アルゴリズム特論（第7回）

北海道大学　大学院　情報科学研究科
アルゴリズム研究室　　湊　真一

前回の内容

- **BDDの変数順序づけ**
  - 順序づけの影響の例
  - 2つの経験則
    - 論理式・論理回路から生成する場合の重み付け法
  - 与えられたBDDの最小化
    - 厳密解法：全解列挙、動的計画法（効果と計算時間）
    - 近似最適化法：隣接交換法、最小幅法（効果と計算時間）
  - 動的順序付け法
    - BDD処理系への組み込み
    - 2変数の入れ替えの局所性
    - 節テーブルハッシュの横割り方式
    - 効果と計算時間

今回の内容

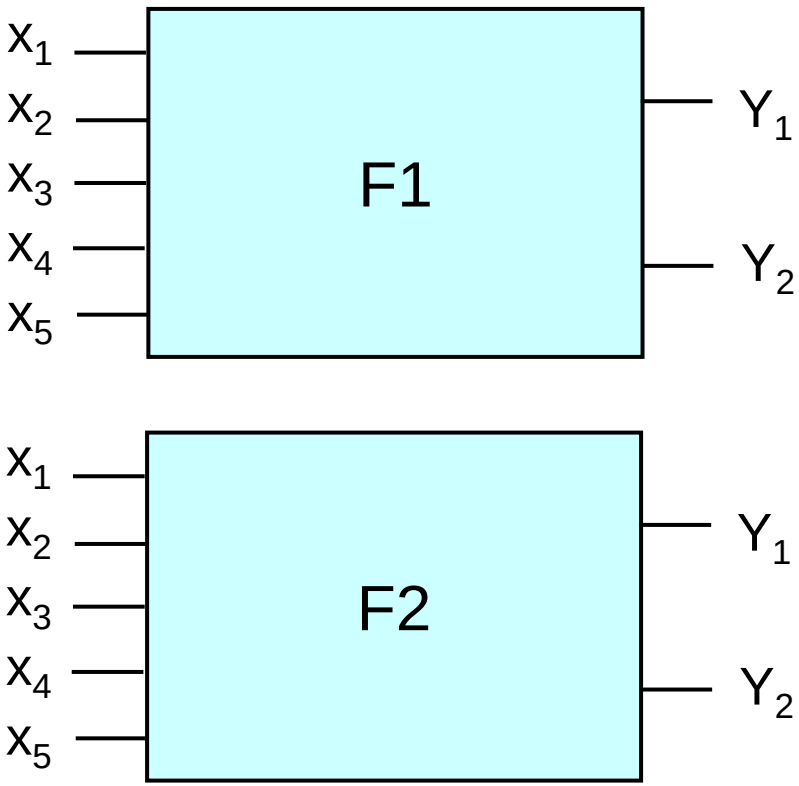
- **BDDの応用**
  - LSI設計技術
    - 論理検証
    - 論理合成・最適化
    - テストパタン生成
  - 組合せ問題・最適化問題
    - ナップサック問題
    - 8クイーン問題、ナイト巡回問題
    - 巡回セールスマン問題
    - グラフ理論の種々の問題
    - 専用の解法との比較
  - 知識データベース、推論エンジン
    - 真偽維持システム
    - 故障木解析
  - 組合せ集合の表現とゼロサプレスBDD

LSI設計技術へのBDDの応用

- BDDは元々は計算機ハードウエア（論理回路）設計のために研究・開発された
  - 論理シミュレーション・論理検証
  - 論理最適化・論理合成
  - テストパタン生成
- LSIの論理設計は非常に大規模な論理関数データを扱う困難な問題
  - 人手では手に負えない規模
  - 誤りが許されない（Pentiumのバグの例）
  - 設計期間の短縮の要求（企業間の開発競争）
  - 巨大なマーケット

