自動推論を用いた論理回路故障診断システムの開発研究

中川嘉宏

①

自動推論を用いた論理回路

故障診断システムの開発研究

中川嘉宏

平成5年7月

目 次

第１章

序論

近年の急速な電子技術の発展はシステムを高級化・複雑化してきている。また巨大化したいくつかのシステムが互いに密に結合されている場合もある。このような状況では，システムに異常や故障が発生したときの社会的影響は非常に深刻で，その正確で高速な予測，検知，診断のための理論や応用技術の開発が急務となってきた。

すべてのシステムは初めに仕様が検討される。システムの仕様が巨大化・複雑化すると，意図しない論理的な矛盾が仕様のなかに含まれる可能性も無視できない。仕様が作成され設計の段階を終了するとシステムが仕様の通り機能を果たしているかを検査する検証段階に入る。この段階で問題点が検出されれば仕様あるいは設計が再検討される。検証の次の段階が診断である。すなわち，システムが運用に供されて保守段階に入ったとき，それが正常な運転なのか故障状態なのかを判断し，もし故障と判断されれば効率的に故障個所の特定を行なう必要がある。

システムを論理回路に限定して，その診断技術のアプローチをみると大きく二つの流れがある。一つは故障シミュレーションを行なう方法である。アルゴリズム的に発生させたテストパターンを回路入力に与え，出力パターンとの比較から故障診断を行なう。検出可能な故障であればそのテストパターンを必ず

求めることができる完全なアルゴリズムが考案されている。他方の診断技術の流れは，人工知能の推論技術に関する研究成果を診断問題へ応用する試みである。このアプローチはさらに大きく二つのタイプがある。一つは診断型エキスパートシステムと呼ばれる推論(浅いモデルによる推論)をベースにした診断システムである。このタイプの診断システムでは既に感染症の診断に成果をあげている。もう一つのタイプは故障の経験的知識または統計的情報に全く依存しない方式がある。この方式は診断対象が内包する論理のみに着目して故障要素を推論(深いモデルによる推論)により検出する。具体的には、診断対象の構成、部品の機能、入出力観測値などを論理的に表現し、自動推論によりそれらの間の論理的矛盾の検査を行なう。矛盾が検出されればそれが故障であると判断できる。以下，これらの方式の考え方について簡単に述べる。

1.1 故障シミュレーション

アルゴリズム的にテストパターンを生成し，それを用いて論理回路の故障を検出する。テスト生成の基本的アルゴリズムとして，Roth の考案した D アルゴリズム が最も広く利用されている。これは，故障による影響が出力に表われるように入力のパターン(テスト入力)を決定するアルゴリズムでる。最近ではVLSI から超 LSI へと回路の集積化が進むにつれて，D アルゴリズムをベースにした高速なテスト生成アルゴリズムが考案されている。ここでは D アルゴリズムによる故障診断の実際について例を用いて述べる [2], [11]。

図 1.1 の回路においてゲート G_1 が故障しているとする。ただし，その出力(信号線 3)が 0 に縮退するモデルを考える。D アルゴリズムでの最初の手続きは素子の故障端子に誤り信号を割り付ける。0 誤り信号および 1 誤り信号は，それぞれ D および $\bar{D}$ を割り付ける。この例でキューブ

1	2	3
1	1	D

を故障の基

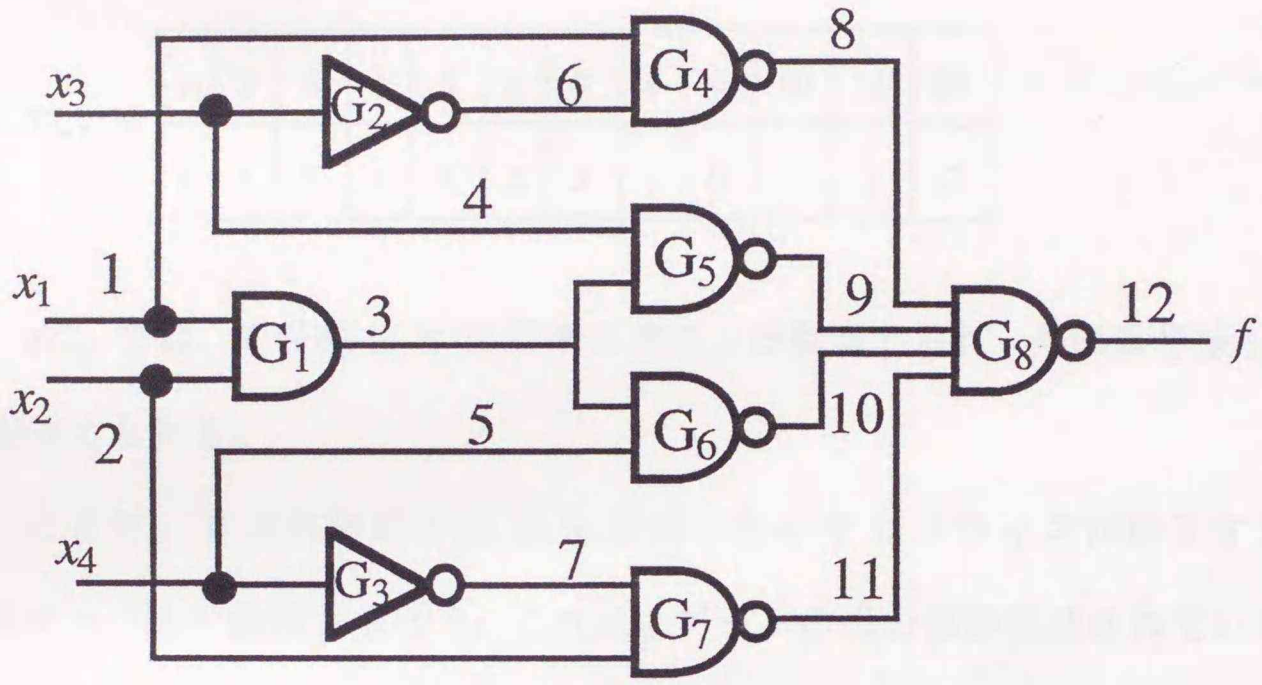


図 1.1: D シミュレーションによる診断回路

本 D キューブという。このとき，故障信号 (D または $\bar{D}$) を入力とするゲートを D フロンティアという。

故障信号をさらに出力側へ伝播させるために D フロンティアからゲートを一つ選び，その出力へ故障信号を伝播させる。このキューブを伝播 D キューブという。このゲートを G_5 とすると，伝播 D キューブは

3	4	9
D	1	$\bar{D}$

となる。この操作を繰り返し，故障信号を外部出力まで伝播させる操作を D ドライブという。ゲート G_5 の伝播 D キューブでは，信号線 4 に 1 が割り当てられる。よって，ゲート G_2 の出力は一意的に 0 になる。このように一意的に値を定める操作を含意操作という。この時点でのテストキューブは

$TC_1 =$

1	2	3	4	5	6	7	8	9	10	11	12
1	1	D	1	X	0	X	1	$\bar{D}$	X	X	X

となり，あらたなDフロンティアは $\{G_6, G_8\}$ である。このうち，出力線に近いゲート G_8 を選びDドライブを行うと，次のようなテストキューブが求められる。

$TC_{11} =$

1	2	3	4	5	6	7	8	9	10	11	12
1	1	D	1	X	0	X	1	$\bar{D}$	1	1	D

TC_{11} では，信号線10と11にそれぞれ1が割当てられ，外部信号線12に D が割当てられる。

ここで，Dが外部信号線12に伝播したのでDドライブが終了する。次のステップは一致操作を行う。これは，ゲートの出力値が記述されているが入力値が定まっていない信号線 (Xで表現) に値を割当てる操作である。TC_{11} では $\{G_6, G_7\}$ である。G_6 の出力線10が1であるためには入力線5は0である必要がある。この値で含意操作を行うと，ゲート G_3 の出力線の値が1となり，さらに G_7 の出力線11の値は0でなければならないが，これは TC_{11} により矛盾である。この場合，アルゴリズムはバックトラックを行う。すなわち，TC_1 で選択したDフロンティア G_8 を G_6 に換えてDドライブを行う。このときのテストキューブは

$TC_{12} =$

1	2	3	4	5	6	7	8	9	10	11	12
1	1	D	1	1	0	0	1	$\bar{D}$	$\bar{D}$	1	D

となり，テストパターン生成に成功する。以上，Dアルゴリズムの手順を例を用いて述べてきた。これをまとめると次のようになる。

(i) 故障を仮定した端子をもつ素子の基本Dキューブを作る。

(ii) 故障信号 (D または $\bar{D}$) をDドライブにより外部出力線まで伝播させる。伝播の経路が複数存在するときはそのすべてを求める。

(iii) D ドライブで得られた各キューブについて，一致操作を行い，テスト入力を求める。

一般に D アルゴリズムでは (iii) においてバックトラックが発生する。この回数を減少させアルゴリズムの効率を上げることが重要な研究課題となっている [11]。

1.2 エキスパートシステム

エキスパートシステム (expert system:ES) とは，人間の専門家が問題を解決するプロセスを計算機にシミュレートさせようとするものである。具体的には，問題領域に関する知識をルール形式で表現し，これと推論を行う機能を計算機に実現させて問題を解決する。推論制御にはプロダクションシステム (production system:PS) とよばれるプログラミング技法がよく用いられる。一般に PS の構成は次のようになっている [25]。

- プロダクション (production:P) の集合，これをプロダクションメモリ (production memory:PM) と呼ぶ。
- データベース，これをワーキングメモリ (working memory:WM) と呼ぶ。
- インタプリタ (PSI)

PM に蓄えられている P は，その P が起動される条件 C_i の集合と，条件が満たされたときに実行される動作 A_i の集合との対からできている。条件の集合部分をその P の LHS(left hand side)，動作の集合を RHS(right hand side) と呼ぶ。 PSI はすべての P について P の LHS を満足する状況が WM のなかにあるのかを監視している。もし，そのような P が見つかれば，その P の RHS で指定される行動が順次実行される。 RHS で示される行動の系列は一般に WM の状況を変化させることになるので，これによって引き起こされる WM の変化がさらに別の

症状	ドライブに電力が供給されない	ディスクの故障	書き込み禁止のプロテクトがかかっている	ドライブのアライメントが狂っている	仮説 常駐プログラムに問題がある	ベルトの故障	ヘッドが汚れている	ディスクが正しく挿入されていない	ディスクのドアが閉じていない
ブートしない	✓	✓		✓		✓	✓	✓	✓
ドライブのランプがつかない	✓								
ドライブから何の音もしない	✓								
ドライブのランプが点灯する		✓	✓	✓	✓	✓	✓	✓	✓
ドライブから異音がする		✓				✓			
読み込みのトラブル		✓		✓	✓	✓	✓	✓	✓
書き込みのトラブル		✓	✓	✓	✓	✓	✓	✓	✓
ディスクが挿入できない		✓							
初期診断においてコードNo.608となる		✓							
トラック/セクタのエラーメッセージがでる		✓		✓			✓		
すべてのドライブが機能しない		✓							
書き込み禁止メッセージが表示される			✓						
FORMATが正しくされていない			✓		✓	✓	✓	✓	✓
読み込みメッセージが準備できていない						✓		✓	✓
別のドライブが働いている	✓						✓		
ディスクが逆向きである								✓	
ドライブのドアが開いている									✓

表 1.1: エキスパート診断における仮説 - 症状一覧

P の LHS を満たせばその P の RHS が起動されて行動の連鎖が続くことになる。

PS による推論過程の動作は次の三つのステップを繰り返し実行する。

(i) PM 中のすべての LHS を，WM の状況と照合し，起動条件が満足されている P の集合を作る。この集合を競合集合 (conflict set) と呼ぶ。

(ii) 競合集合の中から，ある種の戦略に基づいて起動すべき P を選択する。この操作を競合の解消 (conflict resolution) と呼ぶ。

(iii) 選択された P の RHS を実行して WM の状態を変える。

故障診断に限らずエキスパートシステムを成功させるためには，実際に専門家が有している問題領域の知識をよく整理しておく必要がある。表 1.1 はディスク装置の診断に用いられた故障の症状と原因の関係である [25]。この表の各行が一つのプロダクションに対応する。仮説の各列の要素が連言形式で LHS に格納され，症状が RHS となる。表 1.1 はコンピュータ診断における周辺装置診断の部分を構成している。

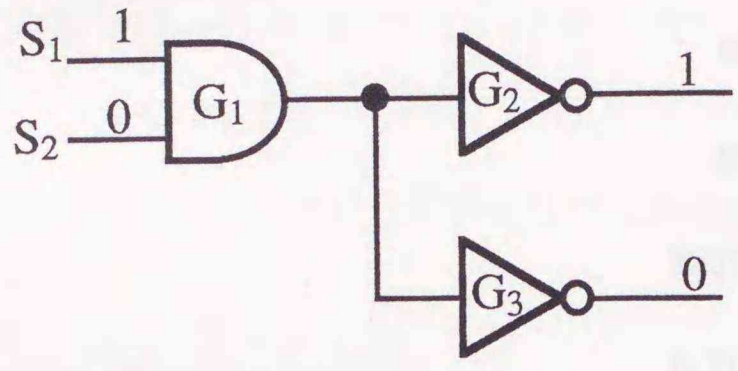


図 1.2: 深いモデルによる診断回路例

1.3 深いモデルによる診断

エキスパートシステムが現象 (または症状) と原因を結び付ける経験的な知識や判断から診断していく方法に対して，問題領域に関する教科書的な基礎知識から論理的に推論し診断を行う方法である [9], [7], [23]。診断対象が内包する論理のみに着目する診断方法で深いモデル (deep models) による推論とよばれる。これに対してエキスパートシステムは浅いモデル (shallow models) による推論とよばれる。実際の診断は、診断対象の構成、部品の機能、入出力観測値などを論理 (述語論理の節集合，方程式-不等式の集合等) で表現し、自動推論システムによりそれらの間の論理的矛盾の検査を行なう。矛盾が検出されれば自動推論システムが作成する証明から故障要素を抽出できる。

図 1.2 に示す回路についての節表現を用いた知識表現 (一部) は次のようになる。

$$\mathbf{ANDG}(x) \wedge \neg\mathbf{AB}(x) \supset \mathbf{EQUAL}(out(x), and(in1(x), in2(x)))$$

$$\mathbf{NOTG}(x) \wedge \neg\mathbf{AB}(x) \supset \mathbf{EQUAL}(out(x), not(in1(x)))$$

$$\mathbf{EQUAL}(in(G_1), out(S_1))$$

$$\mathbf{EQUAL}(in(G_2), out(G_1))$$

$$\mathbf{EQUAL}(not(0), 1)$$

$$\mathbf{EQUAL}(not(1), 0)$$

$$\mathbf{EQUAL}(and(0, x), 0)$$

$$\mathbf{EQUAL}(and(1, x), x)$$

$$\mathbf{EQUAL}(and(x, x), x)$$

$$\neg\mathbf{EQUAL}(1, 0)$$

$$\mathbf{EQUAL}(in1(x), 1) \vee \mathbf{EQUAL}(in1(x), 0)$$

$$\neg\mathbf{EQUAL}(in1(x), 1) \vee \neg\mathbf{EQUAL}(in1(x), 0)$$

$$\mathbf{ANDG}(G_1)$$

$$\mathbf{NOTG}(G_2)$$

$$\mathbf{NOTG}(G_3)$$

$$\mathbf{EQUAL}(out(S_1), 1)$$

$$\mathbf{EQUAL}(out(G_2), 1)$$

$$\neg\mathbf{AB}(G_1)$$

$$\neg\mathbf{AB}(G_2)$$

これらの節の集合は，回路構成素子の動作，回路接続情報，ブール代数の公式，現在の入出力の値 (観測)，素子の種別および現在の回路素子は正常であるという仮定などが記述されている。$\mathbf{AB}(x)$ は素子 x が異常 (abnormal)，$\mathbf{EQUAL}(x, y)$ は x と y が等しいことを意味する述語である。明らかに図 1.2 には故障素子が存在する。この場合，節の集合の部分集合に充足しないものがある。これを自

動推論システム (定理証明器) で検出し，充足しない理由 (証明) から故障素子を推論する (図 1.2 の例では，最終的に $\{G_3\}$ の単一故障と $\{G_1, G_2\}$ の二重故障が推論される)。

述語 **EQUAL** は推論の効率化のため自動推論システムに組込みになっている [14]。また，**AB** は自動推論システムを用いる上で故障診断システムが認識する特別な述語である。**AB** は論理回路の診断以外には次のように使われる [23]。

- 要素がツェナーダイオードの場合，通常 (故障していなければ)，その両端に正でかつブレークダウン電圧以下の電圧がかかっている場合，その電流は零である。

$$\mathbf{ZENER-DIODE}(z) \wedge \neg\mathbf{AB}(z) \wedge \mathbf{GREATER}(voltage(z), 0) \wedge$$
$$\mathbf{LESS}(voltage(z), break-voltage(z)) \supset \mathbf{EQUAL}(current(z), 0)$$

- 要素 C_1 の故障は要素 C_2 の故障を引き起こす。

$$\mathbf{AB}(C_1) \supset \mathbf{AB}(C_2)$$

- あるタイプの要素の故障は，$\mathbf{FAULT}_1, \ldots, \mathbf{FAULT}_n$ のどれかである。

$$\mathbf{TYPE}(x) \wedge \mathbf{AB}(x) \supset \mathbf{FAULT}_1(x) \vee \ldots \vee \mathbf{FAULT}_n(x)$$

以上示してきたように，述語 **AB** を故障要素検出のための一種のタグとして利用できるように知識表現に埋めこんでいる。ここで，注意すべきは，論理回路およびツェナーダイオードの例で示したように，**AB** は「異常」であることのみ意味するもので，それがどのような異常であるかを表現していない。すなわち故障モデルを想定しなくてもよい。また，各知識表現の本質はその要素が正常動作のときの知識表現となっている。これは，診断対象システムの設計時に用いられた知識が大きな変更を必要としないで診断のための知識として利用できることを意味する。

1.4 研究目的と論文構成

はじめに，前節までに述べた各種診断方式の利点と問題点を簡単にまとめ，次に本研究の目的と概要を述べる。

故障シミュレーションによる診断は，高速なテストパターン生成アルゴリズムが開発されデジタル回路の診断では主流となっている。しかし，故障モデルが0縮退（あるいは1縮退）に限られているため，この種の故障しか診断できない。さらに，現在の観測を全く利用することができないためオンライン診断には対応できない。

浅いモデルによる診断では，故障診断のための経験的知識をルール形式で表現し，ルールの連鎖をたどることにより最終的な診断を得る。この方式は，診断対象機器のライフ・サイクルが長く故障原因の統計的データが充分蓄積されている場合や，専門家の知識が最適にルール・ベース化されている場合には，推論方式が単純なため高速に診断結果が得られる。しかし，最近のように開発された装置の製品寿命が短期になってきたり，あるいは複雑大規模化してくると，最適なルール・ベースの構築が難しい問題となり，実用的な故障診断システムの開発は困難である。

深いモデルによる診断では，特定の故障モデルを想定しなくてもよく，また，診断のための経験的あるいは統計的なデータはまったく必要ない。しかし，論理的に表現された診断対象の矛盾を高速に見つけだす高度な自動推論システムとその効果的利用のためのヒューリステックスが必要である。

これらの各方式はいずれも他の方式を排除するものではなく，最終的には各方式が総合的に組み合わされて高度で効率良い故障診断システムとして構築されるべきものである。

本論文は，深いモデルに基づいて著者が開発した論理回路故障診断システムについて述べたものである。この方式のもつ上述の問題点を種々の観点から解

決し，それをシステムとして実現している。このシステムを本論文では DiaLog と呼ぶ。DiaLog の主要部分は診断系と推論系とから成っている。その概要を以下に示す。

DiaLog の診断系は，基本的には，1987 年に Reiter が提案した故障診断の一般理論をアルゴリズムとして実現したものであるが，さらにパッケージ・レベル診断が行なえるようにアルゴリズムを拡張してある。パッケージは回路素子の集合である。実際の回路素子はパッケージ単位で生産・販売・保守されるので，故障診断においては，異常素子を特定できなくても，異常パッケージが特定できれば十分なことが多い。パッケージ・レベル診断はこのことに着目して，診断アルゴリズムの無駄な探索空間を除去し，診断時間の減少に寄与している。

推論系は論理回路に関する一般知識，診断対象である特定の論理回路の構成に関する知識，および診断系から与えられる仮説との間の矛盾を検出するためのアルゴリズムを実現したものである。

DiaLog の推論系は，これまで開発してきた一階述語論理に基づく汎用自動推論システム（定理証明器）Thinker を基に，論理回路故障診断向きに，その推論動作をチューニングし推論効率を大幅に向上させたものである。例えば，組合せ回路や順序回路の知識表現，推論規則および制御戦略が，実験に基づいて適切に設計あるいは選択されている。その結果，これらの工夫のない場合の自動推論システムの動作に比べて，本システムの推論系では無駄な推論が減少し，推論時間の減少に寄与している。

DiaLog の推論系は一階述語論理による自動推論をベースにしているが，必要ならば論理回路を構成する素子の特性に応じて特殊化した推論系を組み込むことができる。これにより 2 値論理回路に限らず今後新たに開発される論理素子に対する故障診断に対しても DiaLog の基本的構成を変更することなく推論部を拡張することで対応できる。本研究ではこの例としてファジィ論理回路の

故障を診断する推論系を構成した。この特殊化した推論系は，ファジィ論理回路向きに効率化した制約伝播方式に基づいた推論アルゴリズムを用いている。このアルゴリズムは，ファジィ論理回路の知識を方程式-不等式系で表現し，局所制約伝播と端点値探索という手法を組み合わせて推論を行なう。その結果，一階述語論理に基づく推論システムと比べて，やや柔軟性が失われる代償として，診断時間の大幅な減少が得られる。実際にはこのアルゴリズムは，最大，最小，補数の各演算ゲートからなるファジィ論理回路向きに設計されているが，その特殊な場合として，2値論理回路にも効率を落とすことなく対処できる。

以後本論文では，便宜上，2値論理回路に対する診断診断システムを DiaLog-I と呼ぶ。同様に，ファジィ論理回路に対する診断システムを DiaLog-II と呼ぶ。

次に，論文の構成を述べる。

第2章では， DiaLog の基本的な診断系を構成する Reiter の故障診断理論を簡潔にまとめて紹介している。診断を数学的に厳密に定義し，それを求めるアルゴリズムを解説している。また， Greiner によって改訂されたアルゴリズムについても解説している。

第3章では，DiaLog-I の推論系の基になっている Thinker の概略を解説している。このシステムが扱うことのできる知識表現の形式，各種の推論規則，推論の制御戦略などが示されている。

第4章は， DiaLog-I の構成と診断について述べている。まず，Thinker のチューニング方法を述べ，ついで，組合せ論理回路のための述語論理による知識表現を説明し，適切な推論規則と制御戦略を決定している。その後に DiaLog-I の構成を示し，実験例として全加算器の故障診断を取り上げ，多重故障の診断結果，および観測データの一部欠損など，診断のための環境が悪化した状況での診断結果を示している。

第5章では，Reiter の診断理論において表われる診断単位を拡張したパッケー

ジの概念を提案している。具体的には，DiaLogの診断系を構成するパッケージ・レベル診断の理論とアルゴリズムを示している。このアルゴリズムによりこれまで述語論理ベースでは扱われることのなかったような規模の論理回路に対する診断も可能としている。この例として順序素子を含む自動販売機制御部に対する故障診断を示している。

第6章では，特殊化された推論系をもつDiaLog-IIによるファジィ論理回路の診断について述べている。まず，ファジィ論理回路の構造および観測される入出力値が方程式，不等式の集合で表現できることを示し，次に，この集合の上での矛盾を検出するために推論系が用いる推論規則を定義している。この推論は，制御機構に局所制約伝播を用いて効率良く実現されている。しかし，この推論のみでは矛盾の検出に不完全な場合があるため，端点値探索法と呼ぶアルゴリズムを提案し，局所制約伝播と組み合わせて全体として完全な診断を求める方法について述べている。

第7章は，本研究の全体的なまとめと今後の研究課題について論じている。

第 2 章

故障診断の理論

システムの構造や動作を規定する理論と現在のシステムの動作状況 (観測) を適当な論理，例えば述語論理などで統一的に表現する。すなわち，論理式の集合で表現する。システムが故障であるのはこの論理式の集合に矛盾しているものが存在していると考える。 Reiter は 1987 年に論理式間の矛盾を機械的に検出することによりシステムの故障要素を推論する診断理論を発表した。本章では，はじめに Reiter の提案した故障診断理論について概略を述べる。Reiter の故障診断理論は， 1989 年に Greiner によってシステムによっては不完全な診断を行うことが指摘された。本章の後半ではこの改訂された Reiter の故障診断理論についても触れる。詳細は文献 [23], [12] を参照。

2.1 問題記述

故障診断の理論を特定な領域だけでなく，一般のシステムに適用することを可能にするためにシステムの概念を特定の領域とは独立に定義する。

定義 1 システムとは、システム記述 (*system discription*, **SD**) とシステム構成要素の集合 **COMPONENTS** の組 $\langle$**SD**, **COMPONENTS**$\rangle$ である。

ただし，SD はシステムの構造や挙動を表す第一階論理式の集合， COMPONENTS はシステムの故障し得る構成要素を表す定項の有限な集合である。

SD には「異常である」を意味する特別な述語 AB を用いる。それ以外の述語記号は適当に定めてよい。

故障診断をするためには，なんらかの形でシステムを観測しなくてはならない。観測を次のように定義する。

定義 2 システム観測 OBS は，第一階論理式の有限な集合で表される。

この表現に基づくと，システムが故障しているというのは，その構成要素 $\{C_1, C_2, ..., C_n\}$ について、

$$\mathbf{SD} \cup \{\neg\mathbf{AB}(C_1), \ldots, \neg\mathbf{AB}(C_n)\} \cup \mathbf{OBS} \tag{2.1}$$

が矛盾を持つことと同じである。

定義 3 $\langle \mathbf{SD}, \mathbf{COMPONENTS}, \mathbf{OBS} \rangle$ についての診断とは、

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\mathbf{AB}(C) \mid C \in \Delta\} \cup \{\neg\mathbf{AB}(C) \mid C \in \mathbf{COMPONENTS} - \Delta\} \tag{2.2}$$

が無矛盾であるような極小集合 $\Delta \subseteq \mathbf{COMPONENTS}$ である。

この定義は次のように解釈できる。Δ^* を診断とする。定義より，Δ^* に属さない要素のみの正常性を仮定すると，システム記述と観測の関係を矛盾なく説明できる。また，論理式の集合は論理的には連言を表わすから，Δ^* を包含する Δ に対しても 2.2 式は無矛盾である。しかし Δ^* の真部分集合 Δ に対しては定義により 2.2 式は矛盾を含む。また，特にシステム記述と観測との間にはじめから矛盾がないときに限り，$\Delta^* = \{\}$ である。

2.2 診断の計算

診断問題は 2.2 式が無矛盾であるような極小集合 $\Delta \subseteq \mathbf{COMPONENTS}$ を求めることに帰着した。これを COMPONENTS のすべての部分集合について直接に計算したのでは構成要素が多くなったときに効率が悪くなる。このため Reiter は de Kleer による、コンフリクト集合 (conflict set) の概念に基づく手法を示した [9]。

定義 4 $\langle \mathbf{SD}, \mathbf{COMPONENTS}, \mathbf{OBS} \rangle$ についてのコンフリクト集合とは、

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\neg \mathbf{AB}(C_1), \ldots, \neg \mathbf{AB}(C_k)\} \tag{2.3}$$

が矛盾となる集合 $\{C_1, \ldots, C_k\} \subseteq \mathbf{COMPONENTS}$ である。

定義 5 集合族 C のヒット集合 (*hitting set*) H は、それぞれの $S \in C$ について $\mathbf{H} \cap S \neq \{\,\}$ であるような集合である。ヒット集合 H のいずれの真部分集合も C のヒット集合でないとき、H を極小ヒット集合という。

次の定理は診断を計算するための基礎を提供する。

定理 1 $\Delta \subseteq \mathbf{COMPONENTS}$ が $\langle \mathbf{SD}, \mathbf{COMPONENTS}, \mathbf{OBS} \rangle$ の診断であることと，Δ が $\langle$ SD, COMPONENTS, OBS $\rangle$ のコンフリクト集合の族についての極小ヒット集合であることとは同値である。

このように $\langle \mathbf{SD}, \mathbf{COMPONENTS}, \mathbf{OBS} \rangle$ の診断を求めるためには，そのすべてのコンフリクト集合の族についての極小なヒット集合を計算すれば良い。実際には，

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\neg \mathbf{AB}(C) \mid C \in \mathbf{COMPONENTS}\} \tag{2.4}$$

が矛盾を生じることを健全で完全な定理証明器 (theorem prover) によって証明し、その証明に使われた $\neg \mathbf{AB}(x)$ のインスタンスを $\{\neg \mathbf{AB}(C_1), \ldots, \neg \mathbf{AB}(C_k)\}$ とすると、$\{C_1, \ldots, C_k\}$ がコンフリクト集合になる。

2.3 HS-木 の生成による診断

Reiter はすべてのコンフリクト集合を求めずに HS-木 と呼ばれる木を生成して極小ヒット集合，すなわち診断を得るアルゴリズムを示した。具体的に HS-木 を生成するアルゴリズムは次のように示されている。ただし，$H(n)$ は，ルートからノード n までのパスについているラベルの集合である。また，定理証明器は 2.4 式の矛盾性を検出するために用いるもので，矛盾ならばコンフリクト集合を値として返し，無矛盾ならば記号 $\surd$ を返すものとする。なお，以下に示すアルゴリズムを本論文では要素レベル診断と呼ぶ。

Step-1 システム構成要素がすべて正常であると仮定して定理証明器を呼び出す。もし $\surd$ が返れば要素レベルの診断は $\{\}$ であり，終了。コンフリクト集合が返れば，そのコンフリクト集合を HS-木 の根のラベルとする。(ラベルがコンフリクト集合でないノードを「閉じられたノード」と呼ぶ)。

Step-2 すべてのノードが閉じられていれば終了。このとき $\surd$ でラベル付けされたノード n の $H(n)$ のうち，極小のものが要素レベル診断となる。

Step-3 まだ閉じられていないノード n を任意に選ぶ。n がコンフリクト集合 S によってラベル付けされているとする。それぞれの $\sigma \in S$ について，σ をラベルとする枝を介して子ノード n_σ を生成する。**COMPONENTS**$-H(n_\sigma)$ の要素を正常と仮定して定理証明器を呼び出し，それが返した値 ($\surd$ またはコンフリクト集合) を n_σ のラベルとする。 **Step-2** へ進む。

Step-3 において n_σ のラベルを計算するときに，すでに求められているコンフリクト集合 S のうちで，$S \cap H(n_\sigma) = \Phi$ を満たすものがあれば，定理証明器を呼び出さずに n_σ のラベルを S としてよい。

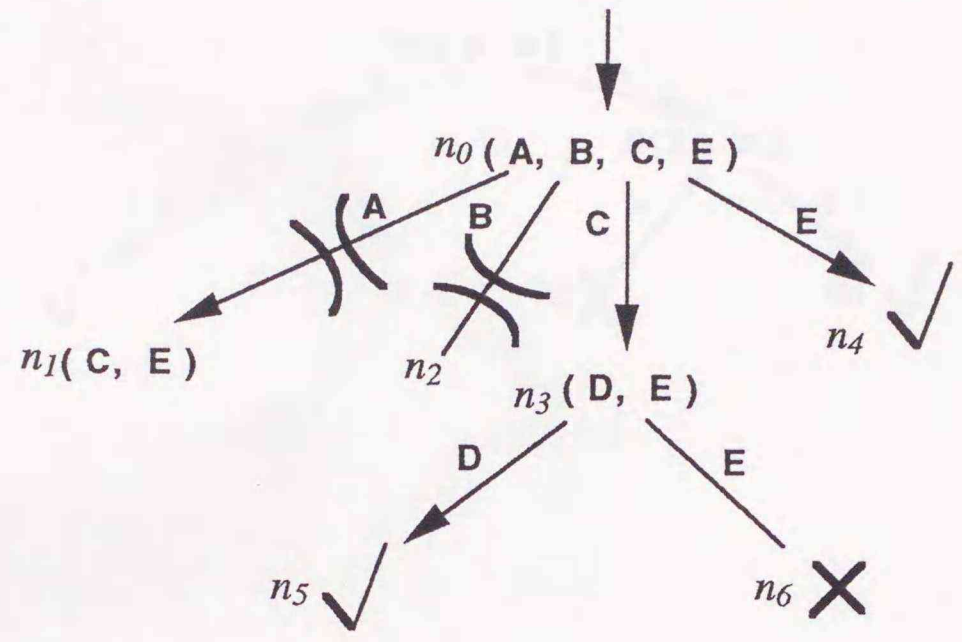


図 2.1: HS-木 生成の例

このアルゴリズムによって生成される HS-木 は冗長な部分があるので，Reiter は診断の極小性に基づく以下の三つの刈り込みを行なって効率を上げている。

(i) $\surd$ がついた他のノードと $H(n)$ が同じか，大きくなるようなノード n は閉じる。

(ii) すでに生成されたノードと同じ $H(n)$ となるノード n は閉じる。

(iii) ノード n のラベル S の真部分集合 S' がノード n' のラベルであるとき，各要素 $\alpha \in S - S'$ に対し α をラベルとして n から出ている枝をカットする。

