の生成時間

（実際の知識集合を用いた公理節集合の場合）

Number of Clauses	Number of Literals	Literals in Query Clause	Minimal Support Clauses	CPU Time (unit:sec)		
				Ordered Consensus Method		Breadth-first Method
				Compiled	Interpreted	
31	25	1	non-empty	1．5	72．3	715．5
		2	empty	3．3	70．2	118．1
31	26	1	non-empty	1．8	53．6	2260．7
		2	empty	11．7	20．9	79．1
60	36	1	non-empty	2．6	248．4	above 50000
		2	empty	24．0	172．0	311．6

3．5　述語論理への拡張

3．5．1　述語論理と命題論理

命題論理においては，真か偽の２値をとる基本命題（素命題）を基に，合成命題（複合命題）の真偽値を論じ，推論を行なう．命題論理は記号論理学の一分野で，２値問題を扱う記号論理の分野には，この他に述語論理や論理関数を扱うブール代数が含まれる[48]．

命題論理は合成命題を基本命題に分解しているものの，命題の内容までに踏み込むことができない限界がある．このような命題論理の限界を超えるものとしては，基本命題をさらに主語と述語に分解した述語論理がある．述語論理では述語記号の他に，主語等を表現する複数の項を用いることが可能であるとともに，変数を用いることが可能であり，命題論理と比較してより高い表現力がある．

変数を用いることにより，表現力が高まるとともに，命題論理で表現した知識集合と同じものを圧縮して表現することが可能であり，知識集合の規模の拡大に対処し，知識集合をコンパクトに表現するためにも効果的である．

項としてのみ変数を扱うものを，第一階述語論理，述語記号でも変数を扱うものを，第二階述語論理というが，ここでは述語論理として，体系の確立している第一階述語論理を対象として，順序づけ節コンセンサス法（ＯＣＣ法）の拡張を図ることにする．

3．5．2　拡張にあたっての検討事項

（1）述語論理への拡張方法

述語論理を対象とするための方法として，述語論理の論理式の中の変数に，エルブラン空間の要素を基礎代入する方法（instantiation）を挙げている論文が多いが，この場合，関数を含まない論理式に限定しても，空間が節の数，定数の数，リテラル中の項の数に比例した大きさになるという問題があり，述語論理を対象とした真理管理機構の開発は，重要な課題として指摘されている．

論理式においては，代入が適用可能であれば，束縛変数を書き換えることが可能である

[35]．このことは自由変数のみならず束縛変数についても，変数に変数を代入することが可能であることを意味している．そこでここでは，変数を当初から全部instantiateすることを考えず，導出に際して，変数を含まない基礎代入ではなく，変数を代入できるものについては変数を代入し，変数を含んだ述語論理節について，Prime Implicant節を導出することを考えた．

ここでは，述語論理節を構成するリテラルの中の項としては，定数と変数のみを考え，関数を考えないこととする．変数の代入を優先する代入をθとし，任意の単一化代入をλとすると，関数を対象としないことから$\lambda=\theta\cdot\sigma$となる代入$\sigma$が存在し，この変数を許容する方法は，この範囲で最汎単一化代入法（most general unifier method）である．最汎単一化代入法を用いたＯＣＣ法を，ここではＣＯＭＰ（ＣＯmpiling Method for Predicate）と呼ぶことに，またリテラルの中の項としての変数に，導出に先だって定数を代入する方法を用いる方法を，instantiation法と呼ぶことにする．

また，ＣＯＭＰにおいては，Minimal Support節の高速生成を目的としているため，再帰的な節を対象とした場合Prime Implicant節が大量に生成されるため，再帰的な節は対象としない．

（２）述語論理への拡張に当たっての検討事項

述語論理で表現された節集合について導出を行なうためには，

① 論理節中の変数に代入（substitution）を行ない，導出のための単一化（unify）を行なうこと，

② 単一化を行なった節同士の間で導出（resolution）を行なうこと，そして

③ 節について因子化（factoring）を行なうことが必要となる．

さらに，ＯＣＣ法を適用するにあたっては，

④ 包含関係による節の削除（suppress）の方法，および

⑤ 順序づけの方法

について検討する必要がある．

（３）単一化代入

論理式においては，代入が適用可能であれば，変数を変更することが可能である．単一化代入を行なうにあたって，まず改名代入（renaming substitution）を検討する．導出

の親節となる二つの節が共通の変数を持つ時には，一方の節の変数名を変え，共通の変数を持たないように改名代入を行なう必要がある．ついで，変数に項を代入するが，変数を代入することが可能な変数については，変数による単一化を行なう．このように項として変数を許容することにより，全部 instantiateする方法と比較して，問題空間の拡大を制限することが可能となる．

（4）導出

ＯＣＣ法においては，二項導出（binary resolution）を用いている．ＣＯＭＰにおける述語論理を対象とした二項導出の方法を，次に例を用いて示す．

① 項として定数を含む節のみに関して導出を行なう場合，命題論理と同じである（図３．２５－例１）．

② 項として定数を含む節と，変数を含む節とに関して導出を行なう場合，変数に定数が代入される（図３．２５－例２）．

③ 項として変数のみを含み定数を含まない節同士に関して導出を行なう場合，変数には変数が代入される（図３．２５－例３）．

例１　ｐ（ａ）と　～ｐ（ａ）の場合，
空節が導出される．

例２　ｐ（ａ）と　～ｐ（Ｘ）の場合，
代入θ＝｛Ｘ：＝ａ｝を適用すると，
空節が導出される．

例３　ｐ（Ｘ）Ｖｑ（Ｙ）と　～ｐ（Ｙ）の場合，
① 共通の変数があるため，まず第２節に対して
改名代入θ１＝｛Ｙ：＝Ｙ２｝を適用，
ｐ（Ｘ）Ｖｑ（Ｙ），　～ｐ（Ｙ２）
② ついで二つの節に対して，
代入θ２＝｛Ｙ２：＝Ｘ｝を適用すると，
ｑ（Ｙ）が導出される．

図３．２５　導出の例

（5）因子化

導出節の中のリテラルとして，同じ述語記号を持つものが存在することがあるが，instantiateする方法を用いる場合には問題とはならない．しかしながら，変数の代入を許容する場合には，因子化を検討する必要がある．

ある節の中に，単一化可能なリテラルが存在する場合には，この一つの節について代入を行ない（因子化），因子（factor）を生成することができる（図3．２６－例１）．リテラルの数や変数の項数が多い場合には，因子は必ずしも１つであるとは限らない（図3．２６－例２，例３）．この結果，導出節と因子とからなる複数の節が，見かけ上導出されたことになる場合がある．

（6）包含関係による節の削除

節集合の中により一般的な節がある場合，一般的ではない節を除く．命題論理の場合は節の包含関係により，一般的でない節を簡単に削除する事ができたが，述語論理の場合には，変数の項と定数の項との関係にも考慮する必要が出てくる．

ＣＯＭＰにおいて，冗長な節を削除する方法を，次に例を用いて示す．

① 項として定数のみを含む節間の場合，命題論理と同じである（図3．２７－例１）．

② 項として定数を含む節と，変数を含む節との場合，変数を含むPrime Implicant節を認め，冗長な節を削除する（図3．２７－例２）．

例1　ｐ（Ｘ）Ｖｐ（Ｙ）の場合，
　因子としては次のものが生成される．
　ｐ（Ｘ）

例2　ｐ（ａ）Ｖｐ（Ｘ）Ｖｐ（Ｙ）の場合，
　因子としては，次の2つのものが生成される．
　ｐ（ａ）Ｖｐ（Ｘ），ｐ（ａ）

例3　ｐ（Ｘ，Ｙ，Ｚ）Ｖｐ（Ｚ，Ｕ，Ｖ）Ｖｐ（Ｖ，Ｙ，Ｕ）の場合，
　因子としては，次の4つのものが生成される．
　ｐ（Ｘ，Ｙ，Ｘ）Ｖｐ（Ｘ，Ｙ，Ｙ）
　ｐ（Ｘ，Ｙ，Ｚ）Ｖｐ（Ｚ，Ｚ，Ｘ）
　ｐ（Ｘ，Ｕ，Ｚ）Ｖｐ（Ｚ，Ｚ，Ｚ）
　ｐ（Ｘ，Ｘ，Ｘ）

図3．２６　因子化の例

③　項として変数のみを含み，定数を含まない節同士の場合，変数を含むPrime Implicant節を認め，冗長な節を削除する（図３．２７－例３）．

（７）順序づけの方法

述語論理を対象としているＣＯＭＰで，正式に順序づけを行なうにあたっては，定数と変数とを区別して組み合わせ数を計算する必要がある．しかしながら，オーバーヘッドの負担を軽くするために，定数を含む節と，変数を含む節とを区別せず，項数が等しく，同じ述語記号を持つリテラルのみに注目して分類し，組み合わせ数を概算して順序づけを行なう略法の方が現実的である．組み合わせ数の計算例を次に示す．

例　｛ｐ（Ｘ），ｐ（ａ），ｐ（ｂ），～ｐ（Ｘ），～ｐ（ａ）｝の場合，
ｐ（Ｘ）については，相補リテラル（complementary pair）が３組，
ｐ（ａ）については２組と数えるところ，双形リテラルｐについて，
２＊３＝６組と概算する．

例１　ｐ（ａ）Ｖｑ（ｂ）と　ｐ（ａ）の場合，
　　ｐ（ａ）Ｖｑ（ｂ）を削除する．

例２　①　ｐ（ａ）と　ｐ（Ｘ）の場合，
　　ｐ（ａ）を削除する．
　　②　ｐ（ａ）Ｖｑ（ｂ）と　ｐ（Ｘ）の場合，
　　ｐ（ａ）Ｖｑ（ｂ）を削除する．
　　③　ｐ（ａ）Ｖｑ（Ｘ）と　ｐ（ａ）の場合，
　　ｐ（ａ）Ｖｑ（Ｘ）を削除する．
　　④　ｐ（Ｘ）Ｖｑ（ｂ）と　ｐ（ａ）の場合，
　　削除できない．
　　⑤　ｐ（Ｘ）Ｖｑ（ｂ）と　ｐ（Ｘ）の場合，
　　ｐ（Ｘ）Ｖｑ（ｂ）を削除する．

例３　ｐ（Ｘ）Ｖｑ（Ｙ）と　ｐ（Ｘ）の場合，
　　ｐ（Ｘ）Ｖｑ（Ｙ）を削除する．

図３．２７　包含関係による節の削除の例

3．5．3　述語を対象としたコンパイル法（ＣＯＭＰ）の効果についての検討

（1）１つの節の中のリテラルの項が，すべて同じ変数である場合

instantiation法では，質問節が与えられた時点で，１回のinstantiateが行なわれ，instantiateされた節集合に対して，ＯＣＣ法を１回適用し，Prime Implicant節を求め，これと質問節との集合演算により，Minimal Support節を生成することになる（図3．２８－①）．

ＣＯＭＰでは，ＯＣＣ法によりコンパイルを１回行なうことにより，変数の項を持ったリテラルによる節の形で，Prime Implicant節を得ることができる．このコンパイルした節集合に対して，１つの定数のみを含む質問節を与えた場合，１回のinstantiateと集合演算とによりMinimal Support節を生成することができる（図3．２８－②）．

この範囲では，コンパイルの計算量，Prime Implicant節の数，Minimal Support節の数，生成に要する計算量とも，instantiation法とＣＯＭＰとで同程度である．

しかしながら，引き続いて異なる定数を持った質問節について，Minimal Support節を生成する場合，instantiation法の場合，先に行なったコンパイルは，このMinimal Support節を生成するためには，何の効果も持たず，ほとんど同様なコンパイルを，もう一度繰り返し行なう必要がある（図3．２８－③）．

これに対し，ＣＯＭＰでは，既にPrime Implicant節が導出されているので，もう一度新たにコンパイルする必要は無く，instantiationと集合演算とのみによりMinimal Support節を生成することができるため，計算量はＣＯＭＰのほうが少なくなる（図3．２８－④）．

（2）１つの節の中のリテラルの項が，すべて変数で２種類以上存在する場合

instantiation法では，定数の組み合わせ数に比例して，公理節集合が拡大することにより，導出のための計算量が大きくなるとともに，Prime Implicant節も多くなり，Minimal Support節を生成するための，集合演算量も多くなる．最大では定数の組み合わせ数をｎ，定数の数をk，変数の数をｍとすると，$n = k^m$ となるが，同じ節も存在するため可能な組み合わせ数は通常これより少ない（図3．２９－①）．

これに対し，ＣＯＭＰでは，Prime Implicant節の数は，instantiation法に比較した場合，大幅に少なく，またMinimal Support節を生成する場合も，Prime Implicant節の数自

例1
公理節
{p(X) V q(X) V r(X), ～p(X2) V q(X2)}
第一の質問節　q(a)
第二の質問節　r(b)

ケース(1)　第一の質問節q(a)

① instantiation法の場合
基礎代入　$\theta 1$={X:=a, X2:=a}
公理節
{p(a) V q(a) V r(a), ～p(a) V q(a)}
Prime Implicant節
{q(a) V r(a), ～p(a) V q(a)}
Minimal Support節
{r(a), ～p(a), ～q(a)}

② COMPの場合
Prime Implicant節
{q(X) V r(X), ～p(X2) V q(X2)}
代入　$\sigma 1$={X:=a, X2:=a}
Minimal Support節
{r(a), ～p(a), ～q(a)}

ケース(2)　第二の質問節r(b)

③ instantiation法の場合
基礎代入　$\theta 2$={X:=b, X2:=b}
公理節
{p(b) V q(b) V r(b), ～p(b) V q(b)}
Prime Implicant節
{q(b) V r(b), ～p(b) V q(b)}
Minimal Support節
{q(b), ～r(b)}

④ COMPの場合
代入　$\sigma 2$={X:=b, X2:=b}
Minimal Support節
{q(b), ～r(b)}

図3．28　述語を対象としたコンパイル法COMPの効果(1)
節中の項：すべて変数，　節中の変数：1種類

例2

公理節

{p(X) V q(Y) V r(Z), ~p(X2) V q(Y2)}

質問節 q(a) V s(b)

① instantiation法の場合

基礎代入

θ1={X:=a, Y:=a, Z:=a, X2:=a, Y2:=a},

θ2={X:=a, Y:=a, Z:=a, X2:=a, Y2:=b},

.

.