組合せ回路の論理検証

- 2つの組合せ回路（ループや記憶素子を含まない回路）が等価（同じ論理を実現しているか）を判定



すべての入力の組合せに対して、同じ出力となるかをチェックしたい（単純に検査すると2^n通り）

対応する出力のBDDを生成することができれば、根の節点の番地を比較するだけで即座に判定できる

BDD生成の実験結果

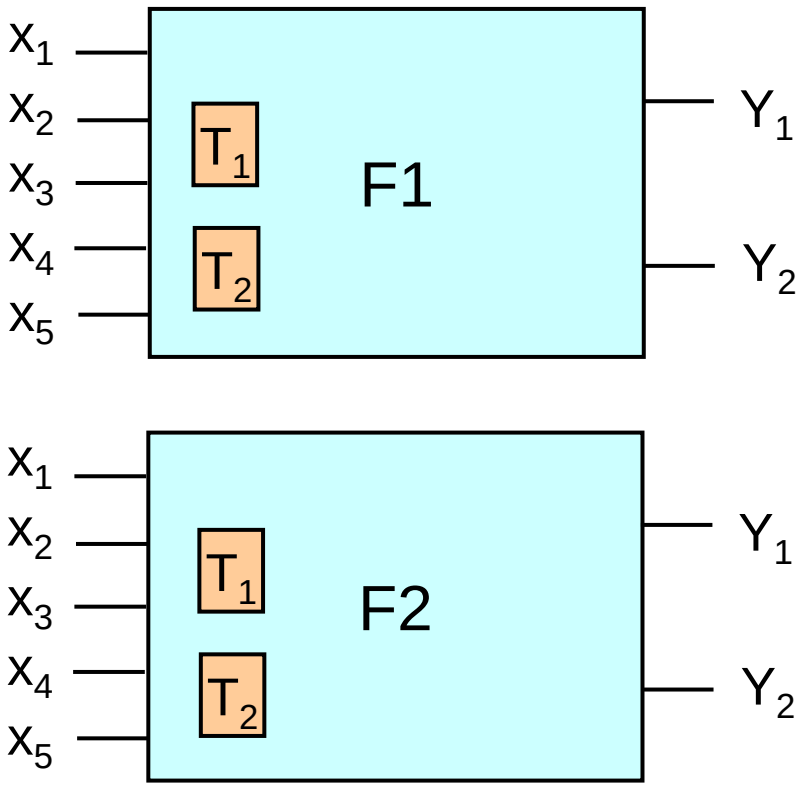
- 実用的な論理回路を集めたベンチマークで実験
  - 数百ゲートの回路でも、
    性質がよければ数分でBDDを生成可能

	回路規模				
回路名	入力数	出力数	内部信号線数	BDD節点数	時間(秒)
sel8	12	2	29	78	0.3
enc8	9	4	31	56	0.3
adder8	18	9	65	119	0.4
adder16	33	17	129	239	0.7
mult4	8	8	97	524	0.5
mult8	16	16	418	66161	24.8
c432	36	7	203	131299	55.5
c499	41	32	275	69217	22.9
c880	60	26	464	54019	17.5
c1355	41	32	619	212196	89.9
c1908	33	25	938	72537	33.0
c5315	178	123	2608	60346	31.3

(計算時間はSun3, 24MByte)

順序回路の論理検証

- レジスタ(記憶素子)を含む順序回路の等価性判定問題

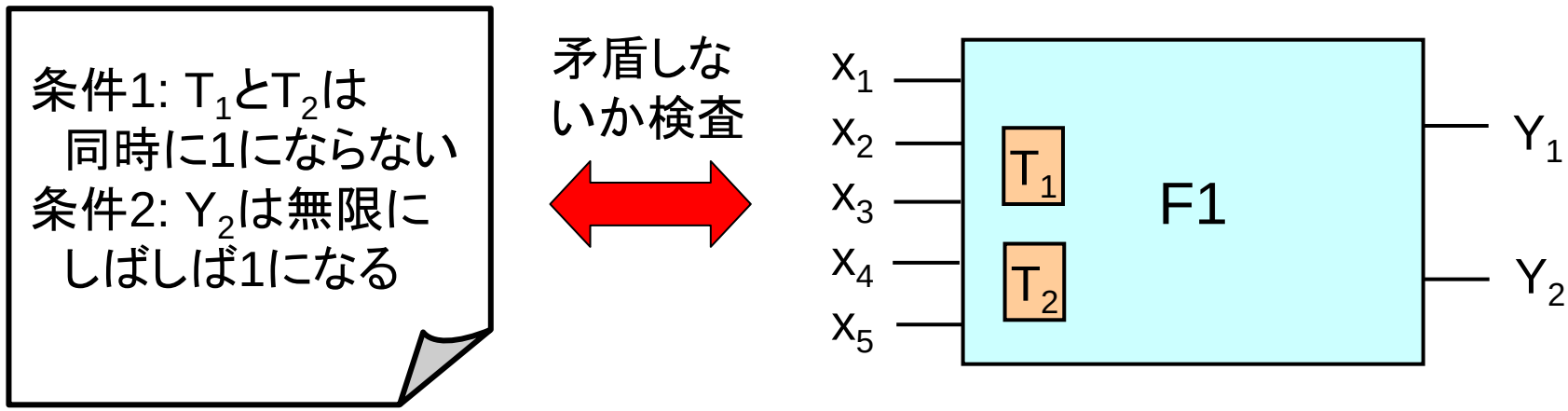


すべての入力の組合せの系列に対して、同じ出力系列となるかをチェックする問題(組合せ回路の場合よりはるかに難しい)