図 2.1 に HS-木 生成の例を示す。最初にコンフリクト集合 {A, B, C, E} が得られたとしこれを根のラベルとする。根の第一子ノードのラベルとしてコンフリクト集合 {C, E} が得られたとする。刈り込み (iii) によりラベル A, B の枝をカットする。第三子ノードにおいてコンフリクト集合 {D, E} が得られたとする。第四子ノードにおいて $\surd$ が得られたとする。すなわち，単一故障の診断 {E} が得られる。次に，{D, E} とラベル付けされたノード (n とする) から D,

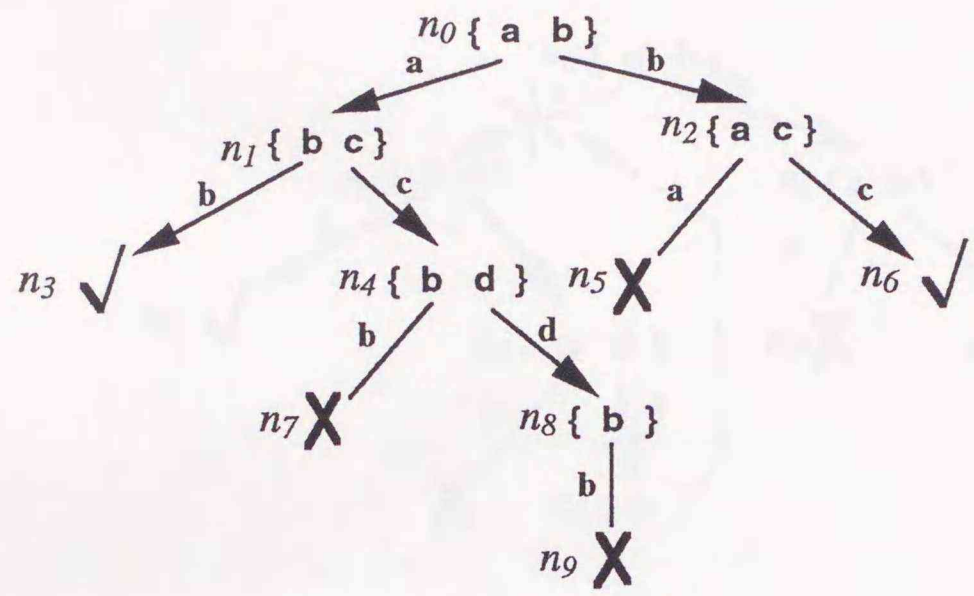


図 2.2: 刈り込みを行わない HS-木

E とラベル付けされる枝をのばす。D に隣接するノード n_D において $\surd$ が得られた場合は，二重故障診断 {C, D} が得られる。ノード n_E は刈り込み (i) により閉じられる。

2.4 Reiter の診断アルゴリズムの問題点

定理 1 よりコンフリクト集合の族に対する極小のヒット集合が診断となる。HS-木 生成による診断はすべてのコンフリクト集合が明示的に求められている必要はなくかつ定理証明器が返すコンフリクト集合は極小でなくてもよい。すなわち，HS-木 の生成時に前述の三つの刈り込みを行うことにより，自動的に $\surd$ でラベル付けされたノード n の $H(n)$ が極小のヒット集合すなわち診断となっている。しかし，この刈り込みは極小のヒットを失う場合があることが Greiner によって指摘された [12]。以下にその例を示す。

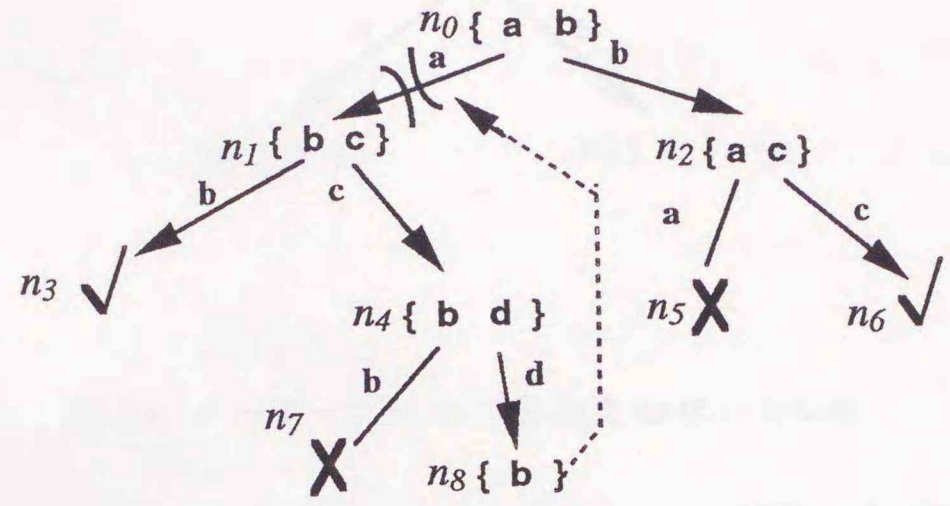


図 2.3: 刈り込みを行った HS-木

不完全な診断が得られる例として次の集合族を考える。

$$\{\{a,b\},\{b,c\},\{a,c\},\{b,d\},\{b\}\}$$

このとき，**HS-木** が図 2.2 のように得られたとする。ただし，刈り込み (iii) は行われていない。図から診断は

$$\{\{a,b\},\{b,c\}\}$$

となる。同じ状況で刈り込み (iii) を使用する。この結果は図 2.3 となる。ノード n_0 から出ているラベル b の枝がカットされる。この結果，診断は

$$\{\{b,c\}\}$$

となり診断 $\{a,b\}$ を失い不完全なものとなる。

次に示す例も不完全な診断が行われる例である。

$$\{\{a,b\},\{a\},\{b\}\}$$

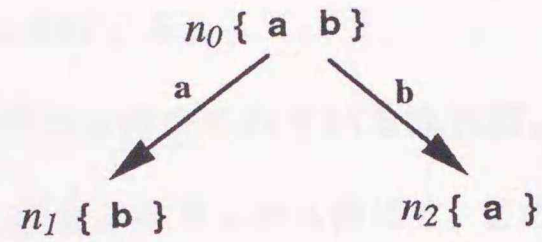


図 2.4: ノード・ラベルの書換えのない HS-木

の集合族を考える。図 2.4 に示すように刈り込み (iii) が行われると枝がすべてカットされてしまう。

最初の例は，刈り込み (i), (ii) と (iii) との刈り込みのタイミングが問題となる。図 2.4 の例は刈り込み (iii) が行われたあとにノード・ラベルの書き換えが必要になる例である。すなわち，ラベル a をもつ枝がカットされた場合，ただちにノード n_0 のラベルを $\{a,b\}$ から $\{b\}$ に書換える必要がある。

次節では Reiter の診断アルゴリズムが持つこれら問題点を改良したアルゴリズムを述べる。

2.5 改訂診断アルゴリズム

HS-木 を生成にする代わりに有向サイクルグラフ **dag** (directed acyclic graph) を用いる。**HS-木** に対応する dag を **HS-dag** と呼ぶ。**HS-dag** の生成アルゴリズムを簡明に示すため，集合族 F は順序付けされているものとする。はじめに刈り込みを行わない **HS-dag** D の生成アルゴリズムを示す。

(i) D の根になるノードを生成する (ラベル付けはここでは行われない)。

(ii) D のノードは幅優先で処理される。ノード n については

(a) $H(n)$ を根から n へのパス上の枝に付けられたラベルの集合とする。

(b) すべての $x \in F$ について，$x \cap H(n) \neq \{\}$ ならば，n を $\surd$ でラベル付けする。そうでなければ，$\Sigma \cap H(n) = \{\}$ となるような F の最初の要素 Σ でラベル付けする。

(c) n が $\Sigma \in F$ でラベル付けされているならば，$\sigma \in \Sigma$ について σ でラベル付けされた枝をノード n から伸ばす。この枝だは $H(m) = H(n) \cup \{\sigma\}$ となるようなノード m に連結する。この新しい m は，n の深さのすべてのノードが展開されたあとに処理される。

(iii) D を最終的な dag として返す。

このアルゴリズムは刈り込みのない **HS-木** の生成のアルゴリズムと基本的には同じになっている。異なる点は，F が順序付けられているので，(iib) に示された条件のもとに，F の最初の要素から順次ラベルに使用される。また，このアルゴリズムはノードに付けるラベルに F の要素を再利用する可能性がある。すなわち，F の中を左からスキャンしたとき，ある Σ がノードのラベルとして何度も使われる特徴がある。

次に極小のヒット集合を得るために **HS-dag** D の生成時に行う三つの刈り込みを示す。

(i) ノードの再利用 (Reusing nodes): $H(n') = H(n) \cup \{\sigma\}$ となるようなノード n' が存在するならば，n からの枝 σ は n' に連結するようにする。したがって，n' は一つ以上の親を持つことになる。

(ii) ノードを閉じる (Closing): すでに生成されたノード n' が $\surd$ でラベル付けされているとする。このとき，$H(n') \subset H(n)$ となるノード n は閉じる。n のラベルは計算されず，従って子ノードも生成されない。

(iii) 枝の刈り込み (Pruning): 集合 Σ をあるノードのラベルとする。このとき，次の処理を行う。

(a) $\Sigma \subset S'$ となっているラベル $S' \in F$ をもつノード n' が存在するなら，

n' のラベルを Σ に書換える。このとき，$S' - \Sigma$ のすべての要素 α についてそれらをラベルとしてもつ枝は刈り込まれる (カット)。これらの枝に連結する子ノードは，n' 以外の親ノードをもつ場合を除いて除去される (当然この刈り込みは現在処理中のノードが除去される可能性もある)。

(b) 集合族 F の中の S' を Σ と交換する (これは F から S' を取り除くことと同じ効果となる)。

完全な診断が得られる HS-dag 生成の例を示す。次の集合族を考える。

$$\{\{a,b\},\{b,c\},\{a,c\},\{b,d\},\{b\}\}$$

これは，Reiter の HS-木 生成のアルゴリズムでは不完全な診断を示した集合族 F である。この F に対して HS-dag は図 2.5 のようになる。刈り込みされた枝を取り除いて診断を表す HS-dag を示すと図 2.6 となり，

$$\{\{b,a\},\{b,c\}\}$$

が診断として求められる。このアルゴリズムでは $\{b,c\}$ を失うことなく完全な診断が得られている。

HS-dag の形は F の順序付けに依存する。置換 Π による F の再配置を $\Pi(F)$ と表すと，$\mathbf{HS-dag}(F)$ と $\mathbf{HS-dag}(\Pi(F))$ は異なった HS-dag となる。Greiner はこのような場合でも HS-dag は同じ極小ヒット集合を作り出すことを次の定理で示した (証明は文献 [12] 参照)。

定理 2 順序付けられた集合族 F が与えられたとき HS-dag 生成アルゴリズムは次の性質をもつラベル付きの dag を返す。

- $\surd$ でラベル付けられたすべてのノード n について，$H(n)$ は極小ヒット集合である。

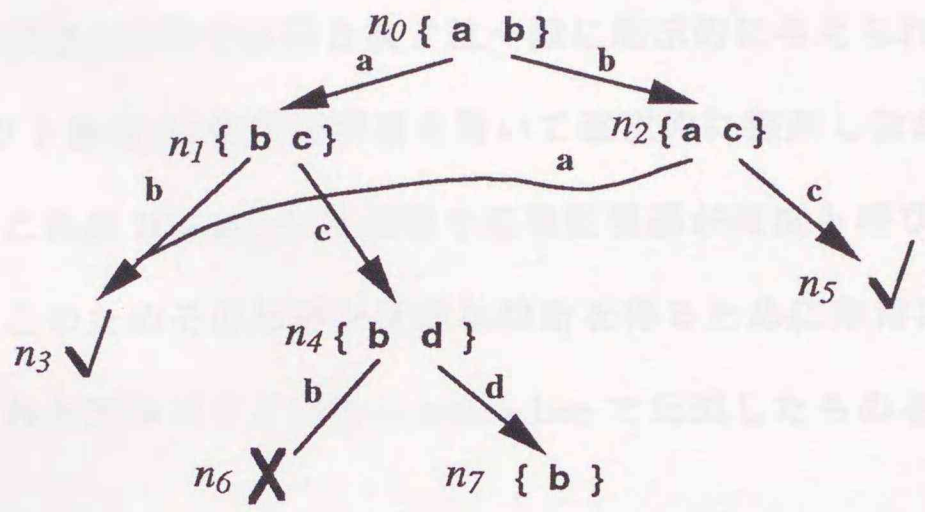


図 2.5: HS-dag 生成の例

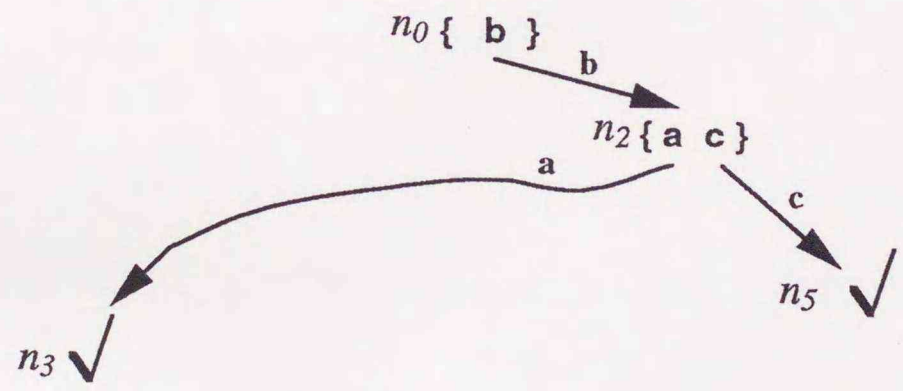


図 2.6: 診断を表す HS-dag

- F のすべての極小ヒット集合は，$\surd$ でラベル付けられたノード n の $H(n)$ となっている。

本論文で示す診断実験はこの改訂されたアルゴリズムを用いている。ここで注意すべきは実際の診断では集合族 F は一般に明示的に与えられない。F の要素 (コンフリクト集合) は定理証明器を用いて逐次的に探索しながら HS-dag を成長させる。これは HS-dag 生成過程で定理証明器が何度も呼び出されることを意味する。このためその効率が高速な診断を得るために非常に重要となる。

なお，改訂されたアルゴリズムを Common Lisp で記述したものを本論文の付録に示してある。

第 3 章

自動推論システム Thinker

診断を 2 章で示したアプローチで得るためには診断対象が適当な論理の集合で表現されていて，かつその集合が矛盾を生じているか否かを検出するために定理証明器と呼ばれるものが必要となる。定理証明器は歴史的には数学定理の自動証明の研究で開発された推論システムであるが，最近では数学の分野に限らず本研究のような診断問題あるいは回路設計・検証，構造モデリングなどその応用範囲は工学の分野にも急速に広がりつつある [14], [5],[24]。このため応用的あるいは独立した汎用推論システムの意味で定理証明器 (theorem prover) という名称よりも自動推論システム (automated resoning system) と呼称される場合が多くなってきている [14]。

本研究による具体的な診断実験は 4, 5 章および 6 章で示される。 4, 5 章では診断対象の知識表現として第一階述語論理に基づく論理式の集合で表す。本章ではこの論理式を扱うことができる自動推論システム **Thinker** の諸機能について述べる。自動推論システムの効率に問題があると実用的な診断システムは構成できない。 **Thinker** では超導出，デモジュレーション， UR 導出など高度な推論規則と各種の戦略を備え効率のよい推論を行うことができる。さらに，**Reiter** の診断アルゴリズムと **Thinker** が組合わされ汎用性の高い診断システム **DiaLog-I** が構成される。なお， **Thinker** の初期バージョンは 1986 年に栗原らが

開発に成功し現在も調整が続けられている [14]。

3.1 知識表現形式

Thinker を利用するためには，問題に関する様々な事実，ルール，仮定は一階述語論理の節形式で表現されている必要がある。例えば，

$$\forall x, y, z(P(x, y)\&Q(y, z) \rightarrow R(x, z))$$

という論理式は節形式で，

$$\neg P(x, y) \mid \neg Q(y, z) \mid R(x, z)$$

と表現する。記号¬は否定，|は選言，&は連言，→は含意を表す。また，x, y, z は変数を表している。

ここで，節が含んでいる正および負リテラルの数に着目し，節を次に示すクラスに分類する。

(i) 空節 (empty clause): リテラルを 1 個も含まない節。矛盾を表す。

(ii) 正単位節 (positive unit clause): 正リテラル 1 個からなる節。ある基本的な肯定的事実が成り立つことを表す。

(iii) 負単位節 (negative unit clause): 負リテラル 1 個からなる節。ある基本的な否定事実が成り立つことを表す。

(iv) 正非単位節 (nonunit positive clause): 2 個以上の正リテラルからなり，負リテラルを含まない節。いくつかの肯定的事実の選言 (or) を表す。

(v) 混合節 (mixed clause): 負リテラル，正リテラル共に 1 個以上含んでいる節。

$$\neg a_1 \mid \neg a_2 \mid \ldots \mid \neg a_m \mid b_1 \mid b_2 \mid \ldots \mid b_n$$

のような混合節は，含意表現 $a \rightarrow b$ を用いた，

$$(a_1\&a_2\& \ldots \&a_m) \rightarrow (b_1 \mid b_2 \mid \ldots \mid b_n)$$

という if-then ルールと考えるとわかりやすい。

(vi) 負非単位節 (nonunit negative clause): 2 個以上の負リテラルからなり，正のリテラルを含まない節。

$$\neg a_1 \mid \neg a_2 \mid \ldots \mid \neg a_m$$

は $a_1, a_2, \ldots, a_m$ が同時には成立しないことを表す。

(ii) および (iii) を単位節 (unit clause)，(iv), (v), (vi) を非単位節 (nonunit clause) という。また，(ii) および (iv) を正節 (positive clause)，(iii) および (vi) を負節 (negative clause) という。

3.2 導出原理による推論

はじめに，最も原理的な導出原理である二元導出について述べる。次に **Thinker** が採用している高度な導出原理に基づく推論規則について述べる。導出原理に関する基本的事項に関しては標準的な解説書を参照されたい [6],[16]。

3.2.1 二元導出

最も基本的な推論規則が二元導出 (binary resolution) である。この推論では二つの親節のそれぞれから相補リテラルを 1 個ずつ取り除き，残りのリテラルから導出節を構成する。例えば二つの節

$$P(a) \mid R(a)$$

$$\neg P(x) \mid Q(x)$$

において，$P(a)$ と $\neg P(x)$ は最汎単一化置換 (most general unifier) $\{x/a\}$ のもとで相補リテラルとなるので，

$$R(a) \mid Q(a)$$

が得られる。

二元導出には効率の面で問題がある。それは，推論規則の適用条件がゆるいので非常に多くの導出節が生成されてしまうことである。例えば，次のような節集合から $R(a)$ を推論する場合を考える。

$$\neg P(x) \mid \neg Q(x) \mid R(x) \tag{3.1}$$

$$P(a) \tag{3.2}$$

$$P(b) \tag{3.3}$$

$$Q(a) \tag{3.4}$$

$$\neg R(c) \tag{3.5}$$

これから得られる導出節は，

(3.1), (3.2) 式より，

$$\neg Q(a) \mid R(a) \tag{3.6}$$

(3.1), (3.3) 式より，

$$\neg Q(b) \mid R(b) \tag{3.7}$$

(3.1), (3.4) 式より，

$$\neg P(a) \mid R(a) \tag{3.8}$$

(3.1), (3.5) 式より，

$$\neg P(c) \mid Q(c) \tag{3.9}$$

(3.6), (3.4) 式より，

$$R(a) \tag{3.10}$$

(3.8), (3.2) 式より，

$$R(a) \tag{3.11}$$

この例から次のことがわかる。

- $R(a)$ を根とする導出木 (演繹木) が二つある。節点集合のみ記すと，

$$\{(3.1), (3.2), (3.6), (3.4), (3.10)\}$$

の各式の集合と，

$$\{(3.1), (3.4), (3.8), (3.2), (3.11)\}$$

である。

- どちらの導出木も深さが 2 であり，目的の節 $R(a)$ を得る途中で中間結果的な節 (3.6) 式および (3.8) 式が生成される。
- 節 (3.7), (3.9) 式はいずれの導出木にも用いられない無駄な節である。

このような問題点を解決するために，親節となる候補に制限を設ける二元導出の変形や戦略が数多く研究されてきた。しかし現在では，二元導出は生成される節が組み合わせ的に急速に増加するので実用的でないとの見方が一般的である。従って，**Thinker** では原則として二元導出は採用せず，超導出および **UR-導出** という強力で実用的な推論規則を採用している。超導出および UR-導出のいずれも 2 個以上の親節から一つの導出節を導くものである。前述の例の場合，両者のいずれの推論規則を用いても以下のことがいえる。

- (3.1), (3.2), (3.4) から 1 ステップの推論 (導出木の深さ 1) でただちに $R(a)$ が得られる。
- $R(a)$ の導出木は上記の一つしかない。
- (3.1) ~ (3.5) 式からは，$R(a)$ 以外の導出節は生成されない。つまり，中間結果や無駄な節は生成されない。

3.2.2 UR-導出

UR-導出は，一つの非単位節 (ここに含まれているリテラルの数を n とする) と $n-1$ 個 (重複を許す) の単位節から一つの単位節を導出する推論規則であ

る [28]。例えば，三つの節

$$\neg\,\mathrm{CONNECT}(x,y) \mid \neg\,\mathrm{CONNECT}(y,z) \mid \mathrm{SAMEVOLTAGE}(x,z)$$

$$\mathrm{CONNECT}(line1, line2)$$

$$\neg\,\mathrm{SAMEVOLTAGE}(line1, line3)$$

より，ただちに次の節が推論される。

$$\neg\,\mathrm{CONNECT}(line2, line3)$$

UR-導出は二元導出に比べて規則の適用条件が厳しいので，生成される節の数がかなり制限される。また，中間的な節は生成されず，最終的に結論された単位節だけが生成される。従って，推論継続の過程で節の数の組合わせ的爆発を軽減するのに非常に有効である。しかも，生成されるのは単位節であるから，問題解決にあたって非常に重要な情報を含んでいることが多い。

3.2.3 超導出

超導出 (hyperresolution) は，一つの混合節または負節 (含まれている負リテラルの個数を m とする) と，m 個 (重複を許す) の正節から一つの正節を導出する推論規則 (正の超導出) である [28]。例えば三つの節，

$$\neg\,\mathrm{CONNECT}(x,y) \mid \neg\,\mathrm{CONNECT}(y,z) \mid \mathrm{SAMEVOLTAGE}(x,z)$$

$$\mathrm{CONNECT}(line1, line2)$$

$$\mathrm{CONNECT}(line2, line3) \mid \mathrm{CONNECT}(line2, line4)$$

より，ただちに次の節が推論される。

$$\mathrm{CONNECT}(line2, line4) \mid \mathrm{SAMEVOLTAGE}(line1, line3)$$

$$\mathrm{CONNECT}(line2, line3) \mid \mathrm{SAMEVOLTAGE}(line1, line4)$$

が生成される。

超導出は **if-then** ルールおよびその前提を満たす事実から肯定的な結論を得るのが目的であり，プロダクション・システムにおける前向き推論 (forward chaining) に類似していることがわかる。

3.3 等式論理による推論

システムを論理的に表現する場合，等号の概念を用いると自然に表現できることが多い。しかし，導出原理に基づく推論規則だけを用いる場合には等号の満たす反射性・対称性・推移性の公理のほか，いま対象としている理論に現れるすべての述語や関数に対して代入則を表す多くの公理を追加しなければならない [6],[16]。**Thinker** では等号を表す述語 **EQUAL** が組込みになっており，等号の概念に基づく推論規則としてデモジュレーション，バックデモジュレーションおよびパラモジュレーションを採用しているので，等号に関する公理を追加する必要がなく効率も向上する。

3.3.1 デモジュレーション

デモジュレーションは，等式 $r = s$ を項 r から s への書換え規則と考え，節を簡単化するのに用いられる [28]。

定義 6 A を **EQUAL**(r, s) の正単位節， B を，r のインスタンスである項 t を含む任意の節とする。デモジュレーションは (*demodulation*) は A を用いて B を次のような節 C に書き換える規則である。なお，このとき B は削除される。

- A 内の変数に適当な項を代入してインスタンス **EQUAL**(r', s') を作る。ただし，$r' = t$ である。
- B 内の項 t を s' に置き換え，C とする。節 A をデモジュレータ (demodulator) という。

例 「サブシステム 1, 2 の両方が正常ならばシステムは正常であり，いずれかが異常ならばシステムは異常である」というルールの論理的表現を考える。

$$\neg\,\mathrm{SUB1}(\mathit{normal}) \mid \neg\,\mathrm{SUB2}(\mathit{normal}) \mid \mathrm{SYSTEM}(\mathit{normal}) \tag{3.12}$$

$$\neg\,\mathrm{SUB1}(\mathit{abnormal}) \mid \mathrm{SYSTEM}(\mathit{abnormal}) \tag{3.13}$$

$$\neg\,\mathrm{SUB2}(\mathit{abnormal}) \mid \mathrm{SYSTEM}(\mathit{abnormal}) \tag{3.14}$$

等式を用いると次のように表現することもできる。

$$\neg\,\mathrm{SUB1}(x) \mid \neg\,\mathrm{SUB2}(y) \mid \mathrm{SYSTEM}(\mathit{check}(x, y)) \tag{3.15}$$

ただし，関数 *check* は次の三つのデモジュレータで定義される。

$$\mathbf{EQUAL}(\mathit{check}(\mathit{normal}, \mathit{normal}), \mathit{normal}) \tag{3.16}$$

$$\mathbf{EQUAL}(\mathit{check}(\mathit{abnormal}, y), \mathit{abnormal}) \tag{3.17}$$

$$\mathbf{EQUAL}(\mathit{check}(x, \mathit{abnormal}), \mathit{abnormal}) \tag{3.18}$$

(3.19)

ここで以下の二つの節

$$\mathrm{SUB1}(\mathit{normal}) \tag{3.20}$$

$$\mathrm{SUB2}(\mathit{abnormal}) \tag{3.21}$$

が与えられたとき，(3.15)~(3.21) 式からどのような推論がなされるかを説明する。はじめに，(3.15), (3.20), (3.21) 式から超導出により

$$\mathrm{SYSTEM}(\mathit{check}(\mathit{normal}, \mathit{abnormal})) \tag{3.22}$$

が導出される。(3.22) 式はただちに (3.18) 式より

$$\mathrm{SYSTEM}(\mathit{abnrmal}) \tag{3.23}$$

に書き換えられる。(3.23) 式が得れた結論である。なお，この例でサブシステムが三つのときは (3.15) 式を次のように変更すればよい。

$$\neg\,\mathrm{SUB1}(x) \mid \neg\,\mathrm{SUB2}(y) \mid \neg\,\mathrm{SUB3} \mid \mathrm{SYSTEM}(\mathit{check}(x, \mathit{check}(y, z))) \qquad (3.24)$$

等号を用いると，この例のように関数が導入され，見通しよくモデル化できることが多い (**Thinker** ではこの関数を組込みとしたり，LISP 関数としてユーザ定義もできる)。また，デモジュレーションでは節は副作用的に書き換えられるので，推論過程で生成される節の数の組合わせ的爆発を軽減できる。すなわち，自動推論システム全体の効率向上に大きく関係してくる重要な推論規則である。そこで，この推論規則についてその停止性および合流性についてさらに検討する。

3.3.1.1 デモジュレーションの停止性

デモジュレーションは等式 $\mathbf{EQUAL}(s,t)$ を左辺のパターン s から右辺のパターン t への書換え規則 (デモジュレータ) とみなし，生成された節中の項を副作用的に書き換える。

推論過程の間に生成された節のうち，等式に対応するものはデモジュレーションの際に適用されるデモジュレータとして登録される。この登録を無秩序に行うと，デモジュレーションがループに陥り停止しない可能性がある。

定義 7 任意の項に対し，どのデモジュレータをどの様な順番でどの部分項に適用しても無限の書換えが生じないとき，これらのデモジュレータの集合は停止性を持つという。

この停止性を保証するために，辞書的経路順序 (lexicographic path ordering, LPO) によって項の集合上に半順序 $\succ_{lpo}$ を定義し，生成された等式が $\langle$ 左辺 $\succ_{lpo}$ 右辺 $\rangle$ を満たすように必要に応じて両辺を入れ換える。

定義 8 $\succ_f$ を関数記号の集合 F 上の半順序とする。F および変数の集合 V から作られる項の集合 $T(F,V)$ 上の辞書的経路順序 $\succ_{lpo}$ は次のように再帰的に定義される [10]。

$s \succ_{lpo} t \Longleftrightarrow$

s は変数でなく，かつ，

- $(s_i \succ_{lpo} t)$ なる $i = 1,\ldots,m$ が存在するか，
- $(f \succ_f g)$ かつすべての $j = 1,\ldots,n$ について $(s \succ_{lpo} t_j)$ であるか，
- $f = g$ かつ $(s_1,\ldots,s_m) \overset{*}{\succ}_{lpo} (t_1,\ldots,t_n)$ かつ，

  すべての $j = 2,\ldots,n$ について $(s \succ_{lpo} t_j)$ である。

ただし，s, t が変数でないとき，$s = f(s_1,\ldots,s_m), t = g(t_1,\ldots,t_n)$ とする。$\overset{*}{\succ}_{lpo}$ は $\succ_{lpo}$ から導かれる辞書的順序である。

定理 3 任意のデモジュレータ **EQUAL**(s,t) が $s \succ_{lpo} t$ を満たすならば，これらの集合は停止性を持つ [10]。

3.3.1.2 デモジュレーションの合流性

デモジュレーションに必要な性質として，停止性の他に合流性がある。これは，ひとつの項を二通り以上に書き換えても，それらを繰り返し書き換えて行くうちに必ず同じ項に書き換えられる性質をいい，次のように定義される。

定義 9 項 s が 0 回以上の書換えで t に書き換えられるとき $s \overset{*}{\Rightarrow} t$ と書く。$s \overset{*}{\Rightarrow} t_1, s \overset{*}{\Rightarrow} t_2$ のように s を二通りに書き換えられるとき，項 u が存在して，$t_1 \overset{*}{\Rightarrow} u, t_2 \overset{*}{\Rightarrow} u$ となるとき，デモジュレータの集合は合流性を持つという。

合流性は最終的な書き換え結果の一意性を保証する [13]。この合流性を検査する手段を定義と定理で示す。

定義 10 二つのデモジュレータ $\mathbf{EQUAL}(s_1[u],t_1),\mathbf{EQUAL}(s_2,t_2)$ を考え，s_2 と u の最汎単一化子を θ とする。ただし，$s_1[u]$ は u を含む項である。項の対 $\langle t_1\theta, s_1[t_2]\theta \rangle$ を**危険対** (*critical pair*) という [13]。

定理 4 (Huet) デモジュレータの集合 D が停止性を満たし，かつすべての危険対 $\langle s,t \rangle$ に対し，s と t の最終的な書換え結果が一致するならば，D は合流性を持つ [13]。

定理 3 と定理 4 により，**Thinker** はユーザーが与えたデモジュレータの停止性と合流性をチェックする。すなわち書換結果は一意に定まりまた無限に項を書換え続けることはない。

デモジュレーションに関連する推論規則にバックデモジュレーション (back demodulation) がある。これは推論の過程の中で，等式が動的に生成されると，それをデモジュレータとして既に存在している節を書き換えの対象とする方式ある。この推論の具体的使用例は **Thinker** の演繹過程のところで説明されている。

3.3.2 パラモジュレーション

パラモジュレーション (paramodulation) は，等式を含む節 $\mathbf{EQUAL}(s,t) \vee C$，および項 s と単一化可能な項 s' を含む節 $D[s']$ とから代入則により，節 $(C \vee D[t])\theta$ を生成する推論規則である。ただし，s,s',t は項，C はリテラルの選言，D は s' を含む節，$D[t]$ は D 中の一つの s' を t で置き換えた節，θ は s と s' の最汎単一化子である。