θ32={X:=b, Y:=b, Z:=b, X2:=b, Y2:=b}

公理節

{p(a) V q(a) V r(a), ~p(a) V q(a),
p(a) V q(a) V r(b), ~p(a) V q(b),
p(a) V q(b) V r(a), ~p(b) V q(a),
p(a) V q(b) V r(b), ~p(b) V q(b),
p(b) V q(a) V r(a),
p(b) V q(a) V r(b),
p(b) V q(b) V r(a),
p(b) V q(b) V r(b)}

Prime Implicant節

{q(a) V r(a), q(a) V r(b),
q(b) V r(a), q(b) V r(b),
~p(a) V q(a), ~p(a) V q(b),
~p(b) V q(a), ~p(b) V q(b)}

Minimal Support節

{r(a), r(b), ~p(a), ~p(b),
~q(a), ~s(b)}

② COMPの場合

Prime Implicant節

{q(Y) V r(Z), ~p(X2) V q(Y2)}

代入 σ1={Y:=a},
σ2={Y2:=a}

Minimal Support節

{r(Z), ~p(X2), ~q(a), ~s(b)}

図3．29 述語を対象としたコンパイル法COMPの効果（2）
節中の項：すべて変数， 節中の変数：2種類以上

体が少ないことから，計算量は大幅に少なくなる（図3．２９－②）．

（3）リテラル中の項として，定数と変数とが存在する場合

節中の定数が１種類だけの場合，質問節中の定数が公理節中の定数と同じの場合には，instantiation法の計算量と，ＣＯＭＰのとはほぼ同程度である．

定数が異なる場合には，instantiation法では基礎代入の数が増加することにより，Prime Implicant節の数も増加し，その分計算量が増加する（図3．３０－①）．

ＣＯＭＰの場合，因子化を行なうことのできる節が存在することがあるが，因子により元の節をsuppressできるとは限らない（図3．３０－②）．

例3
公理節
{p（a）Ｖ q（X）Ｖ r（X），～p（X2）Ｖ q（X2）}
第一の質問節　q（a）
第二の質問節　q（b）

ケース（1）　第一の質問節q（a）
基礎代入法の場合も，最汎単一化代入法の場合も，例1－①，②にほぼ同じ

ケース（2）　第二の質問節q（b）

① instantiation法の場合
公理節
{p（a）Ｖ q（a）Ｖ r（a），～p（a）Ｖ q（a），
p（a）Ｖ q（b）Ｖ r（b），～p（b）Ｖ q（b）}
Prime Implicant節
{q（a）Ｖ r（a），q（a）Ｖ q（b）Ｖ r（b），
～p（a）Ｖ q（a），p（a）Ｖ q（b）Ｖ r（b），
～p（b）Ｖ q（b）}
Minimal Support節
{q（a）Ｖ r（b），p（a）Ｖ r（b），～p（b），
～q（b）}

② ＣＯＭＰの場合
Prime Implicant節
{q（a）Ｖ q（X）Ｖ r（X），q（a）Ｖ r（a），
p（a）Ｖ q（X）Ｖ r（X），～p（X2）Ｖ q（X2）}
Minimal Support節
{q（a）Ｖ r（b），p（a）Ｖ r（b），～p（b），
～q（b）}

図3．30　述語を対象としたコンパイル法ＣＯＭＰの効果（3）
節中の項：定数と変数が混在，　節中の変数：１種類

変数の数が増加すると，instantiation 法では，節中の変数が１種類だけの場合でも，定数の数に比例して公理節の数が増加するし，さらに変数の数にも比例して公理節の数が増加する（図３．３１－①）．

これに対しＣＯＭＰでは，因子により元の節を包含関係から削除できるとは限らないにしても，公理節の数は基礎代入法と比較して少なく，この傾向は定数の数や変数の数の増加に伴い顕著となる．特に変数が多い場合に，ＣＯＭＰは有効である（図３．３１－②）．

（４）リテラル中の項が，すべて定数の場合

項として変数を含んだリテラルが存在しないためinstantiateする必要はない．

```
例4
  公理節
      {p(a) V q(X) V r(Y), ~p(X2) V q(Y2)}
  質問節  q(b)

①  instantiation法の場合
  公理節
      {p(a) V q(a) V r(a), ~p(a) V q(a),
       p(a) V q(a) V r(b), ~p(a) V q(b),
       p(a) V q(b) V r(a), ~p(b) V q(a),
       p(a) V q(b) V r(b), ~p(b) V q(b)}
  Prime Implicant節
      {q(a) V r(a), q(a) V r(b),
       q(b) V r(a), q(b) V r(b),
      ~p(a) V q(a), ~p(a) V q(b),
      ~p(b) V q(a), ~p(b) V q(b)}
  Minimal Support節
      {r(a), r(b), ~p(a), ~p(b), ~q(b)}

②  COMPの場合
  Prime Implicant節
      {q(X) V r(Y), ~p(X2) V q(Y2)}
  Minimal Support節
      {r(Y), ~p(X2), ~q(b)}
```

図３．３１　述語を対象としたコンパイル法ＣＯＭＰの効果（４）
節中の項：定数と変数が混在，　節中の変数：２種類

3．5．4　述語論理を対象とした仮説推論システムの評価

（1）評価方法

述語論理を対象としたシステムを，Ｐｒｏｌｏｇで構築した．システムをＰｒｏｌｏｇを用いて開発したのは，単一化代入，改名代入等で必要となる，変数を変数に代入する処理について，Ｐｒｏｌｏｇがこれを可能とする機能を持っているためである．この述語論理を対象としたシステムを評価するために，述語を対象としたことによる負荷の変化，そしてPrime Implicant節およびMinimal Support節の生成のためのＣＰＵ時間に関する，ＣＯＭＰと，instantiation法との比較を行なった．

（2）述語を対象としたことによる負荷の変化

述語論理を対象としたことにより，リテラルが１つ以上の項を含むことになったことによる，システムの負荷の増大の傾向を見るため，項を持たない（項の数が０に相当する）乱数を用いて発生させた，命題論理節集合を対象としたシステムと，リテラル中の項の数が１と２である述語論理節集合を対象としたシステムとについて，また実際の知識を表現した節集合について，Prime Implicant節の 導出時間の比較を行なった（表３．８）．なお，リテラル中の項の数が２のものについては，変数を共有する場合と，共有しない場合について行なった．比較に用いた節の形式を次に示す．

命題論理節	・・・	p V ～q,
述語論理節，項数１	・・・	p（X） V ～q（Y）
述語論理節，項数２（変数共有）	・・・	p（X，Y） V ～q（Y，X）
述語論理節，項数２（共有せず）	・・・	p（X，Y） V ～q（A，B）

表３．８に示すように，命題論理を対象としたシステムを，述語論理を対象とするように拡張したことにより，ＣＰＵ時間が多少増加する傾向があった．しかしながら，その増加傾向はそれほど著しいものではなく，比較した例においては，乱数による節集合の場合で，最大約１．５倍，知識による節集合の場合で，最大約１．１倍であった．

（3）代入方法の比較

instantiation法 とＣＯＭＰとを，ＣＰＵ時間，および公理節の数について比較を行なった．比較に用いた節集合は，命題論理について用いたものと同じ，乱数を用いて発生させた述語論理節集合である．instantiation法においては，次の方法によりinstantiationと，Prime Implicant節の導出を行なった．

① 公理節集合中および質問節中の定数を全部探す．

② 変数を含む論理式中の変数に定数を代入し，可能なすべての基礎代入を行なう．

③ ＯＣＣ法により，Prime Implicant節の導出を行なう．

④ 質問節に対しMinimal Support節の生成を行なう．

表３．８　述語を対象としたことによる仮説推論システムの負荷の変化

	Number of Clauses and Literals	ＣＰＵ Ｔｉｍｅ (Unit : sec)			
		Proposition	Predicate		
			1 term	2 terms	
				no overlap	overlap
Prime Implicant Clauses	16／4	6.0	8.3	8.5	8.7
	16／8	66.3	81.2	82.7	86.0
	16／16	477.0	565.1	579.0	612.6
Minimal Support Clauses	16／4	0.4	0.3	0.3	0.4
	16／8	1.2	1.2	1.3	0.9
	16／16	9.0	9.6	9.9	10.2

乱数により発生させた公理節集合を対象とした，Prime Implicant節のおよび Minimal Support節 の生成時間を，表３．９に示す．実際の知識を表現した公理節集合を対象とした Prime Implicant節の生成時間を表３．１０に，Prime Implicant節の数とＣＰＵ時間の関係を表３．１１に示す．また，Minimal Support節の生成時間を表３．１２に示す．

乱数により発生させた知識集合を対象とした場合，節中の変数の数が１で，質問節中の定数も１種類の場合には，二つの方法の間には，ほとんど差は見られなかった．しかしながら，節中の変数の数が２で，質問節中の定数も２種類を許容する場合には，公理節の数が４倍の６４となり，Prime Implicant節 を生成するためには，乱数を用いていることにより，６４の公理節の中のリテラルについて，ほとんどすべての組み合わせを計算する必要があるため，７０００ｓｅｃ以内では，Prime Implicant節 を求めることができなかった．（表３．９）

実際の知識を表現した節集合を対象とした場合，乱数の場合ほど極端にはならなかったが，ＣＯＭＰの方が単純なinstantiation法に対して，Prime Implicant節の生成に関しては，約１０－４５倍高速であった．知識を用いた公理節においては，例えば自船が自船自身を避けるといった，変数の間に有り得ない組み合わせが存在しているため，組み合わせに制限を設けた例についても，Prime Implicant節 の生成時間の計測を行なった．この場合，公理節の数が少なくなったことに比例して，ＣＰＵ時間も小さくなり，高速化の程度は，Prime Implicant節 の生成に関しては，約３－７倍となった（表３．１０）．

（４）Prime Implicant節の数と導出時間

あらかじめすべてのinstantiationを行なうinstantiation法では，導出に利用されない節も含めて多数の節が生成されるため，Prime Implicant 節導出のＣＰＵ時間が大きくなり，Prime Implicant 節の数も多くなった．これに対し述語を対象としたＣＯＭＰでは，変数を含んだ項を許容しているため，Prime Implicant 節の数は多くはならず，そのため導出に要するＣＰＵ時間も大きくはならなかった（表３．１１）．

現実の知識の節集合を述語論理を用いて表現すると，命題論理と比較した場合，公理節数の総数が減少し，これに比例して同じ意味を持つ Prime Implicant節を導出するＣＰＵ時間が減少した．今回用いた例においては，基礎項の数は２であるが，たとえば命題論理において節数が５０で表現される知識を，述語論理ではこれより少ない，６割の３１の節で表現することが可能であり，この時には Prime Implicant節を導出する時間は約４割と

表3．9　仮説推論システムにおける代入方法の比較
（乱数により発生させた公理節集合の場合）

			COMP		Instantiation	
	Number of Axiom Clauses		16	16	16	64
	Number of Variables in a Clause		1	2	1	2
Number of Constant			1	2	1	2
Prime Implicant Clauses (Unit : sec)			82.9	82.4	81.3	above 7000
Minimal Support Clause (sec)	Query	Unit Clause	1.2	1.3	1.2	x
		2 Literals Clause	—	13.9	—	x

表3．10　仮説推論システムにおける Prime Implicant節の生成時間
（実際の知識集合を用いた公理節集合の場合）

	Proposition		Predicate					
			COMP		OCC+Simple Instantiation		OCC+Restricted Instantiation	
	Number of Axiom Clauses	CPU Time (sec)	Number of Axiom Clauses	CPU Time (sec)	Number of Axiom Clauses	CPU Time (sec)	Number of Axiom Clauses	CPU Time (sec)
Prime Implicant Clauses	12	8.1	11	9.4	44	78.9	22	25.9
	32	143.9	22	70.6	88	3625.2	44	540.2
	50	257.5	31	100.9	124	4447.7	62	755.0

なった.

命題論理を対象としたシステムでは，当初の節数が２倍になると時間は約６倍に，節数が４倍になると時間は約２０倍になり[23]，述語論理を対象としたシステムでは，当初の節数が２倍になると時間は約７倍に，節数が４倍になると時間は約４５倍になることがある．また，命題論理を対象としたシステムでは，当初のリテラル数が２倍になると時間は約１０倍に，リテラル数が４倍になると時間は約８０倍になる[23]ことから，述語論理を用いて，節の数およびリテラルの数を減少させることは，Prime Implicant節 を導出するＣＰＵ時間を著しく減少させる効果があるということができる.

（５）Minimal Support節の生成時間

Prime Implicant節について検討したものと同じ節集合について，Minimal Support節生成時間の比較を行なった．実際の知識を表現した節集合を対象とした場合，ＣＯＭＰの方が単純な instantiation法に対して，２－１５倍高速であった．組み合わせに制限を設けた例については，公理節の数が少なくなったことに比例して，ＣＰＵ時間も小さくなり，高速化の程度は約１－６倍となった（表３．１２）.

instantiation法では，多数のPrime Implicant節が生成されるため，Minimal Support節生成のためのＣＰＵ時間が大きくなるとともに，instantiationを行なっているため，Minimal Support節の数も多くなった．これに対し，あらかじめ定数によるinstantiationを行なわないＣＯＭＰでは，変数を含んだ項を許容しているため，Prime Implicant節 の数は多くはならず，そのため Minimal Support節生成に要するＣＰＵ時間も大きくはならなかった.

しかしながら，instantiation法の結果に顕著に見られるように，PrimeImplicant節 の数が大きくなった時に，Minimal Support節 の生成時間が，急激に増大する傾向が見られる．これはシステムを汎用のものとしたために，常に節の包含関係の検討を行なうことによるものである．質問節がユニット節に限定される場合，包含関係の検討は不必要であり，さらに高速化を図ることが可能である.