レジスタの値によって回路の「状態」が決まる。2つの回路が取りうる状態の対応関係が分かれば、対応する各状態において、組合せ回路と同様に等価性判定すればよい

順序回路のモデル検査

- 設計した順序回路が、論理式で与えられた条件を満たすかどうかを検査する手法



- T1, T2が取りうる状態の集合をBDDで表現
- 取りうる状態の中で、常に条件を満たすかどうかをBDDで検査

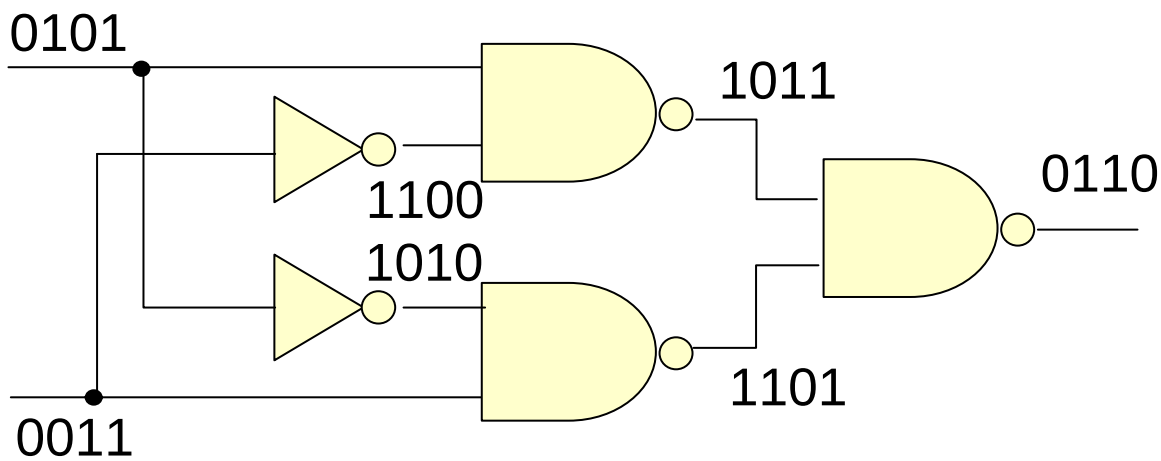
通信プロトコルのような上位の記述の検証にも使える

論理合成・最適化へのBDDの応用

- 与えられた仕様を満たす品質の良い（素子数や段数が小さい）回路を自動生成すること
- ある適当な初期回路を生成し、これを最適化することにより得られる。
- 論理回路の部分的変形を繰り返して品質を改善する方法（トランスダクション法）
  - 部分的変形を行うときに回路全体の論理に影響を与えないことが条件となる
  - 条件判定にBDDを用いることで、それ以前に比べて数十～数百倍も大規模な論理回路を最適化できるようになった。
- 適当な初期回路として積和形論理式を作り、これを因数分解してコンパクトな回路を生成する方法もある
  - BDDを用いて、積和形論理式の生成と因数分解を高速化する技法も開発されている