パラモジュレーションもデモジュレーションと同様，等式を用いて他の節内の項を書き換えるわけであるが，デモジュレーションとの違いは次の二点である。

- デモジュレーションのパターンマッチングが単方向であるのに対し，パラモジュレーションは双方向パターンマッチング (unification) を用いる。
- デモジュレーションでは書換えられた古い節は捨てられる。パラモジュ

レーションでは保存される。

例 1 二つの節

$$\mathbf{EQUAL}(sum(x, minus(x)), 0) \tag{3.25}$$

$$\mathbf{EQUAL}(sum(y, sum(minus(y), z)), z) \tag{3.26}$$

にパラモジュレーションを 1 回適用する。すなわち，(3.25), (3.26) 式に置換

$$\{x/minus(y), z/minus(minus(y))\}$$

が行われ，代入則が用いられて

$$\mathbf{EQUAL}(sum(y, 0), minus(minus(y))) \tag{3.27}$$

が推論される。

例 2 二つの節

$$\mathbf{EQUAL}(f(x, b), x) \vee \neg\, R(x)$$

$$P(g(f(a, x)), x)$$

より $\neg\, R(b) \vee P(g(a), b)$ を得る。

3.4 単位消去

単位消去は，超導出やパラモジュレーションで生成された節 C に対して適用される。C 中の各リテラル L について，$\neg L$ をインスタンスとする単位節がすでに存在するならば，L を C から消去する。例えば単位節 $P(a, x)$ と $\neg\, Q(a)$ が存在しているときに節 $\neg\, P(a, b) \vee R(b) \vee Q(a)$ が生成されると，直ちに二つリテラルが消去され，$R(b)$ を得る。

3.5 包含戦略

超導出，パラモジュレーションなどで生成された節の中には意味的に冗長なものが含まれることがある。これら冗長な節を削除するのに有効なのが節相互の包含関係のチェックである [28],[6]。

定義 11 節 A のインスタンスが節 B に含まれているとき A は B を包含する (*subsume*) という。

A が B を包含するならば $A \rightarrow B$ が成り立つことが知られている。従って，B は冗長なので削除される。

例　三つの節

$$P(x) \tag{3.28}$$

$$P(a) \tag{3.29}$$

$$P(b) \mid Q(b) \tag{3.30}$$

を考える。(3.28) 式は (3.29) 式および (3.30) 式を包含する。従って，(3.29), (3.30) 式は削除される。

3.6 演繹過程の制御

各種の推論規則，戦略を用い **Thinker** は次の手順で演繹を進める [14]。

(i) 三つの節集合 general-axioms, set-of-support, demodulators を準備し，いま考えている問題を表す節をデータとして外部から読み込む。このとき，各節は次の基準で各節集合に配置する。

- いま考えている問題領域で広く成立する一般的な事実やルールを表すもの。$\Longrightarrow$ general-axioms

- 特定の問題に固有の特殊な事実や証明すべき定理の否定など⟹ set-of-support
- デモジュレータ ⟹ demodulators

さらに，一度推論に使われた節を貯える節集合 have-been-given を空にしておく。

(ii) 空節が生成されず，かつ， set-of-support が空でない間，以下の (iii) ~ (vi) を繰返す。もし空節が生成されれば証明成功で推論を停止する。あるいは set-of-support が空になった場合は証明失敗で停止する。

(iii) set-of-support から節を一つ着目し， given-clause とする。given-clause の指定の仕方は任意であるが，ヒューリスティックに基づいて節に重みを付けておき (一般に単位節のように問題解決に重要な情報をもっている節を軽くなるようにする)，最も軽い重みの節を優先的に選ぶのが効果的である。

(iv) あらかじめ指定された推論規則を用いて， given-clause を使って推論される節をすべて求める。推論には given-clause は必ず使い，その他の節は general-axioms か have-been-given から選ぶ。

(v) (iv) で得られた各節 (c とする) に対し，

- デモジュレータを用いて c を可能な限り書き換える (この書換えで新たに得られた節も以下 c と表現する)。
- c が既に存在している節に包含されるならば c を捨てる。そうでなければ c を set-of-support に加える。
- 既に存在している節のうち c に包含される節をすべて削除する。

(vi) (v) で捨てられなかった節のうち等式を表す単位節の集合を new-demodulators とする。この集合を demodulators に付け加える。既に存在する節のうち new-demodulatos の少なくとも一つにより書換えられる節 (d とする) を demodulators 全体を用いて可能な限り書き換える (バックデモジュレーショ

ン)。得られた節を d' とする。d を削除し，d' を set-of-support に付け加える。

(vii) given-clause を set-of-support から have-been-given に移す。

Thinker は現在も推論効率の向上のため調整を継続している。また，現在のバージョンは Common LISP で記述されているのでポータビリティは良好で，Apple 社のパーソナルコンピュータ (Mac II) および Symbolics 社や HP 社の EWS 上で利用可能である。

Thinker はそれ自体独立したインターフェイスを備えたシステムで，ファイルに記述された節の集合を入力として推論を開始できる。もし，演繹過程 (ii) で証明成功で停止 (空節検出) した場合，その証明を表示あるいはファイルに出力することができる。図 3.1 に入力ファイルの例を，図 3.2 に証明の例を示した。この例は，AND, OR, XOR, の各ゲートを任意個用いて全加算器が構成できるかを推論により求めるものである。証明リストの最初の例は XOR, AND ゲート各 2 個と 1 個の OR ゲートで全加算器が構成できることを示している。故障診断実験では **Thinker** の推論部と **Reiter** の診断アルゴリズムとが組合わされ診断システム **DiaLog-I** を構成する。　**DiaLog-I** による実際の診断実験は次章で示される。

```
;;; 1992.6,29
;;; (Thinker "CCL;-Thinker:Thinker.demo3:fulladder-puzzle4.theory")
THEORY FULLADDER-PUZZLE
AXIOMS
 C1   { -OUTPUT(?x1,?x2,?x3,?x4,?x5,?x6,?x7,?x8,?u)
        -OUTPUT(?y1,?y2,?y3,?y4,?y5,?y6,?y7,?y8,?v)
         OUTPUT(and(?x1,?y1),and(?x2,?y2),and(?x3,?y3),and(?x4,?y4),
                and(?x5,?y5),and(?x6,?y6),and(?x7,?y7),and(?x8,?y8),
                andg(?u,?v)) }
 C2   { -OUTPUT(?x1,?x2,?x3,?x4,?x5,?x6,?x7,?x8,?u)
        -OUTPUT(?y1,?y2,?y3,?y4,?y5,?y6,?y7,?y8,?v)
         OUTPUT(or(?x1,?y1),or(?x2,?y2),or(?x3,?y3),or(?x4,?y4),
                or(?x5,?y5),or(?x6,?y6),or(?x7,?y7),or(?x8,?y8),
                org(?u,?v)) }
 C3   { -BOUTPUT(?x1,?x2,?x3,?x4,?x5,?x6,?x7,?x8,?u)
        -BOUTPUT(?y1,?y2,?y3,?y4,?y5,?y6,?y7,?y8,?v)
         BOUTPUT(xor(?x1,?y1),xor(?x2,?y2),xor(?x3,?y3),xor(?x4,?y4),
                 xor(?x5,?y5),xor(?x6,?y6),xor(?x7,?y7),xor(?x8,?y8),
                 xorg(?u,?v)) }
 C4   { -BOUTPUT(?x1,?x2,?x3,?x4,?x5,?x6,?x7,?x8,?u)
          OUTPUT(?x1,?x2,?x3,?x4,?x5,?x6,?x7,?x8,?u) }
 C5   { -OUTPUT (0,1,1,0,1,0,0,1,?u) -OUTPUT (0,0,0,1,0,1,1,1,?v)
         END(?u,?v) }
 C9999  { -END(?u,?v) }
SPECIAL
 C11    { BOUTPUT (0,0,0,0,1,1,1,1,i1)}
 C12    { BOUTPUT (0,0,1,1,0,0,1,1,i2)}
 C13    { BOUTPUT (0,1,0,1,0,1,0,1,i3)}
DEMODULATORS
 C20  { EQUAL(and(0,?x),0) }
 C21  { EQUAL(and(1,?x),?x) }
 C22  { EQUAL(or(0,?x),?x) }
 C23  { EQUAL(or(1,?x),1) }
 C24  { EQUAL(xor(0,1),1) }
 C25  { EQUAL(xor(1,0),1) }
 C26  { EQUAL(xor(?x,?x),0) }
 C27  { EQUAL(and(?x,?x),?x) }
 C28  { EQUAL(and(?x,?y),and(?y,?x),t) }
 C29  { EQUAL(or(?x,?x),?x) }
 C30  { EQUAL(or(?x,?y),or(?y,?x),t) }
 C40  { EQUAL(andg(?x,andg(?x,?y)),andg(?x,?y)) }
 C41  { EQUAL(andg(?y,andg(?x,?y)),andg(?x,?y)) }
 C42  { EQUAL(andg(?x,?x),?x) }
 C43  { EQUAL(andg(?x,?y),andg(?y,?x),t) }
 C44  { EQUAL(andg(:gnd,?x),:gnd) }
 C45  { EQUAL(and(:vcc,?x),?x) }
 C46  { EQUAL(org(?x,org(?x,?y)),org(?x,?y)) }
 C47  { EQUAL(org(?y,org(?x,?y)),org(?x,?y)) }
 C48  { EQUAL(org(?x,?x),?x) }
 C49  { EQUAL(org(?x,?y),org(?y,?x),t) }
 C50  { EQUAL(org(:vcc,?x),:vcc) }
 C51  { EQUAL(org(:gnd,?x),?x) }
 C52  { EQUAL(xorg(?x,?x),:gnd) }
 C53  { EQUAL(xorg(?x,?y),xorg(?y,?x),t) }
;; PARAMODULATORS
;; SUBSUMERS
 WEIGHTS
   BOUTPUT 0    OUTPUT  10   xorg    0   andg    10   org     10   notg    30
  :gnd     50  :vcc     50
OPTIONS
 *HYPERRESOLUTION*    ON    *UR-RESOLVE*          ON    *PARAMODULATION*     OFF
 *DEMODULATION*       ON    *ORTHOPEDICS*         ON    *ORIENTATION*        ON
 *UNIT-DELETION*      ON    *SUBSUMPTION*         ON    *BACK-SUBSUMPTION*   ON
 *BACK-DEMODULATION* OFF   *WEIGHTING*           OFF   *SILENT*             OFF
 *LIMIT-WEIGHT-OF-GENERATED-CLAUSES* OFF
 *TOTAL-PRECEDENCE* ( xorg org andg xor notg i3 i2 i1 or and not 1 0 :vcc :gnd)
 END
```

図 3.1: Thinker 入力ファイル記述例（全加算器の設計）

```
Theory FULLADDER-PUZZLE
Proof-of-205
2 { OUTPUT(0,0,0,0,1,1,1,1,i1) } (C4 C11) 15
3 { BOUTPUT(0,0,1,1,1,1,0,0,xorg(i1,i2)) } (C53 C25 C24 C26 C11 C3 C12) 6
4 { OUTPUT(0,0,1,1,0,0,1,1,i2) } (C4 C12) 15
7 { OUTPUT(0,1,0,1,0,1,0,1,i3) } (C4 C13) 15
8 { BOUTPUT(0,1,1,0,1,0,0,1,xorg(i3,xorg(i1,i2))) } (C53 C25 C24 C26 C13 C3 3) 7
9 { OUTPUT(0,0,1,1,1,1,0,0,xorg(i1,i2)) } (C4 3) 16
17 { OUTPUT(0,1,1,0,1,0,0,1,xorg(i3,xorg(i1,i2))) } (C4 8) 17
23 { OUTPUT(0,0,0,0,0,0,1,1,andg(i1,i2)) } (C43 C21 C20 2 C1 4) 24
31 { OUTPUT(0,0,0,1,0,1,0,0,andg(i3,xorg(i1,i2))) } (C43 C21 C20 7 C1 9) 25
57 { -OUTPUT(0,0,0,1,0,1,1,1,?710) } (C9999 C5 17) 15
205 { } (57 C23 C22 23 31 C2) 0
(REFUTE) took 37533 ticks (625.550 seconds) to run.
Of that, 1421 ticks (23.683 seconds) was spent in GC.

Theory FULLADDER-PUZZLE
Proof-of-234
2 { OUTPUT(0,0,0,0,1,1,1,1,i1) } (C4 C11) 15
3 { BOUTPUT(0,0,1,1,1,1,0,0,xorg(i1,i2)) } (C53 C25 C24 C26 C11 C3 C12) 6
4 { OUTPUT(0,0,1,1,0,0,1,1,i2) } (C4 C12) 15
5 { BOUTPUT(0,1,0,1,1,0,1,0,xorg(i1,i3)) } (C53 C25 C24 C26 C11 C3 C13) 6
7 { OUTPUT(0,1,0,1,0,1,0,1,i3) } (C4 C13) 15
8 { BOUTPUT(0,1,1,0,1,0,0,1,xorg(i3,xorg(i1,i2))) } (C53 C25 C24 C26 C13 C3 3) 7
12 { OUTPUT(0,1,0,1,1,0,1,0,xorg(i1,i3)) } (C4 5) 16
17 { OUTPUT(0,1,1,0,1,0,0,1,xorg(i3,xorg(i1,i2))) } (C4 8) 17
25 { OUTPUT(0,0,0,0,0,1,0,1,andg(i1,i3)) } (C43 C21 C20 2 C1 7) 24
34 { OUTPUT(0,0,0,1,0,0,1,0,andg(i2,xorg(i1,i3))) } (C43 C21 C20 4 C1 12) 25
57 { -OUTPUT(0,0,0,1,0,1,1,1,?710) } (C9999 C5 17) 15
234 { } (57 C23 C22 25 34 C2) 0
(REFUTE) took 5619 ticks (93.650 seconds) to run.
Of that, 245 ticks (4.083 seconds) was spent in GC.


Theory FULLADDER-PUZZLE
Proof-of-265
2 { OUTPUT(0,0,0,0,1,1,1,1,i1) } (C4 C11) 15
3 { BOUTPUT(0,0,1,1,1,1,0,0,xorg(i1,i2)) } (C53 C25 C24 C26 C11 C3 C12) 6
4 { OUTPUT(0,0,1,1,0,0,1,1,i2) } (C4 C12) 15
6 { BOUTPUT(0,1,1,0,0,1,1,0,xorg(i2,i3)) } (C53 C25 C24 C26 C12 C3 C13) 6
7 { OUTPUT(0,1,0,1,0,1,0,1,i3) } (C4 C13) 15
8 { BOUTPUT(0,1,1,0,1,0,0,1,xorg(i3,xorg(i1,i2))) } (C53 C25 C24 C26 C13 C3 3) 7
16 { OUTPUT(0,1,1,0,0,1,1,0,xorg(i2,i3)) } (C4 6) 16
17 { OUTPUT(0,1,1,0,1,0,0,1,xorg(i3,xorg(i1,i2))) } (C4 8) 17
26 { OUTPUT(0,0,0,1,0,0,0,1,andg(i2,i3)) } (C43 C21 C20 4 C1 7) 24
39 { OUTPUT(0,0,0,0,0,1,1,0,andg(i1,xorg(i2,i3))) } (C43 C21 C20 2 C1 16) 25
57 { -OUTPUT(0,0,0,1,0,1,1,1,?710) } (C9999 C5 17) 15
265 { } (57 C23 C22 26 39 C2) 0
(REFUTE) took 6246 ticks (104.100 seconds) to run.
Of that, 266 ticks (4.433 seconds) was spent in GC.
```

図 3.2: Thinker 出力ファイル-証明リスト

第 4 章

診断システム DiaLog

本章では，はじめに，推論系に Thinker を用いて組合せ論理回路の診断を行なう上で考慮すべき知識表現および推論規則と推論戦略について述べる。次に診断の具体例として全加算器を取り上げ実験結果を詳細に示す。その後，診断システム DiaLog-I を構成し，それを用いた 2 ビットデコーダの診断結果についても述べる。

4.1 知識表現

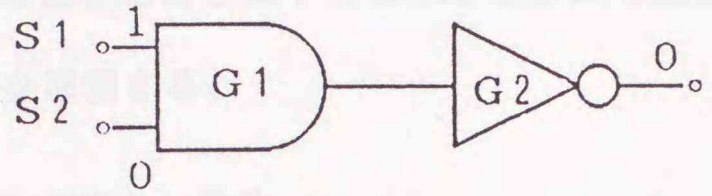
第 2 章で，故障診断は 2.4 式で表せる診断対象システムが矛盾を生じているかを定理証明器で検出し，コンフリクト集合を生成することで得られることを述べた。ここで，診断対象システム SD および OBS は適当な論理で記述される必要がある。本論文ではこれらの記述に第一階述語論理による節形式を用いる。

知識表現の具体例を図 4.1 の論理回路で示す。

4.1.1 COMPONENTS

本論文では，故障し得るのはゲート素子のみと仮定する。図 4.1 の例では，次のようになる。

$$\mathbf{COMPONENTS} = \{G_1, G_2\}$$



C1 ¬ANDG(x) | AB(x) | EQUAL(out(x),and(in1(x),in2(x)))

C2 ¬NOTG(x) | AB(x) | EQUAL(out(x),not(in1(x)))

C3 … EQUAL(in1(G1),out(S1))　C4 … EQUAL(in2(G1),out(S2))

C5 … EQUAL(in1(G2),out(G1))　C6 … EQUAL(not(0),1)

C7 … EQUAL(not(1),0)　C8 … EQUAL(and(0,x),0)

C9 … EQUAL(and(1,x),x)　C10 … EQUAL(and(x,x),x)

C11 … EQUAL(and(x,y),and(y,x))　C12 … EQUAL(x,x)

C13 … ¬EQUAL(1,0)

C14 EQUAL(in1(x),1) | EQUAL(in1(x),0)

C15 EQUAL(in2(x),1) | EQUAL(in2(x),0)

C16 EQUAL(out(x),1) | EQUAL(out(x),0)

C17 ¬EQUAL(in1(x),1) | ¬EQUAL(in1(x),0)

C18 ¬EQUAL(in2(x),1) | ¬EQUAL(in2(x),0)

C19 ¬EQUAL(out(x),1) | ¬EQUAL(out(x),0)

C20 … ANDG(G1)　C21 … NOTG(G2)

C22 … EQUAL(out(S1),1)　C23 … EQUAL(out(S2),0)

C24 … EQUAL(out(G2),0)　C25 … ¬AB(G1)

C26 … ¬AB(G2)

図 4.1: 簡単な論理回路と知識表現

4.1.2 システム記述 SD

SD としてはシステム構成要素の動作，回路接続情報，ブール代数公理，等式の公理，二値性の記述および素子の種別を節形式で記述する。図 4.1 の例では以下のような節を準備する。

(i) システム構成要素の動作: $\{C_1, C_2\}$

(ii) 回路接続情報に関する節: $\{C_3, C_4, C_5\}$

(iii) ブール代数公理: $\{C_6, \ldots, C_{11}\}$

(iv) 等式の公理: $\{C_1\}$

(v) 二値性の記述: $\{C_{12}\}$

(vi) 素子の種別: $\{C_{20}, C_{21}\}$

ここで記号 **ANDG**(x), **NOTG**(x) はそれぞれ，素子 x (変数) が AND ゲート，NOT ゲートであることを表わす述語，**AB**(x) は素子 x が異常 (abnormal)，**EQUAL**(x, y) は x と y が等しいことを意味する述語である (EQUAL は Thinker に組み込まれている特別な述語，AB は故障診断システムが認識する特別な述語である。本論文で現われるそれ以外の述語記号，関数記号，定数記号はすべて任意に定めたものであり，Thinker あるいは故障診断推論システムがあらかじめ特別に定めている解釈はない)。また，記号 $in1(x), in2(x), out(x)$ は，それぞれ素子 x での第１入力，第２入力，出力の値を表している。例えば，節 C_1 は，「x が AND ゲートで，かつ x が異常でなければ，x の出力は x への二つの入力の and に等しい」という論理式，

$$\mathbf{ANDG}(x) \wedge \neg \mathbf{AB}(x) \supset \mathbf{EQUAL}(out(x), and(in1(x), in2(x))) \tag{4.1}$$

を節形式で表現している。ただし，関数 and の意味は節 $C_8, \ldots, C_{11}$ で規定される。また，記号 $\neg, \wedge, |, \supset$ はそれぞれ否定，連言，選言，含意を表す (なお，not は節 C_6, C_7 で規定される $\{0,1\}$ 上の関数を表すためユーザが導入した記号，$\neg$ は

論理式の否定を表すため，あらかじめ Thinker に組込まれた記号である)。回路の入力は仮想信号源 S_1, S_2 の出力から与えられているものとする。

任意の組合せ論理回路に対しても同様の方法で系統的に SD を生成できることが明らかであろう。

4.1.3 OBS の表現例

診断対象システムから現在得られている観測値を節形式で準備する。図 4.1 では，$C_{22}, \ldots, C_{24}$ となる。$\mathrm{out}(G_2)$ はこの回路の出力である。

4.1.4 仮説

診断開始の時点では，C_{25}, C_{26} で示すようにすべての回路素子は故障していないことを示す仮説を置く。

4.2 推論規則と戦略

故障診断を効率的に進めるには自動推論システムが強力な推論規則および戦略をもつ必要がある。現在，Thinker では推論規則として，超導出，パラモジュレーション，デモジュレーション，逆デモジュレーションおよび単位消去の各推論規則を同時にあるいは選択的に使用できる。また推論戦略としては，支持集合戦略，トートロジー除去戦略，包含戦略，EQUAL リテラル向き付け戦略，および節重み付け戦略を実行できる。これらのなかで，診断効率を向上させるために重要なのは，パラモジュレーション，デモジュレーションなどの等式論理に基づく推論規則と戦略である。ここでは，デモジュレーションの停止性を保証するための辞書的経路順序を具体的に決定し，さらにこれと深くかかわる EQUAL リテラル向き付け戦略，両方向デモジュレーションについて述べる。

4.2.1 辞書的経路順序

$\succ_{lpo}$ は以下に示すルールを満たす全順序のうち一つの $\succ_f$ により定めることにした。ここで，ゲートの入力をあらわす関数記号の集合を **IN**，ブール関数を表す関数記号の集合を **BOOL** とする。

rule-1 $C \in$ **COMPONENTS** $\cup \{S_1, \ldots, S_n\}$, $I \in$ **IN**, $B \in$ **BOOL** ならば，$C \succ_f out \succ_f I \succ_f B \succ_f 1 \succ_f 0$.

rule-2 $x, y \in$ **COMPONENTS** $\cup \{S_1, \ldots, S_n\}$，かつ x の出力が y の入力に接続されているならば，$y \succ_f x$.

rule-2 は信号の流入先の素子 y の情報を，流出先の素子 x を用いた表現に書き換えようとの意図である。**rule-1** は，**rule-2** すなわち回路の接続情報で定まる大小関係を最優先に考慮して比較し，以下，**out**,**in1** および **in2**,... の順に小さくなることを意味する。直観的には ($\succ_f$ の意味で) 大きな記号を多く用いた表現は，デモジュレーションにより，小さな記号を多く用いた表現に書き換えられる。

ここで定められた $\succ_{lpo}$ により，デモジュレータに指定された節の集合は停止性を満たすことが証明できる。

例えば，図 4.1 では，

$$G_2 \succ_f G_1 \succ_f S_1 \succ_f S_2 \succ_f out \succ_f in1 \succ_f in2 \succ_f and \succ_f not \succ_f 1 \succ_f 0 \qquad (4.2)$$

とすると，

$$in1(G_1) \succ_{lpo} out(S_1)$$

など，節 C_{11} 以外の等式の左辺が必ず右辺より大きいことがわかる。さらに，その合流性も容易に示される。

また，$\succ_{lpo}$ は推論の過程においても次に示す戦略で使用される。

4.2.2 EQUALリテラル向き付け戦略

推論規則で導出された節がEQUALリテラルを含む場合，EQUALの左辺(第1引数)と右辺(第2引数)を辞書的経路順序で比較し，大小判定が可能ならば左辺を必ず大きくなるようにする戦略である。この戦略は，EQUALリテラルの表現が規格化されるため，超導出や単位消去の推論を行なうときに有効になる。また特にこの節が正の単位節の場合は，この節を新たにデモジュレータとして登録することになるが，その際，停止性が損なわれないことの保証を行なうのにも本質的な役割を果たす。

4.2.3 両方向デモジュレーション

C_{11} のような交換則を表わすデモジュレータ(両方向デモジュレータ)は書換えの停止性を満足しない恐れがある。この場合，左辺と右辺のインスタンスが，$\langle$左辺 $\succ_{lpo}$ 右辺$\rangle$ であるときのみ左辺を右辺に書換えるようにした。

4.3 全加算器の故障診断

図4.2に診断例とした全加算器を示す。また，この回路のSD,OBSおよび仮説を節形式で準備したものを図 4.3 に示した。ここで，

- **SD** $= \{C_{27}, \ldots, C_{66}\}$
- **COMPONENTS** $= \{A_1, A_2, X_1, X_2, O_1\}$
- **OBS** $= \{C_{61}, \ldots, C_{71}\}$

となる。図4.3からわかるように，回路規模が増大しても，知識ベースの構築に関しては機械的な変更を行うだけで故障診断に対応できる。

次に節の配置については試行錯誤の結果以下のようにした。

- 一般公理: $\{C_{27}, C_{28}, C_{29}, C_{45}, C_{46}, C_{47}, C_{49}, C_{50}, C_{51}, C_{72}, \ldots, C_{76}\}$

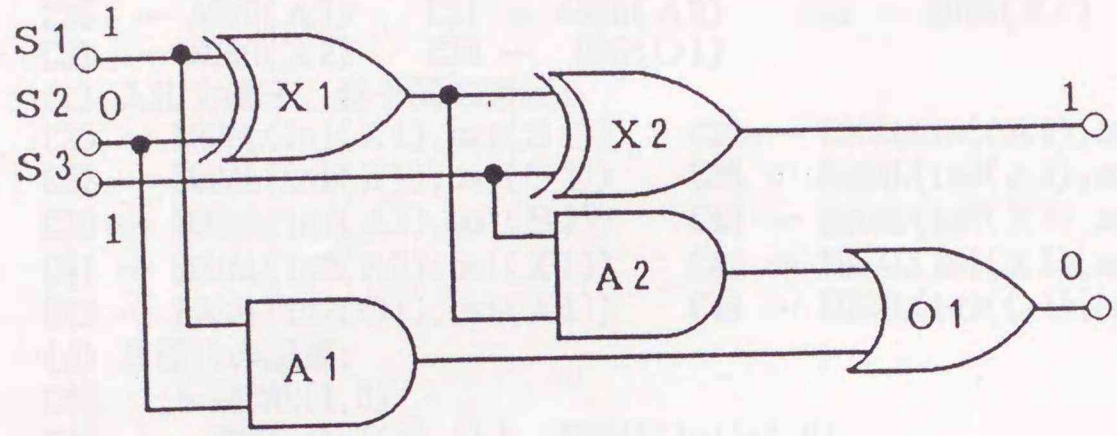


図 4.2: 故障診断回路例-全加算器

- 支持集合: $\{C_{30}, \ldots, C_{34}\}$
- デモジュレータ: $\{C_{35}, \ldots, C_{44}, C_{52}, \ldots, C_{62}, C_{67}, \ldots, C_{71}\}$
- 両方向デモジュレータ: $\{C_{63}, C_{64}, C_{65}\}$
- パラモジュレータ: $\{C_{48}\}$
- 包含節集合: $\{C_{66}\}$

すなわち，一般に

(i) 各素子の種別を表す節を支持集合とする。

(ii) 等式の公理を包含節集合に，交換則を表す等式を両方向デモジュレータにし，それ以外の等式をデモジュレータにする。

(iii) ゲートの出力が 1 または 0 であることを表す節のみをパラモジュレータとする。

```
(a) ゲートの機能:
C27  ¬ANDG(x) | AB(x) | EQUAL(out(x),and(in1(x),in2(x)))
C28  ¬XORG(x) | AB(x) | EQUAL(out(x),xor(in1(x),in2(x)))
C29   ¬ORG(x) | AB(x) | EQUAL(out(x), or(in1(x),in2(x)))
(b) 各素子の種別:
C30  … ANDG(A1)    C31 … ANDG(A2)    C32 … XORG(X1)
C33  … XORG(X2)    C34 …  ORG(O1)
(c) 入出力端子、素子間の接続:
C35 … EQUAL(in1(X1),out(S1))    C36 … EQUAL(in1(A1),out(S1))
C37 … EQUAL(in2(X1),out(S2))    C38 … EQUAL(in2(A1),out(S2))
C39 … EQUAL(in1(A2),out(S3))    C40 … EQUAL(in2(X2),out(S3))
C41 … EQUAL(in2(A2),out(X1))    C42 … EQUAL(in1(X2),out(X1))
C43 … EQUAL(in2(O1),out(A1))    C44 … EQUAL(in1(O1),out(A2))
(d) 2値性の記述:
C45    ¬EQUAL(1,0)
C46      EQUAL(in1(x),1) |  EQUAL(in1(x),0)
C47      EQUAL(in2(x),1) |  EQUAL(in2(x),0)
C48      EQUAL(out(x),1) |  EQUAL(out(x),0)
C49    ¬EQUAL(in1(x),1) | ¬EQUAL(in1(x),0)
C50    ¬EQUAL(in2(x),1) | ¬EQUAL(in2(x),0)
C51    ¬EQUAL(out(x),1) | ¬EQUAL(out(x),0)
(e) ブール代数公理:
C52 … EQUAL(not(0),1)           C53 … EQUAL(not(1),0)
C54 … EQUAL(and(0,x),0)         C55 … EQUAL(and(1,x),x)
C56 … EQUAL(and(x,x),x)         C57 … EQUAL( or(0,x),x)
C58 … EQUAL( or(1,x),1)         C59 … EQUAL( or(x,x),x)
C60 … EQUAL(xor(0,x),x)         C61 … EQUAL(xor(1,x),not(x))
C62 … EQUAL(xor(x,x),0)         C63 … EQUAL(and(x,y),and(y,x))
C64 … EQUAL( or(x,y), or(y,x))  C65 … EQUAL(xor(x,y),xor(y,x))
(f) 等式の公理:
C66    EQUAL(x,x)
(g) システム観測:
C67 … EQUAL(out(S1),1)          C68 … EQUAL(out(S2),0)
C69 … EQUAL(out(S3),1)          C70 … EQUAL(out(X2),1)
C71 … EQUAL(out(O1),0)
(h) 仮説:
C72 … ¬AB(A1)    C73 … ¬AB(A2)    C74 … ¬AB(X1)
C75 … ¬AB(X2)    C76 … ¬AB(O1)
```