表3．１１　Prime Implicant節の数と生成時間

Proposition		Predicate					
		COMP		OCC＋Simple Instantiation		OCC+Restricted Instantiation	
Number of P.I. Clauses	CPU Time (sec)	Number of P.I. Clauses	CPU Time (sec)	Number of P.I. Clauses	CPU Time (sec)	Number of P.I. Clauses	CPU Time (sec)
１９	8.1	１８	9.4	７４	78.9	３８	25.9
１３３	143.9	８９	70.6	６０８	3625.2	２９０	540.2
１７７	257.5	１０９	100.9	８５８	4447.7	３５０	755.0

表3．１２　仮説推論システムにおける Minimal Support節の生成時間
（実際の知識集合を用いた公理節集合の場合）

	Proposition		Predicate					
			COMP		OCC＋Simple Instantiation		OCC+Restricted Instantiation	
	Number of Axiom Clauses	CPU Time (sec)	Number of Axiom Clauses	CPU Time (sec)	Number of Axiom Clauses	CPU Time (sec)	Number of Axiom Clauses	CPU Time (sec)
Minimal Support Clauses	１２	0.7	１１	0.7	４４	1.3	２２	0.9
	３２	9.0	２２	2.7	８８	44.5	４４	16.3
	５０	10.9	３１	3.1	１２４	49.5	６２	18.8

3．6　まとめ

知識ベースの高速コンパイルを可能とする，Prime Implicant節 の高速導出方法（順序づけ節コンセンサス法）の開発，およびコンパイルした知識を利用した Minimal Support 節の高速生成法を実現することができた．このコンパイル技術を応用して，命題論理を対象とした仮説推論システムを構築した結果，Minimal Support 節として空節が生成されない場合には，コンパイルしない場合と比較しても，また単純な幅優先戦略による推論と比較しても，推論の大幅な高速化を図ることができた．この傾向は特に質問節がユニット節の場合に顕著であった．

また，述語を対象として順序づけ節コンセンサス法を適用したＣＯＭＰ（ＣＯmpiling Method for Ｐredicate）と，単純に instantiateする方法とを比較した結果，問題空間の拡大を抑制した，述語論理を対象とした知識ベースの高速コンパイルが可能であり，また高速で Minimal Support節を生成することが可能であった．

このシステムにおいては，節中のリテラルの照合や，包含関係の検討が重要であり，この照合等のために時間がかかっていることから，節データの表現形式を照合に便利な形式にすることは，高速化を図るために検討しなければならない技術である．また，今回は双形リテラルだけに注目したが，補リテラルが存在しない単形リテラルを含む二つの節から導出される節は，双形リテラルから導出される節を包含しない場合，単形リテラルの増加に伴い単調に増加し，空節を導出することがない，という単形リテラルの特性を利用した高速化も，検討するに値する技術であると思われる．

公理集合が大きくなった場合には，全ての Prime Implicant節をあらかじめ導出することはコストが大きいため，de KleerがＡＴＭＳとの類似性を指摘した[45]，ＣＳＰのように部分を対象とすることや，導出との共通性が見出されてきている Groebner Baseで行なわれているように[49]，推論の道筋に沿って Prime Implicant節を導出する方法も，高速化のために検討する必要がある．実用的な仮説推論システムを，Reiter他の提案したＣＭＳの枠組を利用して実現するためには，これらの高速化技術を実現し，仮説推論システムの応用性を高めることが大きな課題である．

3章　参考文献

[1] 上野春樹ほか，「エキスパート・システムの課題」，情報処理，Vol.28，No.2，１９８７

[2] 平井吉光，「わが国におけるエキスパート・システムの開発状況」，人工知能学会誌，Vol.2，No.1，１９８７

[3] 石塚　満：特集「高次人工知能に向けてのパラダイム」について，人工知能学会誌，Vol.2，No.1，pp.4-5，１９８７

[4] 石塚　満：不完全な知識の操作による次世代知識ベース・システムへのアプローチ，人工知能学会誌，Vol.3，No.5，pp.22-32，１９８８

[5] 松田哲史，石塚　満：仮設推論システムの拡張知識表現と概念学習機構，人工知能学会誌，Vol.3，No.1，pp.94-102，１９８８

[6] 鶴田三郎，石塚　満：不完全な知識が係わる知識システムへのＣＭＳの応用に関する基礎研究，人工知能学会全国大会（第３回）論文集，pp.179-182，１９８９

[7] 石塚　満：Dempster & Shaferの確率理論，電子通信学会誌，Vol.66，No.9，pp.900-903，１９８３

[8] 金井直樹，石塚　満：ファジィ論理を組み込んだProlog-ELF，情報処理学会（知識工学と人工知能）研究会資料，No.34-4，pp.1-8，１９８４

[9] 井上克巳：導出を用いた仮説探索，情報処理学会第３５回全国大会論文集1M-2，pp.1567-1568，１９８７

[10] 飛鳥井正道，村田真人，森沢秀一，浅野俊昭：エキスパートシステム構築ツールCHORUS(2)　－仮説推論機構－，情報処理学会第３７回全国大会論文集5G-7，pp.1218-1219，１９８８

[11] 国藤　進，鶴巻宏治，古川康一：仮説推論機構の一実現法，人工知能学会誌，Vol.1，No.3，pp.228-237，１９８６

[12] 藤井裕一，沢本　潤，竹之内博夫，滝　寛和，大木　優：第５研究室の研究状況，ＩＣＯＴジャーナル，No.15，１９８８

[13] Doyle, J.：A Truth Maintenance System, Artificial Intelligence 12, pp.231-272，１９７９

[14] de Kleer, J. : An Assumption-based TMS, Artificial Intelligence 28, pp.127-162, １９８６

[15] Reiter, R., de Kleer, J. : Foundations of Assumption-based Truth Maintenance Systems : Preliminary Report, Proc.AAAI-87, pp.183-188, １９８７

[16] McDermott, D. and Doyle, J. : Non-monotonic logic I, Artificial Intelligence, Vol.13, pp.41-72, １９８０

[17] Reiter, R. : A Logic for Default Reasoning, Artificial Intelligence, Vol.13, pp.81-132, １９８０

[18] McCarthy, J. : Circumscription : A Form of Non-monotonic formalizms, Artificial Intelligence, Vol.13, pp.27-39, １９８０

[19] 人工知能学会：人工知能ハンドブック，オーム社，１９９０

[20] 矢田光治：ＡＩ総覧，フジ・テクノシステム／エス・ディ・シー，１９８７

[21] 伊藤史朗，石塚　満：論理的制約利用による高速仮説推論システム，情報処理学会研究報告，90-AI-70，情処研報，Vol.90，No.32，１９９０

[22] 門脇修司，牧野俊朗，石塚　満：ＲＥＴＥアルゴリズム利用による高速無矛盾性検査機構を備えた仮説推論システム，情報処理学会第４０回全国大会論文集6C-2，pp.184-185，１９９０

[23] 鶴田三郎，石塚　満：命題論理知識ベースのコンパイル法，情報処理学会研究報告，90-AI-70，情処研報，Vol.90，No.32，１９９０

[24] 鶴田三郎，石塚　満：発想的命題論理知識ベースのコンパイル法，人工知能学会誌，１９９０（投稿中）

[25] 鶴田三郎，石塚　満：述語論理知識ベースのコンパイル法，第４回人工知能学会全国大会，１９９０（投稿中）

[26] 青山正弘，石塚　満：”仮説の重み利用による高速化機能を備えた仮説推論システム”，人工知能学会全国大会（第２回）論文集，pp57-60，１９８８

[27] 山口高平，溝口理一郎，田岡直樹，小高浩，野村康雄，角所　収：深い知識に基づく知識コンパイラの基本設計，人工知能学会誌，Vol.2，No.3，pp.333-340，１９８７

[28] 井上克己, 太田好彦：仮説推論システムAPRICOT/0における知識コンパイル, 人工知能学会研究資料, SIG-KBS-8805, pp.51-60, １９８９

[29] 林　秀吉, 山口高平, 溝口理一郎, 角所　収：深い知識に基づく知識コンパイラ, 人工知能学会研究資料, SIG-KBS-8805, pp.61-70 , １９８９

[30] Freuder, E.C. : Synthesizing Constraint Expressions, Communications of the ACM, Vol.21, No.11, pp.958-966, １９７８

[31] Mackworth, A.K. : Constraint Satisfaction, Encyclopedia of Artificial Intelligence, John Wiley and Son, pp.205-211, １９８７

[32] 折原良平, 高柳孝司, 長尾加寿恵, 菅野成一郎, 楠井洋一：発想支援システム「知恵の泉」における類推機構の概要, 情報処理学会「学習のパラダイムとその応用」シンポジウム, pp.89-98, １９８９

[33] 和多田作一郎：ＡＩの基礎を知る事典, 実務教育出版, １９８６

[34] Wos, L., Overbeek, R., Lusk, E. and Boyle J. : Automated Reasoning Introduction and Applications, Prentice-Hall Inc., １９８４

[35] 上野春樹, 石塚　満：知識の表現と利用, オーム社, １９８７

[36] 高井宏幸, 竹中俊夫, 長谷川健介, 森永智昭：実用自動制御ポケットブック, オーム社, １９７４

[37] Quine, W.V. : A Way to Simplify Truth Functions, Amer. Math. Monthly, No.62, pp.627-631, １９５５

[38] McCluskey, E.J.Jr. : Minimization of Boolean Functions, Bell System Tech. J., Vol.35, No.6, pp.1417-1444, １９５６

[39] Slagle, J.R., Chang, C.L. and Lee, R.C. : A New Argorithm for Generating Prime Implicants, IEEE Trans., Computers, C-19, No.4, pp.304-310, １９７０

[40] 上林弥彦, 岡田康治, 矢島脩三：節展開法を用いた　論理関数の主項の生成, 電子通信学会論文誌, J62-D, No.2, pp.89-96, １９７９

[41] Tison, P. : Generalization of Consensus Theory and Application to the Minimization of Boolean Functions, IEEE Trans. Electronic Computers, EC-16, No.4, pp.446-456, １９６７

[42] Morreale, E. : Recursive Operators for Prime Implicant and Irredundant Normal Form Determination, IEEE Trans. Computers, C-19, No.6, pp.504-509, 1970

[43] 松村尚志, 稲石正明, 鶴田三郎, 今津隼馬, 杉崎昭生：船舶航行エキスパートシステムの基礎研究 －システム開発過程における評価について－, 日本航海学会論文集, No.81, pp.31-38, 1989

[44] Matsumura, H., Inaishi, M., Tsuruta, S., Imazu, H. and Sugisaki M. A. : Basic Research on an Expert System for Navigation at Sea, International Federation of Automatic Control Workshop, CAMS'89, pp.37-43, 1989

[45] de Kleer, J. : A Comparison of ATMS and CSP Techniques, IJCAI-89, pp.290-296, 1989

[46] 鶴田三郎, 松村尚志, 稲石正明, 今津隼馬, 杉崎昭生：船舶航行エキスパートシステムの基礎研究 －衝突回避エキスパートシステム－, 日本航海学会論文集, No.77, pp.133-139, 1987

[47] Sugisaki, M.A., Imazu, H., Tsuruta, S., Inaishi, M. and Matsumura, H. : An Expert System for Coping with Collision Avoidance and the EX POST FACTO at Sea, International Navigation Congress, IAIN1988, pp.223-230, 1988

[48] 有川節夫, 原口 誠：述語論理と論理プログラミング, オーム社, 1988

[49] 坂井 公, 相場 亮：CAL：制約論理プログラミングの理論と実例, 電子情報通信学会研究資料, SS87-28, pp.9-17, 1988

4章　船舶航行エキスパートシステムと仮説推論システム

4．1　船舶航行エキスパートシステムへの応用

4．1．1　仮説推論システムの船舶航行エキスパートシステムの知識ベースへの応用

2章においては，船舶航行エキスパートシステムの3つのサブシステムを開発し，システムの妥当性，実時間性および操作性について検討を行なった．そして，そのサブシステムの開発成果を基に，船舶航行エキスパートシステムの開発における課題として，次の5項目を指摘した．

①知識ベースの充実

②管理機構の開発

③推論の高速化

④柔軟な推論

⑤ユーザインタフェースの充実

3章においては，この課題の中で最初に検討すべき課題として，知識管理と推論の高速化を取り上げ，知識ベースの高速コンパイル手法を開発し，このコンパイル手法を応用した仮説推論システムを構築し，その評価を行なった．さらに，対象を述語論理に広げ，システムの応用性を高めた．そして，次の3項目を実用システムに向けて開発すべき課題として指摘した．

①単形リテラルの利用

②部分コンパイルの実現と利用

③推論の道筋に沿ったコンパイル法の実現と利用

この章においては，3章で開発構築した仮説推論システム[1][2][3]を，船舶航行エキスパートシステムの知識ベースに応用することを考えることにする．3章においても，実際の知識ベースを用いて，命題論理を対象とした仮説推論システムや，述語論理を対象とした仮説推論システムの構築・評価を行なってきたが，評価のために用いた知識ベースは，

現実の知識ベースを簡略化したものであった．現実の知識ベースを対象とするためには，簡略化した知識ベースを対象とした場合には特に行なわなかった，知識ベースの特徴や実際の運用方法に配慮することが必要となる．

２章で開発した３つのサブエキスパートシステムの内，衝突回避エキスパートシステム[4]においては，他船の方位・距離といった，船上において取得が一般に容易であるレーダ情報を主として利用していたが，海上衝突予防法の適用を考える場合，レーダにより取得が可能でない視覚情報や，他船の方位・距離に関係した視覚情報を利用することが必要となる．しかしながら，遠距離においては取得が不可能な，もしくは間違える可能性が少なからず存在する視覚情報があることから，法令の規定の適用を厳密に検討するためには，他船が十分に接近するまで待つ必要がある．現実の場においては，確実な判断を行うために十分接近することは危険であり，知識が不備であっても判断を行なうことが必要である．

仮説推論は，常には正しいとは限らない知識や，欠落部分を含む知識のような，不備な知識を対象とするための枠組みであり，衝突回避エキスパートシステムの知識ベースの中の海上衝突予防法に関する部分は，仮説推論システムの応用を検討する対象とするのに適当であると思われる．

また，海上衝突予防法は，４２条からなる法律であるが，衝突を回避するための航行方法（航法）に関する規則は，第４条から第１９条までの計１６条しか規定されていない．規定されていない状況に関する第３８条，第３９条を入れても，高々１８条しか規定が存在せず，知識ベースの規模はそれほど大きくならないこと，そして知識が完結していることから，仮説推論システムの応用を検討する対象として取り上げることとする．

４．１．２　海上衝突予防法を基にした知識ベースの特徴

海上衝突予防法は，航法として，「あらゆる視界の状態における船舶の航法」，「互いに他の船舶の視野の内にある船舶の航法」，および「視界制限状態における船舶の航法」を規定している．この内，２船が互いに他の船舶の視野の内にある場合の規定は，船種，衝突のおそれ，および遭遇状況により，適用航法が決定され，その適用された航法規定により義務の種類が決定され，さらに衝突を避けるための動作が限定されるという構造をしている．

互いに他の船舶の視野の内にある場合について，船種，衝突のおそれの有無と，適用航法，義務，動作との概略の関係を図４．１に示す．例えば「動力船」を例にとると，「帆船」の航法は適用されないものの，「追越し船」，「行会い船」，「横切り船」の航法規定が適用され，さらにはこの適用規定により，他の船舶の針路を避けなければならない義務（避航義務）が発生する場合や，反対に針路・速力を保持する義務（保持義務）が発生する場合がある．しかしながら「追越し船」，「行会い船」，「横切り船」の航法規定が適用されない場合にも，「避航船」としての義務や「保持船」としての義務が発生する場合や，直ちに衝突を避けるための動作（避航動作）を取らなければならない場合もある．

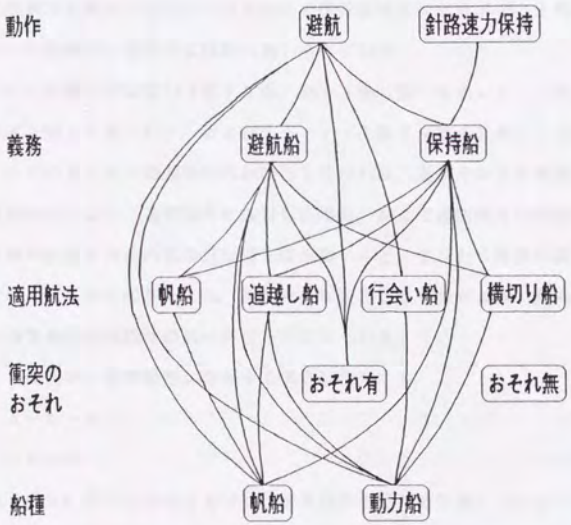


図４．１　海上衝突予防法の航法規定の概略の構造

海上衝突予防法を基にした知識ベースの特徴は，次のようになる．

① 船舶が自船の動作を決定するための経路が，図4．1に示すように，多数あることを，挙げることができる．すなわち，ある状況において適用される規定を，厳密に定義するためには，他の規定が適用されないことを，述べる必要がある場合がある．間近に接近しなければ，適用を検討できない規定があることを考えると，他船と接近するまで，規定の適用および行動の決定ができない事態が発生する可能性がある．船舶を安全に航行させるという面から見た場合，ここでいう厳密な定義を用いることは，現実的な運用方法とはいうことができない．