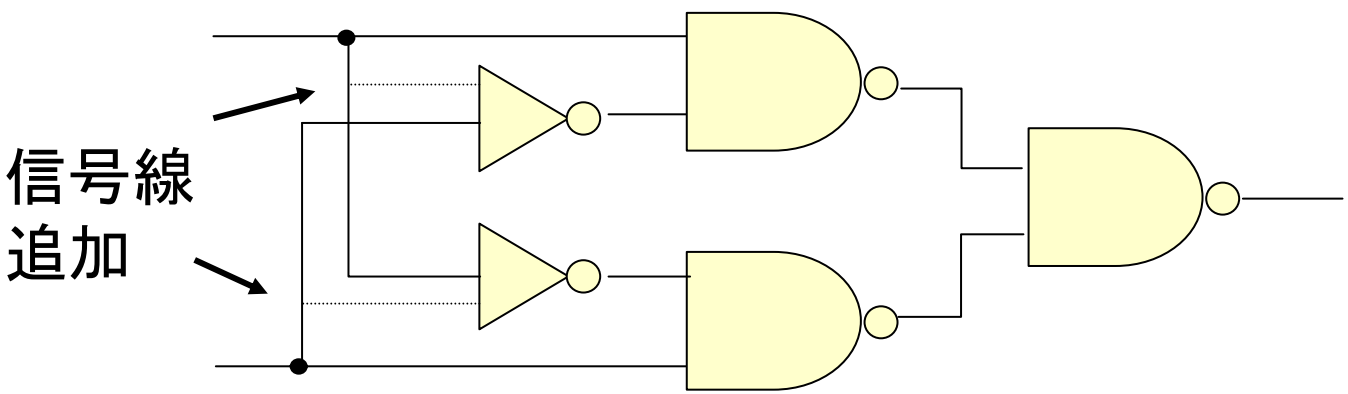
トランスダクション法

- 配線の追加・削除を繰り返し、ゲート数や配線数を削減させて行く。
- 出力に変化を与えずに配線の追加・削除ができるための条件を、途中の配線ごとに計算。



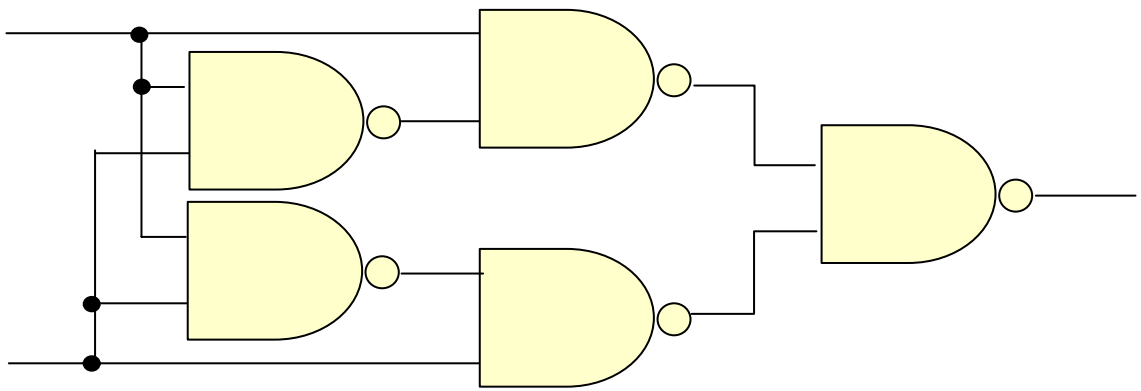
トランスダクション法

- 配線の追加・削除を繰り返し、ゲート数や配線数を削減させて行く。
- 出力に変化を与えずに配線の追加・削除ができるための条件を、途中の配線ごとに計算。



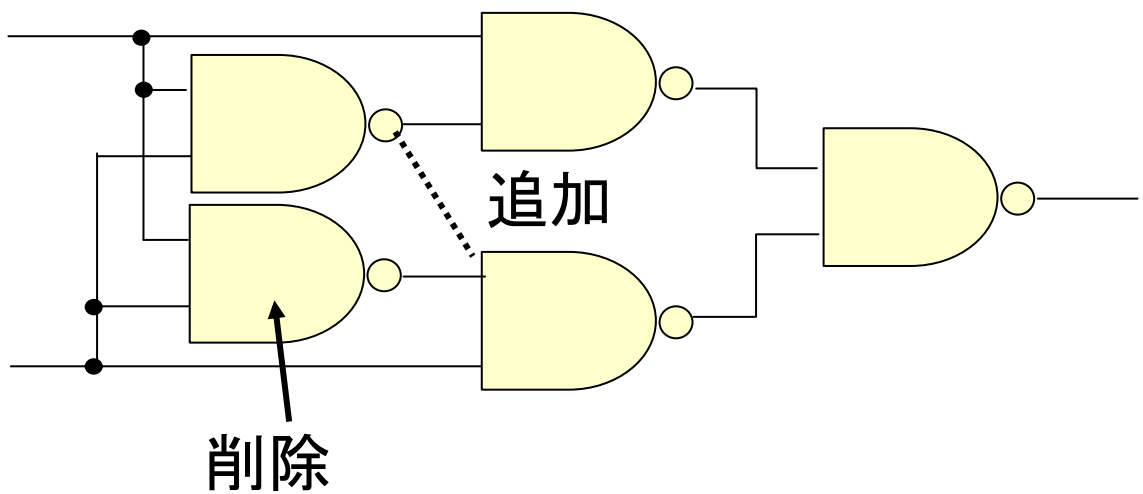
トランスダクション法

- 配線の追加・削除を繰り返し、ゲート数や配線数を削減させて行く。
- 出力に変化を与えずに配線の追加・削除ができるための条件を、途中の配線ごとに計算。



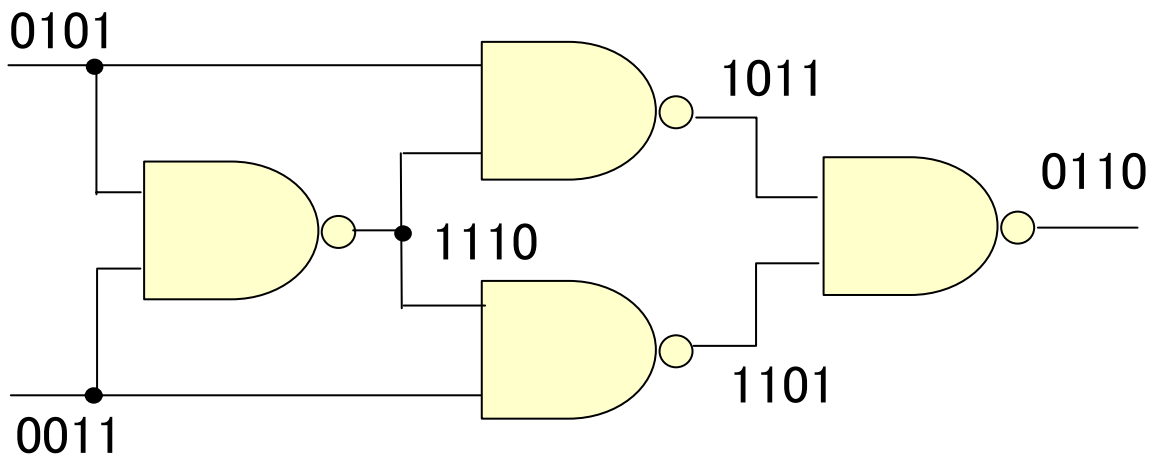
トランスダクション法

- 配線の追加・削除を繰り返し、ゲート数や配線数を削減させて行く。
- 出力に変化を与えずに配線の追加・削除ができるための条件を、途中の配線ごとに計算。



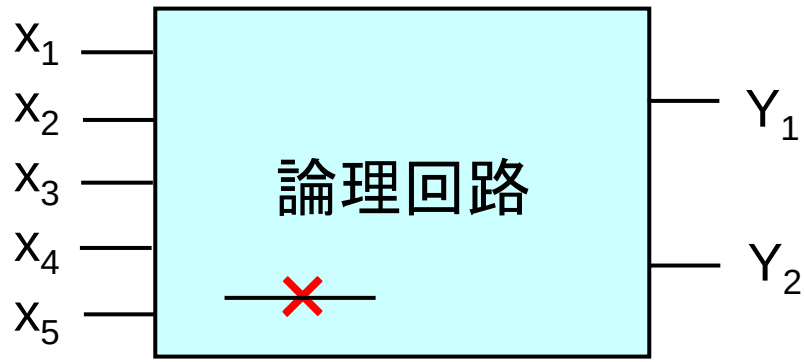
トランスダクション法

- 配線の追加・削除を繰り返し、ゲート数や配線数を削減させて行く。
- 出力に変化を与えずに配線の追加・削除ができるための条件を、途中の配線ごとに計算。



テストパタン生成

- LSI製造時に小さなチリが混入すると、その部分に故障が発生する
  - 何%かの確率で発生
  - 出荷時に不良品を検査して取り除く必要がある



- 出荷時にはLSI内部を直接見ることはできない
  - ある入力パタンを与えたときに、期待値と異なる出力が観測されたときに初めて不良品と確認できる
  - 故障を発見するための入力パタン(＝テストパタン)を、すべての故障箇所について求めておく必要がある
  - 出荷時の手間を減らすため、テストパタンはなるべく少なくしたい。(少ないパタンでなるべく多くの故障をみつけたい)
- テストパタン生成には大規模な論理演算処理が必要
  - BDDを用いた高速化が有効

組合せ問題・最適化問題へのBDDの応用