図 4.3: 全加算器の知識表現

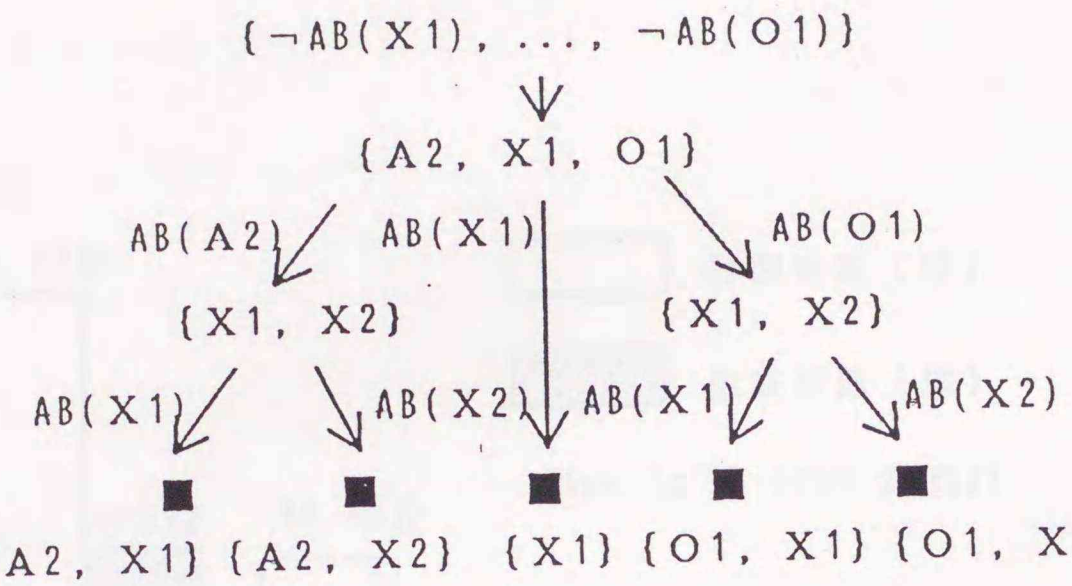


図 4.4: 全加算器診断 HS-木

4.3.1 診断実験結果

4.2 節で述べた推論規則，戦略で定理証明器 Thinker を実行すると， Thinker は空節を導出 (証明成功) して停止した。証明を以下に示す。

$\{C_{29}, C_{34}, C_{43}, C_{44}, C_{64}, C_{71}, C_{76}\}$ より，ただちに，

C_{77} **EQUAL**$(or(out(A_1), out(A_2)), 0)$

を得た。この導出に使われた推論規則を説明する。

初めに C_{29} と C_{34} の超導出により，

$$\mathbf{AB}(O_1) \mid \mathbf{EQUAL}(out(O_1), or(in1(O_1), in2(O_1)))$$

が生成される。これに $\{C_{43}, C_{44}, C_{64}, C_{71}\}$ のデモジュレーション，および **EQUAL** リテラル向き付け戦略により，

$$\mathbf{AB}(O_1) \mid \mathbf{EQUAL}(or(out(A_1), out(A_2)), 0)$$

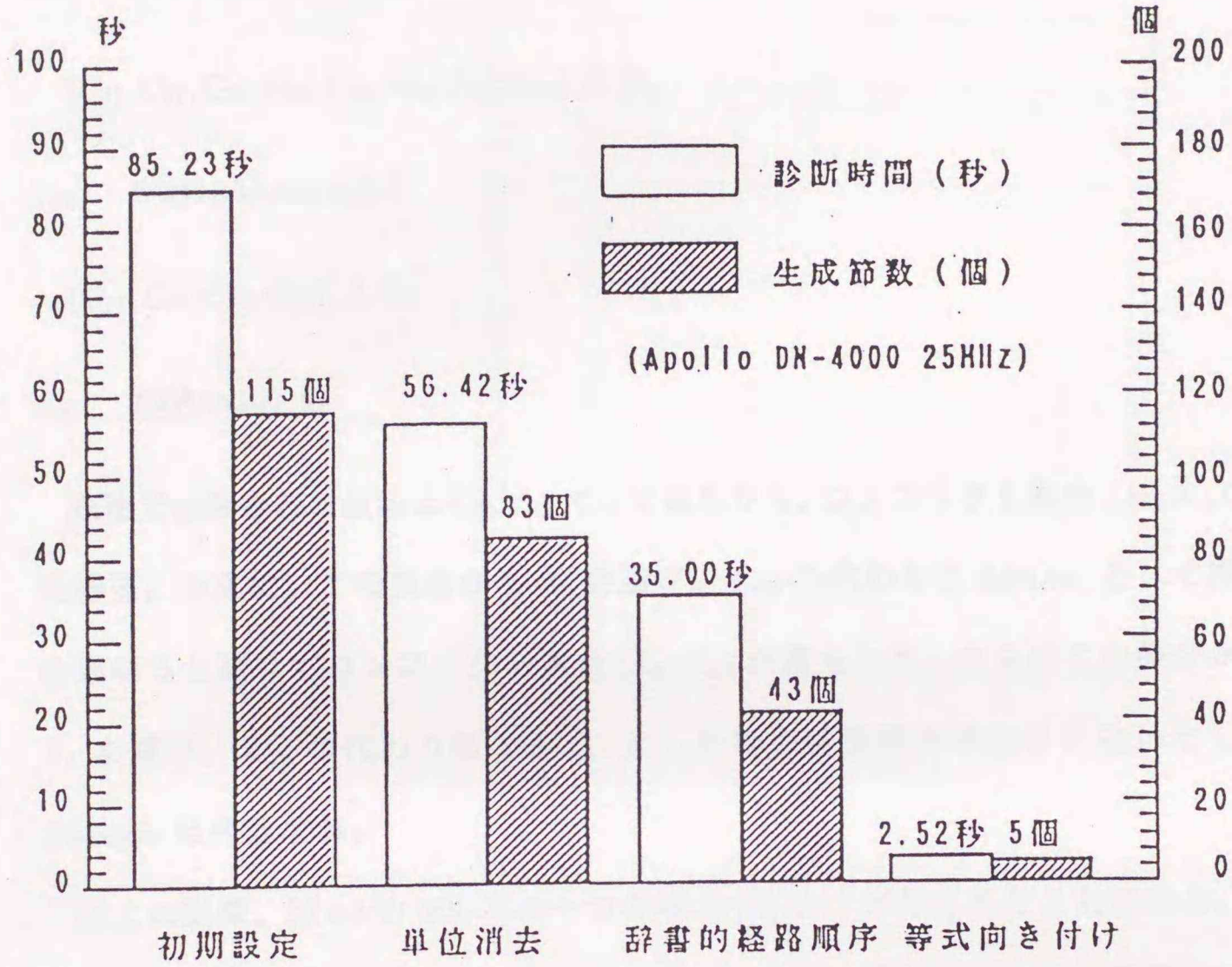


図 4.5: 全加算器診断時間と生成節数

を得る。その後，C_{76} により単位消去が行なわれて最終的に C_{77} が結果として得られる。このとき，途中で生成された中間結果の節はすべて消去される。

$\{C_{32}, C_{28}, C_{35}, C_{67}, C_{37}, C_{68}, C_{61}, C_{52}, C_{74}\}$ より，

C_{78} **EQUAL**$(out(X_1), 1)$

$\{C_{31}, C_{27}, C_{39}, C_{69}, C_{41}, C_{78}, C_{55}, C_{73}\}$ より，

C_{79} **EQUAL**$(out(A_2), 1)$

$\{C_{77}, C_{64}, C_{58}, C_{79}\}$ より，

C_{80} **EQUAL**$(1, 0)$

証明で参照された仮説は C_{73}, C_{74}, C_{76} であるから，コンフリクト集合 $\{A_2, X_1, O_1\}$ を得る。つぎに，この集合から A_2 を選び，C_{73} の代わりに **AB**(A_2) として推論を進めると続いてコンフリクト集合 $\{X_1, X_2\}$ が得られた。さらにこの集合から X_1 を選び，C_{74} の代わりに **AB**(X_1) とした場合は空節が検出されないとして **Thinker** は停止する。

以上の結果，図 4.4 の **HS-木**の一つの端点 $\{A_2, X_1\}$ が得られたことになる。これは素子 A_2 故障かつ X_1 故障ならばシステム記述と観測のあいだに矛盾はないことを意味する。

同様の手順を繰り返すと，図 4.4 が得られる。**HS-木**の端点は 2.2 式を無矛盾とする Δ の要素となっている。これらのうち極小のもの，すなわち $\{X_1\}, \{A_2, X_2\}, \{O_1, X_1\}$ が最終的に得られる診断である。

ここで，最初のコンフリクト集合 $\{A_1, X_1, O_1\}$ が得られるまでの実行時間と，生成された節の数を図 4.5 に示す。図 4.5 では，単位消去，辞書的経路順序および **EQUAL** リテラル向き付け戦略の効果を表している。初期設定とはこれらをすべて行わない場合である。適切な推論規則，戦略を採用すれば推論効率が著しく改善されるのがわかる。

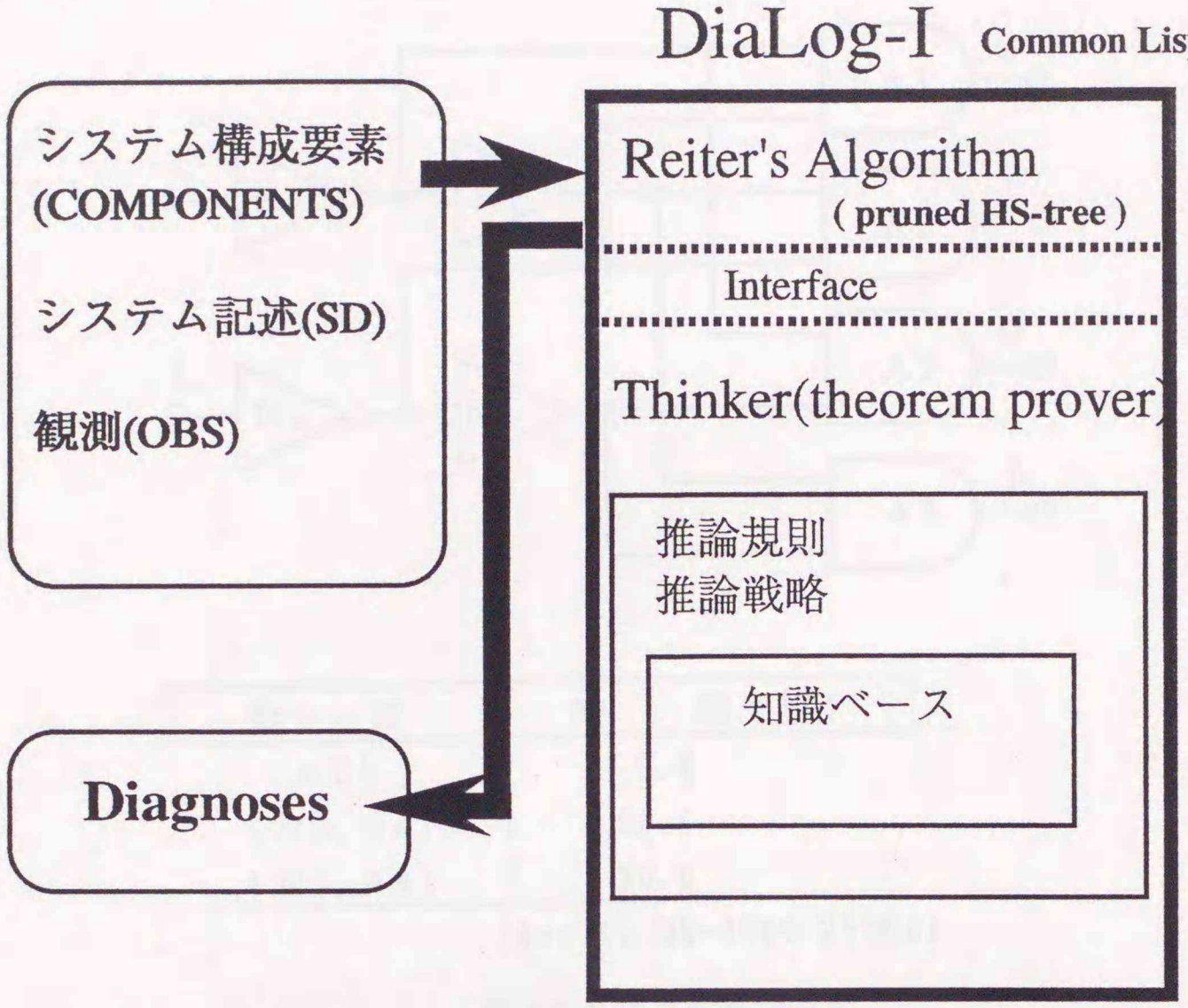
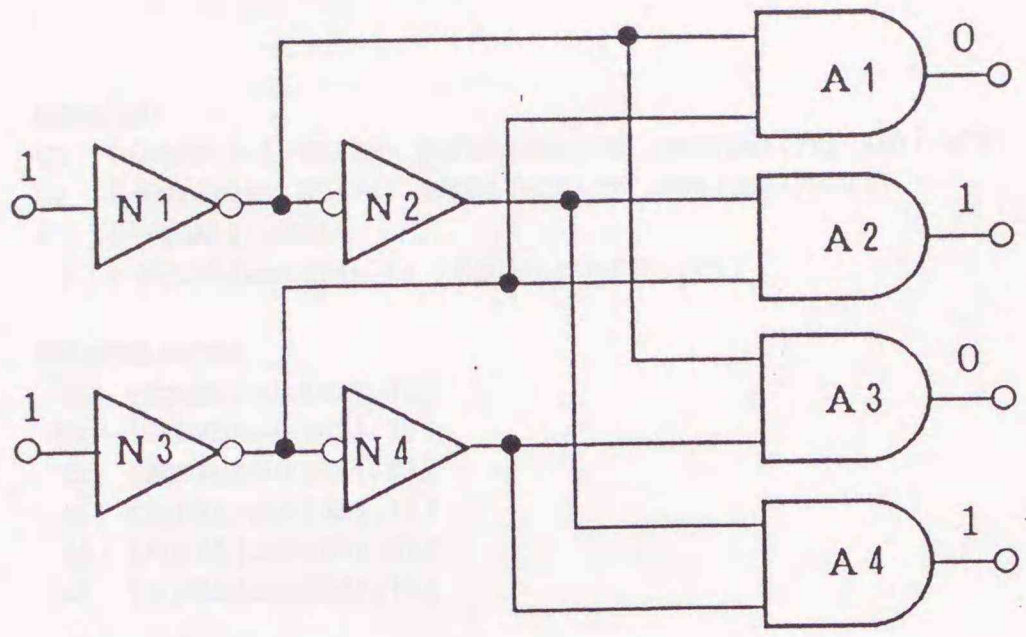


図 4.6: 診断システム DiaLog-I の構成

4.4 診断システムの構成と診断実験

4.4.1 DiaLog-I の構成

全加算器の診断実験を通して Thinker が備える超導出，単位消去および EQUAL リテラル向き付け戦略などが推論の効率向上に効果があることがわかった。そこで，第 2 章でのべた Reiter の診断アルゴリズムと組合わせた診断システム DiaLog-I を作成した。このシステムは common lisp で記述されている。 DiaLog-I の構成は図 4.6 のようになっている。外部 (ファイルに記述) から入力された診断対象，すなわちシステム構成要素 (COMPONENTS)，システム記述 (SD) および観測 (OBS) は知識ベースに蓄えられる。DiaLog-I が起動されると，はじめは知識ベースの COMPONENTS はすべて正常であるとの仮定のもと



診　　断	時　　間（秒）
{A2}	9.8
{N3, N4}	26.5
{N3, A4}	30.0

(Apollo DN-4000 25MHz)

図 4.7: 2 ビットデコーダ

に Thinker に知識ベースの矛盾を検査させる (反駁手続き)。 Thinker は，知識ベースが充足可能 (無矛盾) であればそれを示す記号 (NIL)，充足不能であればコンフリクト集合を診断部へ返す。診断アルゴリズムはこれらを用いて HS-木 を展開し診断を構成する。

4.4.2　2 ビットデコーダの診断

DiaLog-I による診断例である 2 ビットデコーダの回路を図 4.7 に示す。明らかにこの回路には故障素子が存在する。図 4.8 はこの診断に用いた入力ファイルである。全加算器の故障診断で得られたヒューリスティクスを利用して節を配置した。また，DiaLog-I の診断実行の様子を図 4.9 に示した。このバージョンは診断の過程で HS-木 のどのような刈り込みが行われているかを示すようになっている。

```
GENERAL
C1   {-ANDG(?x) AB(?x) EQUAL(out(?x),and(in1(?x),in2(?x)))}
C2   {-NOTG(?x) AB(?x) EQUAL(out(?x),not(in1(?x)))}
C3   {-EQUAL(1,0)}
C4   {-EQUAL(out(?x),1) -EQUAL(out(?x),0)}

DEMODULATORS
 M0   {EQUAL(out(S1),1)}
 M1   {EQUAL(out(S2),1)}
 M2   {EQUAL(out(A1),0)}
 M3   {EQUAL(out(A2),1)}
 M4   {EQUAL(out(A3),0)}
 M5   {EQUAL(out(A4),1)}

C10   {EQUAL(in1(N1),out(S1))}
C11   {EQUAL(in1(N3),out(S2))}
C12   {EQUAL(in1(N2),out(N1))}
C13   {EQUAL(in1(A1),out(N1))}
C14   {EQUAL(in1(A3),out(N1))}
C15   {EQUAL(in1(N4),out(N3))}
C16   {EQUAL(in2(A1),out(N3))}
C17   {EQUAL(in2(A2),out(N3))}
C18   {EQUAL(in1(A2),out(N2))}
C19   {EQUAL(in1(A4),out(N2))}
C20   {EQUAL(in2(A3),out(N4))}
C21   {EQUAL(in2(A4),out(N4))}
```

図 4.8: DiaLog-I インプットファイル (2 ビットデコーダ)

診断結果は図 4.7 にあるように一個の単一故障と二個の二重故障が得られた。人間にとって素子 A2 の故障は直感的に判断できる。しかし，A2 が正常である と判明したとき，残りの素子の異常を仮定して現在の回路の状況を説明するのは簡単ではないと考えられる。

4.5 システム観測が不完全な場合の故障診断

4.3 節で示した全加算器の診断で，診断環境に問題があるような比較的高度な故障診断が必要となる場合を示す。特に，次に示すような不完全なシステム観測での推論はこれまで例がない。

```
      DiaLog for Macintosh II Ver.1.0

      Use the HELP command to display the list of all the DiaLog commands.

DIALOG-TP NIL
CONFLICT-Set: (N3 N2 N1 A2)
(REFUTE) took 476 ticks (7.933 seconds) to run.

DIALOG-TP (N3)
CONFLICT-Set: (N4 N2 N1 A4 A2)
(REFUTE) took 503 ticks (8.383 seconds) to run.

DIALOG-TP (N2)
CONFLICT-Set: (N3 A2)
(REFUTE) took 357 ticks (5.950 seconds) to run.

PRUNING TREE:: (n0)-N1-
PRUNING TREE:: (n0)-N2-(n2)-N3-
PRUNING TREE:: (n0)-N2-(n2)-A2-
DIALOG-TP (A2)
CHECKED
(REFUTE) took 792 ticks (13.200 seconds) to run.

DIALOG-TP (N4 N3)
CHECKED
(REFUTE) took 969 ticks (16.150 seconds) to run.
Of that, 208 ticks (3.467 seconds) was spent in GC.

DIALOG-TP (N2 N3)
CONFLICT-Set: (N4 A4 A2)
(REFUTE) took 857 ticks (14.283 seconds) to run.

PRUNING TREE:: (n0)-N3-(n1)-N1-
PRUNING TREE:: (n0)-N3-(n1)-N2-(n5)-N4-
PRUNING TREE:: (n0)-N3-(n1)-N2-(n5)-A4-
PRUNING TREE:: (n0)-N3-(n1)-N2-(n5)-A2-
DIALOG-TP (A4 N3)
CHECKED
(REFUTE) took 773 ticks (12.883 seconds) to run.

DIAGNOSES COMPONENTS::
(A2)
(N4 N3)
(A4 N3)
?
```

図 4.9: DiaLog-I の実行

4.5.1 システム観測の欠損

4.3節では4.3(g)に示すように全加算器の入出力端子でのシステム観測が完全に揃っていた場合の故障診断を示した。ここでは，入力側の観測値を2個失った場合の推論結果を示す。

4.5.1.1 システム観測

4.3(g)のみを次のようにする。その他は同様である。この例では回路は故障している。

C_{81} **EQUAL**$(out(S_3), 0)$

C_{82} **EQUAL**$(out(X_2), 1)$

C_{83} **EQUAL**$(out(O_1), 1)$

4.5.1.2 実験結果

以下に得られた証明を示す。

$\{C_{34}, C_{29}, C_{83}, C_{44}, C_{43}, C_{64}, C_{76}\}$ より，

C_{84} **EQUAL**$(or(out(A_1), out(A_2)), 1)$

$\{C_{33}, C_{28}, C_{82}, C_{42}, C_{40}, C_{81}, C_{65}, C_{60}, C_{75}\}$ より，

C_{85} **EQUAL**$(out(X_1), 1)$

$\{C_{32}, C_{28}, C_{85}, C_{35}, C_{37}, C_{65}, C_{74}\}$ より，

C_{86} **EQUAL**$(or(out(S_2), out(S_1)), 1)$

$\{C_{31}, C_{27}, C_{39}, C_{81}, C_{41}, C_{85}, C_{54}, C_{73}\}$ より，

C_{87} **EQUAL**$(out(A_2), 0)$

$\{C_{87}, C_{84}, C_{64}, C_{57}\}$ より，

C_{88} **EQUAL**$(out(A_1), 0)$

$\{C_{30}, C_{27}, C_{88}, C_{36}, C_{38}, C_{63}, C_{72}\}$ より，

C_{89} **EQUAL**$(and(out(S_2), out(S_1)), 1)$

$\{C_{48}, C_{89}, C_{54}, C_{45}\}$ より，

C_{90} **EQUAL**$(out(S_2), 1)$

ここで，C_{90}の導出を説明する。

C_{48} の二つめの等号リテラルの左辺 $out(x)$ から C_{89} 中の項 $out(S_2)$ へのパラモジュレーションにより，

$$\textbf{EQUAL}(and(0, out(S_1)), 1) \mid \textbf{EQUAL}(out(S_2), 1)$$

が新たに生成される。これに C_{54} によるデモジュレーションと 4.2.2 節で述べた EQUAL リテラル向き付け戦略により，

$$\textbf{EQUAL}(1, 0) \mid \textbf{EQUAL}(out(S_2), 1)$$

を得る。さらに C_{45} により単位消去され C_{90} が導出される。これは，「AND ゲートの出力が 1 ならばその入力は 1 である」ことを導いたことになる。さらに，以下のように推論が続く。

$\{C_{48}, C_{89}, C_{63}, C_{54}, C_{45}\}$ より，

C_{91} **EQUAL**$(out(S_1), 1)$

最後に，$\{C_{48}, C_{90}, C_{91}, C_{62}, C_{45}\}$ より，空節を導出して停止。以上の証明過程からわかるように，与えられていなかった入力側の二個の観測値 $out(S_2), out(S_1)$ を導出して証明に成功した。

これまでの導出例から次のことが言える。

- 多数の節が一度に関与して人間からみても自然な結論が推論されている。
- 陽に後向きのルールが記述されていないのにもかかわらず，パラモジュレーションという等式に対する強力な推論規則により，後向き推論が行

なわれた。

最終的な診断として，$\{X_1\},\{X_2\},\{A_1\},\{A_2\},\{O_1\}$ の五つを得る。すなわち，いずれかの素子の”単一故障”で回路の異常を説明できる。

4.5.2 非数値的システム観測

これまで観測に必ず $0,1$ の定数を含むものを扱った。次に示すのは定数をまったく含まない例である。

4.5.2.1 システム観測

4.3(g) を次のようにする。

$$\mathbf{EQUAL}(out(S_1), out(S_2))$$

$$\mathbf{EQUAL}(out(S_2), out(S_3))$$

$$\neg\mathbf{EQUAL}(out(X_2), out(O_1))$$

この例も全加算器としては故障である。

4.5.2.2 実験結果

途中の導出の過程は省略するが，

$$\mathbf{EQUAL}(out(X_1), 0)$$

$$\mathbf{EQUAL}(out(A_2), 0)$$

などを生成し，最終的にはシステム観測の欠損の場合と同じ診断結果が得られた。

以上，システム観測が不完全，あるいは非数値的な場合の実験結果を示した。これは次の二つの状況で有用である。

(i) システム観測用センサーが故障の場合

(ii) 入出力値を直接には観測できない場合

(i) は一般にリアルタイム故障診断で生起し得る事態であり，観測値の一部が得られなくなる。(ii) は算術論理演算ユニット (ALU) のように，全加算器 (FA) を内部に埋め込んでいる装置の故障診断の際に起こり得る。この場合，システム観測として ALU の入出力値のみが与えられると，FA の入出力値ははじめのシステム観測から推論により求めるしかない。この場合，FA のレベルにおいてはシステム観測が不完全あるいは非数値的となる。ここで示したような，知識 (観測) に問題があっても可能な限り推論を継続できる故障診断システムの開発は，診断システムをより頑健 (robust) で信頼性の高いものにするためにも重要となる。

4.6 議論

本章の実験で用いた知識表現と推論方式をまとめ，本研究の新規性と有用性を議論する。

4.6.1 知識表現

本研究では知識表現として論理を用いている。その第一の理由は Reiter の故障診断理論が論理をベースにしているからである。論理は曖昧な知識の表現に適しておらず，また，論理的推論は一般に効率が良くない (と信じられている) こともあり，論理が最良かつ唯一の知識表現であるとは思わない。しかし，論理回路に関する知識は (文字通り) 十分論理的であり，本章で述べたようにに比較的効率よい推論が可能なので，論理的知識表現が適している分野の一つであると考える。

実用システムの多くは主としてプロダクション・システム (production system, PS) を用いているが，推論システムの基礎という観点から見た場合，PS には一

つ重要な欠陥があることを指摘したい。それは推論の不完全性である(知識の集合 A の任意の論理的帰結 p を推論できるシステムは完全であるという)。以下では論理回路に関する推論のごく自然な状況においてこの問題が生じ得ることを具体的に示す。

入力端子 a, b と出力端子 c を持つゲートを考える。x と y のブール積が z であることを $AND(x,y,z)$ と書き，次の四つの事実で表現する。

$$AND(0,0,0) \tag{4.3}$$

$$AND(0,1,0) \tag{4.4}$$

$$AND(1,0,0) \tag{4.5}$$

$$AND(1,1,1) \tag{4.6}$$

端子 t の値が v であることを $V(t,v)$ で表すと，次のプロダクション・ルールを得る。

rule-1 もし，$V(a,x)$ かつ $V(b,y)$ かつ $AND(x,y,z)$ ならば $V(c,z)$

ここで次のように入力端子 a の値を 0 とする。

$$V(a,0) \tag{4.7}$$

PS はこれらの知識から出力が 0 であることを推論できない。なぜなら，ルール中の $V(b,y)$ にマッチする事実が存在しないからである。実際には，この推論に成功するには，すべての値は 1 か 0 であるという知識

$$V(x,0) \mid V(x,1) \tag{4.8}$$

を加える必要があるが，それでも上の状況は変らない。

完全性を有する論理的推論は，この問題を解決する。まず **rule-1** を論理の節形式に変換しておく。

$$\neg V(a,x) \mid \neg V(b,y) \mid \neg AND(x,y,z) \mid V(c,z) \tag{4.9}$$

推論規則は超導出を用いる。(4.7) 式に着目して，(4.7),(4.9),(4.8),(4.3) 式より

$$V(b,1) \mid V(c,0) \tag{4.10}$$

を得る。つぎにこの (4.10) 式に着目して，(4.10),(4.9),(4.7),(4.4) より所望の結果である次式を得る。

$$V(c,0) \tag{4.11}$$

PS にこの推論をさせるには,「もし，$V(a,0)$ ならば $V(c,0)$」のようなアドホックなルールを加える必要がある。アドホックな知識で知識ベースにパッチを当てることにより推論の失敗を補う方法は，**AI** をベースとしたシステム開発の特徴ではある。しかし，**PS** が，将来の高度でかつ信頼性を有するエキスパートシステムの推論メカニズムの中核となり得るか否かには議論の余地がある。不完全な推論システムにいくら知識を付加しても決して完全になったという保証は得られないのに対し，完全なシステムに対する冗長な知識の付加は，純粋に効率向上のためのオプションである。

4.6.2 推論方式

本章で示した推論方式をまとめ，その新規性と有用性を論じる。**Reiter** が多重故障までも含む統一的な故障診断理論を発表して以来，そのシステムをインプリメントしたという報告はまだない。実際，彼の興味は工学的な実用性よりはむしろ，故障診断を論理の枠組みで形式的にとらえ，特に彼の専門である非単調推論との関連を考察することにあったと思われる。また，このインプリメントのためには，特定の問題分野における完全でかつ効率良い定理証明器を必要とする。本論文では，問題分野に依存しない完全な定理証明器である **Thinker** を採用し，組合せ論理回路の診断に対して適切な知識と推論規則および推論戦略を取捨選択し，適切にパラメータ設定することにより効率を高めることを試みた。実験による試行を通して，次のヒューリスティクスを得た。

- 各素子のゲート種別を示す節を支持集合とする。
- 回路の接続情報とブール代数の公理を表す等式をデモジュレータ集合とし，停止性と合流性を満たすようにする。交換則を表す公理は，両方向性デモジュレータとする。
- 「任意の素子の出力は0か1の一方である」ことを表す節のみをパラモジュレータとする。
- 項の大小関係 $\succ_{lpo}$ を定める関数記号の選好順序 $\succ_f$ に回路のトポロジーを反映させる。等号リテラルは，可能なら必ず〈左辺 $\succ_{lpo}$ 右辺〉となるよう，必要に応じて両辺を入れ替える。
- 超導出と単位消去を併用し，故障していないと仮定している素子の入出力を関係付ける等式を直接生成させる(中間に介在する節を生成させない)。
- パラモジュレーション，デモジュレーション，逆デモジュレーションにより，等式の代入則を中心とした推論を進める。

本実験は全加算器のみを対象としたため，このヒューリスティクスが他の回路に対してもうまく働くか否か疑問に思われるかもしれない。しかし，これらは「Reiter流に等号を用いて論理回路を表現したときの故障診断」という分野に対しては特殊化されているが,「全加算器」に対して特殊化されたものではない。今後多くの回路に対して実験を行う必要は残されているが，著者は本方式またはそれを自然に発展させたものが他の回路に対しても有効であろうと予想している。なお，本方式を他の回路に適用するためには接続記述と $\succ_f$ の一部の変更など組織的な最少の変更でよい。

この推論方式は推論に関する要素技術を，この問題分野に適合するようシステム化したものである。個々の要素技術自体は良く知られたものである。しかし，論理回路の診断に対して，それらの効果的な組み合せを追及したシステム

工学的な試みとその結果である上記のヒューリスティクスは独創的なものである。また，4.5 節で示した発展的な例題は他のシステムではこれまで試みられていない (あるいは解けない) 新しいものである。最終的には，ヒューリスティクスがなかった本研究の初期の時点に比べると，図 4.5 に示すように飛躍的に効率が向上した。従って，その有用性は十分主張できると考える。ただし，実用性についてはただちに結論付ける訳にはいかない。論理回路の故障診断システムは，単一故障あるいは特定の故障パターンあるいは特定の観測パターンに対するものについてはすでにある程度実用的なものがあり，本方式は現在のところそれを置き換えるものではない。しかし，Reiter の理論は非常に簡潔で一般的であり，広い範囲の高度な故障診断システムの構成に対する統一的で見透しのよい枠組みを与えている。特に，完全な定理証明器による論理的推論に基づくので知識ベースを複雑にすることなく高度な推論が可能となる。本研究の結果は，この枠組みに沿ってさらに研究を進めるにあたっての基礎資料として，今後の技術の発展に寄与し得るものと考える。

第5章

パッケージ概念導入による診断系の拡張

本章では，はじめに，DiaLogの診断系の拡張を行なう。すなわち，Reiterの理論において表われる診断単位COMPONENTSの概念をパッケージへ拡張する。これは回路の素子レベルで故障を特定できても，実際の保守の場面ではその素子を含むパッケージ交換となる場合が多いことに着目し，実用性を失うことなしに大規模システムの診断をより効率化させるものである。次に，順序回路の故障診断を実験する。順序回路は現在の入出力観測だけでは故障検出は不可能で，回路の履歴をコントロールする必要があり，知識表現に工夫が必要である。本章では時間素子に関する知識表現を述語論理の範囲内で記述できることを示し，代表的な順序回路であるカウンタと大規模な順序回路を例にとり診断実験を行なった。

5.1 パッケージ診断理論

Reiterの理論は，故障診断理論を特定の問題領域だけではなく，一般のシステムに適用することを可能とするために，システムの概念を問題領域とは独立に定義している。ここで示すパッケージ診断理論においてもその手法を利用す

るのだが，Reiterの理論との相違点は，システム構成要素の上位にパッケージが存在することである。例えば現実の論理回路では数個の素子を含む一つのICパッケージが実装上の最小単位となっている。このことは，故障素子を特定できたとしても，その保守においてはパッケージ交換となることを意味している。したがって，実用上は，素子レベルより粗いパッケージ・レベル診断で十分であり，かつ，診断に要する計算時間の短縮が期待できる。そこでReiterの診断理論を拡張し，診断をパッケージ単位で行えるパッケージ診断理論を提案する。

5.1.1 診断対象システム

定義12 システムとは、$\langle \mathbf{SD}, \mathbf{COMPONENTS}, \mathbf{PACKAGES} \rangle$ である。

ただし，SD, COMPONENTSは2章の定義1で定義したものと同じものである。また，PACKAGESはCOMPONENTSの分割である。その各要素をパッケージと呼ぶ。

例えば$\mathbf{COMPONENTS} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}, \mathbf{G}\}$のとき，$\{\{\mathbf{A}, \mathbf{B}\}, \{\mathbf{C}, \mathbf{D}\}, \{\mathbf{E}, \mathbf{F}, \mathbf{G}\}\}$は三つのパッケージからなるPACKAGESの例である。