② 規定の複雑さ，不明確さを挙げることができる．「できるかぎり」，「やむを得ない場合を除き」，「・・・・の限りではない」，「ありえる」，「確かめることができない場合」，「・・・・にかかわらず」，「この法律の規定によらないことができる」，「船員の常務としてもしくはその時の特殊な状況により必要とされる注意をする」といった表現が，規定中に頻繁に用いられている．

「ただしこの限りではない（第１２条，第１４条，第１５条）」，「同項の規定にかかわらず（第１７条　）」，および「・・・・を除く（第１８条）」という表現の意味は，いずれもこれらの適用除外が該当しなければ，通常そのまま規定が適用されることを意味しており，適用除外に該当した場合，始めて適用除外の内容が有効になる．適用除外に示された内容の方が優先度が高いこと，すなわち複数の規定の優先度を示している．しかしながらまた，適用除外に該当しない方がより一般的であること，すなわち通常適用の可能性の高い規定を示してもいる．

次のような例が，適用除外が存在する場合に相当する．

$a \wedge \sim b \rightarrow c$,

$a \wedge b \rightarrow d$

この場合，｛a｝が入力されても｛c｝が導出されるためには，｛～b｝か｛～d｝が必要である．現実の場面においては，相手船を発見した時点から判断を行なう必要がある．すべてのデータが収集できる，一般的には接近した時点で判断を開始するのは危険であり，完全なデータが揃わない内に判断を行なう必要がある．この場合，

$a \wedge \sim b \rightarrow c$

の代わりに

$a \rightarrow c$

を用いる方が，より現実的である．しかしながら，この判断はあくまでも仮説であり，{b}という事実が確認された場合，矛盾するため撤回されなければならない．{b}という入力は最後までなされない場合もあり，仮説を用いて推論する必要性がある．

「確かめることができない場合（第１２条，第１３条，第１４条）」という表現の意味は，確かめることができるまでは，その規定が適用されることを意味している．

$$a \rightarrow b,\ \sim a \rightarrow c$$

である時に，{a}もしくは{～a}が事実として入力されるまで，

$$\rightarrow b$$

を用いる場合に相当する．{a}もしくは{～a}が確かめられた時点で，当該規定は適用されなくなる．規定の適用は確認されるまで有効な仮説である．

「やむを得ない場合を除き（第１５条，第１７条，第１８条）」，および「できる限り（第１８条）」という表現の意味は，当該規定が優先的に適用されることを意味している．しかしながら言葉の意味から，特殊な事情がある場合，規定は適用されないことになる．この意味でこの規定の適用も仮説に相当する．

③ 互いに排他的である内容が多いことを挙げることができる．図４．１において，「おそれ有」，「おそれ無」といった，横並びの内容は互いに排他的であり，一方もしくは，その並びの内の一つが成立した場合，他のものは成立しない．次のような例が，この場合に相当する．

$$a \rightarrow c,\ b \rightarrow d,$$

であり，かつ

$$c \rightarrow \sim d,\ d \rightarrow \sim c.$$

これらの公理を節で表現した場合，３番目の公理と４番目の公理は，１つの節で表現することが可能であり（{～c V～d}），公理節は３つとなるが，公理節からはリテラル数の等しい３つの定理節が導出され，節集合は大きくなる．排他的関係が多数存在する場合，節集合が大きくなる可能性が高い．

④ この他の特徴として，現実の場においては，観測結果はルールの形ではなく，事実の形で通常，時間経過に従って，一つずつ入力されることを挙げることができる．３章で開発した仮説推論システムは，一般の節を対象としていたが，質問節をユニット節に限定することにより，推論の高速化を図ることができる可能性がある．

4．1．3　仮説推論システムを応用するに当たっての課題

すべての観測結果が入力される前であっても，衝突を避け安全に航行するために，欠落部分を含む観測結果から，自船の行動を決定する必要があるし，また常には正しいとは限らない規定が存在する問題領域であり，仮説推論を応用する問題としては適当な問題領域である．しかしながら，何を仮説とし，その仮説をどのように表現するかという問題がある．

質問節がユニット節に限定できることから，コンパイル終了後は，ユニット導出を利用することにより，Minimal Support節の生成，矛盾管理の高速化を図ることが可能である．3章の汎用仮説推論システムにおいても，ユニット節に対しては，Minimal Support節を高速で生成することが可能であったが，質問節がユニット節に限定されたことを有効に利用することが必要である．

除外規定を含む公理（例：｛ａ∧～ｂ∧～ｃ∧～ｄ∧～ｅ→ｆ｝）は，節中のリテラルの数が多いため，コンパイルにより定理節の数が多くなる可能性が高い．一つの節中のリテラル数の最大をｎとし，正リテラル，負リテラルの合計を２ｍとすると，最大次の組み合わせが存在する可能性がある．

$$2m + 2m(2m-2)/2! + \cdots + \{2m(2m-2)\cdots(2m-2n+2)/n!\}$$

組み合わせ数は，節中のリテラル数の最大が大きくなると，急激に増加する傾向にあり，導出を行うと節中のリテラル数が，増加する傾向にあることから，定理節の増加をできるだけ抑制した，除外規定を含む公理の取扱いを検討する必要がある．

また排他的関係を表現した公理が節集合中に存在すると，存在しない場合には単形リテラルであったリテラルが，双形リテラルになる場合があり，組み合わせ数が増大することになる．衝突回避に関する公理のリテラルの間に，排他的関係が多く存在することから，コンパイルにより定理節の数が多くなる可能性が高い．

プリコンパイルの高速化を図るためには，また，真理管理の高速化を図るためには，公理・定理節集合の増大を抑制することが必要である．そのためには，一つの節中のリテラルの最大の数を抑制すること，文献［５］［６］において指摘されている，部分コンパイルを検討すること，文献［７］で行なわれているような，推論の道筋に沿ってコンパイル

を行うこと，さらに文献［1］［2］で指摘した，単形リテラルの性質を利用する方法を検討することが考えられる．

一つの節中のリテラルの最大の数を抑制するためには，そして述語論理を対象とした仮説推論システムを応用するためには，質問節がユニット節に限定されたことの有効利用，公理および仮説の表現方法，仮説を含めたリテラルの種類，変数とする項目について検討する必要がある．

部分コンパイルとしては，あるリテラルを含む部分をブラックボックス化し，コンパイル終了後，そのリテラルを含む節を，真理管理の対象としない方法や，一部分についてのみコンパイルし，他については推論の道筋に沿ってコンパイルを行ってゆく方法，この他に全体をいくつかに分割し，それぞれについてコンパイルする方法等が考えられる．

あるリテラルを含む部分を対象としない，部分コンパイルにおいては，どのような理由により，何をそのリテラルとするかという問題がある．推論の道筋に沿ってコンパイルを行なう方法は，コンパイルする計算量が少なくなる利点があり，質問節がユニット節であることから，ユニット導出を有効に利用することが可能である．しかしながら，矛盾が検出された時，どこまで戻るのか，どの矛盾を削除するのかといった，矛盾への対処方法を検討することが必要となる．

3章の仮説推論システムにおいては，主として双形リテラルだけに注目したが，補リテラルが存在しない単形リテラルを含む節からの導出節は，空節となることがない，という単形リテラルの特性を利用した高速化も，検討するに値する技術であると思われる．すなわち，真理管理においては，導出節が空節になるかどうかが最大の関心事であり，これに関係しない節は，あらかじめ節集合から除外することが可能であり，節集合の増加を抑制するために効果がある．

そこで，ユニット導出を利用した手法，単形リテラルの利用方法，および部分コンパイルの手法について，次節（4．2）で検討することにする．また，公理や仮説の表現方法，および仮説推論システムにおける矛盾検出時の対処方法については，4．3において検討することにする．

4．2　コンパイル手法の検討

4．2．1　ユニット導出を利用した方法

親節から相補リテラルの一方のリテラルを除いた節が，他の親節から同様にリテラルを除いた節を包含する（もしくは両方の節が等しくなる）場合，リテラル数の多い方の親節よりも，リテラルの数が1少ない導出節を生成することができる．ユニット導出（unit resolution）はこの特殊な例であり，ユニット節については導出リテラルを除いた部分節は空節であることから，包含関係の検討が不要であり，空節の導出を目指して導出を行なうことが可能である．

ユニット導出を行なうことにより，導出節のリテラル数は必ず減少し，この導出節によりユニット節以外の親節の削除が可能になり，さらにその他の節も削除が可能となる場合があることから，ユニット導出を先行させることにより，高速化を図ることができる可能性がある．

しかしながら次に示すようなケースにおいては，必ずしも高速になるとは限らない．

$\{pVtVa_1,\ pVtVa_2,\ \cdots,\ pVtVa_n,\ \sim p,$

$rVt,\ \sim rVt\}$

このケースについて見ると，ユニット導出の場合，リテラル｛p｝についてn回，｛r｝について1回の計n＋1回の導出を行うことにより，Prime Implicant 節　｛～p｝，｛t｝を導出することが可能であるが，先に｛r｝について導出を行った場合，リテラル｛r｝について1回の導出を行うことにより，Prime Implicant節を導出することが可能である．すなわちユニット導出による導出節が他のリテラルについての導出節を包含する場合，先行させることの効果が無くなる．

3章の命題・述語論理を対象とした仮説推論システムにおいては，ユニット導出を，ＯＣＣ法（順序づけ節コンセンサス法）の第一ステップ，およびMinimal Support節の生成において利用している．Minimal Support節の生成において，質問節がユニット節である場合に，Prime Implicant節と質問節の包含関係から，Minimal Support節を生成する方法は，ユニット導出を応用した例である．

船舶の衝突回避を対象としたシステムにおいて，観測結果を質問節として，公理・定理

節集合に付加してゆく場合，質問節がユニット節でしかないことから，ユニット導出を利用することが可能である．また，観測結果が一つずつ入力される度に，すなわち推論の道筋に沿って，コンパイルを行うことを検討の対象とすることも可能となる．非ユニット節も含めて対象とした仮説推論システムにおいては，Minimal Support節を生成する場合，生成された節の間で包含関係を検討することが必要であったが，質問節をユニット節に限定した場合，この過程を削除することが可能となる．すなわち高速で，コンパイルおよびMinimal Support節の生成を行なうことが可能となる．

4．2．2　ゴール，入力に注目した部分コンパイル

ゴール節，入力節がユニット節の時に，ゴール節，入力節に存在しないリテラルを含む節を，公理節集合から削除することを考える．すなわちこれらのリテラルを含む節を，ブラックボックスの中に入れ，外部とはゴール節，入力節のみを介して情報のやりとりをするシステムを考える．

ここでは，ゴール節，入力節を「入出力節」，入出力節に含まれているリテラルを「入出力リテラル」，入出力リテラル以外のリテラルを「非入出力リテラル」，非入出力リテラルを含む節を「中間節」と呼ぶことにする．言い替えると，この項では，Prime Implicant節の増大の抑制を目的として，中間節を削除する部分コンパイルを考える．

船舶の衝突回避を対象とする問題領域において，ゴールが公理節集合から証明できない場合，ゴールが証明されるためには，ゴールを質問節とした Minimal Support節が，入力の否定のリテラルの選言から構成される節の形をとることが必要である．言い替えると，非入出力リテラルを含む節からは，非入出力リテラルが導出によって削除されることがないため，ゴール節を証明することができない．

ゴール節の証明可能性を検討する場合，非入出力リテラルである単形リテラルを含む節は，公理節集合から直ちに削除することが可能である．また双形リテラルの形をした非入出力リテラルについては，導出終了後そのリテラルを含む節を，公理・定理節集合から削除することが可能であり，Prime Implicant節集合の増大を抑制することが可能である．

公理・定理節の数の最大は，公理節集合中に含まれるリテラルの数をmとすると，正リテラル，負リテラルの合計は２ｍとなり，次式で表現することができる．

$$
\begin{aligned}
&2m+2m(2m-2)/2!+\cdots \\
&\quad +\{2m(2m-2)\cdots(2m-2n+2)/n!\} \\
&\quad +\{2m(2m-2)\cdots 2/m!\} \\
&=(1+2)^m-1 \\
&\fallingdotseq 3^m
\end{aligned}
$$

リテラルの数を入出力リテラルに限定すると，リテラルの数はn（n＜m）に減少し，公理・定理節の数の最大は，3^nとなる．すなわち中間節を削除することにより，公理・定理節を$1/(3^{m-n})$に減少させることが可能であり，Prime Implicant節の導出の高速化が可能となる．当然，他の節を包含する節が存在するため，公理・定理節の最大数が，Prime Implicant節の最大数になるわけではないが，最大数のオーダーは示しているということができる．

4．2．3　部分コンパイルアルゴリズム

単形リテラルの形の非入出力リテラルに関しては，導出が行なわれないので，このリテラルは節の中から消えることがない．すなわちこのリテラルを含む節は，ゴールを証明するための Minimal Support節とはなることができないため，ＯＣＣ法を適用する前に公理節集合から削除することが可能であり，Prime Implicant節集合の増大を抑制することを目的として，公理節集合を小さくすることが可能である．

非入出力リテラルが正のユニット節であり，その非入出力リテラルの間に矛盾が存在する場合，それが入力リテラルから構成される節から導かれるものであれば，公理節をコンパイルすれば，入力リテラルの間の矛盾としても表現されるため，矛盾を表現している非入出力リテラルに関する節も，部分コンパイルにおいて削除することが可能である．

例えば，公理節集合を，

$\{\sim a \vee c,\ \sim b \vee d,\ \sim c \vee e,\ \sim d \vee f\}$

とし，矛盾が

$\{\sim c \vee \sim d\}$

で表現される例（図4．2）においては，

入力を

{a, b},

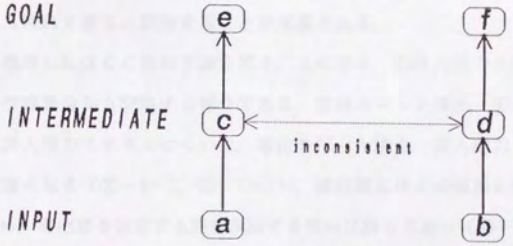
ゴールを

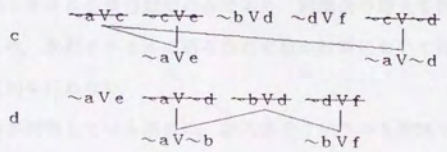
{e, f}

とすると，非入出力リテラルを含む節を削除しながら行ったコンパイル結果は，

{～a V～b, ～a V e, ～b V f}

となり，非入出力リテラル間の矛盾は，入力間の矛盾に変換されている．





Prime Implicant節　　{～a V～b, ～a V e, ～b V f}

図4．2　部分コンパイルの例

同様に，ゴール節が正のユニット節であり，入力節を公理節に付加することによりゴール節を証明可能な場合，ゴール間の矛盾は，入力間の矛盾に変換され，同じ矛盾を表現する冗長な節は不要となる．ゴール間の矛盾を，直接に表現している節は，ゴール節を２つ以上含んでいる節の中に有る．またゴール節を２つ以上含んでいる節は，単一のゴールを証明するという視点からは冗長な節であるため，削除することが可能である．すなわちコンパイル終了後，節集合には，Prime Implicant節として，入力節からのみ構成される節と，ゴール節をただ１つ含む節を残せば良いことになる．

