間接含意の効率的な計算方法

九州大学
松永　裕介

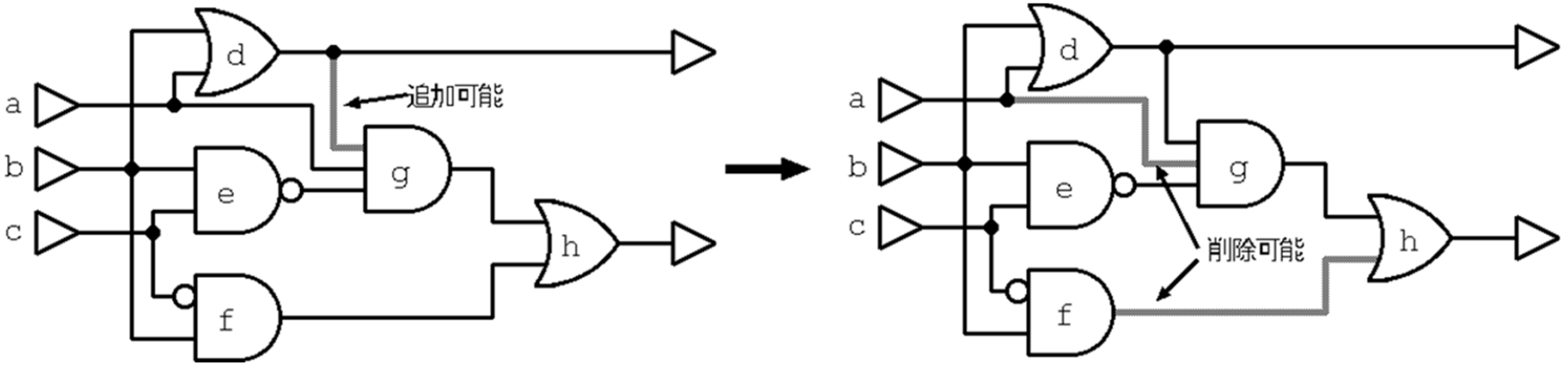
間接含意

- 直接含意から求めることのできない含意
- 単純に求めることはできないが、間接含意がわかってしまえば冗長故障の検出が容易になることが多い。

- ATPG的には間接含意を求める手間と間接含意による高速化のトレードオフが難しい。

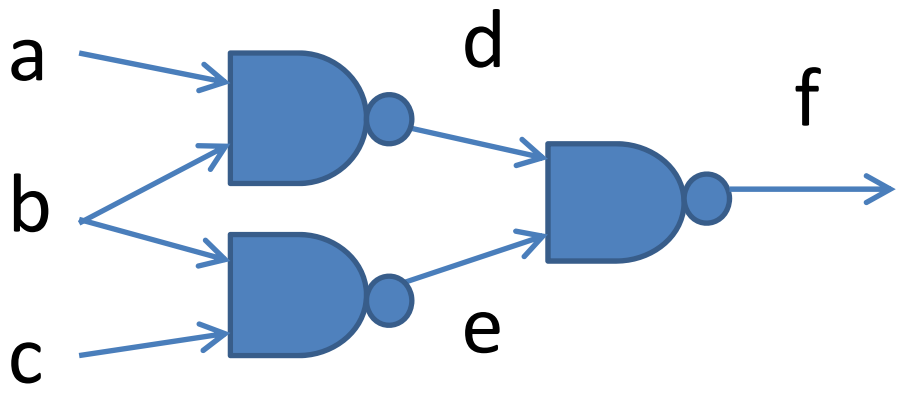
間接含意の応用

- local rewiring
  - 昔は redundancy addition and removal と呼ばれた。
  - 追加可能な信号線も削除可能な信号線も冗長故障。
  - ただし、全部を調べると手間がかかるので間接含意からわかるものだけを使う。