5.1.2 パッケージ診断

次に，本章で新しく提案するパッケージ診断を定義する。

以下πは要素$c \in \mathbf{COMPONENTS}$を引数とし，$c$が属するパッケージ$P \in \mathbf{PACKAGES}$を値とする関数である。またパッケージの集合$\Delta_p$中に現れる要素の集合を

$$comp(\Delta_p) = \{c \mid (\exists P) c \in P \in \Delta_p\}$$

要素の集合Δを包含するための最小のパッケージ集合を

$$pack(\Delta) = \{\pi(c) \mid c \in \Delta\}$$

とする。明らかに

$$\Delta \subseteq comp\,(pack(\Delta))$$

$$\Delta_p = pack(comp\,(\Delta_p))$$

が成り立つ。

定義 13 $\langle \mathbf{SD}, \mathbf{COMPONENTS}, \mathbf{PACKAGES}, \mathbf{OBS} \rangle$ についてのパッケージ診断とは、

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\neg \mathbf{AB}(c) \mid c \in \mathbf{COMPONENTS} - comp\,(\Delta_p)\} \tag{5.1}$$

が無矛盾であるような極小集合 $\Delta_p \subseteq \mathbf{PACKAGES}$ である。

5.1 式はパッケージ内の素子毎にまとめて正常性 (または異常性) を仮定することを意味している。従って，この定義に基づく診断は，Reiter の理論の枠内で診断が可能である。さらに，診断空間の大幅な縮小から，考察すべき仮定の組合せが少なくなり，明らかに要素レベルの診断より計算時間が短くなる。その反面，この様な直接パッケージ診断は診断結果の理由説明が粗くなる。例えば，単一パッケージ {E, F, G} の故障とのパッケージ診断が得られた場合， E, F, G のいずれかの単一故障なのか，あるいは二重，三重の故障なのかの説明が得られない。特に別のパッケージ診断 {A, B} も得られているときには，いずれがより「もっともらしい」かの判断にとってこの理由説明は重要である。従って，本論文では直接パッケージ診断の効率を活かしながらも，要素レベルの理由説明 (パッケージ内の少なくともどの素子が異常であるかの説明) もできるような診断手法を求めることにする。これは Reiter の提案した素子レベル診断と，直接パッケージ診断との中間的な診断アプローチである。

命題 1 Δ が診断ならば， $\Delta_p \subseteq pack(\Delta)$ を満たすパッケージ診断 Δ_p が存在する。

証明 $\Delta'_p = pack(\Delta)$, $\Delta' = comp\,(\Delta'_p)$ とすると， $\Delta \subseteq \Delta'$ であり， Δ' は要素レベルの診断 2.2 式を無矛盾とする。よって， Δ'_p は 5.1 を無矛盾とする。従っ

て，パッケージ診断 $\Delta_p \subseteq \Delta'_p$ が存在する。 □

この命題において，$\Delta_p = pack(\Delta)$ になるとは限らない。例えば，$\mathbf{PACKAGES} = \{\{\mathbf{A},\mathbf{B}\},\{\mathbf{C},\mathbf{D}\}\}$ とし，二つの診断 $\Delta_1 = \{\mathbf{A}\}$, $\Delta_2 = \{\mathbf{B},\mathbf{C}\}$ があるとしよう。$\Delta_{p1} = pack(\Delta_1) = \{\{\mathbf{A},\mathbf{B}\}\}$ はパッケージ診断である。$\Delta_{p2} = pack(\Delta_2) = \mathbf{PACKAGES}$ は極小性を満たさないので，パッケージ診断ではない。ただし，命題 1 で存在が保証されている Δ_2 に対するパッケージ診断は Δ_{p1} である。

命題 2 Δ_p がパッケージ診断ならば，$pack(\Delta) = \Delta_p$ を満たす診断 Δ が存在する。また，$pack(\Delta') \subset \Delta_p$ を満たす診断 Δ' は存在しない。

証明 $\Delta^* = comp(\Delta_p)$ とする。Δ_p は 5.1 式を無矛盾にするから，Δ^* は 2.2 式を無矛盾とする。よって，$\Delta \subseteq \Delta^*$ を満たす診断 Δ が存在する。$pack(\Delta) \subseteq pack(\Delta^*) = \Delta_p$ であるから，$pack(\Delta') \subset \Delta_p$ を満たす Δ' 診断が存在するとして矛盾を示せばよい。すなわち，命題 1 より，$\Delta'_p \subseteq pack(\Delta') \subset \Delta_p$ を満たすパッケージ診断 Δ'_p が存在する。これは，Δ_p の極小性に矛盾する。 □

定理 5 Δ_p がパッケージ診断であることと，Δ_p が集合 $pack(\Delta_1),\ldots,pack(\Delta_n)$ のうち極小なものに一致することは同値である。

ただし，$\Delta_1,\ldots,\Delta_n$ は $(\mathbf{SD},\mathbf{COMPONENTS},\mathbf{OBS})$ に対するすべての診断である。

証明 Δ_p がパッケージ診断ならば，命題 2 より $pack(\Delta) = \Delta_p$ を満たす診断 Δ が存在し，$pack(\Delta') \subset pack(\Delta)$ を満たす診断Δ' は存在しない。よって，$pack(\Delta)$ は極小である。

逆に，Δ_p が $pack(\Delta_1),\ldots,pack(\Delta_n)$ の極小のもの，例えば $pack(\Delta)$ に一致するとき，命題 2 より $\Delta'_p \subseteq pack(\Delta) = \Delta_p$ を満たすパッケージ診断 Δ'_p が存在する。もし，$\Delta'_p \subset pack(\Delta) = \Delta_p$ ならば，命題 2 より，$pack(\Delta') = \Delta'_p \subset pack(\Delta)$ を満たす診断 Δ' が存在し，$pack(\Delta)$ の極小性に反する。よって，

$\Delta'_p = pack(\Delta) = \Delta_p$, すなわち$\Delta_p$ はパッケージ診断である。 □

ここで，$\Delta_p = pack(\Delta)$ のとき，Δ を Δ_p の根拠という。

5.2 パッケージ診断の計算

パッケージ診断の計算は，基本的には Reiter の方法に基づいて要素レベルの診断を計算し，定理 5 によりその極小なものを求める考え方で行なう。ただし，この仮定において明らかに極小性を満たさない解の候補は刈込みにより探索の範囲から除外される。以下，第 2 章で述べた **HS-木** を用いてパッケージ診断アルゴリズムを示す。ただし，$H(n)$ は，ルートからノード n までのパスについているラベルの集合で，$HP(n) = pack\{H(n)\}$ である。また，定理証明器は 2.4 式の矛盾性を検出するために用いるもので，矛盾ならばコンフリクト集合を値として返し，無矛盾ならば記号 $\surd$ を返すものとする。

Step-1 システム構成要素がすべて正常であると仮定して定理証明器を呼び出す。もし $\surd$ が返ればパッケージ診断は $\{\}$ であり，終了。コンフリクト集合が返れば，そのコンフリクト集合を **HS-木** の根のラベルとする。(ラベルがコンフリクト集合でないノードを「閉じられたノード」と呼ぶ)。

Step-2 すべてのノードが閉じられていれば終了。このとき $\surd$ でラベル付けされたノード n の $HP(n)$ のうち，極小のものがパッケージ診断となる。

Step-3 まだ閉じられていないノード n を任意に選ぶ。n がコンフリクト集合 S によってラベル付けされているとする。それぞれの $\sigma \in S$ について，σ をラベルとする枝を介して子ノード n_σ を生成する。**COMPONENTS**$-H(n_\sigma)$ の要素を正常と仮定して定理証明器を呼び出し，それが返した値 ($\surd$ またはコンフリクト集合) を n_σ のラベルとする。 **Step-2** へ進む。

Step-3 において n_σ のラベルを計算するときに，すでに求められているコンフリクト集合 S のうちで，$S \cap H(n_\sigma) = \Phi$ を満たすものがあれば，定理証明器を呼び出さずに n_σ のラベルを S としてよい。

このアルゴリズムによって HS-木 を生成する過程では 2 章 2.3 節で述べた三つの刈り込みを行なって効率を上げる。パッケージ診断ではさらに次の (iv) に示すパッケージ診断の極小性による刈り込みを追加して用いることができる。

(i) $\surd$ がついた他のノードと $H(n)$ が同じか，大きくなるようなノード n は閉じる。

(ii) すでに生成されたノードと同じ $H(n)$ となるノード n は閉じる。

(iii) ノード n のラベル S の真部分集合 S' がノード n' のラベルであるとき，各要素 $\alpha \in S - S'$ に対し α をラベルとして n から出ている枝をカットする。

(iv) ノード n が $\surd$ によってラベル付けされていて，ルートから n へのパス上にないノード n' が $HP(n) \subseteq HP(n')$ を満たすとき，n' を閉じる。

図 5.1 にパッケージ診断時に生成される HS-木 の例を示す。最初にコンフリクト集合 {A, B, C, F, G} が得られたとしこれを根 (n_0 とする) のラベルとする。根の第一子ノード (n_1) のラベルとしてコンフリクト集合 {C, F, G} が得られたとする。刈り込み (iii) によりラベル A, B の枝をカットする。第三子ノード (n_3) においてコンフリクト集合 {D, F, H} が得られたとする。第四子ノード (n_4) において $\surd$ が得られたとする。診断 {F} を根拠にパッケージ診断 {{E, F, G, H}} が得られる。この結果，第五子ノード (n_5) は刈込み (iv) により閉じられる ($\otimes$ で示される)。次に，ノード n_3 から D, F, H とラベル付けされる枝をのばす。D に隣接するノード n_6 において $\surd$ が得られた場合は，診断 {C, D} を根拠にパッケージ診断 {{A, B, C, D}} が得られる。ノード n_7 は刈り込み (i) により閉じられる。ノード n_8 は既に得られているパッケージ診断 {{E, F, G, H}} により刈り込み (iv) が行われて閉じられる。(素子レベルの診断ではノード n_5, n_8 の刈込み

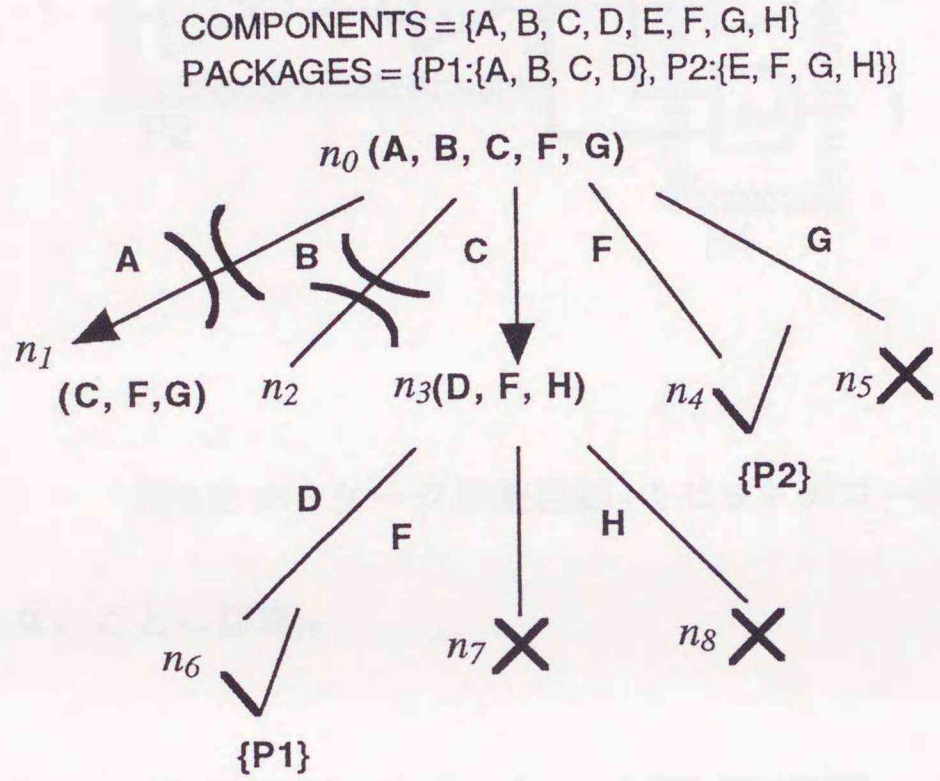


図 5.1: パッケージ診断における HS-木 生成の例

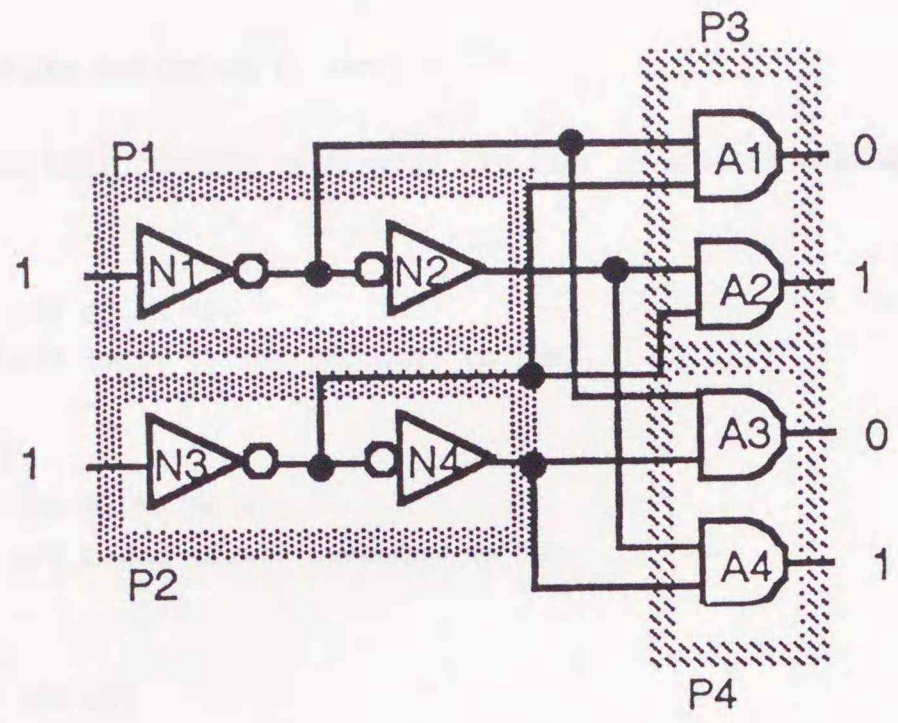


図 5.2: パッケージ診断回路 (2 ビットデコーダ)

は行われないことに注意)。

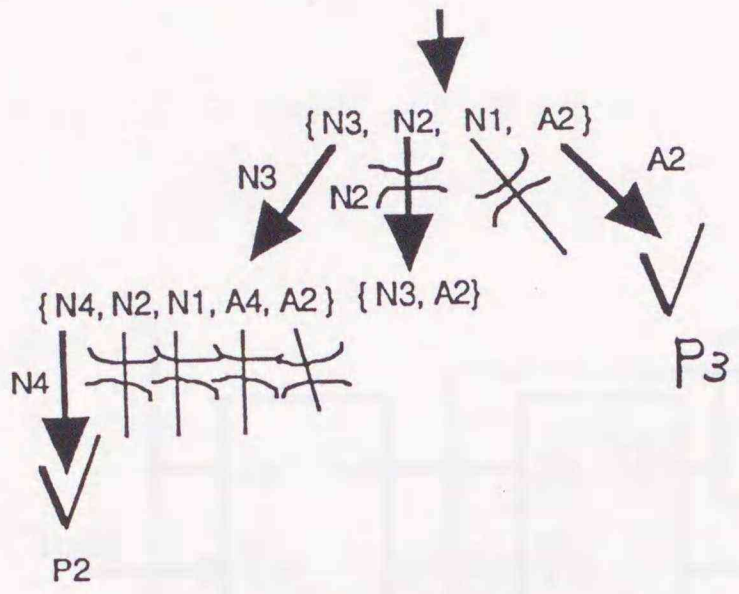
5.3 組合せ論理回路のパッケージ診断実験

第 4 章でのべた診断システム DiaLog-I の診断アルゴリズムをパッケージ診断ができるように拡張し，新たに PACKAGES をデータとして入力できるようにした。

診断回路を図 5.2 に示した。図に示した例では 4 個のパッケージで 8 個の素子を拘束して回路が構成されている。この回路の診断の様子を図 5.3 に示した。診断結果は根拠 {N4, N3}, {A2} をそれぞれ根拠にした 2 個のパッケージ単一故障 {P2}, {P3} が得られている。また，図 5.4 は，この診断が得られたときに生成された HS-木およびパッケージ診断を行なわないときの診断 (素子レベル診断) との比較を示している。パッケージ診断では診断個数が一つすくなくなっている。これはパッケージ診断 {P2} の極小性による。また，図からわかるように

```
        DiaLog for Macintosh II Ver.1.0

        Use the HELP command to display the list of all the DiaLog commands.

DIALOG-TP NIL
CONFLICT-Set: (N3 N2 N1 A2)
(REFUTE) took 477 ticks (7.950 seconds) to run.

DIALOG-TP (N3)
CONFLICT-Set: (N4 N2 N1 A4 A2)
(REFUTE) took 505 ticks (8.417 seconds) to run.

DIALOG-TP (N2)
CONFLICT-Set: (N3 A2)
(REFUTE) took 357 ticks (5.950 seconds) to run.

PRUNING TREE:: (n0)-N1-
PRUNING TREE:: (n0)-N2-(n2)-N3-
PRUNING TREE:: (n0)-N2-(n2)-A2-
DIALOG-TP (A2)
CHECKED
(REFUTE) took 794 ticks (13.233 seconds) to run.

DIALOG-TP (N4 N3)
CHECKED
(REFUTE) took 973 ticks (16.217 seconds) to run.
Of that, 210 ticks (3.500 seconds) was spent in GC.

DIAGNOSES PACKAGES::
(P3) BASED-ON: (A2)
(P2) BASED-ON: (N4 N3)
?
```

図 5.3: パッケージ診断の実行



A pruned HS-tree for the 2-inputs decoder

Macintosh-II (MC68020,15MHz) Thinker (Common Lisp)

Package-level diagnosis	TP Calls	Time(SEC.)
{P3 : A2}, {P2: ,N4, N3}	5	51.4
Component-level diagnosis	TP Calls	Time(SEC.)
{ A2 }, { N4, N3 }, { A4, N3 }	7	78.4

図 5.4: 診断結果 (2 ビットデコーダ)

明らかに大幅な刈り込みが行われ，したがって定理証明器 (TP) の呼び出し回数も減少している。汎用の定理証明器の呼び出し回数の減少は本章の後半で示すような大規模な論理回路の診断では診断効率の向上に大きな効果をもたらす。

5.4 順序回路の故障診断実験

はじめに順序回路の故障検出を行なう上で考慮すべき知識表現について述べる。次に故障診断の具体例としてカウンタ及び自動販売機について実験結果を示す。本実験に使用した定理証明器および診断システムのバージョンはアポロ社 Domain Common Lisp で記述し，同社 DN3500(25MHz-MC68030) 上で診断時間を実測したものである。

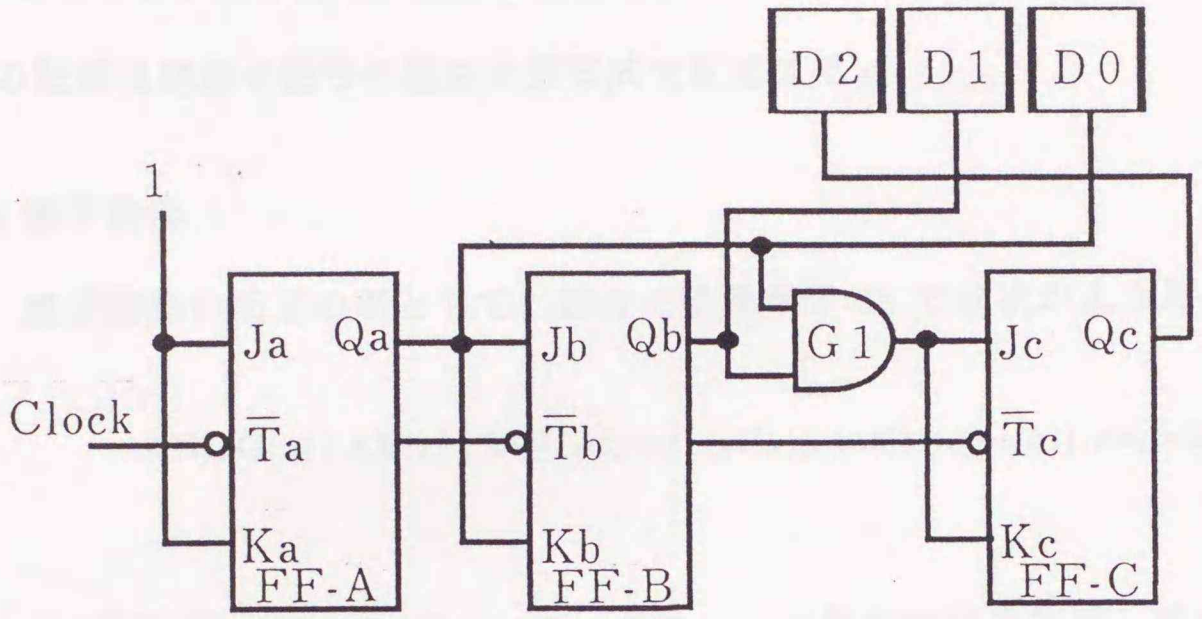


図 5.5: 診断回路 - 8 進同期式カウンタ

5.4.1 知識表現

基本的には組合せ回路に対する表現法を拡張して使うことができる。ただし，順序回路では，時刻パルス (ビット時刻) を組み込んだ表現を工夫する必要がある。具体的に図 5.5 の 8 進同期式カウンタの例で示す。**FF-A, FF-B, FF-C** はマスタ・スレーブ型同期式 **JK** フリップ・フロップ，G_1 は **AND** ゲートである。

5.4.1.1 COMPONENTS

故障診断の対象となる素子の集合で，図 5.5 の回路では，**COMPONENTS** = {G_1, **FF-A, FF-B, FF-C**} である。

5.4.1.2 システム記述 SD

SD としては，素子動作，回路接続情報，ブール代数の公理，等式の公理，二値性の記述及び素子記号の種別を節形式で記述する。

(i) 素子動作

素子動作の記述の例として，組合せ論理素子 G_1 では次のようにした

$$\neg\mathbf{ANDG}(x) \mid \mathbf{AB}(x) \mid \mathbf{EQUAL}(v(t,out(x)), and(v(t,in1(x)), v(t,in2(x)))) \tag{5.2}$$

5.2 式は「x が AND ゲートで，かつ，x が異常でなければ，ビット時刻 t での x の出力はその時刻での x の二つの入力値の and 関数の値に等しい」という意味を表している。ここで，ANDG, AB, EQUAL は述語，$and, v, in1, in2$ は関数，x, t は変数である。次に，FF-A のような時間素子の節表現は次のようになる。

$$\neg\mathbf{JKFF}(x) \mid \mathbf{AB}(x) \mid \mathbf{EQUAL}(v(s(t),q(x)), jk(v(t,j(x)), v(t,k(x)), v(t,q(x)))) \tag{5.3}$$

関数 j, k, q は JK-FF の入出力端子を意味する。$s(t)$ は任意の時刻 t の 1 クロック後，つまり，$(t+1)$ ビット時刻を意味する。関数 jk の意味は次の等式で規定される。

$$\mathbf{EQUAL}(jk(x,y,z), or(and(x,not(z)), and(not(y),z))) \tag{5.4}$$

5.3, 5.4 式より，$(t+1)$ ビット時刻での JK-FF の動作を記述したことになる。

(ii) 回路接続情報

接続されている端子での信号値 (1 または 0) が任意の時刻で等しいことを等式で記述する。例えば，FF-A と FF-B の部分の接続は，次式で記述

される。

$$\mathbf{EQUAL}(v(t, j(FF_B)), v(t, q(FF_A))) \tag{5.5}$$

$$\mathbf{EQUAL}(v(t, k(FF_B)), v(t, q(FF_A))) \tag{5.6}$$

(iii) ブール代数の公理

関数記号 not, and, or 等の意味を記述する。例として， and に関しては具体的には次のようになる。

$$\mathbf{EQUAL}(and(0, x), 0) \tag{5.7}$$

$$\mathbf{EQUAL}(and(1, x), x) \tag{5.8}$$

$$\mathbf{EQUAL}(and(x, x), x) \tag{5.9}$$

$$\mathbf{EQUAL}(and(x, y), and(y, x)) \tag{5.10}$$

(iv) その他

次に示す例のように，等式の公理，二値性の記述及び素子記号の種別を節形式で記述する。

$$\mathbf{EQUAL}(x, x) \tag{5.11}$$

$$\mathbf{EQUAL}(v(t, out(x)), 1) \mid \mathbf{EQUAL}(v(t, out(x)), 0) \tag{5.12}$$

$$\mathbf{ANDG}(\mathbf{G}_1) \tag{5.13}$$

$$\mathbf{JKFF}(\mathbf{FF}_A) \tag{5.14}$$

5.4.1.3 システム観測 OBS

図 5.5 のカウンタの場合，各ビット時刻で得られたフリップ・フロップの出力値を等式で記述する。

$$\mathbf{EQUAL}(d_0(0), 0) \tag{5.15}$$

$$\mathbf{EQUAL}(d_1(0), 0) \tag{5.16}$$

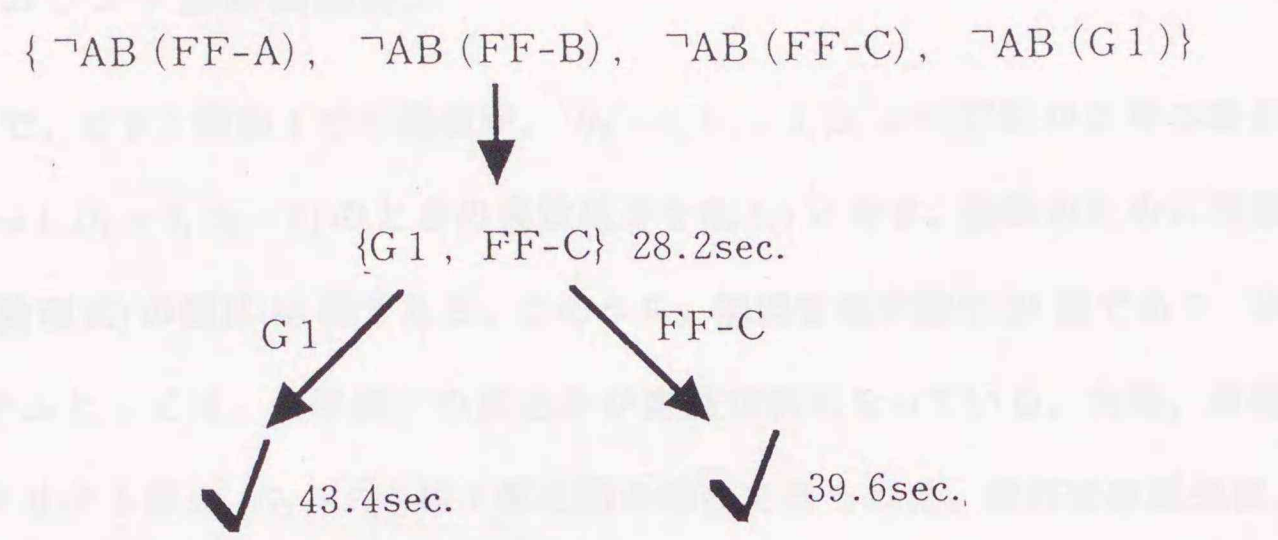


図 5.6: HS-木 - 8 進同期式カウンタ

$$\mathbf{EQUAL}(d_2(0), 0) \tag{5.17}$$

$$\mathbf{EQUAL}(d_0(s(0)), 1) \tag{5.18}$$

$$\mathbf{EQUAL}(d_1(s(0)), 0) \tag{5.19}$$

$$\mathbf{EQUAL}(d_2(s(0)), 0) \tag{5.20}$$

同様にビット時刻 7 まで記述する。5.18 式はビット時刻 1 にカウンタの最下位桁がカウント・アップしたことを示している。

5.4.1.4 仮説

最初のコンフリクト集合を得るために診断開始の時点では，すべての素子は異常ではないという仮説を置く。図 5.5 では，次のようにする。

$$\neg\mathbf{AB}(\mathbf{G}_1) \tag{5.21}$$

$$\neg\mathbf{AB}(\mathbf{FF}_A) \tag{5.22}$$

$$\neg\mathbf{AB}(\mathbf{FF}_B) \tag{5.23}$$

$$\neg\mathbf{AB}(\mathbf{FF}_C) \tag{5.24}$$

5.4.2 カウンタ診断実験結果

図 5.5 で，ビット時刻 4 での観測が，$D_2 = 0, D_1 = 0, D_0 = 0$ (回路が正常の場合は，$D_2 = 1, D_1 = 0, D_0 = 0$) のときの実験結果を図 5.6 に示す。診断のために用意した節 (論理式) の数は 58 個である。このうち，観測を表す節は 24 個であり，診断システムとしては，故障素子の絞込みが容易な例となっている。実際，最初のコンフリクト集合 $\{G_1, FF_C\}$ は 7 個の節の導出で得られた。最終診断結果は，$\{G_1\}$ あるいは $\{FF_C\}$ の単一故障であることがわかる。これは素子 G_1 (または，FF_C) が故障ならばシステム記述とシステム観測のあいだに矛盾は無いことを意味する。定理証明器 (**Thinker**) の呼び出しは 3 回であった。

5.4.3 自動販売機のパッケージ診断

次に，やや大規模な論理回路の診断実験として，自動販売機のパッケージ診断の例を述べる。この機械は入力として 10 円，50 円および 100 円の三種類の硬貨を受付け，出力として定価 80 円の品物と釣銭をだす。制御部の回路図を図 5.7 に示す。

図 5.7 で $\mathbf{I}_H, \mathbf{I}_L$ は硬貨検出器からの 2 進コード入力端子を表し，$\mathbf{I}_H, \mathbf{I}_L = 00, 01, 10, 11$ はそれぞれ，入力無し，10 円，50 円及び 100 円の入力を表す。また，**Out-J** はジュース出力信号，**Out-A**,...,**Out-D** は釣銭とすべき 10 円硬貨の枚数の 2 進コード出力端子である。ここで，回路が異常な動作をしている状況を，「50 円硬貨を 2 度投入したとき，品物は得られたが，釣銭が 10 円のみであった」とした。この状況を観測とし，14 個の等式で記述した。次に，回路素子が，それぞれどのパッケージに属するかを表 5.1 のように決めた。なお，診断のために最終的に用意した節の総数は，98 個となった。

カウンタの例と同様，定理証明器 **Thinker** を用いて実験を行なった。初めにサイズ 40 のコンフリクト集合が得られた。この集合をもとにパッケージ単一故