ＯＣＣ法においては，一つのリテラルに関しては一回しか導出を行わないため，導出の終了したリテラルは，その後の導出によって節から削除されることがない．すなわち，双形リテラルの形の非入出力リテラルを含む節は，ゴールを証明するための節の候補から，導出が終了した時点で直ちに削除することが可能である．

この方法を適用したＯＣＣ法の手順を図４．３に示す．①は入出力に関係のない単形リテラルを，公理節集合から削除する部分である．②はユニット導出，④は順序づけ導出の部分である．非入出力リテラルについて，導出を行った場合，非入出力リテラルを含む親節の削除の可能となる（②－(ハ)，④－(ロ)）．導出節にはその導出に用いたリテラルは含まれないため，導出節を包含する節を削除する部分は同じであり（②－(ニ)，④－(ハ)），節集合が大きくなると包含関係を検討するための時間がかかることになる．③は順序づけの部分である．導出終了時②－(ニ)，④－(ハ)以前に，直ちに削除が可能なのは，導出に用いた非入出力リテラルを含む親節のみであり，節集合の増大を抑止する効果は，順序づけの方が高いため，双形リテラルの組み合わせ数の計算においては，入力リテラル，非入力リテラルの区別を行わない．

ゴール・入力が判明している場合の，非入出力リテラルを削除する，部分コンパイルアルゴリズムを，文献［８］で用いられている例について適用し，その概要を示す．

完全な知識を，

{aV～bV～c，aV～bV～d，bV～eV～f，eV～gV～h，eV～gV～i，jV～h，f}

制約を，

{～cV～g，～dV～h}

とし，

{a}，{c，d，g，h，i}

をそれぞれ，ゴール，入力とする知識ベースを考える．図４．４は，ネットワークを用い

① (イ) 非入出力単形リテラルを含む節を削除する.
(ロ) ①－(イ)の削除により新たに単形リテラルとなるリテラルを含め,
非入出力単形リテラルを含む節が無くなるまで①－(イ)を繰り返す.
② (イ) ユニット導出を行なう.
(ロ) ユニット節ではない方の, 導出の親節を削除する.
(ハ) 導出終了後, 非入力リテラルのユニット節は削除する.
(ニ) 導出節を包含する節を削除する.
(ホ) すべてのユニット節について②－(イ)(ロ)(ハ)(ニ)を繰り返す.
③ 双形リテラルについて, 順序づけを行なう.
④ (イ) 組み合わせの少ない順に導出を行なう.
(ロ) 導出終了後, 非入力リテラルについては, そのリテラルを含んでいる
節をすべて削除する.
(ハ) 導出節を包含する節を削除する.
(ニ) 順序づけを行なったすべての双形リテラルについて④－(イ)(ロ)(ハ)
を繰り返す.

図4．3　非入出力リテラルを削除する部分コンパイルアルゴリズム

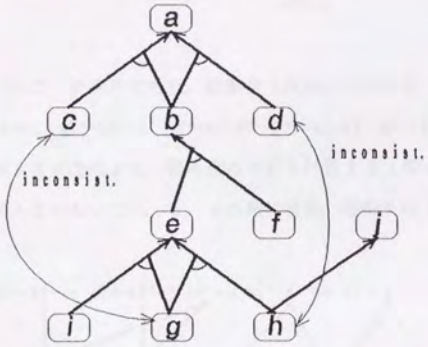


図4．4　検討例のネットワーク表現[8]

て表現したものである.

公理節集合をΣとすると,

Σ＝{aV～bV～c, aV～bV～d, bV～eV～f, eV～gV～h, eV～gV～i, jV～h,

f, ～cV～g, ～dV～h},

であり, 入出力リテラルは

{a, c, d, g, h, i},

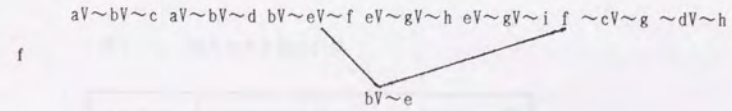
である. {j}は出力リテラルにもなりうるが, ゴール{a}とは無関係であり, この場合非入出力リテラルとなる. 結果として非入出力リテラルは

{b, e, f, j}

となる.

まずアルゴリズム①を適用し, リテラル{j}を含む節を削除する. {j}を含む節を削除した段階において, この{j}以外には該当するリテラルは無い.

次いでアルゴリズム②を適用し, {f}についてユニット導出を行う.

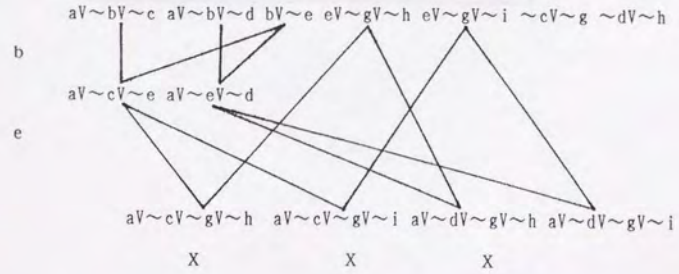


{f}は非入出力リテラルであり, 節集合は次のようになる.

{aV～bV～c, aV～bV～d, bV～e, eV～gV～h, eV～gV～i, ～cV～g, ～dV～h}

次いでアルゴリズム③により, 組み合わせ数を計算する（表4. 1）.

アルゴリズム④においては, b, eの順に導出, 削除を行う.



非入出力リテラル｛b, e｝を含む節は，④－(ハ)により節集合から削除され，また"X"は制約により削除され，その結果Prime Implicant節は，

{～cV～g, ～dV～h, aV～dV～gV～i}

となる．

導出回数は7回である．このPrime Implicant節の中で，ゴールを証明することができる節は{aV～dV～gV～i}だけである．ゴール・入力を特別に扱わない場合，導出回数は9回，Prime Implicant節は14である．この場合，ゴール｛a｝を含んでいる節は5つ存在するが，入力によってゴールを証明できる節は前述の場合と同じく，{aV～dV～gV～i}だけであり，結果は同じである．

ここで示したケースにおいては，ゴールおよび入力は，正のリテラルによって表現されるユニット節であったが，この方法は，負のリテラルが存在しても，また一般の節の形をした入力にも適用することは可能である．

表4．1　組み合わせ数の計算

リテラル	正リテラル	負リテラル	組み合わせ数
a	2	0	0
c	0	2	0
d	0	2	0
g	0	3	0
h	0	2	0
i	0	1	0
j	1	0	0
b	1	2	2
e	2	1	2

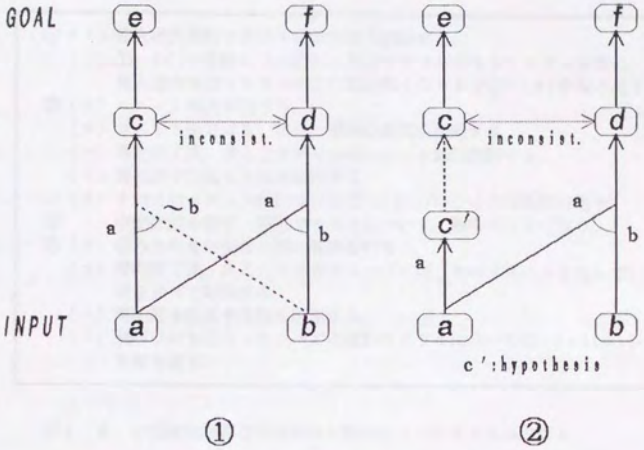
4．2．4　中間仮説が存在する場合の部分コンパイル法

入力が｛a，b｝，ゴールが｛e，f｝であり，矛盾｛～cV～d｝が存在する，図4．5の①の公理節集合ににについて，考えることとする．

衝突回避に関する現実の運用の場においては，図4．5の①の公理節集合における，｛～aVbVc｝の代わりに，｛～aVc｝を中間仮説とする，図4．5の②の公理節集合を用いるようなことが必要になる場合がある．これは，この公理節集合において，｛a｝と｛b｝の入力に，時間的な差がある場合や，｛b｝という入力が稀にしか発生しない場合が相当する．｛b｝の入力を待つことにより，より危険な状況になる可能性がある場合や，｛b｝という入力が無い可能性がある場合，常には正しいとは限らないが，現在の状況でさしあたり可能な判断を行なうことは，日常的に行なわれていることである．　この中間仮説は，ユニット節の仮説c'を用いると，｛～aVc'，～c'Vc｝の形で表現することが可能であり，ルールとしての中間仮説を，ユニット節の仮説c'を用いて表現することが可能である．この仮説c'についても導出を行なうと，仮説を選択したことになるため，プリコンパイルにおいては，あらかじめ仮説を選択することの無いように，仮説のリテラルについては導出を行なわないことが必要となる．仮説リテラルを入出力リテラルと同様に扱い，プリコンパイルでは入出力リテラルや仮説のリテラルを除いたリテラルについて導出を行なう必要がある．

図4．5の②において，部分コンパイルを用いたプリコンパイルを行なうと，リテラルdは非入出力リテラルであるため，矛盾｛～cV～d｝の一方のリテラルであるdを含む節は，導出終了後削除されることになる．しかしながらこの矛盾は形を変え，｛～aV～bVf｝，｛～aV～bV～c'｝の形でPrime Implicant節に表現されている．

中間仮説が存在する場合の部分コンパイル法のアルゴリズムは，非入出力リテラルを削除する部分コンパイルアルゴリズムの③を，図4．6に示すように順序づけのステップにおいて，中間仮説を除くように変更したものとなる．



① 仮説を用いない場合

入力　{a, b},
ゴール　{e, f}
公理節集合1　{～a∨b∨c, ～a∨～b∨d, ～c∨e,
～d∨f, ～c∨～d}

② 中間仮説を用いた場合

入力　{a, b},
中間仮説　{c'}
ゴール　{e, f}
公理節集合2　{～a∨c', ～c'∨c, ～a∨～b∨d,
～c∨e, ～d∨f, ～c∨～d}

公理節集合2を，部分コンパイルした時のPrime Implicant節
{～a∨c', ～c'∨e, ～a∨～b∨f, ～a∨～b∨～c'}

図4．5　中間仮説が存在する場合の部分コンパイル

①（イ）非入出力単形リテラルを含む節を削除する．
（ロ）①－(イ)の削除により新たに単形リテラルとなるリテラルを含め，
非入出力単形リテラルを含む節が無くなるまで①－(イ)を繰り返す．
②（イ）ユニット導出を行なう．
（ロ）ユニット節ではない方の，導出の親節を削除する．
（ハ）導出終了後，非入力リテラルのユニット節は削除する．
（ニ）導出節を包含する節を削除する．
（ホ）すべてのユニット節について②－(イ)(ロ)(ハ)(ニ)を繰り返す．
③'　中間仮説を除き，双形リテラルについて，順序づけを行なう．
④（イ）組み合わせの少ない順に導出を行なう．
（ロ）導出終了後，非入力リテラルについては，そのリテラルを含んでいる
節をすべて削除する．
（ハ）導出節を包含する節を削除する．
（ニ）順序づけを行なったすべての双形リテラルについて④－(イ)(ロ)(ハ)
を繰り返す．

図4．6　中間仮説が存在する場合の部分コンパイルアルゴリズム

4．3　衝突回避と仮説推論システム

4．3．1　衝突回避に関する知識ベース

仮説推論システムの応用を検討する知識ベースの対象としては，船舶航行エキスパートシステムのサブシステムとして構築した，衝突回避エキスパートシステムにおいて，知識ベースの重要な部分を構成している，海上衝突予防法の衝突を回避するための航行方法（航法）を考える．主として，航法に関する規則のうち，「第２節　互いに他の船舶の視野の内にある船舶の航法」を対象として検討することとする．第２節の第１１条から第１８条，およびこの第２節の規定との関係が深く，知識ベースを完結させるために必要な，３条（用語の定義），７条（衝突のおそれ），第３８条（切迫した危険のある特殊な状況），第３９条（注意等を怠ることについての責任）を加えた，合計１２条を，検討する知識ベースの対象とする．

規定を節形式に変換するに当たっては，コンパイルしたPrime Implicant節を，任意の２船間に適用することが可能である，述語論理を対象とした仮説推論システム[3]の応用を考えるとともに，Prime Implicant節の節数の増大を抑制する効果の高い，部分コンパイル法の応用を考える．

中間仮説が存在する場合の部分コンパイルアルゴリズムや，推論の道筋に沿った部分コンパイルの手順を利用するためには，入力リテラル，ゴールリテラルと，非入出力リテラルとを分類する必要がある．入力としては，視覚等により直接観測・入手することが可能な，船の種類，見合い関係，他船の方位などを考える．ゴールとしては，「避航する」，「針路速力を保持する」，および「衝突のおそれはない」の３つを考える．これ以外をゴールとすることも可能であるが，部分コンパイルをより有効とするためには，非入出力リテラルが多い方が効果的であり，またこの３項目が最終的に必要とされる基本的項目であるため選択した．「避航する」を，さらに「右に変針する」，「左に変針する」，および「減速・停止する」の３つに分類したり，変針・変速の程度を分類することも可能であるが，ここでは最初に示した３つのゴールだけを考えることにする．

ユニット導出を利用しやすいように，入力，ゴールは，正のリテラルで表現する．リテラルの表現としては，「白馬は馬にあらず」的な表現が必要となる．たとえば，「漁労に

従事している」「帆船」は，「帆船」としてではなく，「漁船」として扱うことが必要となる．

入力，ゴール以外を表現するリテラルは，非入出力リテラルとする．適用条文名や，「衝突のおそれがある」，「避航義務がある」「針路速力保持義務がある」などを，非入出力リテラルとした．適用条文名や，「衝突のおそれがある」，「避航義務がある」「針路速力保持義務がある」などを，ゴールとして考えたい場合もあるが，ここでは前述した理由により，３つの自船の動作をゴールとする．

中間仮説としては，一義的に確定できないものを考える．第１２条，第１４条，第１５条において，「ただしこの限りではない」と，第１７条において「同項の規定にかかわらず」と，第１８条において「・・・を除く」と表現されている内容は，厳密には確定することが可能であるが，中間仮説とした方が，現実の運用方法に適合している．第１２条，第１３条，第１４条において「確かめることができない場合」と，第１５条，第１７条，第１８条において「やむを得ない場合を除き」と，第１８条において「できる限り」と表現されている内容は，内容が不明確であり，この法律の表現しているところからだけでは，内容を表現することができないが，仮説と考えるべきであり中間仮説とする．中間仮説を表現するための仮説は，{rule_12_po(X,Y)} のように，”＿ｐｏ”を付けたリテラルを用いて表現した．仮説を項の一つとして表現することも可能であるが，導出時の識別の煩雑さを避けるために，船舶を表すもののみを項とした．仮説の集合はリストにして，仮説であるかどうかの判断に用いる．