対偶による間接含意

- a=1 ⇒ b=1 の対偶は b=0 ⇒ a=0
- この対偶の関係の含意が直接含意でなければこれだけで間接含意（の一部）を求めることができる。



b = 0 ⇒ d = 1 & e = 1
⇒ f = 0
f = 1 ⇒ b = 1

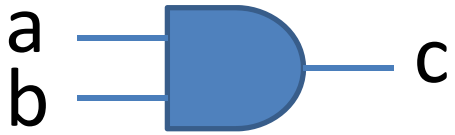
Recursive Learning

- 直接含意で値が決まらないゲート（unjustified gate）に対して考えられるすべての値の組合せを試し、それぞれの含意結果の共通部分を求める。
- 上記の処理を再帰することで複雑な間接含意も理論的にはもとめることができる。
- [脇道] Kunz のRecursive Learning の論文の大きな貢献は justified/unjustified gate の数学的に完全な定義を与えたこと

完全な含意の列挙

- 単純な手法
  - 乱数シミュレーションで含意関係の候補を求める。
  - 各信号線間の関係を和積形論理式（CNF: Conjunctive Normal Form）で表す。
  - 候補の一つ一つを取出し、SATを用いて反例が存在しないことを確かめる。たとえば a=1 ⇒ b=1 を証明するには上で作った CNF式に a ∧ b’ を加えた式が充足可能かどうかを調べる。充足不能なら a ∧ b’ の否定 a’ ∨ b が成り立つということ。

CNF式の生成



- a ∨ c’
- b ∨ c’
- a’ ∨ b’ ∨ c

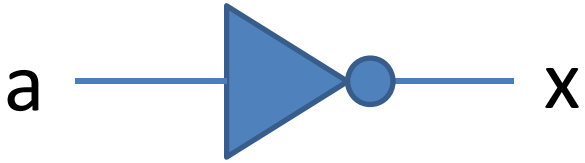
- ２入力ANDの場合２つの2リテラル節と1つの3リテラル節で表せる
- 回路サイズに比例したサイズのCNF式が生成される。

原因リストを用いた手法

- 信号線 a を “1”にする原因となる信号線割り当ての集合（リスト）を原因リストと呼び、Laと表すことにする。
- 同様に信号線aを”0”にする原因リストはLa’とする。
- おおまかには、各信号線の０・１の値割り当ての原因リストが完成したら（原因⇒結果）の含意関係がすべて得られたことになる。

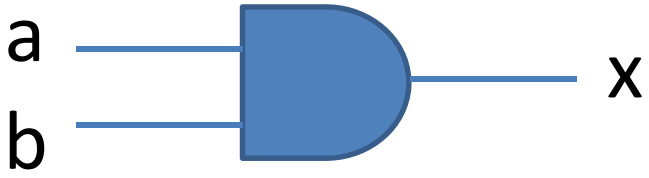
原因リストの計算

- INVERTER



Lx = La’
Lx’ = La

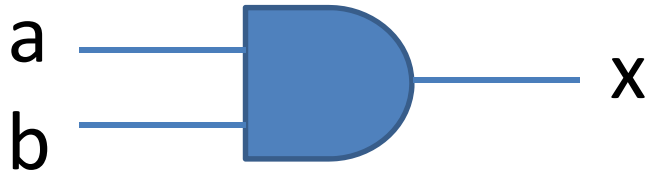
- AND



Lx = La & Lb
Lx’ = La’ + Lb’