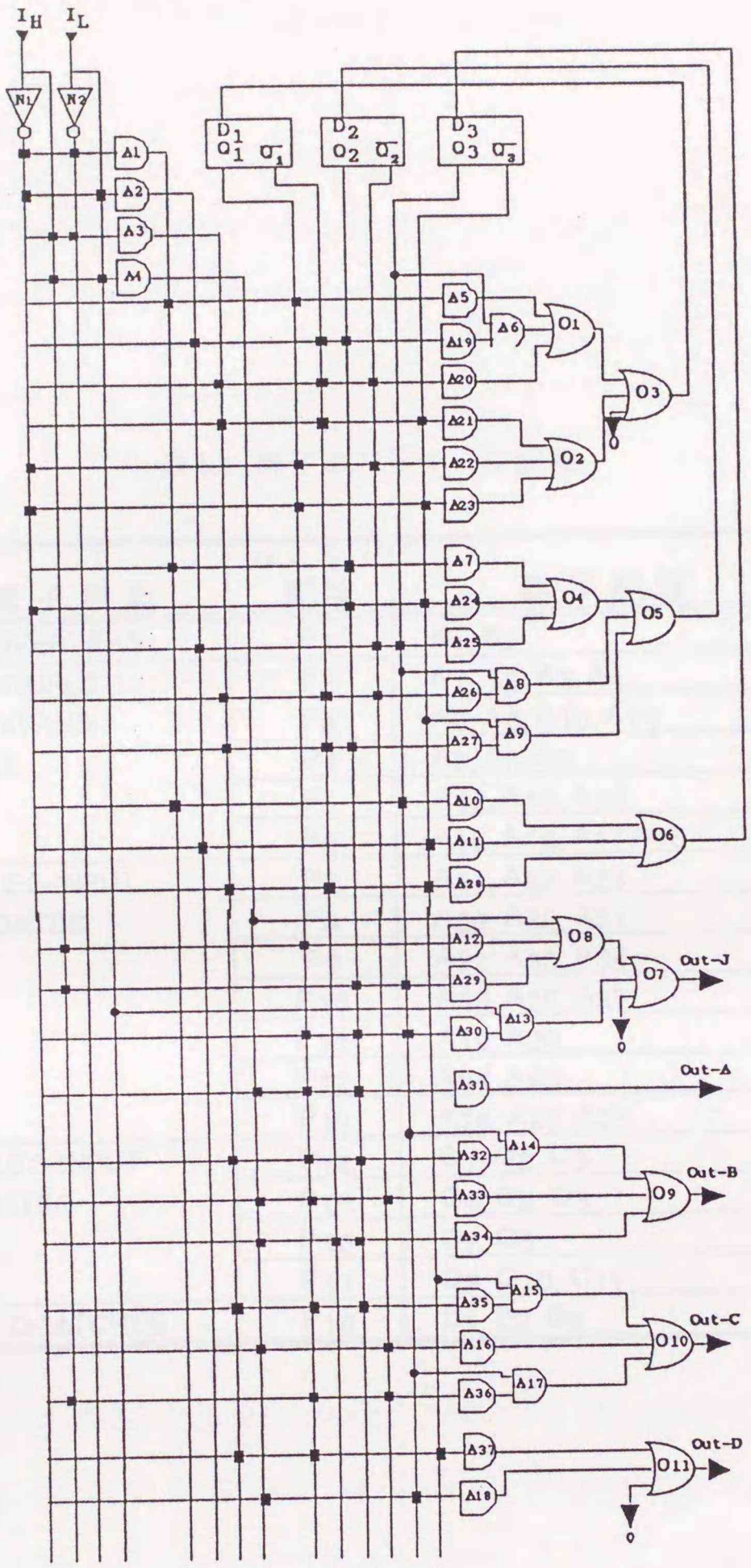


図 5.7: 故障診断回路 - 自動販売機

表 5.1: 素子のパッケージ配置

素子機能	パッケージ記号	素子配置
HEX INVERTERS	P_1	N_1, N_2
QUADRUPLE 2-INPUT AND GATES	P_2	A_1, A_2, A_3, A_4
	P_3	A_5, A_6, A_{10}, A_{11}
	P_4	A_7, A_8, A_9
	P_5	A_{12}, A_{13}, A_{14}
	P_6	$A_{15}, A_{16}, A_{17}, A_{18}$
TRIPLE 3-INPUT AND GATES	P_7	A_{21}, A_{22}, A_{23}
	P_8	A_{29}, A_{30}, A_{31}
	P_9	A_{32}, A_{33}, A_{34}
	P_{10}	A_{35}, A_{36}, A_{37}
	P_{11}	A_{19}, A_{20}
	P_{12}	A_{24}, A_{25}
	P_{13}	A_{26}, A_{27}, A_{28}
TRIPLE 3-INPUT OR GATES	P_{14}	O_1, O_2, O_3
	P_{15}	O_4, O_5, O_6
	P_{16}	O_7, O_8
	P_{17}	O_9, O_{10}, O_{11}
4-BIT D-LATCHES	P_{18}	D_1, D_2, D_3

表 5.2: 自動販売機診断結果

パッケージ単一故障	根 拠	パッケージ二重故障	根 拠
$\{P_{13}\}$	$\{A_{28}\}$	$\{P_2, P_3\}$	$\{A_3, A_6\}$
$\{P_{15}\}$	$\{O_6\}$	$\{P_2, P_{11}\}$	$\{A_3, A_{20}\}$
	TP Calls 37	$\{P_2, P_7\}$	$\{A_3, A_{23}\}$
	Time(min.) 24.8	$\{P_4, P_2\}$	$\{A_9, A_{24}\}$
$\{P_6\}$	$\{A_{17}, A_{18}\}$	$\{P_{12}, P_2\}$	$\{A_{25}, A_4\}$
$\{P_{10}\}$	$\{A_{36}, A_{37}\}$	$\{P_1, P_3\}$	$\{N_2, A_6\}$
$\{P_{17}\}$	$\{O_{10}, O_{11}\}$	$\{P_1, P_{11}\}$	$\{N_2, A_{20}\}$
$\{P_{18}\}$	$\{D_{3\text{-}q}, D_{3\text{-}nq}\}$	$\{P_1, P_7\}$	$\{N_2, A_{23}\}$
		$\{P_{14}, P_2\}$	$\{O_3, A_3\}$
		$\{P_{14}, P_1\}$	$\{O_3, N_2\}$
		TP Calls 95	
		Time(min.) 94.7	

障と二重故障を求めた結果を表 5.2 に示す。根拠のサイズが 1 のパッケージ単一故障が 2 個 (診断時間，約 25 分) 求まった。**Thinker** の呼出回数は 37 回であった。さらに，呼出回数 95 回 (診断時間，約 1 時間半) で根拠のサイズが 2 のパッケージ単一故障が 4 個と二重故障が 10 個得られた。

次に，表 5.3 に素子レベルの根拠を求めない直接パッケージ診断の結果を示す。診断としては表 5.2 と同一となった。しかし，**Thinker** の呼出回数が大幅に減少しているにもかかわらず診断時間は回数に比例して減少していない。これは，各パッケージ内の素子毎にまとめて異常を仮定することは，回路的にはそれ

表 5.3: 根拠なし直接パッケージ診断

パッケージ単一故障	TP Calls	Time(min.)
{P_6}, {P_{10}}, {P_{13}}, {P_{15}}, {P_{17}}, {P_{18}}	14	4.3
パッケージ二重故障	TP Calls	Time(min.)
{P_1, P_3}, {P_1, P_7}, {P_1, P_{11}}, {P_1, P_{14}}, {P_2, P_3 } {P_2, P_4}, {P_2, P_7}, {P_2, P_{11}}, {P_2, P_{12}}, {P_2, P_{14}}	27	78.5

らを同時に開放除去したことに相当し，入出力関係からきまる推論の制約条件がゆるくなることを意味する。このため推論時の場合分けが多くなり Thinker の矛盾検出時間が増大したことが原因である。また，従来の Reiter の理論による素子レベル診断では，二重故障の診断結果を得るためには， Thinker の呼出回数は最悪の場合約 400 回以上と見積もれるが，途中まで実験を行なって経過をみたところ大幅なカットも現れなかった。仮にカットにより Thinker の呼出しが半分になったとしても時間的に現実的な診断となり得ない。本章で提案したパッケージ・レベル診断では，パッケージの二重故障まで約 2 時間の診断時間で求めることができた。比較的大規模な順序回路であっても，本方式の診断システムが有効であることがわかる。

5.5 議論

ここで扱った故障診断に関する問題は以下の二つである。

(i) Reiter の理論に基づいた組合せ論理回路の故障診断システムが順序回路にも適用可能であるか。

(ii) 大規模な回路も実用的な時間で診断可能か。

この章で示した実験結果は，以上の二点について解答を与えるものである。すなわち，(i) に対しては，具体的な知識表現法を述語論理の範囲内で提示し，実験により組合せ回路のときと同様に順序回路の診断が可能であることを示した。また，(ii) に対しては，パッケージの概念を導入して Reiter の理論を拡張し，探索空間の縮小と効果的な枝の刈込みによる大規模回路診断へのアプローチを提案した。

Reiter の理論は，原理的に任意のシステムの故障診断に適用できる非常に一般的な理論であり，将来の高度な故障診断システムの統一的な構成に明確な見通しを与えている。システムの異常検知という観点からは，システムのモデル (知識表現) と観測に基づいて自動的に異常を検知し，その異常であるという理由 (証明) に基づいて，論理的かつ効率的に異常原因を推論している。本章で示した結果は，Reiter の理論の実際的な応用可能性を拡大した新しい成果であり，論理回路の分野に限らず，故障診断一般について今後の研究に有効に寄与するものと考える。

しかし，本実験では，汎用の定理証明器の機能を用いて推論部を構成している。大規模回路に対して飛躍的に効率を向上させるには，より限定されたクラスの定理のための効率的な証明器が必要である。次章ではファジィ論理回路の故障診断を例にとり，より限定した (制約論理に基づく) 定理証明器の構成方法と診断結果を示す。

第6章

ファジィ論理回路の故障診断

本章では，ファジィ論理回路の故障診断を効率良く行なうために，DiaLogの推論系を新たに構成する。この推論系をもつ診断システムを，便宜上，DiaLog-IIと呼ぶ。

本章では，はじめにファジィ論理回路をどのように診断対象システムとして形式化を行なうかを示す。具体的には，ファジィ論理回路の構造および観測される入出力値を方程式および不等式の集合で表現する。この集合の上で診断の計算を定義し，診断に必要となる定理証明器が用いる推論規則を示す。今回の診断方法では，推論の制御機構に局所制約伝播を用いて効率を向上させている。しかし，これらの推論規則のみでは故障の検出に不完全な場合があることを指摘し，端点値探索法と呼ぶ充足性判定法を提案し，これと組み合わせて診断を行なう方法について述べる。最後に，いくつかの実験結果を示す。

6.1 診断システムの形式化

故障診断アルゴリズムは，基本的には，2章で示したReiterの診断理論を利用する。Reiterの理論では，診断対象システムの構造や挙動を，第一階の述語論理式の集合で表し，その中の互いに矛盾した論理式を定理証明器により検出することによって診断を行った。今回扱う診断問題では，診断対象システムを

方程式 - 不等式の集合で表現する。定理証明器はこの系に解が存在するか否かの判定に利用する。以下，Reiter の診断理論をなるべく自然に適用できるように，診断対象システムの形式化を行う。

6.1.1 システム記述

ファジィ論理回路は，MAX(∨),MIN(∧), CMP(¬) のゲートが結線されたものである。これは，以下のように表現できる。

定義 14 ファジィ論理回路のシステム記述 **SD** はラベル付き方程式の有限集合で，以下の条件を満たすクラス **S** に属するものである。

(i) $\{\} \in \mathbf{S}$

(ii) $\mathbf{SD} \in \mathbf{S}$ ならば，

(a) $\mathbf{SD} \cup \{C : x \vee y = z\} \in \mathbf{S}$

(b) $\mathbf{SD} \cup \{C : x \wedge y = z\} \in \mathbf{S}$

(c) $\mathbf{SD} \cup \{C : \neg x = z\} \in \mathbf{S}$

(d) $\mathbf{SD} \cup \{C : x = z\} \in \mathbf{S}$

ただし，C はラベル (素子名) で，**SD** で用いられていないものである。x, y は変数である。z は変数で，**SD** に用いられていないものである。すなわち，ループのあるファジィ論理回路は診断対象としない。(iid) は回路中の 2 点 x, z の値が常に等しいことを宣言するもので，この場合 C は x-z 間の結線をも素子とみなしている。結線を素子とみなさないときには必要ない。以後，**SD** の全ラベルの集合を **COMPONENTS** とし，**SD** 中のすべての変数の集合を V とする。

例 図 6.1 の回路に対するシステム記述 **SD** は次のようになる。

$$\mathbf{SD} = \{A : x \vee y = z,\ B : z = v,\ C : \neg v = w\}$$

$$\mathbf{COMPONENTS} = \{A, B, C\},\ \mathbf{V} = \{v, w, x, y, z\}$$

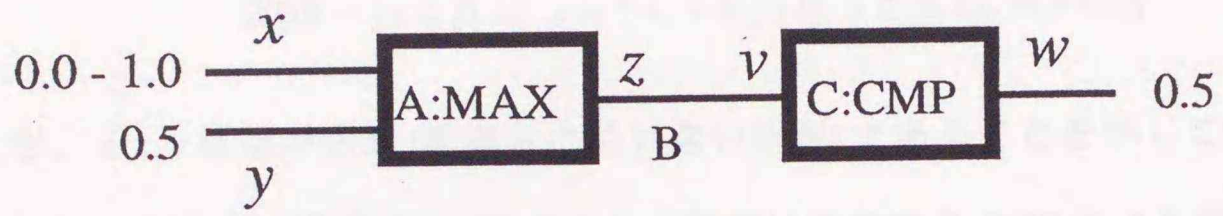


図 6.1: 簡単なファジィ論理回路例

6.1.2 観測

観測は，変数の集合 V の部分集合で与えられる変数のいくつかに，それぞれ区間 [0, 1] 内の実数値を割り当てたものである。変数 x_i に観測値 v_i を与えたとき $v_i \leq x_i \leq v_i$, 変数 x_i を観測しないとき $0 \leq x_i \leq 1$ と表現することにすれば，観測はすべての変数 x_i に対する不等式 $l_i \leq x_i \leq u_i$ の集合として形式的に表現できる。これはまた，観測値に $\pm e_i$ の誤差を含むとき $v_i - e_i \leq x_i \leq v_i + e_i$ と表現することも可能にしている。表記の便宜上，このような不等式を $x_i \in [l_i, u_i]$ の形で表す。$[l_i, u_i]$ は l_i 以上 u_i 以下の実数値の集合を表す。特に， $x \in [l, l]$ のとき，$x = l$ と書く。

定義 15 システム **SD** に対する観測 **OBS** は，すべての変数 $x_i \in \mathbf{V}$ に対する（上下限）不等式の集合，

$$\mathbf{OBS} = \{x_i \in [l_i, u_i] \mid x_i \in \mathbf{V}\}$$

である。ただし，$0 \leq l_i \leq u_i \leq 1$ である。

観測されたシステムを (SD, OBS) で表す。

例　図 6.1 の回路の観測は次のようになる。

$$\mathbf{OBS} = \{x \in [0,1],\ y = 0.5,\ z \in [0,1],\ v \in [0,1],\ w = 0.5\}$$

ここで，x, z, v は値が未知 (観測されていない状況) であることを示している。

システム記述は方程式系 SD で表され，観測は不等式系 OBS により表されている。方程式 - 不等式系 SD ∪ OBS に解が存在しなければ，SD で表現されたファジィ論理回路は故障していると言える。

6.1.3 意味論

SD ∪ OBS 中の各方程式，不等式の意味は直観的に明らかと思われるが，以下で形式的に定義しておく。

$x \in V$ に対して $[0,1]$ の実数を対応づける関数 σ を解釈と呼ぶ。σ を用いて個々の方程式，不等式の真 (T)，偽 (F) を次の規則で定める。

- $x \vee y = z$ が T となる。$\Longleftrightarrow max(\sigma(x), \sigma(y)) = \sigma(z)$
- $x \wedge y = z$ が T となる。$\Longleftrightarrow min(\sigma(x), \sigma(y)) = \sigma(z)$
- $\neg x = z$ が T となる。$\Longleftrightarrow cmp(\sigma(x)) = \sigma(z)$
- $x = z$ が T となる。$\Longleftrightarrow \sigma(x) = \sigma(z)$
- $x \in [a,b]$ が T となる。$\Longleftrightarrow a \leq \sigma(x) \leq b$

ただし，max は最大値，min は最小値，cmp は補数 $(cmp(a) = 1 - a)$ をとる演算である。

A ⊆ SD ∪ OBS のすべての要素を T とする解釈を A の解と呼ぶ。A の解が存在するとき A は充足可能，さもなくば A は充足不能であるという。なお，構文的な要素 $(\vee, \wedge, \neg)$ と意味的な要素 (max, min, cmp) を区分していることに注意。

6.1.4 (SD, OBS) に対する診断

準備として，SD 中のラベル C を持つ方程式を返す関数 $\mathbf{E}_{SD}(C)$ を定義する。図 6.1 の回路では，

$$\mathbf{E}_{SD}(A) = \langle A : x \vee y = z \rangle$$

である。さらに，これをラベルの集合を扱えるように次のように拡張する。

$$\mathbf{E}_{SD}(X) = \{\mathbf{E}_{SD}(C) \mid C \in X\}$$

定義 16 (SD, OBS) に対する診断とは，$\{\mathbf{SD} - \mathbf{E}_{SD}(\Delta)\} \cup \mathbf{OBS}$ が充足可能であるような極小集合 $\Delta \subseteq \mathbf{COMPONENTS}$ である。

直観的には，Δ に含まれるすべての素子の異常を仮定すると観測の異常を説明できるが，Δ より小さな集合に対して同様の仮定をしても観測の異常を説明できないことを表している。この場合注意すべき点は，ある素子の異常を仮定するとは，その素子に対応するラベルを持つ方程式を，単に，もとのシステム SD から取り除く操作を行えばよく，特別に異常動作の記述が必要とされるわけではない。もとの回路に異常がないときは $\Delta = \{\ \}$ である。

次に，定義 16 を用いて Reiter の診断のアルゴリズムで用いられたコンフリクト集合を定義する。

定義 17 $\Gamma \subseteq \mathbf{COMPONENTS}$ で，$\mathbf{E}_{SD}(\Gamma) \cup \mathbf{OBS}$ が解を持たないとき，Γ を (SD, OBS) に対するコンフリクト集合という。

コンフリクト集合を用いると，診断を次のように形式化できる。

命題 3 $\Delta \subseteq \mathbf{COMPONENTS}$ が (SD, OBS) に対する診断であることは，Δ が次の条件を満たす極小集合であることと同値である。

- $\mathbf{COMPONENTS} - \Delta$ が (SD, OBS) に対するコンフリクト集合ではない。

ファジィ論理回路の診断においても命題 3 の Δ を直接求めず，定義 5 で示した極小ヒット集合を利用する。すなわち，2 章の定理 2 に基づいて HS-dag を生成することで診断を得る。HS-dag を成長させる過程では定理証明器が使用される。ただし，本診断で用いる定理証明器は，4, 5 章で用いた導出原理と等式理論に基づく定理証明器ではなくその主な推論制御に制約伝播を用いるものである。具体的には，$\mathbf{SD}' \subseteq \mathbf{SD}$ と OBS を受け取り $\mathbf{SD}' \cup \mathbf{OBS}$ が充足可能であるか否かを局所制約伝播により判定し，充足可能であればそれを示す記号 √，充足不能であれば $(\mathbf{SD}', \mathbf{OBS})$ に対するコンフリクト集合を返す。ただし，局所制約伝播のみ推論制御では不完全な診断を行う場合があるので端点値探索法と呼ぶ充足性判定アルゴリズムを組込んだ定理証明器を提案する。

6.2 推論規則 (制約式)

$\mathbf{SD}' \cup \mathbf{OBS}$ の充足性を調べる定理証明器の構成について述べる。充足性の判定には推論規則が必要となる。以下，ファジィ論理の基本的な演算である，$\vee, \wedge, \neg$ および同値関係を条件とし，変数の区間を導出する推論システム I を考える。なお，$\mathbf{SD}' \cup \mathbf{OBS}$ は以後 A と表現する。

6.2.1 関係および区間制約

推論システムの全体 I を表 6.1 に示す。ただし，$x \vee y, x \wedge y, x = y, \neg x = y$ は，それぞれ x と y を入れ替えた式と同一視している。変数の関係が，$x = y$ のとき，x, y は同値関係にあるという。また，$\neg x = y$ のとき，変数 x, y は補関係にあるという。

$\vee$ に関係する推論規則は，表 6.1 の (1) ~ (5) で示される。ここで，$x \vee y = z$ で $\mathbf{OBS} = \{x \in [0.1, 0.5], y = 0.4\}$ のように観測が区間をもつ場合は，推論規則 (1) より $z \in [0.4, 0.5]$ となり，推論結果にも区間が得られる。また，$x \vee y = z$ で $z = 0.7$

表 6.1: ファジィ論理素子に関する推論規則

(1) ∨-順方向 $x \vee y = z$ $x \in [l, u]$ $y \in [l', u']$ --- $z \in [l'', u'']$, where $l'' = \max(l, l')$, $u'' = \max(u, u')$	(2) ∨-逆方向 $x \vee y = z$ $z \in [l, u]$ --- $y \in [0, u]$	(3) ∨-順方向選択 $x \vee y = z$ $x \in [l, u]$ $y \in [l', u']$ $u \leq l'$ --- $z = y$
(4) ∨-べき等 $x \vee y = z$ $x = y$ --- $z = x$	(5) ∨-逆方向選択 $x \vee y = z$ $x \in [l, u]$ $z \in [l', u']$ $u < l'$ --- $y = z$	(6) ∧-順方向 $x \wedge y = z$ $x \in [l, u]$ $y \in [l', u']$ --- $z \in [l'', u'')]$, where $l'' = \min(l, l')$, $u'' = \min(u, u')$
(7) ∧-逆方向 $x \wedge y = z$ $z \in [l, u]$ --- $y \in [l, 1]$	(8) ∧-順方向選択 $x \wedge y = z$ $x \in [l, u]$ $y \in [l', u']$ $u' \leq l$ --- $z = y$	(9) ∧-べき等 $x \wedge y = z$ $x = y$ --- $z = x$
(10) ∧-逆方向選択 $x \wedge y = z$ $x \in [l, u]$ $z \in [l', u']$ $u' < l$ --- $z = y$	(11) 同値関係推移 $x = y$ $y = z$ --- $x = z$	(12) 同値関係 $x = y$ $x \in [l, u]$ --- $y \in [l, u]$
(13) 補関係 $\neg\, x = y$ $x \in [l, u]$ --- $y \in [1-u, 1-l]$	(14) 二重否定 $\neg\, x = y$ $\neg\, y = z$ --- $x = z$	(15) 補関係推移 $\neg\, x = z$ $x = y$ --- $\neg\, y = z$
(16) 区間縮小 $x \in [l, u]$ $x \in [l', u']$ --- $x \in [l'', u'']$, where $l'' = \max(l, l')$, $u'' = \min(u, u')$		

の場合は，推論規則 (2) より，$y \in [0, 0.7]$ となり変数に区間が推論される。このように，一般的には区間が推論される。

変数が持つ区間は，(16) の規則によって更新される。(16) は，変数に二つの異なる制約が課せられた状態のときの推論規則である。この場合，区間を実数値の集合と見て，二つの集合の交わりを得ることによって区間が更新される。このとき，$l'' > u''$ となったとき，これを空区間と呼び，[ ] と表現する。[ ] は実数の空集合を表し，任意の解釈 σ のもとで $x \in [\ ]$ は常に偽 (F) である。

6.2.2 健全性

p を方程式または不等式とする。方程式-不等式系 A のすべての式を真とする解釈 σ に対して，いつでも p が T となるとき，$\mathbf{A} \models p$ で表す。また，推論システム I により，A から p が導き出されるとき，$\mathbf{A} \vdash p$ と表現する。このとき，健全性と完全性を次のように定義する。

定義 18 $\mathbf{A} \vdash p$ ならば $\mathbf{A} \models p$ であるとき，推論システム I は健全であるという。

定義 19 A が充足不能ならばある $x \in \mathbf{V}$ に対して $\mathbf{A} \vdash x \in [\ ]$ であるとき，I は (反駁に関して) 完全であるという。

表 6.1 の推論システム I に関して，次の命題を得る。

命題 4 推論システム I は健全である。

証明 表 6.1 の各規則の健全性のみをチェックすればよい。I の推論規則のうち (1) の健全性を証明する。

(i) $max(\sigma(x), \sigma(y)) = \sigma(z), l \leq \sigma(x) \leq u, l' \leq \sigma(y) \leq u'$ より，$\sigma(z) \geq \sigma(x) \geq l, \sigma(z) \geq \sigma(y) \geq l'$ であるから，$\sigma(z) \geq max(l, l')$ を得る。

(ii) また，$\sigma(z) = \sigma(x)$ ならば，$\sigma(z) \leq u, \sigma(z) = \sigma(y)$ ならば，$\sigma(z) \leq u'$ であるから $\sigma(z) \leq max(u, u')$

(i), (ii) より，$max(l, l') \leq \sigma(z) \leq max(u, u')$ が成立。I のその他の推論規則の健全性も容易に証明できる。 □

推論システム I の健全性により $x \in [\ ]$ の形の式が推論されたとき，系は充足不能と判定でき，以後の推論を停止させる。

命題 5 ある変数 $x \in \mathbf{V}$ に対し，$\mathbf{A} \vdash x \in [\ ]$ ならば A は充足不能である。

証明 推論システム I の健全性から明らか。 □

6.2.3 停止性

推論システム I の停止性，すなわち無限に新しい等式または不等式を推論し続けないことを示す。I は次の何れかで停止する。

(i) 新たな等式または不等式を推論しえなくなったとき。

(ii) 空区間が推論されたとき。

(i) の状況の系を定常状態と呼ぶ。(ii) のとき方程式-不等式系 A を満足する解は無い。

はじめに，$\mathbf{OBS} = \{x_i \in [l_i, u_i] \mid 1 \leq i \leq m\}$ とし，次の有限集合（端点値集合と呼ぶ）B を定義する。

定義 20 端点値集合 B は 以下の有限集合である。

$$\mathbf{B} = \{0, 1\} \cup L \cup U \cup L^c \cup U^c$$

ただし，$L = \{l_i \mid 1 \leq i \leq m\}, U = \{u_i \mid 1 \leq \mathbf{I} \leq m\}, L^c = \{1 - l_i \mid 1 \leq i \leq m\}, U^c = \{1 - u_i \mid 1 \leq i \leq m\}$ である。

端点値集合 B の各要素を昇順 $(0 = b_1 < b_2 < \ldots < b_n = 1)$ にしたときに，隣合う要素 b_i, b_{i+1} から作られる区間 $[b_i, b_{i+1}]$ を端点区間と呼ぶ。また，B の要素を端点値と呼ぶ。

B の性質に関する次の命題は明らかに成立する。

命題 6

(i) 任意の x に対し，$x \in \mathbf{B}$ ならば，$1-x \in \mathbf{B}$ である。

(ii) $[l,u]$ が端点区間ならば，$[1-u,1-l]$ も端点区間である。

命題 7 任意の A に対して I は停止する。

証明　まず，変数の上下限値 u,l の値は任意の時点で端点値集合 B の要素である。これは B の定義と推論規則 I により数学的帰納法で容易に証明できる。B は有限集合であるから，それから作られる区間の数は有限個である。変数の数も有限個なので，推論されうる不等式は有限個しかない。同様に同値関係および補関係を表わす等式は有限個しかない。ゆえに I は無限に新しい等式および不等式を推論し続けることはない。 □

次に，解の存在と定常状態に関してつぎの命題を得る。

命題 8 A が充足可能ならば，I は定常状態になる。

証明　命題 7 により I は停止する。このとき (i) 定常状態，または (ii) ある変数が空区間となるの何れかが成立する。命題 5 により A 充足可能ならば (ii) ではない。よって，I は定常状態になる。 □

以上，推論システム I は健全であり停止性を満足することを示した。この I は次に述べる局所制約伝播によるインプリンメンテーションに適した規則の集合となっている。

6.3 制約伝播と端点値探索法

表 6.1 に示した推論システム I を用いて解の存在の有無を推論する訳であるが，規則適用の制御を制約伝播を用いて行う。本節でははじめに制約伝播のインプリメンテーションについて述べる。また，端点値探索法と呼ぶ充足性判定

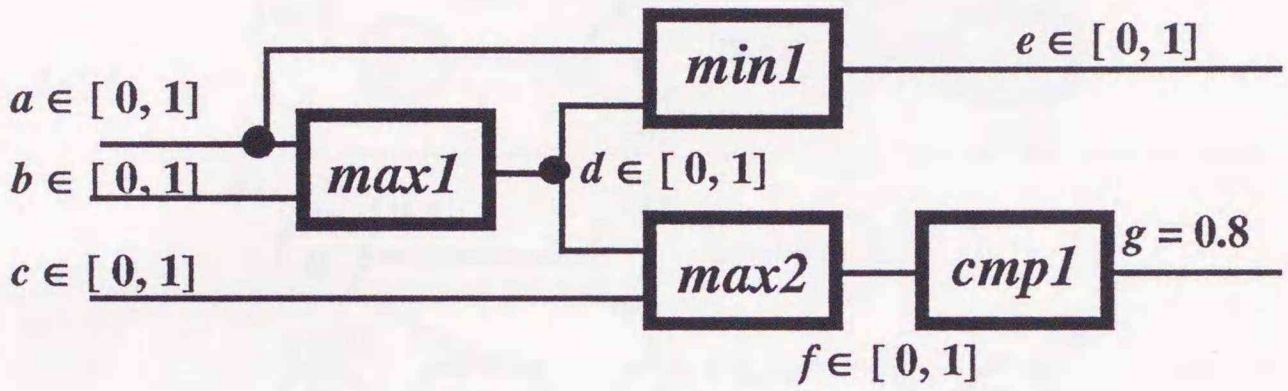


図 6.2: ファジィ論理回路例

アルゴリズムについても述べる。これと制約伝播を組み合わせて全体として完全な故障診断のための推論システムを構築する。最後にいくつかの実験結果を示す。

6.3.1 局所制約伝播

ここで用いる用いた制約伝播の方法は **Abelson** らが用いた局所制約伝播 (local constraint propagation) の方式を用いる [1]。局所制約伝播では，回路網を，その構造を反映した制約網 (constraint network) と呼ばれるネットワークで表現する。制約網は基本制約 (primitive constraint) と結合子 (connector) で構成される。

ここで，図 6.2 の回路を例として，具体的に制約網の上で局所制約伝播が行われる様子を図 6.3 で説明する。図 6.3-(a) は，結合子 g の観測 0.8 が (制約) とし

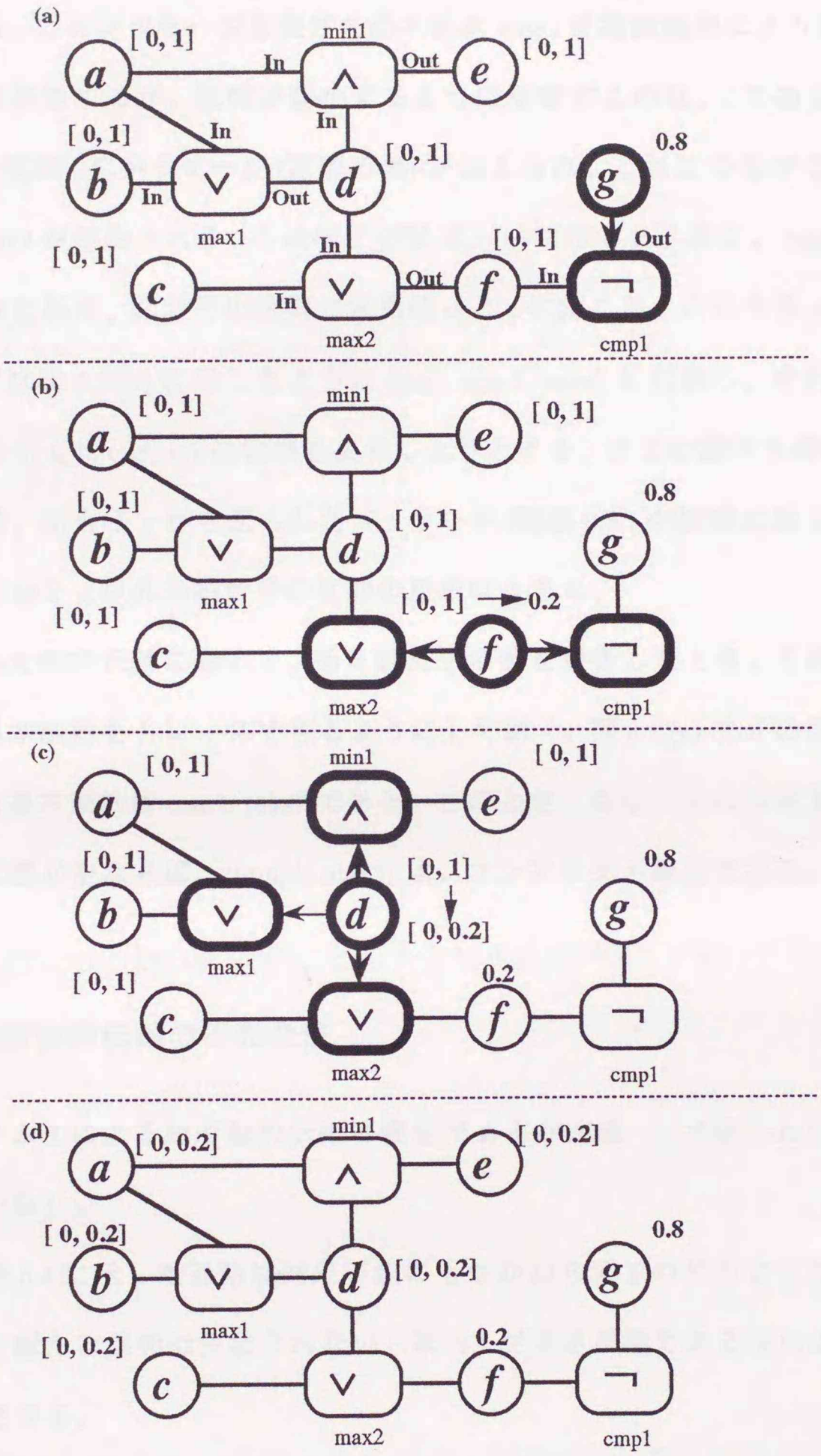


図 6.3: 局所制約伝播の動作

て g につながっているすべての基本制約 (この場合 cmp1 のみ) に伝達されたところである。このメッセージを受けた基本制約 cmp1 は推論規則により結合子 f と g の値を計算するが，区間が縮小するように更新するのは，f である。したがって，今度は f にメッセージ (区間の値) が伝えられ，これにつながる基本制約 cmp1, max2 が起動される。この様子を図 6.3-(b) に示した。次に， max2, cmp1 が起動された結果，結合子の区間の変更は d と c に起こる。このうち d からのメッセージは図 6.3-(c) に示したように min1, max1, max2 を起動し，それらにつながる結合子 a, b, c, d, e, f の区間を更新しようとする。以上の動作を繰り返し，最終的には，結合子 g から送られたメッセージ (観測 0.8) の影響は図 6.3-(d) に示すようになり g 以外の結合子の区間の更新は止まる。