４．３．２　衝突回避を対象とした仮説推論システムの概要

（１）推論の道筋に沿った部分コンパイル

仮説を用いた部分コンパイル法により，生成されるPrime Implicant節は，仮説について導出を行なっていないため，仮説を含んだ公理節及び定理節より構成される節集合である．そこで，このプリコンパイルの結果を利用して，仮説の選択を行い，推論の道筋に沿ったコンパイルを行なう仮説推論システムの構築を考える．３章で構築したシステムにおいては，仮説生成機能や仮説検証機能を持っているものの，仮説選択機能は持っていなかったが，ここでは仮説の選択も可能とするシステムの構築を考える．

推論の道筋に沿った部分コンパイルの手順を，図4．7に示す．入力が，事実として観測される毎に，入力についてのユニット導出を行なう．導出を行なった結果について，ゴール節を質問節としてMinimal Support節の生成を行なう．空節が生成される場合，入力からゴール節を証明することが可能である．ゴール節を証明することができない場合，仮説を質問節としてMinimal Support節の生成を行なう．空節が生成される場合，入力から仮説を証明することが可能である．仮説が証明可能な場合，仮説についてユニット導出を行なう．言い替えると，ゴール節が節集合中でユニット節となった時，証明可能であり，仮説が節集合中でユニット節となった時，初めて仮説のリテラルに関しユニット導出を行なう．

ある入力が入力された時，空節が生成されたならば，矛盾が存在する．矛盾が存在した場合，部分コンパイル終了時点まで戻り，それまでに事実となっている入力について，先にユニット導出を行なう．以降の手順は同様である．

① 仮説が存在する場合の部分コンパイルを行なう．
② ①の結果としてのPrime Implicant節を，作業用節集合とする．
③ 入力が事実として観測されたならば，入力のリテラルを用いて，作業用節集合についてユニット導出を行なう．
④（イ）全ての入力の入力が終了したら終了する
（ロ）空節が生成されなければ，⑥に行く．
⑤（イ）空節が生成されたならば，すべての仮説を破棄する．
（ロ）①の結果としてのPrime Implicant節を，再び作業用節集合とし，それまでに事実となっているすべての入力について，ユニット導出を行なう．
（ハ）全ての入力の入力が終了したら終了する
⑥（イ）仮説が節集合中でユニット節となった時，空節を導出する仮説の否定のリテラルが存在したならば，その仮説を含む節を全て削除する．
（ロ）空節を導出しない場合，仮説のリテラルに関しユニット導出を行なう．
⑦ ゴール節が節集合中でユニット節となれば，そのゴールは証明可能である．
⑧ ③に戻る．

図4．7 推論の道筋に沿った部分コンパイルの手順

推論の道筋に沿った部分コンパイルの手順の，適用例を，図4．5の部分コンパイルを行なった時のPrime Implicant節を用いて，図4．8に示す．(1)では，手順①の部分コンパイルの結果を作業用節集合としている（手順②）．(2)では，入力｛a｝に関しユニット導出を行なっている（手順③）．(3)では，空節が生成されないかったので，手順⑥（ロ）により，ユニット節となった仮説｛c'｝に関して導出を行なっている．一つのゴール｛e｝がユニット節となったので，現在までの入力により｛e｝は証明可能である（手順⑦）．まだ全ての入力がなされていないので，続行することが可能である．(4)では，時間の経過に伴う新しい入力｛b｝について導出を行ない（手順③），その結果空節が導出されたので，仮説｛c'｝を破棄している（手順⑤(イ)）．(5)では，手順⑤(ロ)により，既に入力された節のみについて，プリコンパイルされた結果を基に導出を行なっている．先ほどとは異なる新しいゴール｛f｝が導出されている．全ての入力がなされたので終了する（手順⑤(ハ)）．

この方法を用いた場合，矛盾が検出された時点で，毎回，当初の公理節まで戻り，改めてコンパイルし直す必要が無く，プリコンパイルした結果まで戻れば良いため，コンパイル時間の節約を図ることが可能である．プリコンパイルにおいては，仮説について導出を行なわないため，仮説を用いない場合と比較した場合，節集合の増大を抑制することが可能で，プリコンパイルに要する時間が短くなる．また，推論の道筋に従って行なう導出は，全てユニット導出であり，高速で行なうことが可能である．さらに，仮説を用いて，推論の道筋に従って，また新しい入力がある度に，コンパイルを行なう方法は，現実の推論状況に適合しており実用的である．

（2）推論の道筋に沿った部分コンパイルを行う仮説推論システムの特徴

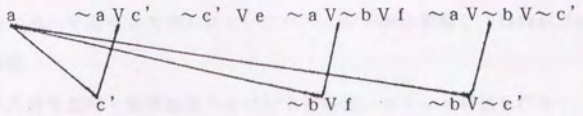
この仮説推論システムにおいては，当初，リテラルおよび節の数は，多少多くなるが，部分コンパイルを用いることにより，導出の回数が減少し，Prime Implicant節 の導出時間を減少させることが可能となる．矛盾が発生した場合，矛盾の無いところまで戻る必要があるが，この仮説推論システムでは，公理節集合をプリコンパイルした結果である，Prime Implicant節まで 戻れば済み，当初の公理節集合から再びコンパイルをし直す必要は無い．

質問節にリテラルが2つ以上含まれている場合，たとえば質問節が｛a V d｝で，Minimal Support節を生成する場合，公理節を｛a V b V c，c V d｝とすると，質問節の

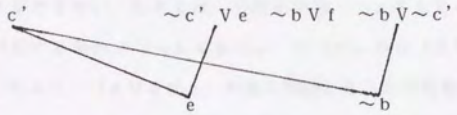
（1）部分コンパイルした時のPrime Implicant節をΣとする.

Σ＝｛～a V c'，～c' V e，～a V～b V f，～a V～b V～c'｝

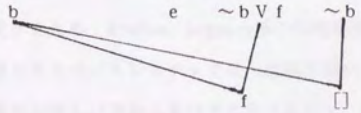
（2）最初の入力をaとし，aに関する導出を行なう.



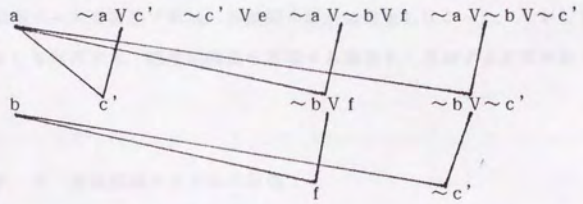
（3）仮説c'がユニット節となったので，c'に関する導出を行なう.



（4）eはゴールである．引き続いて新たな入力bに関する導出を行なう.



（5）空節が導出されたので，Σに対して入力a，bに関する導出を行なう.



（6）fはゴールである.
導出はもう行なうことができないので，終了する

図4．8　推論の道筋に沿った部分コンパイルの手順の適用例

否定は｛～ａ∧～ｄ｝となり，｛ａＶｂＶｃ｝と｛～ａ｝から｛ｂＶｃ｝が，｛ｃＶｄ｝と｛～ｄ｝から｛ｃ｝が導出され，一般のＯＣＣ法においては，｛ｂＶｃ｝と｛ｃ｝との間で suppressが必要となる．こういう一般の場合，Minimal Support節を生成するためには，suppressが必要となるが，質問節をユニット節に限定した場合，このsuppressの必要が無くなるため，今回のシステムにおいては，この手順を省略し Minimal Support節生成の高速化を図った．

新しい入力節を公理・定理節集合に付加する場合，ユニット導出を行ない，Prime Implicant節を導出することになるが，Prime Implicant節の導出においては，suppressは欠かすことができない．たとえば，公理節集合｛～ａＶｂＶｃ，ｄＶｂＶｃ｝に，入力｛ａ｝を付加する場合，｛～ａＶｂＶｃ｝と｛ａ｝から｛ｂＶｃ｝が生成され，この｛ｂＶｃ｝により，｛ｄＶｂＶｃ｝の節を削除することが可能であるからである．

質問節をユニット節に限定したことにより，Minimal Support節，Prime Implicant節とも，高速で生成することが可能であるが，Prime Implicant節については，節集合の大きさの影響を受けるため，Minimal Support節の生成時間よりは，時間がかかることになる．

衝突回避エキスパートシステムでは，仮説を用いる代わりに，危険度に応じて優先順位を付け，条件の厳しい方から実行するようにして，競合の解消を図っていた．この場合，問題領域に対する専門的知識が必要となるが，今回構築した仮説推論システムにおいては，条文通り知識ベースを構築すれば，仮説間の競合は発生しなかった．しかしながら暗黙の了解事項として存在する，排他的関係を表現する節等を，追加する必要があった．

４．３．３　仮説推論システムの評価

（１）評価方法

仮説推論システムの評価を目的として，Prime Implicant節数 等の数や，コンパイル時間等のＣＰＵ時間について，コンパイルの方式による相違，公理節集合の大きさによる相違を検討する．

検討するコンパイルの方式としては，仮説を用いない公理節集合を用いた場合について，そのまま導出を行なう方式の１種類，仮説の存在する公理節集合を用いた場合について，仮説について導出を行なわない方式，非入出力リテラルを削除する方式，および仮説につ

いて導出を行なわず，非入出力リテラルを削除する方式の３種類の合計４種類とした（表４．２）．

公理節の大きさによる影響を検討するため，対象船舶および適用航法により整理し，仮説を用いない公理節集合について１種類と，仮説を用いた公理節集合について３種類の，合計４種類の公理節集合を作成した．仮説を用いた公理節集合は，衝突事故の多い，動力船同士の横切りに関する節集合，対象を横切りだけではなく，全航法に広げるとともに，対象とする片方の船舶を動力船，もう一方を全船舶にと拡大したもの，そして対象を帆船を含めた全船舶に拡張したものである（表４．４）．

それぞれのケースについて，Prime Implicant節 生成時間（プリコンパイル時間），質問ユニット節に対する Minimal Support節生成時間，および質問ユニット節の付加に要する時間を測定した．Minimal Support節 生成時間，および節の付加に要する時間については，それぞれ３つの質問節に対する平均値を測定し，ＣＰＵ時間 50000ｓｅｃで計測を打ち切った（表４．３，表４．５）．

（２）コンパイル方式の影響

４種類のコンパイル方式による，プリコンパイルによる Prime Implicant節生成時間，Prime Implicant節数，Minimal Support節生成時間，および節集合への新しい節の付加に要する時間のＣＰＵ時間を，表４．３に示す．

仮説を用いない公理節集合について，そのまま導出を行なう方式では，高々公理節は４６しか存在しないが，50000ｓｅｃ以内では，Prime Implicant節を求めることができなかった．導出を行なっている中途であるが，制限時間を超え節の生成を打ち切った時点で，この場合 Prime Implicant節の候補が，約２５００存在しており，１つの節の中のリテラルの数が最大１４に達する節も存在していた．導出の中途における節の増大，および一つの節が含んでいるリテラル数の増大が，制限時間内に Prime Implicant節を求めることができなかった理由であると思われる．

これに対し仮説を用いた場合，Prime Implicant節 生成時間は，制限時間以内に納まっている．仮説について導出を行わない方式の場合（②），および非入出力リテラルを含む節を導出終了後削除する方式の場合（③）については，約４０倍以上の高速化を図ることができている．両方を同時に行う場合（④），後で入力がある度に節集合に付加することが可能か，充足可能性を検討する必要があるが，約１００倍以上の高速化を図ることがで

表4．2　コンパイルの方式

仮説を用いない公理節集合
① そのまま導出を行なう方式

仮説を用いた公理節集合
② 仮説について導出を行なわない方式，
③ 非入出力リテラルを含む節を，導出終了時削除する方式
④ 仮説について導出を行なわず，非入出力リテラルを削除する方式

表4．3　節数・リテラル数とＣＰＵ時間
（コンパイル方式の影響）

	公理節	リテラル	Prime Implicant節		Minimal Support節	ユニット節
	数	数	数	生成時間 (sec)	生成時間 (sec)	付加時間 (sec)
①	４６	３１	－	above 50000	－	－
②	５２	３４	１４６	１３２９	１．７	２６４
③	５２	３４	７７	１１６２	２．０	４１７
④	５２	３４	４７	４７４	１．０	３０

きている．仮説を用いるとともに非入出力リテラルを削除する方式は，Prime Implicant節数の増加の抑制効果が高く，生成時間も小さくなっている．さらに，仮説について導出を行なわない方式は，後で充足可能性を検討する必要はあるものの，非常に有効な方式である．

表には示していないが，仮説を用いて節中のリテラルの数を抑え，導出の制限や，非入出力リテラルを含む節の削除を行なわず，仮説についてもそのまま導出を行なった場合，公理節数が５２の場合，約 16000ｓｅｃで，Prime Implicant節 を求めることができた．即ち，否定のリテラルを多く含む節を，仮説を用い節の長さを短くすることだけによっても，約３倍以上の高速化を図ることができており，仮説を用いることの効果は高い．また，順序づけの方式について検討するため，削除される非入出力リテラルについて，先に導出を行なうことを，③のケースについて行なったが，約5400ｓｅｃかかっており，アルゴリズム通りの順序で導出を行なった方が，約４．５倍速い結果となっており，順序づけの効果の高さが示された．

Minimal Support節生成時間は，ユニット節だけを対象とし，ユニット導出を有効に利用した結果，Prime Implicant節数の大きさに対して抑制された時間となっている．節数が５２である④の場合，同規模の節集合を対象とした命題論理・述語論理を対象とした汎用システムでは，従来２．６－３．１ｓｅｃかかっていたものが，約１ｓｅｃで生成することができている．

節の付加に要する時間は，節の間で包含関係を検討する必要があり，④の場合で，３０ｓｅｃかかっている．Prime Implicant節 生成時間が小さく，推論の道筋に沿ってコンパイルするするのに要する時間もそれほど大きくないことから，仮説についてあらかじめ導出を行なわず，部分コンパイルを行ない，推論の道筋に沿ってコンパイルを行なう方式は，高速化のために，効果の高い方式であるということができる．しかしながら，実用レベルから考えた場合には，３０ｓｅｃという時間はまだまだ大きいということができる．

表４．３においては，Minimal Support節 生成時間も，節の付加に要する時間のＣＰＵ時間も，知識ベースの内容の影響を受け，Prime Implicant節 の数と，必ずしも一次的な関係があるとはいえない結果となっている．

（３）公理節の大きさの影響

４種類の大きさの公理節集合（表４．４）における，プリコンパイルによる Prime Implicant節生成時間，Prime Implicant節数，Minimal Support節生成時間，および節集合への新しい節の付加に要する時間のＣＰＵ時間を，表４．５に示す．

「右に見る」に対する「左に見る」のように，負リテラルを用いて表現可能なリテラルも存在するが，入力，ゴールを，正のリテラルで表現したため，リテラル数は増加することとなる．また，仮説リテラルを導入したため，さらにリテラル数は増加している．公理節の数は，仮説を用いた場合，仮説の数だけ増加している．しかしながら仮説を使用しない場合であっても，不明確な内容を表現するリテラルが必要であり，仮説を使用した場合とのリテラル数の差は，①と②の差に見られるように，それほど大きいわけではない．リテラル数は概略公理節の数に比例して増加している．

Prime Implicant節および生成時間は，公理節およびリテラルの増加に伴い，急速に増加している．公理節数が１４６の，フルサイズの公理節集合については，Prime Implicant節を50000ｓｅｃ以内で求めることができなかった．この例については 100000ｓｅｃ以上まで実行してもみたが，結果は同じであった．