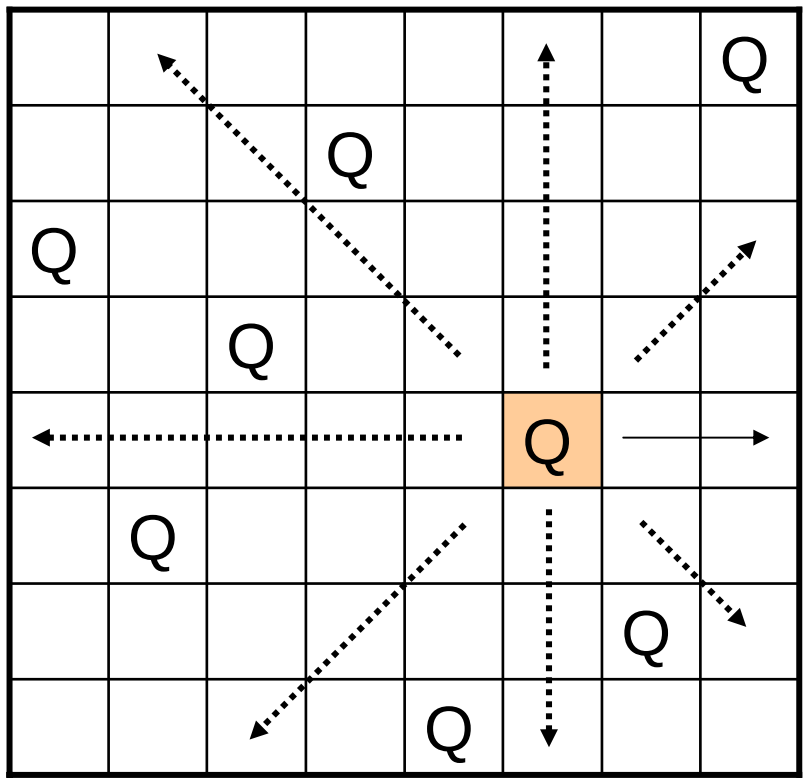
- グラフ理論で扱われるいくつかの基本的な問題にBDDを応用する試みも行われている
  - （例） 最大クリーク問題、最大独立集合問題、等
- 基本的に、与えられたグラフの枝（または節点）の集合の中から、ある条件を満たす部分集合を見つけ出す問題
- グラフの枝（または節点）に論理変数を割り当て、変数の組合せの集合をBDDで表現
- 満たすべき条件を論理式で表し、そのBDDを生成できれば、コスト最小となる変数組合せを即座に求められる。
  - ただし、大規模な例題に対してはBDDのサイズが爆発的に増大して主記憶に入りきらない場合がある。常に簡単に解けるわけではない。

組合せ最適化問題の例（ナップサック問題）

- 商品がn個ある。各商品はそれぞれ値段と重さが決まっている。手持ちのナップサックの許容重量の範囲で、合計金額が最大となるように商品を組合せて買いたい。
  - 買うかどうかを変数x_iの1/0で表現する。
  - 各商品の重さをa_i, 値段をc_i, ナップサックの許容重量をbとすると、$\Sigma\, a_i\, x_i \leqq b$ の条件下で、$\Sigma\, c_i\, x_i$ が最大になるようなx_iの0,1の組合せを求めればよい。
- 制約条件 $\Sigma\, a_i\, x_i \leqq b$は、$x_1 \ldots x_n$を入力とする論理関数となる。
  - これをBDDで表すと、根から1定数節点に至る経路が、ナップサックの許容範囲の商品の組合せを示している。
  - 全ての経路の中で、コストの合計が最大になるような経路を選べばよい。これはBDDの節点数に比例する時間で求められる。

組合せ問題の例（8クイーン問題）

- ８×８のチェス盤上にクイーン（縦横斜めに利く）を8個、お互いに取り合わないように配置する問題
- 論理変数を64個用意する
  - 各論理変数は、各マス目のクイーンの有無を0/1で表現する
- クイーンの条件を論理式で記述する
  - 各列にクイーンは1つだけ
  - 各行にクイーンは1つだけ
  - 各斜め筋にクイーンは0または1個
- 全ての条件の論理積(AND)を表すBDDを生成すれば解の個数が直ちにわかる