以上述べた制約伝播において，ある結合子の値を更新したとき，それまでに関わった基本制約をトレースできるようにしておく。図 6.3-(c) で d の区間更新に関わった基本制約は cmp1, max2 である。このとき，もし， d の区間更新の結果，その区間が空ならば， {cmp1, max2} は，コンフリクト集合である。

6.3.2 局所制約伝播の不完全性

論システム I による局所制約伝播は健全であるが定義 19 で示した完全性を満足していない。

例えば図 6.4 に示した回路は充足不能にもかかわらず I のどのような規則を適用しても新たな情報は生成されない。図 6.4 が充足不能である理由は次のように説明できる。

出力の観測に 0.2 が現われている。また，回路には否定 (CMP) が含まれていない。よって，入力に観測 0.2 がなければならない。第 2 入力が 0.2 であると，第 1, 3 出力の観測が 0.2 でなければならない。これは観測 0.1 に矛盾する。

以上，局所制約伝播は完全性を満足していない。このため診断回路の構成に

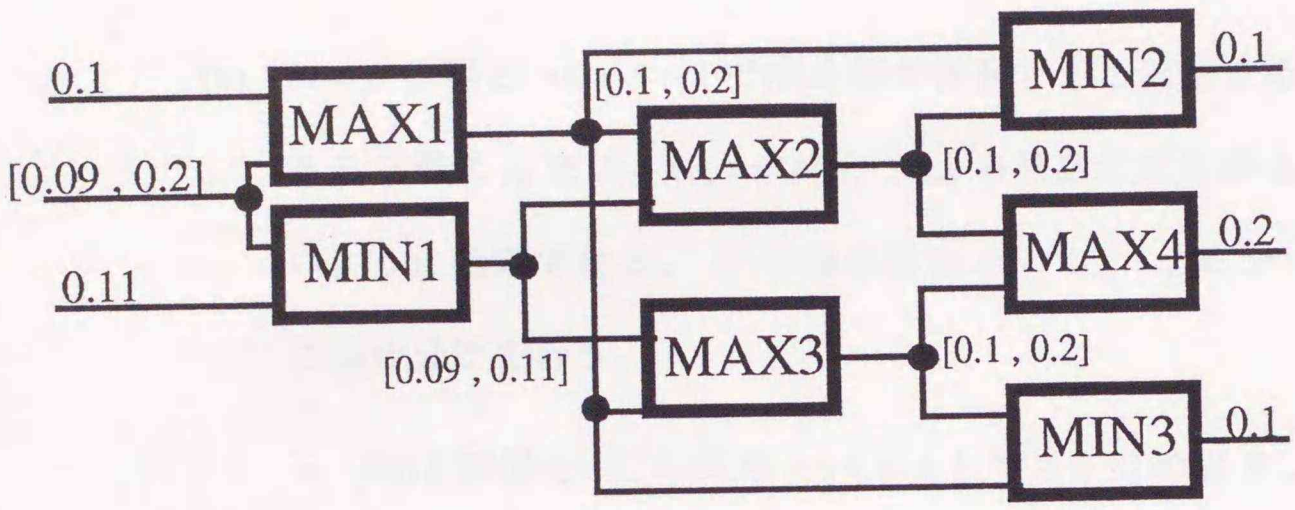


図 6.4: 局所制約伝播で定常状態になる回路

よっては誤った診断を導く可能性がある。よって，定理証明器は制約伝播の他に，系の大域的な一貫性 (解) を判定するような推論制御機能を持つ必要がある。そこで，大域的な充足性判定アルゴリズムである端点値探索法と呼ぶ方法を以下に提案する。この方法は局所制約伝播と共に定理証明器に組み込まれる。

6.3.3 端点値探索法

定義 21 すべての変数 $x \in V$ に対して $\sigma(x) \in \mathbf{B}$ であるような解釈 σ を端点解釈という。また，方程式-不等式系 **A** を満たす端点解釈を端点解と呼ぶ。

定理 6 方程式-不等式系 **A** に解が存在するならば，その系には端点解が存在する。

証明 **A** の解を σ とする。各変数 $x \in V$ に対して，$\sigma(x)$ の含まれる端点区間を $[l_x, u_x]$ とする $(l_x \leq \sigma(x) \leq u_x)$。このとき，$\sigma'(x) = l_x$ または $\sigma'(x) = u_x$ と

なるような端点解 σ' が存在することを，A の全方程式の数 m についての帰納法で証明する。

(i) $m=0$ の場合: この場合は系は不等式のみである。B の定義より明らかに端点解が存在する。

(ii) $m=k$ の場合: $m=k-1$ で端点解が存在すると仮定する。このとき，ある方程式 e_k を $\mathbf{A}-\{e_k\}=\mathbf{A}'$ かつ e_k の右辺の変数が $\mathbf{A}'$ に出現しないように指定できる。$\mathbf{A}'$ の端点解を σ' とし，e_k について次のように場合分けを行う。

- MAX の場合: この式を $a \vee b = c$ とする。このとき，右辺の変数 c は $\mathbf{A}'$ に出現しない。$\sigma(a), \sigma(b)$ の値が含まれている端点区間を $[l_a, u_a], [l_b, u_b]$ とする。なお，$\sigma(c)=\sigma(a) \geq \sigma(b)$ を仮定しても一般性は失われない。このとき，端点区間 $[l_a, u_a]$ は $\sigma(c)$ を必ず含む。はじめに，

  (1) 端点区間 $[l_a, u_a], [l_b, u_b]$ が異なる場合を考える。この場合の端点値の関係は，$l_b \leq u_b \leq l_a = l_c \leq u_a = u_c$ となる。よって，端点値 $\sigma'(a), \sigma'(b)$ が，それぞれ，$\sigma'(a) = (l_a$ または $u_a), \sigma'(b) = (l_b$ または $u_b)$ のどの組み合せにおいても，$max(\sigma'(a), \sigma'(b)) = \sigma'(a)$ となる。故に，$\sigma'(c) = \sigma'(a)$ となるようにσ' を拡張すれば，A の端点解となる。次に，

  (2) 端点区間 $[l_a, u_a], [l_b, u_b]$ が一致している場合 を考える。このとき，$l_a = l_b = l_c \leq u_a = u_b = u_c$ であり，$max(\sigma'(a), \sigma'(b)) = \sigma'(c)$ となるようにσ' を拡張する。

  以上，(1), (2) より，端点解が存在する。

- MIN の場合: MAX の場合と同様に端点解の存在が示される。

- CMP の場合: この式を $\neg a = b$ とする。このとき，右辺の変数 b は

A′ に出現しない。$\sigma(a)$ の値が含まれている端点区間を，$[l_a, u_a]$ とする。このとき，$l_a \leq \sigma(a) \leq u_a$ なので，$1-u_a \leq 1-\sigma(a) \leq 1-l_a$ が成立。また，命題 6 により，$[1-u_a, 1-l_a]$ は端点区間であり，$\sigma(b) = 1-\sigma(a)$ であるから，この端点区間は $\sigma(b)$ を必ず含む。よって，$\sigma'(a)$ が，l_a, u_a の何れであっても，$\sigma'(b) = 1-\sigma'(a)$ となるように σ' を拡張すれば，A の端点解となる。

- 結線の場合: この式を $a = b$ とする。$\sigma'(b) = \sigma'(a)$ とすればよいのは明らか。 □

定理 6 は，A の解の存在を有限回の探索により判定できることを示している。なぜならば，変数の数および B 共に有限なので，端点解釈の数も有限となるからである。具体的には，定理証明器として用いる端点値探索法のアルゴリズムは次のようになる。

(i) A の中のどの等式の右辺にも表れない変数 (独立変数) に対してのみ B の要素を割り当てる。

(ii) 順方向の制約伝播 (端点値伝播) を行ない，すべての変数の値を求める（この値は一意に定まる)。

(iii) 観測と矛盾が無ければ，それが端点解である。記号 √ を返す (A に大域的な解が存在)。

(iv) もし矛盾があれば，独立変数の割り当てを他の端点値の組み合わせにして，(ii) 以下を繰り返す。

(v) B の要素のすべての組み合わせに対して矛盾があれば端点解は存在しない。コンフリクト集合を生成する (端点値伝播にかかわった素子名の集合)。

このアルゴリズムは端点値探索法のみでも A の充足性の判定が可能であることを示している。しかし，この方法は，B が大きくなると，一般に効率は悪

い。一方，局所制約伝播による方法は不完全であるが効率は良い。さらに，実際の制約網では多くの場合，A が充足不能ならば局所制約伝播により変数の空区間を推論できる。そこで，実際の診断では，この二つの方法を組み合せて使用する。つまり，はじめに局所制約伝播による解の探索を行い，それが定常状態になった場合のみ端点値探索法を用いる。この結果，端点値探索法を用いるときには，すでに局所制約伝播によって各変数の区間が一般に縮小しているので，探索空間が縮小し，全体として完全で効率的な解の探索が期待できる。

6.4 制約網とデータ構造

図 6.5 は，制約網の例を表している。具体的な基本制約と結合子のデータ構造の例として，それぞれ max1 と d を示した。これらのデータは実際のプログラミングではオブジェクト指向で表現し max1 は制約クラス (MAX-CONSTRAINT)，d は結合子クラス (CONNECTOR) のそれぞれインスタンスである。

6.4.1 基本制約データ構造

max1 のデータ構造で，スロット Output, Input-a, Input-b はこの基本制約とつながる結合子へのポインタを意味する。max1 が属するクラスにつながるメソッドは，MAX 制約に関係する推論規則 I の (1) ~ (5) に基づいて得た値をこれらスロットのデータを参照して各結合子にメッセージ伝達する。他の基本制約のクラスにインプリメンテーションしたメソッドも同様な働きをする。すなわち，MIN-CONSTRAINT のメソッドは (6) ~ (10), CMP-CONSTRAINT については (13) の推論規則を適用する。

Active は，素子の異常を仮定するか否かのフラグである。もし，max1 の Active が記号 NIL の場合は，max1 の異常を仮定したことになり，max1 においては結合子にメッセージ伝達を行なわない。これは，システムから，ラベル max1 をも

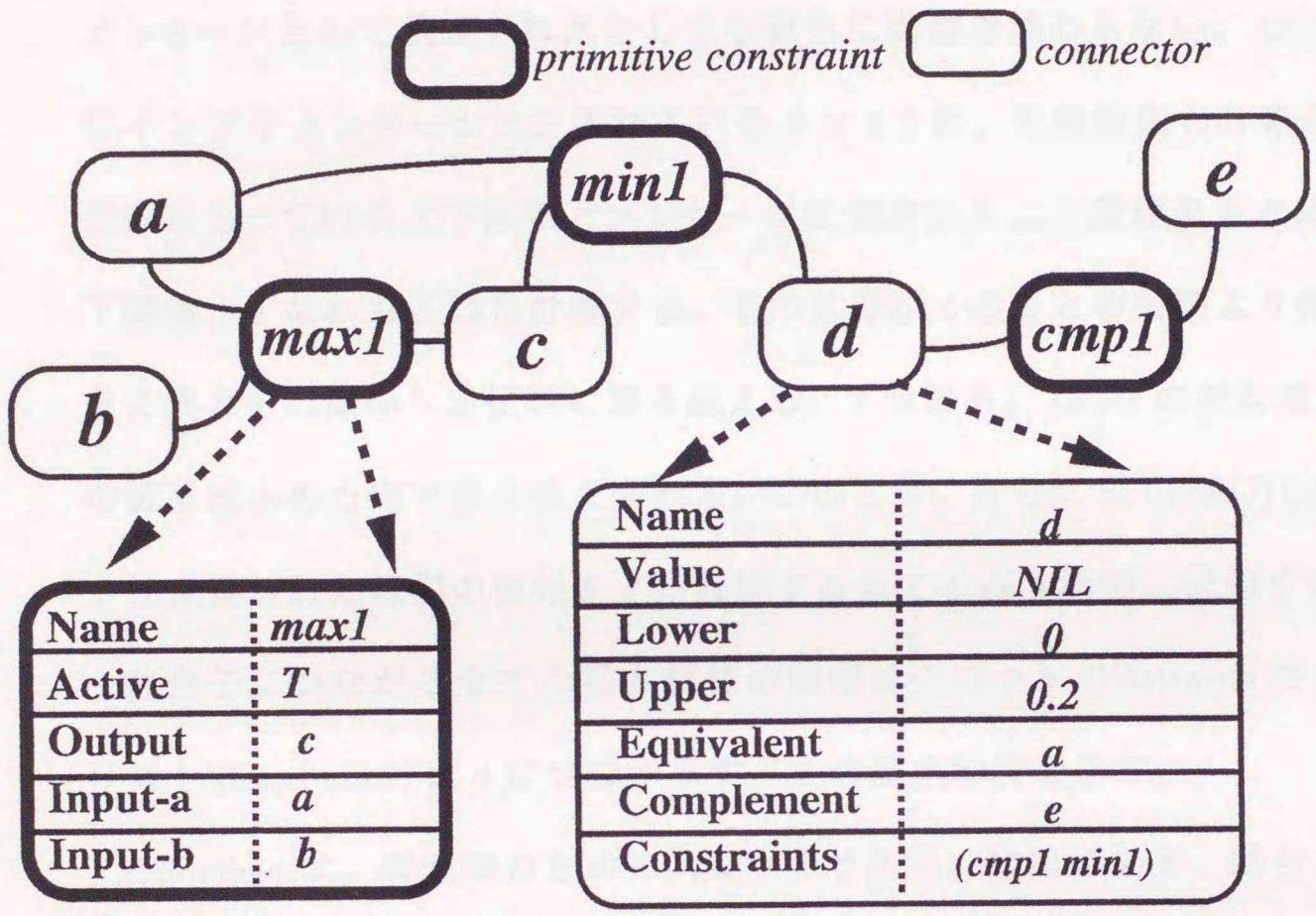


図 6.5: 制約網による回路表現とデータ構造

つ方程式を取り除く操作に対応する。

6.4.2 結合子データ構造

結合子 d のデータ構造で，スロット Lower, Upper は d のもつ区間の下限と上限の値を格納する。これらの値は，d につながる基本制約によって新しい区間がメッセージとして伝達されたとしても直ちには書き換わらない。CONNECTOR にインプリメンテーションされているメソッドは，推論規則 (16) を用いて，d が既にもっている上下限値とメッセージ伝達された上下限値をもとに新しく上下限値，すなわち区間を計算する。その区間が d のもとの区間より縮小したときに限りその縮小した区間に書き換える。すなわち，Lower の値は増加，Upper の値は減少の方向で書き換えられる。このとき，さらに CONNECTOR のメソッドは変更された区間の情報を d に接続する全ての基本制約に伝達する。

結合子につながる全ての基本制約の情報はスロット Constraints で示される。リスト (cmp1 min1) は d につながるすべての基本制約を示す。

Equivalent は，同値類のためのスロットである。推論の結果，結合子 $c_1, \ldots, c_5$ が，$c_1 = c_2, c_2 = c_3, c_4 = c_5$ のような等式が推論されたとする。これらを，同値類 $\{c_1, c_2, c_3\}, \{c_4, c_5\}$ で表現し，それぞれ唯一の代表元，例えば，c_1, c_4 に辿れるようにする。この同値類をデータ構造上では，代表元を根にもつ木で実現する (前述の例では，c_1 と c_4 を根にもつ二つの木)。Equivalent にはこの木の枝を表現するためのポインタを格納する。これらの木に対して Union-Find 操作が定義されている [3]。ここで，Find 操作とは，Equivalent ポインタをたどることにより結合子の代表元を求める操作を言う。Union 操作とは，異なる同値類に属する二つの要素 (結合子) c_2, c_5 に対して，$c_2 = c_5$ という新たな等式が推論されたときに，この二つの同値類を合併して一つの同値類 (木) にする操作である。

結合子に新たな等式が推論されると，Equivalent の値が調べられ，これが記

号 NIL であれば代表元とみなし，Lower, Upper が有効となる。そうでなければ，Find 操作が行なわれ，唯一の代表元に到達する。その代表元に対し推論規則 (12) が適用される。また，結合子に新たな等式が推論されると，区間が縮小したとき同様に Constraints に示される全てのオブジェクトにその推論結果が伝達される。

結合子 c_1 と c_2 が補関係にあるとき，c_1 の属する同値類の代表元 c_{01} と c_2 の属する同値類の代表元 c_{02} が，Complement 中のポインタにより，互いに他を指すようにしておく。結合子に Equivalent および Complement をもたせることにより，暗黙に表 6.1 に示した中の同値，補関係の推移律 (11), (14), (15) が適用可能となる。

Value スロットは，6.3.3 の端点値探索法で述べた端点値を格納する。

以上述べてきた局所制約伝播の機構とデータ構造は LISP によるオブジェクト指向プログラミングの方法 (CLOS) で記述した。MAX, MIN, CMP の基本制約のクラスと，結合子（変数）のクラスを準備した。なお，CLOS による制約伝播のインプリメンテーションについての基本的な考え方は，文献 [26] に述べられている。

6.5 ファジィ論理回路診断実験

これまで述べた診断アルゴリズム，局所制約伝播および端点値探索法を組み込んだ診断システム **DiaLog-II** の構成図を図 6.6 に示す。なお，以下の **DiaLog-II** による診断実験は **Macintosh II** 上で実行されたものである。

はじめに，図 6.7 の回路について診断を行なった。この回路の **SD** および **OBS** は次のようになる。

$$\mathbf{SD} = \{\mathrm{MIN1} : min(in1, in2) = l_1,$$

$$\mathrm{MAX1} : max(in2, in3) = l_2,$$

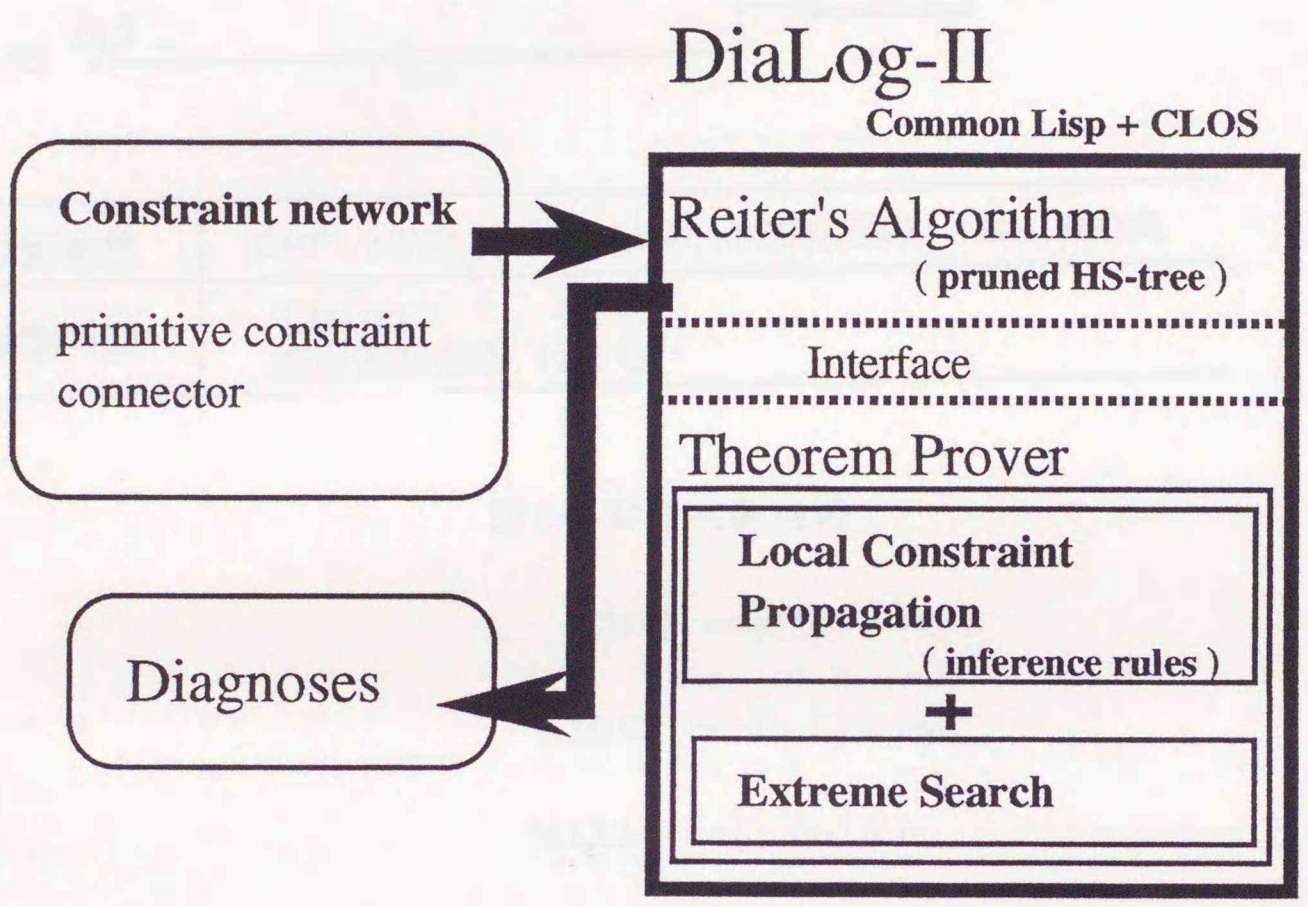
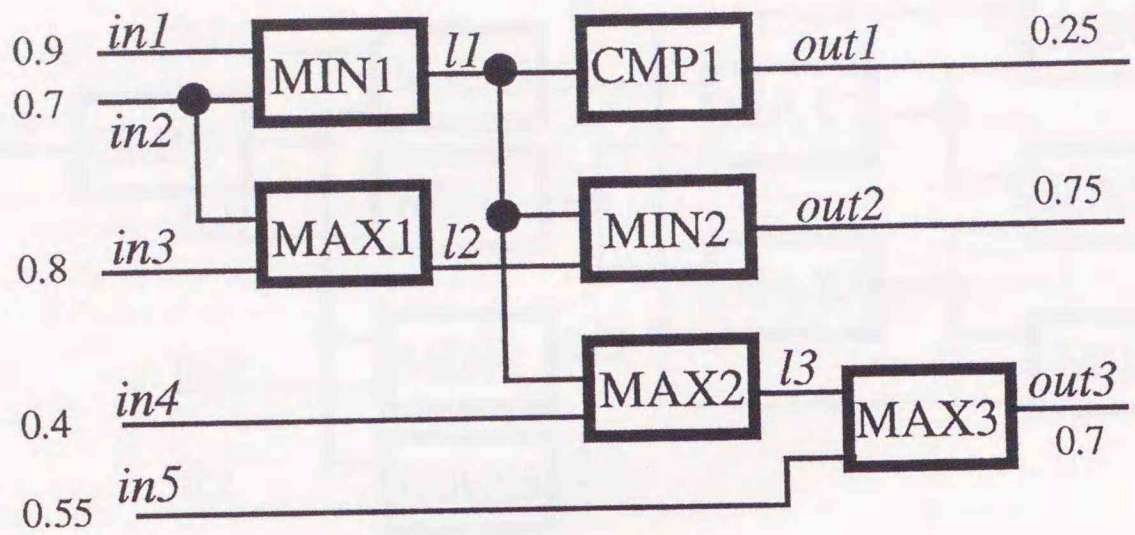


図 6.6: ファジィ論理回路診断システム DiaLog-II



診断結果	{CMP1, MIN2}, {CMP1, MIN1, MAX2}, {CMP1, MIN1, MAX3}
診断時間	DiaLog II 4.1 秒 端点値探索法 13.8 秒

図 6.7: 診断実験回路 1

$$\mathrm{CMP1} : cmp1\,(l_1) = out1,$$

$$\mathrm{MIN2} : min(l_1, l_2) = out2,$$

$$\mathrm{MAX2} : max(l_1, in_4) = l_3,$$

$$\mathrm{MAX3} : max(l_3, in_5) = out3\}$$

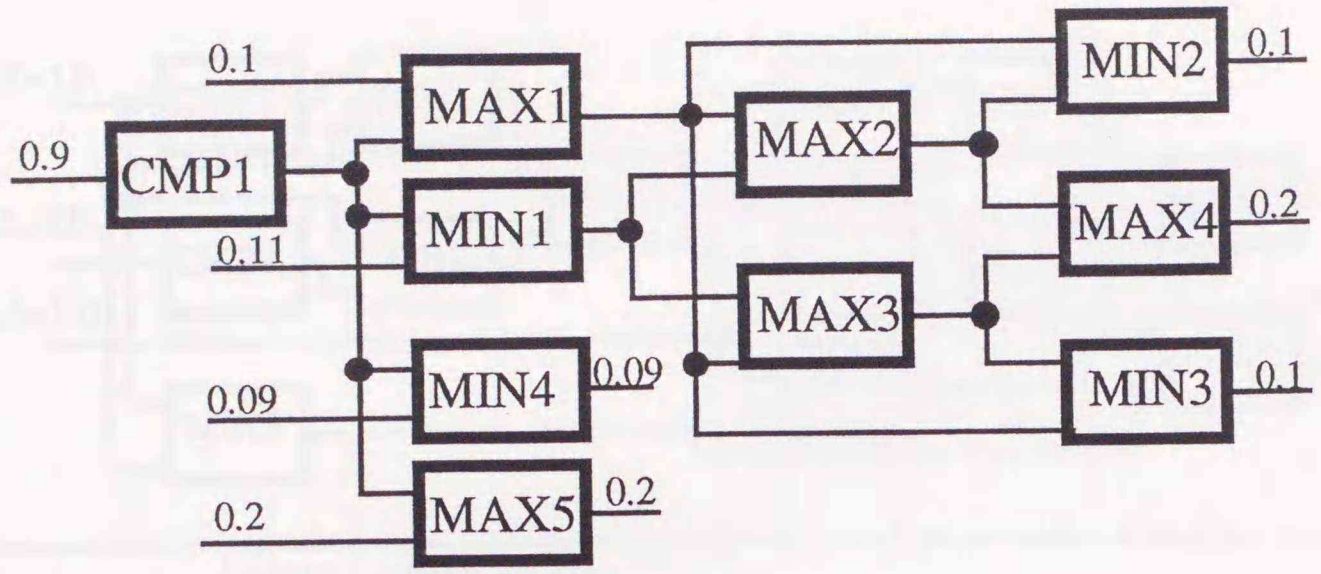
$$\mathbf{OBS} = \{in1 = 0.9, in2 = 0.7, in3 = 0.8, in4 = 0.4,$$

$$in5 = 0.55, out1 = 0.25, out2 = 0.75, out3 = 0.7,$$

$$l_1 \in 1], l_2 \in [0, 1], l_3 \in [0, 1]\}$$

COMPONENTS = {MAX1, MAX2, MAX3, MIN1, MIN2, CMP1 }

図 6.7 に示すように，この回路の故障は単一の要素では回路の異常 (単一故障) を説明できず，一つの二重故障と二つの三重故障が候補として生成された。なお，定理証明器に端点値探索法のみを用いたときの診断時間は局所制約伝播を組み合わせた診断時間の約 3 倍となった。

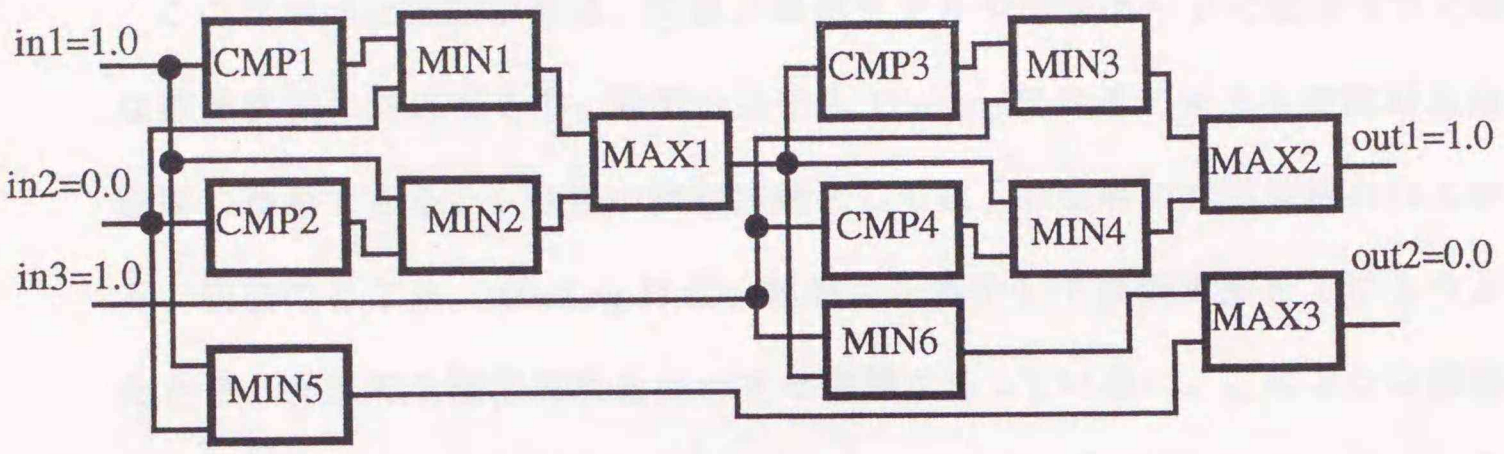


診断結果	{MAX2}, {MAX3}, {MAX4}, {MIN1}, {MAX1, MIN2, MIN3}, {CMP1, MIN2, MIN3}
診断時間	DiaLog II 17.9 秒 端点値探索法 11.7 秒

図 6.8: 診断実験回路 2

次に図 6.8 の回路について診断した。四つの単一故障と二つの三重故障が得られた。図 6.8 では，CMP1 を故障と仮定したとき，図 6.4 に示した回路と同じ状況になる。したがって，端点値探索法を用いなければ充足不能性を検出できないので，CMP1 の単一故障が得られてしまったところである。診断結果から分かるように DiaLog-II では完全な推論が行なわれている。なお，この実験では，端点値探索法のみを用いたときの方が短時間で診断が得られている。これは，図 6.7 の回路の端点値集合の要素数が 16 に対し，図 6.8 では 10 と端点値集合が小さいことが考えられる。一般に，端点値集合が大きい場合，または多重故障が多く表われる診断では DiaLog-II の方が効率が良い (n 重故障の場合，独立変数の未知数は n 個であり，考慮すべき端点解釈の数は n に関して指数的に増大する)。

DiaLog-II は観測が 0, 1 のみであれば 2 値論理回路も診断可能である。図 6.9 は 2 値論理回路 (全加算器) の診断を示している。汎用定理証明器による診断と



診断結果	{MAX1}, {MIN2}, {CMP2}, {MAX3, MAX2}, {MAX3, MIN3}, {MAX3, CMP3}, {MAX3, MIN4}, {MAX3, CMP4} {MIN6, MAX2}, {MIN6, MIN3}, {MIN6, CMP3}, {MIN6, MIN4}, {MIN6, CMP4}
診断時間	DiaLog II 16.4 秒 端点値探索法 16.3 秒 汎用定理証明器 109.7 秒

図 6.9: 診断実験回路 3

は，4, 5 章で述べた定理証明器，すなわち診断対象を一階述語論理の節集合で表現し，導出原理による定理証明器を使用した診断時間である。

以上示したように，DiaLog-II による診断では，多重故障を含めすべての診断を求めることができ，機能的には十分実用的であると考えられる。また，汎用定理証明器による診断と比較して性能面でも有効性が確認できた。