Minimal Support節生成時間は，Prime Implicant節数が４７から２２１へと，約５倍になったのに対し，約３倍にしかなっておらず，Prime Implicant節数の大きさの影響を，余り顕著に受けていない．これは，ユニット導出に限定したことの効果であると思われる．

節の付加に要する時間は，Prime Implicant節数の大きさの影響を大きく受け，推論の道筋に沿ってコンパイルを行なうには，実用的な範囲を超えた，例えば公理節数９３の時約1500ｓｅｃという，非常に大きな値となっている．表４．５のユニット節付加時間についての括弧中の数値は，suppressを省略した場合についての時間である．無矛盾での充足可能性を高速で検討するために，suppressを行なうのであるが，このシステムにおいてはその負荷が大きくなっている．プリコンパイル後，suppressは無矛盾性での充足可能性を検討するために，必ずしも必要では無いので省略した結果，６－２０倍高速で，新しい節を付加することが可能となった．

推論の道筋に沿ってコンパイルを行なうこと，および矛盾が発見された場合に，元の公理節集合に戻らず，再びプリコンパイルの状態から再開することは可能であった．プリコンパイル，部分コンパイルを利用した，推論の道筋に沿って仮説の選択を行なうことのできる，仮説推論システムが構築できたということができる．再びプリコンパイルの状態か

表4．4　公理節集合の種類

仮説を用いない公理節集合
① 動力船同士，横切りの航法

仮説を用いた公理節集合
② 動力船同士，横切りの航法
③ 動力船と全船舶，全航法
④ 全船舶，全航法

表4．5　節数・リテラル数とＣＰＵ時間
（公理節の大きさの影響）

	公理節数	リテラル数	Prime Implicant節		Minimal Support節生成時間 (sec)	ユニット節付加時間 (sec)
			数	生成時間 (sec)		
①	４６	３１	－	above 50000	－	－
②	５２	３４	４７	４７４	１．０	３０ (5.2)
③	９３	５８	２２１	５０７３	３．３	１４８７ (73.5)
④	１４６	８７	－	above 50000	－	－

ら再開する場合，概略それまでの入力ユニット節を付加する時間の合計の時間が必要であった．公理節が高々９３の時に，１入力あたり約７０秒かかるということは，実用レベルから考えた場合，まだまだ遅いということができる．

4．4　まとめ

この章においては，仮説推論システムを，衝突回避エキスパートシステムの知識ベースを構成する，海上衝突予防法の航行方法に関する規則を対象として，応用することを考えてきた．

この海上衝突予防法を基にした知識ベースの特徴は，次の通りである．

① ある入力に対してゴールに到る経路が複数存在すること，

② 規則が複雑であること，不明確であること，

③ 排他的な内容が多いこと，

④ 入力は「事実」の形で，時間経過に従って一つずつ，行なわれること．

そこで，この知識ベースに仮説推論システムを応用するため，次のような手法の開発，およびシステムの改造を行なった．

① Minimal Support節の生成の高速化と，推論の道筋に沿ったコンパイルを可能とする，ユニット導出の利用方法，

② Prime Implicant節集合の増大の抑制を目的とした，入力とゴールおよび単形リテラルを利用した部分コンパイル法，

③ 欠落している知識を表現する仮説の表現方法，および仮説が存在する場合の部分コンパイル法

④ 時間経過，推論の道筋に沿ったコンパイルを可能とする，仮説推論システムの構築

これらの開発・構築した手法・システムの評価を，コンパイル方式の影響および公理節の大きさの，ＣＰＵ時間・生成Prime Implicant節数への影響を見ることにより行なった．

部分コンパイルを用いることにより，用いない場合には，ＣＰＵ時間 50000ｓｅｃ以内では，Prime Implicant節集合を求めることができなかった公理節集合について，Prime Implicant 節集合の，ゴールを証明するために必要な部分集合を求めることができた．少なくとも約４０倍以上の高速化が図れたことになる．仮説を用いて表現し，プリコンパイ

ルを行なう方式も，後で仮説が選択された時に，仮説について導出を行なう必要があるものの，同程度の高速化が図れている．両方の方式を同時に利用した場合，少なくとも約１００倍以上の高速化を図ることができた．また，仮説を用いることは，１つの節中のリテラル数の減少を図るためにも効果があった．

知識ベースの特徴に応じた，推論の道筋に沿ってコンパイルを行なう，仮説推論システムを構築することができた．公理節の数が９３の場合でも，Prime Implicant節を ２２１に抑制し，約３ｓｅｃで Minimal Support節を生成することが可能であり，また，新しいユニット節の充足可能性を，約７０ｓｅｃで検討することが可能であった．

これらの開発・構築した手法・システム等は，衝突回避に関する知識ベースのみならず，他のシステムにも適用・応用可能なものである．

しかしながら，この評価のための動作テストは，２隻の船を対象としたのにも関わらず，航行方法の全てを覆う，節数１４６のフルサイズの知識ベースについては，Prime Implicant節集合を生成することができなかった．またMinimal Support節生成や，節の充足可能性検討のためのＣＰＵ時間も，手法的には高速化が図れているものの，実用システムへの適用を考え，絶対値で見た場合，依然として時間がかかりすぎている．

高速化を図るための，第一の方法は，プログラム技術的な高速化を図ることである．このシステムが，Prologの基本的な機能を利用して構築されていることから，リストのメンバーの探索に時間がかかるなどの問題点は，仮説をビットベクトル等で表現する手法を利用すれば，解決可能であり，ここに示したアルゴリズムや手順により，どこまで高速化が可能か，検討する必要がある．また，自船を中心としたシステムにすることにより，変項を減少することが可能であるし，相手船毎にコンパイルすることにより，Prime Implicant節の数の 増大を抑制することが可能であるなど，問題の特性に応じた高速化も，必要に応じて検討する必要がある．

今回はゴールとして，３つの自船の動作のみを考えたが，適用条文名や，「衝突のおそれがある」，「避航義務がある」，「針路速力保持義務がある」などを，ゴールとして考えたい場合もある．ゴールが増加すれば，部分コンパイルの効果は減少することになるため，ゴール別にコンパイルする方法などを検討する必要もある．また，数値を含む命題は今回対象としていないが，針路の変更，速力の変更など，数値を含む命題の表現方法・処理方法の開発が必要である．

4章　参考文献

[1] 鶴田三郎，石塚　満：命題論理知識ベースのコンパイル法，情報処理学会研究報告，90-AI-70，情処研報，Vol.90，No.32，１９９０

[2] 鶴田三郎，石塚　満：発想的命題論理知識ベースのコンパイル法，人工知能学会誌，１９９１（予定）

[3] 鶴田三郎，石塚　満：述語論理知識ベースのコンパイル法，第４回人工知能学会全国大会，pp.81-84，１９９０

[4] 鶴田三郎，松村尚志，稲石正明，今津隼馬，杉崎昭生：船舶航行エキスパートシステムの基礎研究　－衝突回避エキスパートシステム－，日本航海学会論文集，No.77，pp.133-139，１９８７

[5] de Kleer, J.：A Comparison of ATMS and CSP Techniques，IJCAI-89，pp.290-296，１９８９

[6] de Kleer, J.：Exploiting Locality in a TMS，AAAI-90，pp.264-271，１９９０

[7] 坂井　公，相場　亮：CAL：制約論理プログラミングの理論と実例，電子情報通信学会研究資料，SS87-28，pp.9-17，１９８８

[8] 伊藤史朗，石塚　満：論理的制約利用による高速仮説推論システム，情報処理学会研究報告，90-AI-70，情処研報，Vol.90，No.32，１９９０

5章　結論

5．1　まとめ

現在，船舶システムの自動化の流れの中で，船舶システムの設備・機器の自動化や，近い将来の知能化船，無人化船の実現を目指した研究が進められている．しかしながら，機械の操作・処理の自動化等が進められているものの，運航者の専門的知識や経験に裏打ちされた判断の自動化は，ほとんど進められていなかった．そこで，安全かつ能率的な船舶航行のための専門家の機能の補完，専門家の負荷軽減のための支援，そして運航に関するのノウハウの蓄積による専門家の教育・訓練を目的とする，船舶航行エキスパートシステムの開発に着手した．この研究は，船舶航行に関する問題領域を対象とした初めての研究であり，また開発したサブエキスパートシステムは，いずれもそれぞれの問題領域に関する最初の研究である．

船舶航行エキスパートシステムの開発においては，一般のエキスパートシステムの開発と同様に，知識ベースの構築と，推論機構の構築が大きな課題となる．

知識ベースを構築するためには，専門家の持つ知識・ヒューリスティックス，推論・判断のプロセスなどを抽出整理し，専門家の知識およびその構造，推論のパターン，知識量を明らかにし，獲得した知識を，その構造，推論のパターンにあった知識表現モデルを利用して表現することが必要である．そこで，船舶航行エキスパートシステムにおける知識ベースの開発対象として，知識の特性の異なる，衝突事故対応エキスパートシステム，衝突回避エキスパートシステム，および機関操作エキスパートシステムを選定し，この3つのサブエキスパートシステムの構築を通して，専門家の知識とその構造，推論のパターン，および知識量について検討を行なった．

推論機構は知識の操作性の向上を目的としており，一般的な高速化を図るためには，問題領域に関係の無い，高速推論を可能とする推論機構の構築の必要性が高まる．また無矛盾性を維持したまま，知識の追加，削除，変更等の操作を行なうためには，推論機構にはこれを解決する知識管理機能も，高速推論機能に併せて求められ，これらを可能とする推論機構の構築が必要である．そこで，知識の操作性の向上をめざして，推論の高速化，および知識管理機構の構築を目的とする，知識のコンパイル技術を，命題論理および述語論

理を対象として開発した．この知識のコンパイル技術を利用した，仮説推論システムを構築し，その効果について検討すると共に，開発したサブエキスパートシステムの一つが対象とした，衝突回避に関する知識ベースに対して，その応用を検討した．

（1）船舶航行エキスパートシステムの知識ベースについて

システムの推論結果は，船長等の運航の専門家に，想定状況を示してヒアリングした結果とほぼ同様の，実際の判断や処理に近いものであった．ルールの内容の明確化や，表現の統一を行なったのにも関わらず，推論結果および推論過程が妥当なものとの評価が得られたことは，知識の内容の明確化が適当であったこと，ルール間およびルールグループ間の整合性を保った知識の構造化が可能であったことを意味している．また，推論形式の状況による変化，機能に特有な推論形式，競合解消等の，専門家の推論パターンを明らかにすることができたとともに，ある程度システムに取り入れることができた．この結果，推論結果だけでなく，推論過程も実際のパターンに近いものとすることができた．

衝突回避エキスパートシステムの１つの推論を導き出す全所要時間は，推論時間の７０％以上を占める従来型の計算時間を含めても，相手船が１船の場合約２０ｓｅｃであった．衝突事故対応エキスパートシステムの１つの推論を導き出す全所要時間は，２０～２５ｓｅｃであり，このうち計算時間を除いた純粋の推論時間は，単純平均で約６ｓｅｃであった．また，機関操作エキスパートシステムの，ある時刻における一連の推論を行なう時間は約２０ｓｅｃであった．このように，開発したいずれのエキスパートシステムにおいても，推論時間は２０ｓｅｃ程度であり，これを専門家に対する，ヒヤリングの結果による許容推論時間１ｍｉｎ以内や，船上での情報収集に要する時間，および操作上の入力間隔と比較した場合，実用に耐える速度を実現できているということができる．

知識はプロダクションルールで表現したが，モジュール性が高いという長所を生かしながら，船舶航行に関する知識や，推論の特徴を表現することが可能であった．知識ベースの規模は，開発したいずれのサブエキスパートシステムにおいても，ルール数で２００－３００であった．また，１つのルールの中の条件の数は，どのシステムにおいても大差無く約３であった．３つのサブエキスパートシステムの開発を通して得られた知見を基に，実用の船舶航行エキスパートシステムを構築するために必要な，最低のルール数を推定すると，約２５００ルールとなる．また開発実績から，この２５００ルールのシステムを構築するのに要する時間を推定すると，知識の抽出源により異なるが，約９００人日から

２０００人日となる．特徴の異なる３つの問題領域から，知識を抽出し，特徴，構造化および表現方法について検討するとともに，システムの最低の規模および開発所用時間を明らかにすることができた．

（２）知識のコンパイル技術の開発と，仮説推論システムについて

船舶航行エキスパートシステムの，実用システムの規模が，最低でも２５００ルールであることから，知識ベースが大きくなると重要性の高まる，推論の高速化と無矛盾性管理を目的として，知識ベースの高速コンパイルを可能とする，Prime Implicant節の高速導出方法としての順序づけ節コンセンサス法（Ordered Clause Consensus method：ＯＣＣ法）の開発，およびコンパイルした知識を利用した，Minimal Support節の 高速生成法の開発を行なった．Prime Implicant節 導出時間は，幅優先法と比較した場合約５－９０倍高速であった．このコンパイル技術を応用して，命題論理を対象とした仮説推論システムを構築した結果，質問節がユニット節で Minimal Support節として空節が生成されない場合には，単純な幅優先戦略による推論と比較した場合，６０－１００００倍高速で，またＯＣＣ法を利用し高速化を図ったインタプリタ形式による場合と比較した場合でも，１０－１００倍高速であり，推論の大幅な高速化を図ることができた．

また，従来の真理管理機構が命題論理しか対象としていなかったため，表現力の向上を目指して，ＯＣＣ法を述語論理を対象とすることができるように拡張した．述語を対象として順序付け節コンセンサス法を適用したＣＯＭＰ（COmpiling Method for Predicate）と，単純にinstantiateする方法とを，実際の知識を対象として比較した結果，Prime Implicant節導出時間は，instantiateする方法と比較した場合約１０－４５倍高速であり，推論の大幅な高速化を図ることができた．述語表現を用いることにより，問題空間の拡大を抑制した，知識ベースの高速コンパイルが可能であり，また高速でMinimal Support節を生成することが可能となった．

（３）船舶航行に関する知識ベースに対する仮説推論システムの応用について

開発した仮説推論システムを，船舶航行エキスパートシステムに応用するために，応用するに当たっての課題の検討と，課題を解決するための手法の開発，システムの改造を行なった．仮説推論システムの応用を検討する対象としたのは，衝突回避エキスパートシステムの知識ベースを構成する，海上衝突予防法の航行方法に関する規則である．海上衝突

予防法を対象としたのは，この知識ベースを対象とする場合，不備な知識を用いて推論する必要があり，仮説推論システムを応用するのにふさわしい問題領域であるからである．

この知識ベースに仮説推論システムを応用するため，次のような手法の開発し，これに伴いシステムの改造を行なった．

① 入力に対してゴールに到る経路が複数存在するなど，Prime Implicant節集合が増大する可能性が高いため，この抑制を目的とした部分コンパイル法の開発．

② 不備な知識に対する推論を可能とする仮説の表現方法，および仮説が存在する場合の部分コンパイル法の開発．

③ 見張り等による入力が時間経過に従って一つずつ行なわれるため，ユニット導出を利用した，推論の道筋に沿ったコンパイルを可能とするシステムの開発．

仮説を用いて知識を表現すると共に，部分コンパイルを用いてプリコンパイルを行なう方式は，後で仮説が選択された時に，仮説について導出を行なう必要があるものの，用いない場合に対して，少なくとも約１００倍以上の高速化を図ることができた．すなわち，部分コンパイル法を実現することができたと共に，推論の道筋に沿ってコンパイルを行なう，仮説推論システムを構築することができた．