N-Queens問題の実験結果

- N×Nのマス目にN個のクイーンを配置する問題:
  実際にBDDを作り、解の総数をカウントした結果
  - 実用的にはN=13くらいが限界
  - N-Queens専用のプログラムではN=100でも解を求められる物がある(ただしBDDのように全ての解を列挙することはできない)

N	論理変数	BDD節点数	解の個数	計算時間(秒)
8	64	2,450	92	6.1
9	81	9,556	352	18.3
10	100	25,994	724	68.8
11	121	94,821	2,680	1081.9

(SPARC station2, 40MByte)

その他の組合せ問題

- ナイト巡回問題:
  - チェス盤上をナイト（8方向の桂馬）が、各マス目1回ずつ通って元の位置に戻ってくる一種の一筆書きパズル
  - BDDで解いたという論文[Lobbing他1996]がある。
- 巡回セールスマン問題
  - N個の都市をなるべく短い経路で1回ずつ巡回する問題。
  - BDDで解く方法の一例：
    2つの都市を結ぶ枝はN(N-1)/2個存在。各枝に論理変数を割り当てる。通る通らないを1,0で表現。一巡して元に戻る経路（ハミルトン路）の全てを表すBDDを作り、その中でコスト（距離）最小の組合せを選ぶ。（下記は実行例）

N	論理変数	BDD節点数	解の個数	計算時間（秒）
8	28	2,054	2,520	8.7
9	36	20,160	66,136	28.8
10	45	19,972	181,440	216.5

(SPARC station2, 40MByte)

グラフ理論の種々の問題

- その他にBDDで扱えるグラフ理論の問題の例:
  - 最大クリーク問題(Maximal cliques)
  - 最大kカバー問題(Maximum k-cover)
  - 最小α被覆問題(Minimum α-covering)
  - 最大独立集合(Maximum independent set)
  - 最小節点カバー(Minimum vertex cover)
  - 最小彩色問題(Minimum coloring)
  - 最小クリーク分割(Minimum clique aprtition)
  - 最小クリークカバー(Minimum clique cover)
  - 最小支配集合(Minimum dominant set)
- ただし、BDDを用いない方が効率が良い場合もある。問題の性質に依存するので注意が必要。

専用の解法との比較

- BDDを用いた解法では、制約条件を論理式で記述できれば、後は機械的に解くことができる。
  - 与えられた問題に対して、解法を素早くプログラミングできる
  - 多数の解をBDDで共有して全部列挙できる
- 一方、有名な問題に対しては、専用の効率的なアルゴリズムが開発されていることが多い。
  - 問題固有の性質を利用しているので、BDDを用いた場合よりも高速に解けることが多い
  - 特に、解を1つだけ見つける問題に対しては専用の解法が強いことが多い。例えば、巡回セールスマン問題では、ある条件のもとで、1000都市でも最適解を見つける方法が知られている。BDDで全解を列挙する方法では15都市程度が限界。

知識データベース、推論エンジンへのBDDの応用

- 人工知能の分野でもBDD処理系を利用した例がある
- 命題論理の推論や証明は、基本的には論理関数・論理式の演算処理として定式化できる。→BDDを利用可能
  - BDD処理系を用いる場合、
    - 変数の展開順序をすべての命題で合わせなければならない
    - データが主記憶に収まらなければならない

    という制限がある。
  - 問題の規模や性質が、BDD処理の特長とうまくマッチすると大きな効果を発揮する場合がある。

真偽維持システムへの応用

- 真偽維持システム(TMS: Truth Maintenance System)
  - 命題の真偽を管理するデータベース
  - 仮説推論、非単調推論、定性推論などの基礎となる技術
  - 命題同士の矛盾関係や従属関係を証明したり、極小支持集合（ある命題が真となるために最小限必要な仮定）を求める処理がしばしば必要となる
  - 処理手順を論理演算で定式化することにより、BDD処理系を利用できる。論理データが主記憶に収まる範囲であれば、非常に高速に動作する。
  - 高炉制御システムの一部として実際に応用された例がある。[Madre他1991]

故障木解析への応用

- 故障木解析(FTA: Fault Tree Analysis)
  - 発生して欲しくない事象（故障）について、それがどのような原因の組合せで起こるかを、木構造のグラフで場合分けして、安全性の評価・解析を行う手法
  - 原子力発電所や化学プラント等の安全性評価に利用される
  - 故障因子を論理変数とすると、故障を起こす因子の組合せ（の集合）は論理関数となる。
  - これをBDDで表すことにより、従来は近似的にしか求められなかった故障発生確率を、初めて厳密に計算することができたという報告がある[Coudert他1992]

論理関数と組合せ集合

- 組合せ集合とは:
  n個のアイテムの中から自由に選ぶ組合せ方（2^n通り）
  を要素とする集合（2^{2^n}通り）
  - a,b,c,d,e という5つのアイテムに関する組合せ集合の例:
    { a b, e }　　{a b c, c d e, b d, a c d e, e}
    {ϕ, c d }　　{ }
  - 多くの組合せ問題や制約充足問題を処理する際に必要となる
    基礎的なデータ表現の1つ