6.6 議論

本章では，初めに，ファジィ論理回路を診断対象システムとして方程式-不等式の有限集合で表現した。すなわち診断対象を論理の集合で表現し，この集合に対して Reiter の診断理論に現われた故障の定義，コンフリクト集合，診断の計算などを対応付けた。次に，定理証明器を構成するための推論規則およびその制御に用いた局所制約伝播と端点値探索法について述べた。さらに診断理論と定理証明器を組み込んだシステム DiaLog-II を用いて実際にファジィ論理回

路の故障診断実験を行った。

ここで述べた診断手法は，特定の故障モデルや故障モードに依存せずに様々な故障状況に対応できる。診断システム DiaLog-II はそのまま 2 値論理回路の診断に適用できる。 今後の検討課題としては，診断時の対話環境の向上がある。現在のところ，DiaLog-II は，観測点を増やして診断精度を上げようとしたとき，効果的な観測場所を指示する機構はもっていない。このような観測点の最良選択に関する研究は，de Kleer によって最近発表された [8]。また，将来の展望としては，ファジィ論理回路が 2 値論理回路またはアナログ電子回路と結合されているときの故障診断，さらにはファジィ・フリップ・フロップなど時間素子を含む回路の診断技術の開発が考えられる。

第 7 章

結論

7.1 各章の要約

7.1.1 第 1 章

第 1 章は序論であり，本研究の目的，意義，研究の新規性を述べている。また論理回路の診断技術の大きな二つのアプローチ，すなわちシミュレーションと推論による方式を示し，本研究での診断アプローチと比較している。

7.1.2 第 2 章

第 2 章では，本研究で開発した診断システムで採用した Reiter の故障診断理論を簡潔にまとめている。 Reiter の故障診断理論では，診断対象とするシステムの概念を特定の領域に依存しないように考慮した汎用性の高いものとなっている。すなわち，システムの構造や動作を規定する理論と現在のシステムの動作状況 (システム観測) を適当な論理，例えば述語論理などで統一的に表現し，システムが故障であるとはこの論理式の集合に矛盾しているものが存在していると考えた。この考えに基づき次のように診断を定義した。

(**SD**, **COMPONENTS**, **OBS**) についての診断とは、

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\mathbf{AB}(C) \mid C \in \Delta\} \cup \{\neg\mathbf{AB}(C) \mid C \in \mathbf{COMPONENTS} - \Delta\}$$

が無矛盾であるような極小集合 $\Delta \subseteq$ COMPONENTS である。

ただし，SD はシステムの構造や挙動を表す第一階論理式の集合，COMPONENTS はシステムの故障し得る構成要素を表す定項の有限な集合である。SD には「異常である」を意味する特別な述語 AB を用いている。また，OBS はシステム観測で第一階論理式の有限な集合で表される。

Reiter は，極小集合 Δ を求める効率のよい診断アルゴリズムを示した。このアルゴリズムは次に示すコンフリクト集合を利用する。

コンフリクト集合とは、

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\neg \mathbf{AB}(C_1), \ldots, \neg \mathbf{AB}(C_k)\}$$

が矛盾となる集合 $\{C_1, \ldots, C_k\} \subseteq$ COMPONENTS である。

この集合の意味は，コンフリクト集合に含まれるすべての要素の動作の正常性は仮定できないことを示している (この集合の中に少なくとも一個の故障要素が含まれる)。このコンフリクト集合を (SD, COMPONENTS, OBS) の中から推論システムにより探索し，これとヒット集合の概念を用いて極小集合 Δ を決定する。しかし，後に Greiner によって Reiter の診断アルゴリズムではシステムによっては不完全な診断を行うことが指摘され，アルゴリズムが改訂された。2 章の後半ではこの改訂された故障診断アルゴリズムについても解説している。本研究で行った各種の診断実験も Greiner の改訂されたアルゴリズムで行っている。このアルゴリズムを common lisp でインプリメントしたプログラムを本論文の付録で示した。

7.1.3 第 3 章

第 3 章では，汎用自動推論システム Thinker の詳細を示してある。自動推論システムの効率に問題があると実用的な診断システムは構成できない。

Thinker では，原理的な推論方式である二元導出を採用していない。第 3 章の前半では，二元導出は結果 (導出節) を得るまでに中間節が多数生成され，節が組み合わせ的に急速に増加し，したがって推論効率が著しく損なわれることを実例を示して説明している。Thinker では原則として二元導出は採用せず，超導出および UR-導出という強力で実用的な推論規則を採用している。超導出および UR-導出のいずれも中間節を生成することなく直接複数の親節から一つの導出節を導くものである。第 3 章ではさらに等式論理による推論方式（デモジュレーションおよびパラモジュレーション）についても述べている。この結果，Thinker では，推論時に等号の満たす反射性，対称性あるいは代入則などを表す公理を準備する必要がなく，自動推論システム全体として推論効率を上げることに成功している。なお，この汎用自動推論システムはそれ自体独立のインターフェイスを備え，反駁完全性に基づく証明を出力できる。

第 3 章の最後に，Thinker がそれ自体独立に動作するシステムであることを示すため，推論による全加算器構成の実例 (入力ファイル，証明リスト) が示されている。

7.1.4 第 4 章

第 4 章は，3 章で示した汎用自動推論システム Thinker を用いた組合せ論理回路に対する故障診断実験が示されている。章のはじめでは，全加算器を例にして組合せ論理回路の故障診断を行なう上で必要な知識表現の全体を示している。すなわち，SD, COMPONENTS および OBS を実際の回路ではどのように述語論理の節で表現するのかを具体的に示した。Thinker では効率的に推論を行なうため，推論の開始前にこれらの節を一般公理集合 (問題領域で広く成立する知識)，支持集合 (特定の問題に固有の知識)，パラモジュレータおよびデモジュレータ (等式論理で使われる知識) 等に分割して配置する。この節配置の仕

方が診断効率に重要な影響を及ぼすことが実験により明らかにされる。

次に示されるのは，診断のための知識の一部欠損など，診断のための環境が悪化した状況での診断結果である。これは，リアルタイム診断で，いままで得られた診断対象からの観測がとだえた状況(センサーの故障)である。これらの実験を通していくつかのヒューリステックスが得られた。その主なものは，節の配置に関するヒューリステックスおよび超導出，単位消去およびデモジュレーションの使用が効率向上に有効であること，パラモジュレーションは後ろ向き推論の効果を発揮して推論継続能力が増大し，推論システムが頑健(robust)になることなどである。これらヒューリステックスは，Thinkerの推論部の調整に利用され，さらにこの推論部とReiterの診断アルゴリズム(厳密にはGreinerによって改訂されたアルゴリズム)とが組合わされて診断システムDiaLog-Iが作成される。なお，章の後半ではプロダクション・システムによる推論制御が簡単な論理回路に対しても不完全な診断を行う場合があることを議論している。

7.1.5 第5章

第5章では，大規模論理回路および順序回路の診断を扱う。このためReiterの理論のパッケージ・レベル診断への拡張をおこなっている。これは回路の素子レベルで故障を特定できても，実際の保守の場面ではその素子を含むパッケージ交換(あるいは基板交換)となる場合が多いことに着目したアルゴリズムである。例えば，ICではその規模によらずその中身を修理することは考えられない。また最近は，ある基板上(例えばメモリ基板，インターフェイス基板など)での動作不良が発見されても，取り敢えず予備の基板交換で装置を復旧させ後に故障部品が探索交換されるのが一般的である。パッケージ・レベル診断により故障素子推論のため探索空間が大幅に縮小され大規模回路に対して効率的な診断が行われる。

パッケージ診断の定義は次のように示される。

PACKAGES を COMPONENTS の分割とすると，

(SD, COMPONENTS, PACKAGES, OBS) についてのパッケージ診断とは、

$$\mathbf{SD} \cup \mathbf{OBS} \cup \{\neg \mathbf{AB}(c) \mid c \in \mathbf{COMPONENTS} - comp(\Delta_p)\}$$

が無矛盾であるような極小集合 $\Delta_p \subseteq$ PACKAGES である。

ここで，パッケージの集合 Δ_p 中に現れる要素の集合を

$$comp(\Delta_p) = \{c \mid (\exists P) c \in P \in \Delta_p\}$$

とする。

第 5 章では，この定義に基づいた診断アルゴリズムが示されている。ただし，このアルゴリズムは少なくともパッケージ内のどの要素が異常なのかを説明できるような診断 (根拠付き診断) となっている。パッケージ診断の実験例として 2 ビットデコーダが示されている。続いて，5 章では順序素子の知識表現が示される。順序回路は現在の入出力観測だけでは故障検出は不可能で，回路の時間履歴をコントロールする必要がある。

JK フリップ・フロップでは節表現で次のように示される。

$$\neg \mathbf{JKFF}(x) \mid \mathbf{AB}(x) \mid \mathbf{EQUAL}(v(s(t), q(x)), jk(v(t, j(x)), v(t, k(x)), v(t, q(x))))$$

$$\mathbf{EQUAL}(jk(x, y, z), or(and(x, not(z)), and(not(y), z)))$$

関数 j, k, q は JK-FF の入出力端子を意味する。関数 s (successor) を導入して，時刻 t の 1 クロック後，すなわち，$(t+1)$ ビット時刻でのフリップ・フロップの出力を規定するようにした。

パッケージ診断アルゴリズムおよび時間素子の知識表現が示されたので，診断への応用として自動販売機制御部の故障診断を行なっている。この実験によ

り比較的大規模な順序素子を含む回路であっても，Reiter のアルゴリズムを拡張することにより述語論理の範囲内で実用的な診断が可能であることが示された。

7.1.6 第 6 章

第 6 章では，方程式-不等式系論理に基づく診断システム (DiaLog-II と呼ぶ) を新たに構成し，それをファジィ論理回路の診断に応用している。ファジィ論理回路とは，MAX(∨),MIN(∧), CMP(¬) のゲートが結線されたものである。これは，以下のように診断対象システムとして表現している。

ファジィ論理回路のシステム記述 SD は (ラベル付き) 方程式の有限集合で，以下の条件を満たすクラス S に属するものである。

(i) $\{\} \in \mathbf{S}$

(ii) $\mathbf{SD} \in \mathbf{S}$ ならば，

(a) $\mathbf{SD} \cup \{C : x \vee y = z\} \in \mathbf{S}$

(b) $\mathbf{SD} \cup \{C : x \wedge y = z\} \in \mathbf{S}$

(c) $\mathbf{SD} \cup \{C : \neg x = z\} \in \mathbf{S}$

(d) $\mathbf{SD} \cup \{C : x = z\} \in \mathbf{S}$

ただし，C はラベル (素子名) で，SD で用いられていないものである。また，回路はループをもっていないものとする。

この定義はファジィ論理回路の構造および観測される入出力値を方程式と不等式の集合で表現したことになる。この集合の上で，Reiter の診断理論で表われるコンフリクト集合を定義した。この結果，診断対象の知識表現が方程式- 不等式系であっても Reiter の診断アルゴリズムはそのまま使用できる。しかし，コンフリクト集合の検出は述語論理の節表現を対象としたものは利用できない。そこで，各ファジィ素子に対して推論規則を定め，推論制御に局所制約伝

播を用いてコンフリクト集合を検出した。しかし，局所制約伝播で用いた推論規則のみでは故障の検出に不完全な場合があることが判明し，新たに端点値探索法と呼ぶ充足性判定法を提案し，最終的に局所制約伝播と組み合わせて診断システムを構築した(DiaLog-II)。章の後半ではこのシステムによるファジィ論理回路の診断実験が示されている。

7.2 まとめ

本論文で示したシステム DiaLog (I および II) は，論理をベースとした知識工学の手法により開発された深いモデルに基づく故障診断システムであり，その主要部分は診断系と推論系とから成っている。診断系は Reiter の数学的に厳密な故障診断理論に基づいて構築されており，さらに，著者が考案したパッケージ・レベル診断アルゴリズムによりその機能が拡張されて，効率と実用性が改善されている。推論系は論理回路の問題分野特有の知識表現，推論規則，推論制御戦略により，論理的な完全性を失うことなしに効率良く推論を行なう。実際，組合せ回路，順序回路，ファジィ論理回路を含むいくつかの実験により，その機能および効率の面で満足のゆく結果が得られた。

しかし，次にあげる検討課題が残されている。

一つはユーザインターフェースの問題である。例えば，診断精度が上がらないとき（すなわち診断対象システムの構成要素とほぼ同じ数の単一故障が診断されるとき），診断を絞りこむのに必要な最も効果的な観測点を診断システムからユーザに指示できれば，ユーザにとって使い易い故障診断システムとなる。

次の検討課題は知識獲得の問題である。ある製品が開発される場合，膨大な知識が集約される。例えば，コンピュータ関連の機器の場合，ブール代数，物性工学，電子回路工学，実装技術などの理論が駆使されて設計される。これらは当然その分野毎に適した知識表現をもつ。これらを論理，例えば本研究で採

用した述語論理の節表現などに機械的に変換できれば，知識獲得のコストは大幅に低減できる。

最後に，各種のアプローチを融合した故障診断システムの構築がある。故障シミュレーション，浅いモデル，深いモデルの各アプローチはそれぞれ一長一短がある。これらが互いに他の欠点を補完し合うように融合し，総合的に高機能で効率的な故障診断システムを構築することがこの分野の最終目標の一つである。

謝辞

北海道大学工学部大内 東教授には本研究を遂行するにあたり深い議論を通してご指導いただくとともに多くの御援助，励ましをいただきました。深甚の謝意を表します。本研究に対して有意義な御示唆と詳細な議論をいただいた北海道大学工学部新保 勝教授，青木由直教授，嘉数侑昇教授に深く感謝申し上げます。また，研究全般にわたり終始あたたかい御教授，具体的御助言をいただきました北海道大学工学部栗原正仁助教授に心から感謝申し上げます。本研究の機会を与えていただいた北海道自動車短期大学小澤保知学長および数多くの御支援をいただきました北海道工業大学加地郁夫教授に末尾ながら厚くお礼申し上げます。

参考文献

[1] H. Abelson, G.J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MaGraw-Hill, New York, 1985.

[2] 安居院, 内藤. 論理回路の故障診断. 電子科学シリーズ 68. 産報出版, 1978.

[3] エイホ A.V., ほか, (訳) 野崎ほか. アルゴリズムの設計と解析 I. サイエンス, 1984.

[4] 新井ほか. 診断型エキスパート・システム. 情報処理, Vol. 28, No. 2, pp. 177+, 1989.

[5] G. Birtwistle and Subrahmanyam P.A., editors. *Current Trends in Hardware Verification and Automated Theorem Proving*. Springer-Verlag, 1989.

[6] C. Chang, et al. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[7] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, Vol. 24, pp. 347–410, 1984.

[8] J. de Kleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, Vol. 45, pp. 381–391, 1990.

[9] J. de Kleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, Vol. 32, pp. 97–130, 1987.

[10] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, Vol. 3, pp. 69–116, 1987.

[11] 藤原. テスト生成と故障シミュレーション. 情報処理, Vol. 25, No. 10, pp. 1119–1124, 1984.

[12] R. Greiner, B.A. Smith, and R.W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, Vol. 41, pp. 79–88, 1989.

[13] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, Vol. 27, No. 4, pp. 797–821, 1980.

[14] 栗原, 大内, 加地. Automated reasoning に基づくシステム問題へのアプローチ. 電気学会論文誌 C, Vol. 107, No. 2, pp. 141–148, 1987.

[15] 水本. ファジィ理論とその応用. Information & Computing 19, サイエンス, 1988.

[16] 長尾, 淵. 論理と意味. 岩波講座情報科学 7. 岩波, 1983.

[17] Y. Nakagawa, M. Kurihara, and A. Ohuchi. Diagnosis of fuzzy logic circuits based on constraint propagation. In *2nd Pacific Rim International Conference on Artificial Intelligence*, Vol. 2, pp. 1298–1302, sep 1992.

[18] Y. Nakagawa, T. Oda, M. Kurihara, and A. Ohuchi. Package-level fault diagnosis of digital circuits based on automated resoning. In *Proceedings of the 29th SICE Annual Conference*, Vol. 2-II, pp. 685–688, jul 1990.

[19] 中川, 栗原, 大内. パッケージの概念を導入した故障診断アルゴリズムの改訂. Technical report, 情報処理学会研究報告, 1990. 90-AL-16.

[20] 中川, 北谷, 栗原, 加地. 論理回路の故障診断における自動推論システムの実験とヒューリスティクス. 電気学会論文誌 C, Vol. 109, No. 10, pp. 731–738, 1989.

[21] 中川, 小田, 栗原, 大内. 自動推論による順序回路のパッケージ・レベル故障診断. 電気学会論文誌 C, Vol. 110, No. 8, pp. 500–507, 1990.

[22] 中川, 扇, 栗原, 大内. 制約伝播を利用したファジィ論理回路の故障診断. 情報処理学会論文誌, Vol. 34, No. 2, pp. 331–341, 1993.

[23] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, Vol. 32, pp. 57–95, 1987.

[24] G. Saucier, A. Ambler, and M.A. Breuer, editors. *Knowledge Based Systems for Test and Diagnosis*. North-Holland, 1989.

[25] タウンゼント C., (訳) 玄ほか. TURBO Prolog エキスパート・システム設計入門. HBJ 出版局, Nov 1989.

[26] P.H. Winston and B.K.P. Horn. *LISP*. Addison-Wesley, third edition, 1989.

[27] ウォス L., ほか, (訳) 川越ほか. コンピュータによる推論技法. マグロウヒル, 1989.

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

付録 A

図・表目次

図目次

表目次

付録 B

診断アルゴリズム・ソースリスト

DiaLog-I , DiaLog-II で用いられた **Reiter** の診断アルゴリズムを Common Lisp でインプリメンテーションした全リストを以下に示す。

定理証明器は関数名 **TP** としてある。**TP** は関数名 **NODE-EXPAND** で使われる。**TP** の引数は，故障と仮定した要素のリストであり，返す値は診断対象が無矛盾の場合は **NIL**, 矛盾の場合はコンフリクト集合を想定している。

```
;;; Diagnose system based on Logic(DiaLog) was implemented in 1989.
;;; Next, this system is to be developed including a concept of Package.
;;; To represent a HS-node, use a structure.
;;; A HS-node is represented using a structre as follows.
;;;
;;; A number of the node is in NUMBER slot.
;;; A list of lists which contains a parent node's number
;;; and edge's label  is in PARENT slot.
;;; A list of lists which containes child nodes' numbers
;;; and edge's label is in CHILDREN slot.
;;; A conflict set, nil, 'CHECKED or 'CLOSE is in LABEL solt.
;;; A depth from the root node is in LEVEL slot.
;;; A H(node) is in H slot.
;;; A HP(node) is in HP slot.

(DEFSTRUCT HS-NODE
  (NUMBER NIL)
  (PARENTS NIL)
  (CHILDREN NIL)
  (LABEL NIL)
  (LEVEL NIL)
  (H NIL)
  (HP NIL))

;;; Gloval variables and initializing functions.

;; Nodes are stored in HASH-TABLE
```

```
(DEFVAR *HS-NODE-TABLE* NIL)

(DEFUN INIT-HS-NODE-TABLE ()
  (SETF *HS-NODE-TABLE* (MAKE-HASH-TABLE :TEST #'EQL
                                         :SIZE 1000)))

(DEFUN GET-NODE (NODE-NUMBER)
  (GETHASH NODE-NUMBER *HS-NODE-TABLE*))

(DEFVAR *HS-NODE-COUNT* 0)

(DEFUN INIT-HS-NODE-COUNT ()(SETF *HS-NODE-COUNT* 0))

(DEFUN PRINT-HS-NODE-TABLE ()
  (DOTIMES (I *HS-NODE-COUNT* NIL)
    (FORMAT T "~%HS-NODE(~S) is ~S" I (GET-NODE I))))

;; Test of root node.
(DEFUN HS-NODE-ROOT-NODE-P (NODE-NUMBER)
  (= 0 (HS-NODE-LEVEL (GET-NODE NODE-NUMBER))))

;; HS-dag queue
(DEFVAR *HS-NODE-QUEUE* NIL)

(DEFUN INIT-HS-NODE-QUEUE ()
  (SETF *HS-NODE-QUEUE* NIL))

;; Confict sets
(DEFVAR *SET-OF-CONFLICT-SETS* NIL)

(DEFUN INIT-SET-OF-CONFLICT-SETS ()
  (SETF *SET-OF-CONFLICT-SETS* NIL))

;; Diagnoses and Package diagnoses

(DEFVAR *DIAGNOSIS* NIL)

(DEFUN INIT-DIAGNOSIS ()
  (SETF *DIAGNOSIS* NIL))

(DEFVAR *PACKAGE-DIAGNOSIS* NIL)

(DEFUN INIT-PACKAGE-DIAGNOSIS ()
  (SETF *PACKAGE-DIAGNOSIS* NIL))

;; Stop level

(DEFVAR *DIAGNOSIS-LEVEL* NIL)

(DEFUN INIT-DIAGNOSIS-LEVEL ()
  (SETF *DIAGNOSIS-LEVEL* NIL))

(DEFUN SET-DIAGNOSIS-LEVEL (LEVEL)
  (SETF *DIAGNOSIS-LEVEL* LEVEL))
```

```
;;;  Reusing node
;;;
(DEFUN REUSE-NODE (H)
  (DOTIMES (I *HS-NODE-COUNT* NIL)
    (LET ((H-OF-GIVEN-NODE (HS-NODE-H (GET-NODE I))))
      (IF (SET-EQUAL-P H H-OF-GIVEN-NODE)
        (RETURN-FROM REUSE-NODE I)))))

(DEFUN SET-EQUAL-P (S1 S2)
  (AND (LISTP S1)(LISTP S2)
       (SUBSETP S1 S2)(SUBSETP S2 S1)))

;;; Closing
;;;
(DEFUN CLOSE-NODE-P (NODE-NUMBER)
  (LET ((NODE (GET-NODE NODE-NUMBER)))
    (DOLIST (DIAGNOSIS *DIAGNOSIS* NIL)
      (WHEN (SUBSETP DIAGNOSIS (HS-NODE-H NODE))
        (SETF (HS-NODE-LABEL NODE) 'CLOSED)
        (RETURN-FROM CLOSE-NODE-P T)))))


;;; Reuse label
;;;
(DEFUN REUSE-LABEL (NODE-NUMBER)
  (LET ((H-OF-NODE (HS-NODE-H (GET-NODE NODE-NUMBER))))
    (DOLIST (GIVEN-CONFLICT-SET *SET-OF-CONFLICT-SETS* NIL)
      (IF (NULL (INTERSECTION GIVEN-CONFLICT-SET H-OF-NODE))
        (RETURN-FROM REUSE-LABEL GIVEN-CONFLICT-SET)))))


;;; PRUNING
;;;
(DEFUN DO-PRUNING (NEW-CONFLICT-SET)
  (DOLIST (CONFLICT-SET *SET-OF-CONFLICT-SETS*)
    (WHEN (PROPER-SUBSETP NEW-CONFLICT-SET CONFLICT-SET)
      (DOLIST (NODE-NUMBER (LABELED-NODES CONFLICT-SET))
        (LET ((NODE (GET-NODE NODE-NUMBER)))
          ; remove descendants
          (DOLIST (EDGE-LABEL (SET-DIFFERENCE CONFLICT-SET
                                              NEW-CONFLICT-SET))
            (LET ((REDUNDANT-CHILD (GET-CHILD NODE EDGE-LABEL)))
              (REMOVE-EDGE REDUNDANT-CHILD NODE-NUMBER)
              (REMOVE-DESCENDANTS REDUNDANT-CHILD)))
          ; relabel
          (RELABEL NODE NEW-CONFLICT-SET)))
      (ELIMINATE CONFLICT-SET))))

(DEFUN PROPER-SUBSETP (S1 S2)
  (AND (SUBSETP S1 S2)(NOT (SUBSETP S2 S1))))

(DEFUN LABELED-NODES (CONFLICT-SET)
  (LET ((NODES NIL))
    (DOTIMES (I *HS-NODE-COUNT* NODES)
      (LET ((NODE (GET-NODE I)))
```

```
        (IF (SET-EQUAL-P CONFLICT-SET (HS-NODE-LABEL NODE))
          (PUSH I NODES))))))

(DEFUN RELABEL (NODE NEW-CONFLICT-SET)
  (SETF (HS-NODE-LABEL NODE) NEW-CONFLICT-SET))

(DEFUN GET-CHILD (NODE EDGE-LABEL)
  (DOLIST (CHILD-LIST (HS-NODE-CHILDREN NODE) NIL)
    (IF (EQ EDGE-LABEL (SECOND CHILD-LIST))
      (RETURN-FROM GET-CHILD (FIRST CHILD-LIST)))))

(DEFUN REMOVE-EDGE (CHILD-NUMBER PARENT-NUMBER)
  (SETF (HS-NODE-PARENTS (GET-NODE CHILD-NUMBER))
        (REMOVE-IF #'(LAMBDA (X)
                       (= PARENT-NUMBER (FIRST X)))
                   (HS-NODE-PARENTS (GET-NODE CHILD-NUMBER))))
  (SETF (HS-NODE-CHILDREN (GET-NODE PARENT-NUMBER))
        (REMOVE-IF #'(LAMBDA (X)
                       (= CHILD-NUMBER (FIRST X)))
                   (HS-NODE-CHILDREN (GET-NODE PARENT-NUMBER)))))

(DEFUN REMOVE-DESCENDANTS (NODE-NUMBER)
  (LET ((NODE (GET-NODE NODE-NUMBER)))
    (WHEN (NULL (HS-NODE-PARENTS NODE))
      (REMOVE-NODE NODE-NUMBER)
      (DOLIST (CHILD-LIST (HS-NODE-CHILDREN NODE))
        (REMOVE-DESCENDANTS (FIRST CHILD-LIST))))))

(DEFUN REMOVE-NODE (NODE-NUMBER)
  (LET ((NODE (GET-NODE NODE-NUMBER)))
    (DOLIST (CHILD-LIST (HS-NODE-CHILDREN NODE))
      (REMOVE-EDGE (FIRST CHILD-LIST) NODE-NUMBER))
    (SETF (HS-NODE-CHILDREN NODE) NIL)
    ; relabel the node by 'CUT
    (SETF (HS-NODE-LABEL NODE) 'CUT)))

(DEFUN ELIMINATE (LABEL)
  (SETF *SET-OF-CONFLICT-SETS*
        (REMOVE LABEL *SET-OF-CONFLICT-SETS* :TEST #'SET-EQUAL-P)))

(DEFUN CUT-NODE-P (NODE-NUMBER)
  (EQ (HS-NODE-LABEL (GET-NODE NODE-NUMBER)) 'CUT))


;;  Level stopping
;;
(DEFUN LEVEL-STOP-P (NODE-NUMBER)
  (AND *DIAGNOSIS-LEVEL*
       (> (HS-NODE-LEVEL (GET-NODE NODE-NUMBER)) *DIAGNOSIS-LEVEL*)))


;;; NODE-EXPAND returns a list of children nodes and
;;; expands a HS-DAG.  If TP(theorm prover) returns NIL, a diagnosis
;;; is found, otherwise HS-DAG is expanded further.
```

```
(DEFUN NODE-EXPAND (NODE-NUMBER)
  (LET* ((NODE (GET-NODE NODE-NUMBER))
         (H-OF-NODE (HS-NODE-H NODE))
         (LEVEL-OF-NODE (HS-NODE-LEVEL NODE))
         (LABEL (OR (REUSE-LABEL NODE-NUMBER)
                    (TP H-OF-NODE))))
    (COND ((NULL LABEL)
           ; conflict set was not found.
           (SETF (HS-NODE-LABEL NODE) 'CHECKED)
           NIL)
          (T
           ; confict set was found.
           (SETF (HS-NODE-LABEL NODE) LABEL)
           (UNLESS (MEMBER LABEL *SET-OF-CONFLICT-SETS* :TEST #'EQUAL)
             ; conflict set was not reused
             (DO-PRUNING LABEL)
             ; update *SET-OF-CONFLICT-SETS*
             (SETF *SET-OF-CONFLICT-SETS* (CONS LABEL *SET-OF-CONFLICT-SETS*)))
           (UNLESS (CUT-NODE-P NODE-NUMBER)
             (LET ((NEW-CHILDREN-LIST NIL))
               (DOLIST (COMPO LABEL)
                 (LET* ((H-OF-CHILD (CONS COMPO H-OF-NODE))
                        (REUSED-NODE-NUMBER (REUSE-NODE H-OF-CHILD)))
                   (COND (REUSED-NODE-NUMBER ; node was reused
                          (PUSH (LIST NODE-NUMBER COMPO)
                                (HS-NODE-PARENTS (GET-NODE REUSED-NODE-NUMBER)))
                          (PUSH (LIST REUSED-NODE-NUMBER COMPO)
                                (HS-NODE-CHILDREN NODE)))
                         (T  ; new node must be made
                          (LET ((NEW-CHILD
                                 (MAKE-HS-NODE :NUMBER *HS-NODE-COUNT*
                                               :PARENTS (LIST (LIST NODE-NUMBER
                                                                    COMPO))
                                               :LEVEL (1+ LEVEL-OF-NODE)
                                               :H H-OF-CHILD
                                               :HP (GET-HP-FROM-H H-OF-CHILD))))
                            ; link the new child to the parent
                            (PUSH (LIST *HS-NODE-COUNT* COMPO)
                                  (HS-NODE-CHILDREN NODE))
                            ; entry the new node to the registry
                            (SETF (GETHASH *HS-NODE-COUNT* *HS-NODE-TABLE*)
                                  NEW-CHILD)
                            (PUSH *HS-NODE-COUNT* NEW-CHILDREN-LIST)
                            ; update node's counter
                            (INCF *HS-NODE-COUNT*))))))
                 (REVERSE NEW-CHILDREN-LIST)))))))


;;  Dummy TP
;;
(DEFUN TP (LST)
  (FORMAT T "~%Call TP ~S " LST)
  (READ))

(DEFUN GET-HP-FROM-H (H)
```

```
  (REMOVE-DUPLICATES (MAPCAR #'PI H)))

(DEFUN PI (COMPONENTS)
  (LIST COMPONENTS))
;;;
;;; Component Level Diagnose Ver.0.4
;;;

(DEFUN COMPONENT-DIAGNOSE ()
  (LOOP
    (IF (NULL *HS-NODE-QUEUE*)
      (RETURN-FROM COMPONENT-DIAGNOSE *DIAGNOSIS*)
      (LET ((NODE-TO-BE-EXPANDED (POP *HS-NODE-QUEUE*)))
        (UNLESS (OR (CUT-NODE-P NODE-TO-BE-EXPANDED)
                    (CLOSE-NODE-P NODE-TO-BE-EXPANDED)
                    (LEVEL-STOP-P NODE-TO-BE-EXPANDED))
          (LET ((NEW-CHILDREN (NODE-EXPAND NODE-TO-BE-EXPANDED)))
            (IF (EQ (HS-NODE-LABEL (GET-NODE NODE-TO-BE-EXPANDED)) 'CHECKED)
              ; diagnosis was foud
              (LET ((NEW-DIAGNOSIS (HS-NODE-H (GET-NODE NODE-TO-BE-EXPANDED))))
                ; store the new diagnosis
                (SETF *DIAGNOSIS* (APPEND *DIAGNOSIS* (LIST NEW-DIAGNOSIS)))
                (FORMAT T "~%Find new diagnosis ~A~%" NEW-DIAGNOSIS))
              ; expand HS-DAG (conflict set was foud)
              (SETF *HS-NODE-QUEUE* (APPEND *HS-NODE-QUEUE* NEW-CHILDREN)))))))))


;;; TOP
;;; 90/01/24

(DEFUN TOP ()
  (INIT-HS-NODE-TABLE)
  (INIT-HS-NODE-COUNT)
  (INIT-HS-NODE-QUEUE)
  (INIT-SET-OF-CONFLICT-SETS)
  (INIT-DIAGNOSIS)
  (SET-DIAGNOSIS-LEVEL 5)
  (PUSH *HS-NODE-COUNT* *HS-NODE-QUEUE*)
  (SETF (GETHASH *HS-NODE-COUNT* *HS-NODE-TABLE*)
        (MAKE-HS-NODE :NUMBER *HS-NODE-COUNT*
                      :LEVEL 0))
  (INCF *HS-NODE-COUNT*)
  (COMPONENT-DIAGNOSE))
```

inches 1 2 3 4 5 6 7 8

cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Kodak Color Control Patches

Blue Cyan Green Yellow Red Magenta White 3/Color Black

Kodak Gray Scale

C Y M

A 1 2 3 4 5 6 **M** 8 9 10 11 12 13 14 15 **B** 17 18 19