システムの改造や新しい手法の開発が必要であったが，仮説推論システムを船舶航行エキスパートシステムの知識ベースに応用することが可能であった，さらに，ここで開発・構築した手法・システム等は，衝突回避に関する知識ベースのみならず，他の一般のシステムにも適用可能なものであり，応用範囲の広いものである．

本研究により，船舶航行エキスパートシステムに関する知識ベース構築の実現性と開発目標を提示することができた．また，この研究は，高次人工知能機能に向けての基礎研究として位置づけられるものでもある．

５．２　今後の課題

開発したサブエキスパートシステムについても，推定した最低ルール数の，まだ４０－５０％程度の知識しか収集されておらず，フィールドテスト等を行なうためには，また実用的な船舶航行エキスパートシステムを構築するためには，知識ベースを充実する必要が

ある．現実の場において推論の高速化が求められる時には，専門家は高速化を可能とする知識を使用している．これらの知識はたぶんに感覚的であり，この素早い判断を可能としている専門的知識を，知識ベースに組み込むために，たとえば視覚情報の高速自動入力等の開発も必要となる．

システムを使用評価した専門家の希望事項は，操作性に関するものが大半であった．実用性を考えた場合，ユーザのレベルやユーザの能力の進歩に伴って，さらに進んだ柔軟な問い合わせが行なえるようなユーザインタフェース機能が必要である．実際の場面においては，推論の進行，状況の変化に伴い，判断の変更等が必要となる場合が少なくなく，推論形式を，運航機能別に，また処理状況の変化によって，変更することが可能なようにする等，これに対応するための柔軟な操作性を実現する必要がある．特に，緊急時のように，エキスパートシステムへの期待が大きい状況では，容易に利用できるように操作の複雑さ，煩雑さを改良する必要がある．

仮説推論システムを海上衝突予防法の知識ベースに適用したが，２隻の船を対象としたのにも関わらず，航行方法の全てを覆う，節数１４６のフルサイズの知識ベースについては，Prime Implicant節集合を生成することができなかった．またMinimal Support節生成や，節の充足可能性検討のためのＣＰＵ時間も，手法的には高速化が図れているものの，実用システムへの適用を考え，絶対値で見た場合，依然として時間がかかりすぎている．

このシステムにおいては，節中のリテラルの照合や，包含関係の検討が重要であり，この照合等のために時間がかかっていることから，プログラム技術的な高速化を図り，ここに示したアルゴリズムや手順により，どこまで高速化が可能か，検討する必要がある．

部分コンパイルを適用するに当たって，本研究においては自船の３つの動作のみをゴールとして考えたが，ゴールを増加した場合，部分コンパイルの効果は減少することになるため，ゴール別にコンパイルする方法などを検討する必要もある．また，針路の変更，速力の変更など，数値を含む命題の表現方法・処理方法の開発も必要である．

これらの課題を解決してゆくことにより，船舶航行エキスパートシステムの実用システムの構築が可能となると共に，高次人工知能の実現に向けての着実な進歩が可能となる．

謝　舌辛

本論文を作成するに当たっては，多くの方々から貴重な御指導と御助言を賜った．

特に，東京大学生産技術研究所助教授石塚満先生には，研究の機会を提供していただいたと共に，研究の遂行に際し，あらゆる面において適切な御指導と御鞭撻を賜った．ここにここに深甚の謝意を表します．

また，著者に船舶航行に関するエキスパートシステムの研究の重要性を示唆され，研究を進めるに当たって御指導を賜った東京商船大学教授杉崎昭生先生，そしてエキスパートシステムの研究グループの今津隼馬先生，稲石正明先生，松村尚志先生に感謝の意を捧げます．

東京大学生産技術研究所石塚研究室の皆様には，有益な討議を賜り，併せて感謝の意を申し上げます．

5. 学術論文

[1] 岡田三郎：ホタテ貝－貝殖の研究の現状，日本応用動物学会誌 [illegible]，[illegible] pp.27-31，1961年9月

[2] 河合信人，成瀬二郎，[illegible]，[illegible]：新の無機加速度と電圧応答特性，日本応用動物学会誌 [illegible]，[illegible]号，pp.47-56，1961年9月

[3] [illegible]：[illegible]，[illegible]：[illegible]の[illegible]について，日本応用動物学会誌 [illegible]，[illegible]号，pp.71-75，1961年9月

[4] 鈴木三郎：下高マイヤ[illegible]に基く[illegible]について，日本応用動物学会誌 [illegible]，[illegible]号，pp.17-21，1961年9月

[5] 岡田三郎，小山[illegible]，[illegible]：[illegible]と[illegible]の[illegible]について，日本応用動物学会誌 [illegible]，[illegible]号，pp.27-31，1961年9月

[6] 鈴木[illegible]，成瀬二郎，[illegible]：[illegible]の[illegible]に関する[illegible]，日本応用動物学会誌 [illegible]，[illegible]号，pp.37-43，1961年9月

[7] [illegible]：[illegible]の[illegible]，日本応用動物学会誌 [illegible]，[illegible]号，pp.47-51，1961年9月

[8] 岡田三郎：合成[illegible]の[illegible]と[illegible]について，日本応用動物学会誌 [illegible]，[illegible]号，pp.131-135，1961年9月

[9] [illegible]：[illegible]，[illegible]，[illegible]，

論　文　一　覧

1．学術論文

[1]鶴田三郎：カテナリー理論の応用限界，日本航海学会論文集，５７号，pp.139-144，1977年８月

[2]田辺行夫，鶴田三郎，神田寛，後藤大二：船の動揺加速度と動揺病発症率，日本航海学会論文集，５７号，pp.77-84，1977年８月

[3]小川征克，鶴田三郎，杉崎昭生：見張りの信頼性について，日本航海学会論文集，６３号，pp.11-17，1980年８月

[4]鶴田三郎：ドルフィン係留船舶に働く係留力について，日本航海学会論文集，６５号，pp.13-21，1981年９月

[5]鶴田三郎，小山荒雄，鞠谷宏士：静止把駐力を越える力が加わった場合の錨の挙動について，日本航海学会論文集，６７号，pp.21-26，1982年８月

[6]鞠谷宏士，鶴田三郎，福谷恒男：錨鎖張力の動的解析に関する実験研究，日本航海学会論文集，６９号，pp.17-23，1983年９月

[7]小山荒雄，鶴田三郎：新型錨の開発，日本航海学会論文集，７１号，pp.47-54，1984年９月

[8]鶴田三郎：合成繊維索の引張り疲労強度について，日本航海学会論文集，７３号，pp.151-155，1985年９月

[9]SABURO TSURUTA：RESIDUAL STRENGTH OF ARAMID ROPE，OFFSHORE TECHNOLOGY CONFERENCE，OTC86，pp.223-230，MAY，1986

[10]鶴田三郎，田中義久：合成繊維索の摩耗特性について，日本航海学会論文集，７５号，pp.87-94，1986年９月

[11]鶴田三郎，稲石正明，松村尚志，杉崎昭生：衝突事故対応エキスパートシステムの基礎研究，日本航海学会論文集，７５号，pp.131-139，1986年９月

[12]鶴田三郎, 松村尚志, 稲石正明, 今津隼馬, 杉崎昭生：船舶航行エキスパートシステムの基礎研究－衝突回避エキスパートシステム－, 日本航海学会論文集, ７７号, pp.133-139, 1987年９月

[13]M. AKIO SUGISAKI, HAYAMA IMAZU, SABURO TSURUTA, MASAAKI INAISHI, HISASHI MATSUMURA：AN EXPERT SYSTEM FOR COPING WITH COLLISION AVOIDANCE AND THE EX POST FACTO AT SEA, INTERNATIONAL NAVIGATION CONGRESS, IAIN1988, pp.223-230, FEB. 1988

[14]稲石正明, 松村尚志, 鶴田三郎, 今津隼馬, 杉崎昭生：船舶航行エキスパートシステムの基礎研究－集中タワーにおける機関操作エキスパートシステム－, 日本航海学会論文集, ７９号, pp.9-16, 1988年９月

[15]HISASHI MATSUMURA, MASAAKI INAISHI, SABURO TSURUTA, HAYAMA IMAZU, M. AKIO SUGISAKI：BASIC RESEARCH ON AN EXPERT SYSTEM FOR NAVIGATION AT SEA －ENGINE OPERATION EXPERT SYSTEM AT THE CONTROL TOWER－, 2ND INTERNATIONAL OCEAN AND DEVELOPMENT EXHIBITION AND SYMPOSIUM, TECHNO-OCEAN'88, pp.429-433, Nov. 1988

[16]松村尚志, 稲石正明, 鶴田三郎, 今津隼馬, 杉崎昭生：船舶航行エキスパートシステムの基礎研究－システム開発過程における評価について－, 日本航海学会論文集, ８１号, pp.31-38, 1989年５月

[17]HISASHI MATSUMURA, MASAAKI INAISHI, SABURO TSURUTA, HAYAMA IMAZU, M. AKIO SUGISAKI：BASIC RESEARCH ON AN EXPERT SYSTEM FOR NAVIGATION AT SEA, INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL WORKSHOP, CAMS'89, pp.37-43, Aug. 1989

[18]鶴田三郎, 石塚　満：発想的命題論理知識ベースのコンパイル法, 人工知能学会誌, Vol.6, No.1, 1991年（予定）

2. 修士論文

振れまわり運動のシミュレーションについての一考察, 東京商船大学修士論文, 1977年３月

3．研究発表

[1]鶴田三郎，林尚吾，及川　清，巻島　勉：京浜港横浜区の横浜航路における船舶交通実態調査について，東京商船大学第２７回学術講演会論文集，pp.41-44，1979年２月

[2]安藤定雄，鶴田三郎：ゴムフェンダーの非線形特性について（その１）－模型化と強制変位－，第３４回船舶技術研究所研究発表会講演集，pp.77-80，1979年１２月

[3]松本隆夫，鞠谷宏士，鶴田三郎：TYPHOON BUOYの基礎的研究，東京商船大学第２８回学術講演会論文集，pp.44-47，1980年２月

[4]小川征克，鶴田三郎，杉崎昭生：見張りの信頼性について，東京商船大学第２８回学術講演会論文集，pp.56-59，1980年２月

[5]鶴田三郎，林尚吾，及川　清，巻島　勉：京浜港横浜区の横浜航路における船舶交通実態調査について，東京商船大学研究報告３０号，pp.171-191，1980年３月

[6]及川　清，鶴田三郎，小山荒雄，小山悟郎：救命設備の使用効果に関する研究，東京商船大学研究報告３２号，pp.117-136，1981年１２月

[7]鶴田三郎，小山荒雄，小山悟郎，鞠谷宏士：錨の把駐特性に及ぼす引張速度の影響に関する実験研究，東京商船大学研究報告３２号，pp.75-88，1981年１２月

[8]鞠谷宏士，鶴田三郎，渡辺　豊：直線移動にともなう貨物のふれについて（第１報）一本吊りの場合，東京商船大学研究報告３４号，pp.95-110，1982年１２月

[9]鶴田三郎，稲石正明，松村尚志，杉崎昭生：船舶航行エキスパートシステムの基礎研究－衝突事故対応エキスパートシステムについて－，東京商船大学第３５回学術講演会論文集，pp.25-28，1987年２月

[10]鶴田三郎，小山悟郎，橋本一明：錨の把駐性能に関する実船実験，東京商船大学研究報告３９号，pp.31-51，1988年１２月

[11]橋本一明，小山悟郎，鶴田三郎：錨の把駐性能に関する実船実験，東京商船大学第３７回学術講演会論文集，pp.29-32，1989年２月

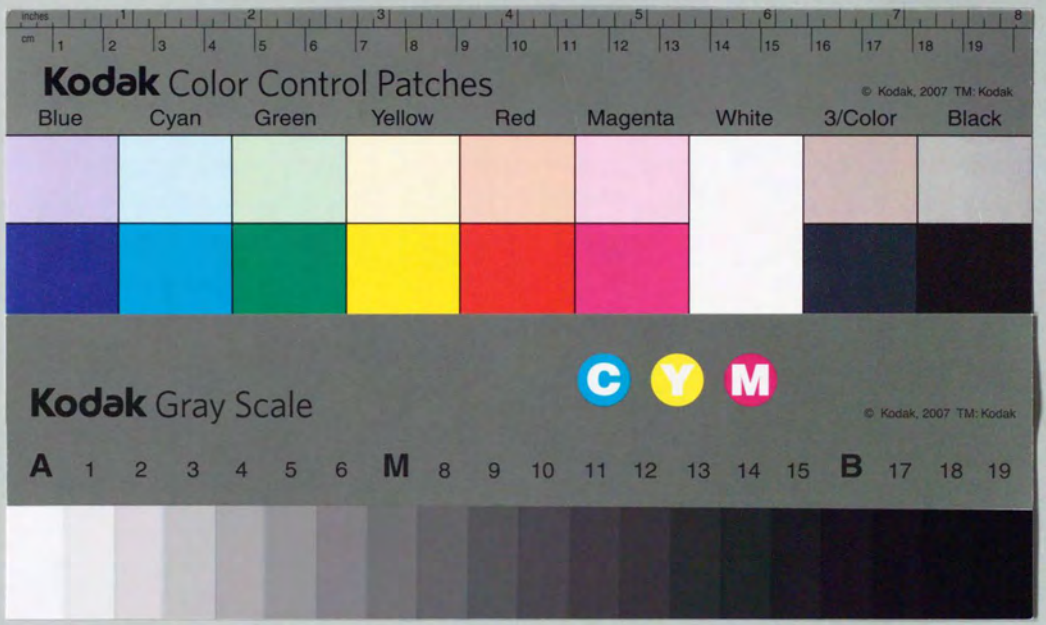
[12]今津隼馬，杉崎昭生，鶴田三郎，稲石正明，松村尚志：船舶航行エキスパートシステムの基礎研究，日中航海学会学術交流会論文集，pp.69-78，1989年

[13]鶴田三郎，石塚　満：不完全な知識が係わる知識システムへのＣＭＳの応用に関する基礎研究，第３回人工知能学会全国大会，pp.179-182，1989年７月

[14]小山悟郎, 橋本一明, 鶴田三郎：合成繊維索の強度の経年変化, 東京商船大学第３８回学術講演会論文集, pp.57-60, 1990年２月

[15]鶴田三郎, 石塚　満：命題論理知識ベースのコンパイル法, 情報処理学会研究報告, 90-AI-70, Vol.90, No.32, 1990年５月

[16]鶴田三郎, 石塚　満：述語論理知識ベースのコンパイル法, 第４回人工知能学会全国大会, pp.81-84, 1990年７月


inches
cm
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
C
Y
M
Kodak Gray Scale
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19