論理関数と組合せ集合の対応

a	b	c	F	
0	0	0	0	→ c
0	0	1	1	
0	1	0	0	→ bc
0	1	1	1	
1	0	0	0	
1	0	1	0	
1	1	0	1	→ ab
1	1	1	1	→ abc

論理関数とみた場合

F = a b + ~a c

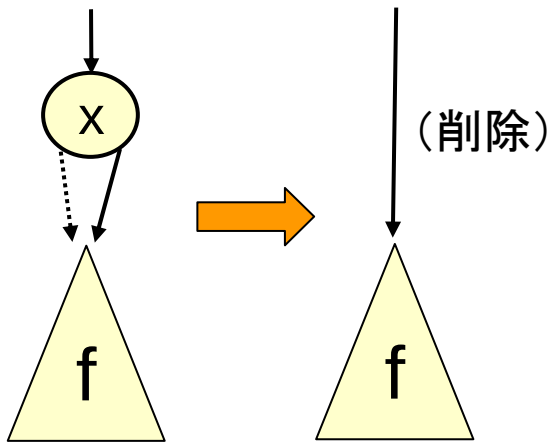
組合せ集合とみた場合

F = {c, bc, ab, abc}

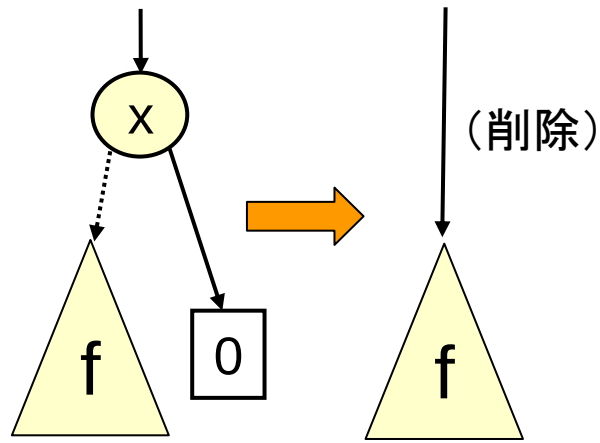
1対1の対応関係がある
(ただし意味づけが異なる.)

ゼロサプレス型BDD(Zero-suppressed BDD)

- 組合せ集合を効率的に表現するためのBDD[Minato93]
- 通常のBDDとは異なる節点省略規則を使用する。
  - 「1枝」が直接「0定数節点」を指しているときに削除
  - 従来は削除していた冗長節点は削除せずに残す。
  - 等価な節点の共有は従来同様に行う。



通常のBDDの節点省略規則

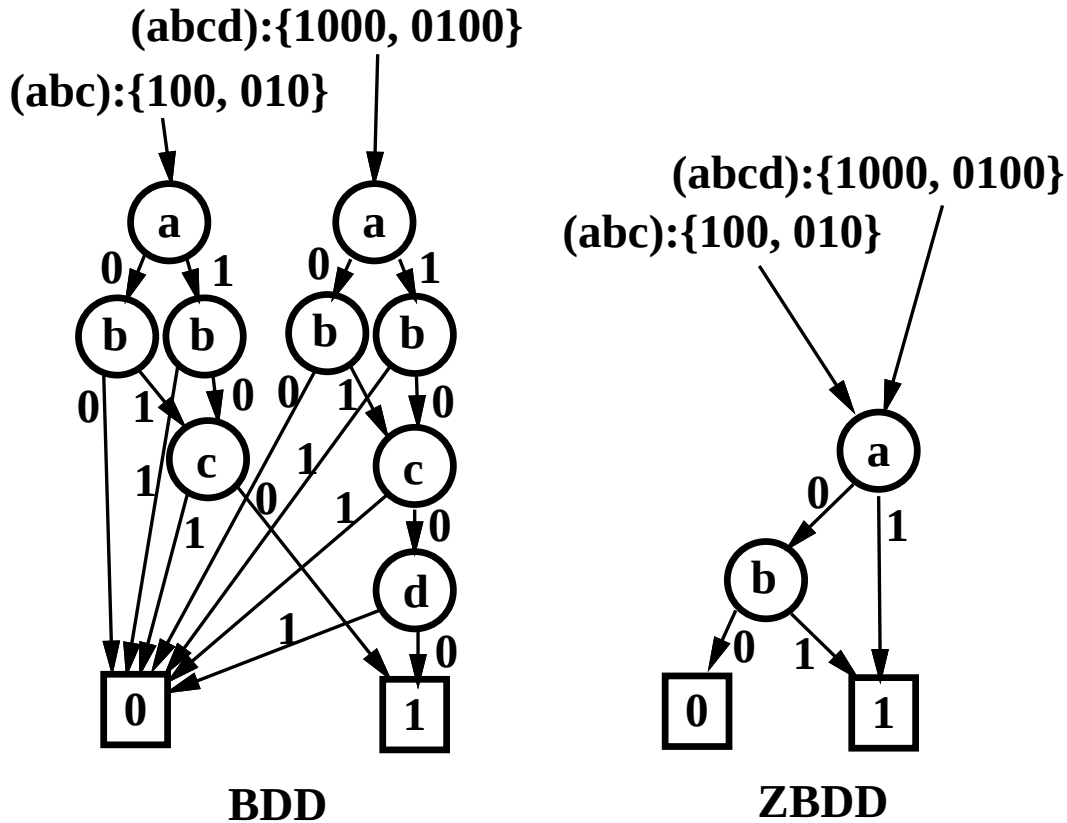


ZBDDの節点省略規則

ZBDDの効果

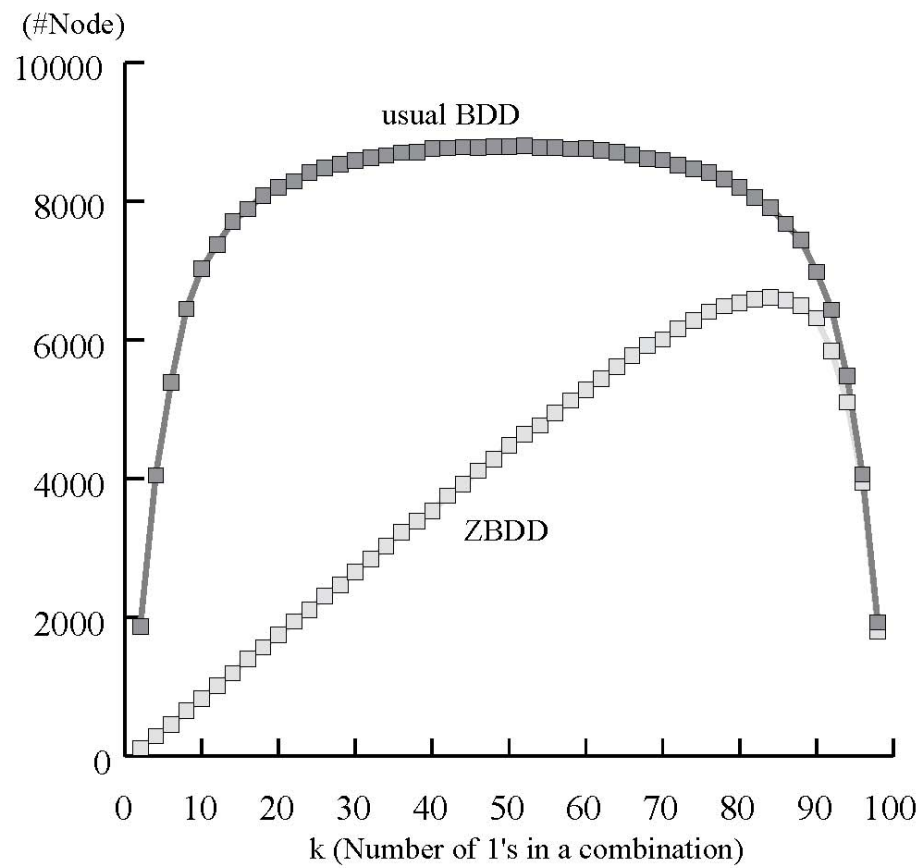
- 通常の論理関数では、式に現れない変数は0でも1でも結果に影響を与えないと仮定
  - BDDの簡約化規則が有効
- 組合せ集合では、式に現れない変数は、集合に含まれない（常に0）と仮定
  - ゼロサプレス型の簡約化規則が有効

（例） S(a b c) = { a, b }
S(a b c d) = { a, b }
という組合せ集合を表す場合



ZBDDの特長

- 組合せ集合に出現しないアイテムに関する節点が自動的に取り除かれる。
  - 通常のBDDでは無駄に残ってしまい、BDDの共有化による効果を帳消しにしてしまう場合があった。
- 疎な組合せの集合に対して顕著な削減効果がある
  - 例えば1000個のアイテムから平均10個しか選ばないような組合せ集合では、通常のBDDに比べて約100倍の節点削減効果



ZBDDの応用

- 大規模な組合せアイテム集合データを扱う
  データベース解析（データマイニング・知識発見）
  - ZBDD上の集合演算を組合せることにより、
    データ内に頻出するパタンを高速に抽出するアルゴリズム
- 人工知能システムにおける知識インデックス表現
  - ロボットが自分で推論しながら次の動作を決めるような
    複雑な演算処理を、ZBDDで効率よく実現できる可能性がある。
  - 機械学習システムにおいて、学習状況をZBDDを用いて
    メモリ上に保存して、学習を高速化する。
- その他、グラフ理論の様々な問題に応用可能
  - 特に疎なグラフを表現する場合にZBDDは効果的

BDDの応用のまとめ

- LSI設計分野を中心とした多くの市販ツールの内部でBDD処理系が実際に利用されている。
- BDDは「効率的しらみつぶし法」
- 論理式で定式化できればあとは機械的に解ける。
- 問題の規模が主記憶に収まる範囲で高速に動作する
  - 規模が大きいと記憶あふれを起こして失敗する
- 問題特有の性質を究めて開発された専用アルゴリズムより低速であることが多い
  - ただし解法プログラムの開発期間が短くてすむこともある
- 多数存在する解の1つをみつけるような問題よりも、全ての解を数え上げたり、反例が1つもないことを証明する問題に適している。
- 組合せ集合に適した「ゼロサプレス型BDD」もある。データマイニングや人工知能の分野にも応用が進んでいる。