原因リストの計算（続き）

- AND



La = Lx
Lb = Lx
La’ = Lx’ & Lb
Lb’ = Lx’ & La

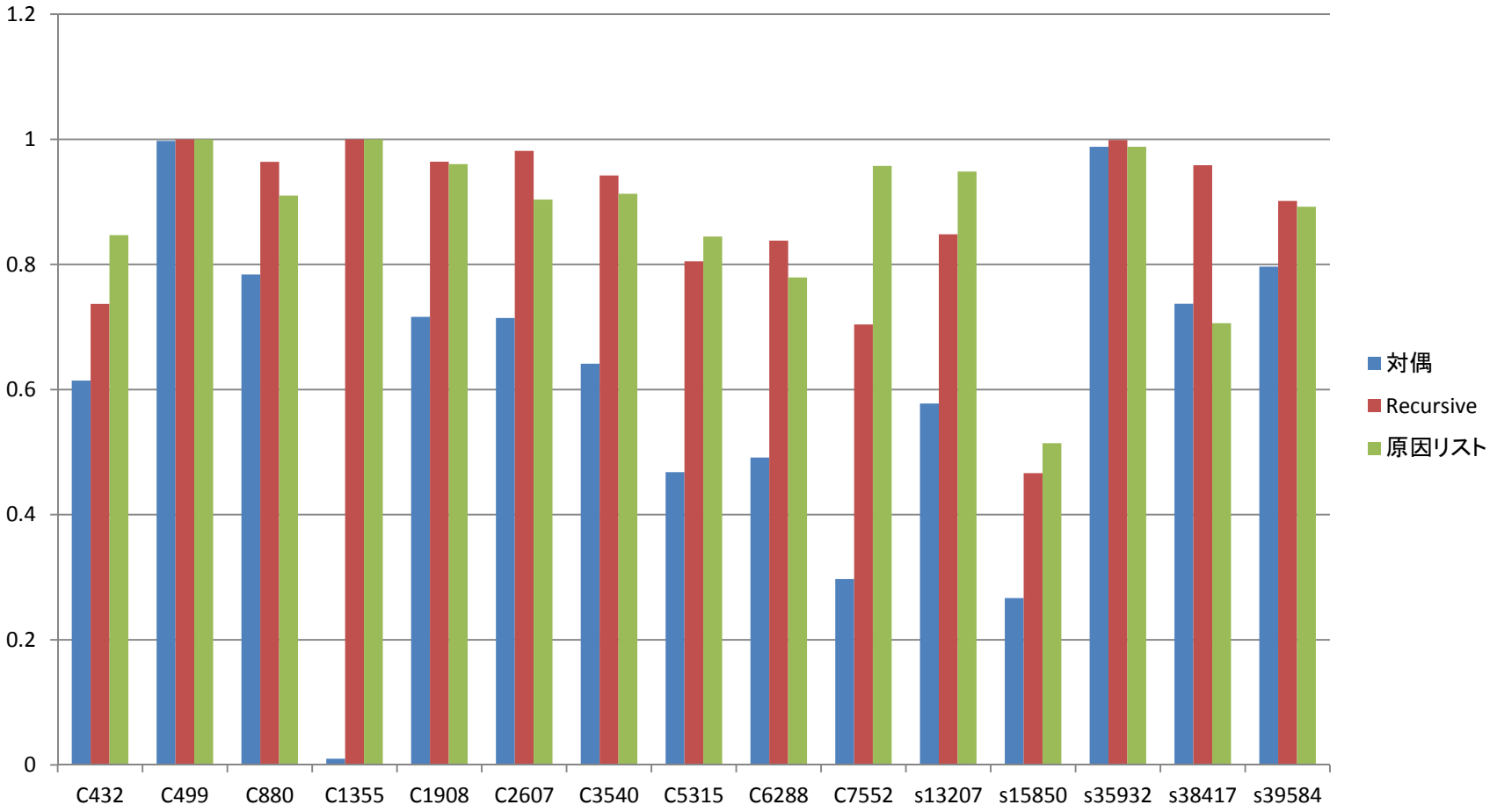
- &演算は手抜きで集合としてのインターセクションで代用する⇒完全な含意は求められない。ここが将来的な工夫のポイント

実験

- 対偶や Recursive Learning、原因リスト でどれくらいの間接含意を求めることができるのか？
- 間接含意の全列挙はどれくらいの時間でできるのか？

- 実験環境
  - 回路はすべて２入力AND＋NOT（AIG）に変換
  - 計算機 Intel Core I7 たぶん 3GHz くらい

実験結果(1)
間接含意数


1.2
1
0.8
0.6
0.4
0.2
0
C432
C499
C880
C1355
C1908
C2607
C3540
C5315
C6288
C7552
s13207
s15850
s35932
s38417
s39584
対偶
Recursive
原因リスト

計算時間

対偶	RL	原因	SAT
0.72	236.94	3.00	254.49

- 個々の回路によって計算時間は大きくばらつくが原因リストは常に recursive learning よりもはるかに速い。
- 場合によってはSATのほうが recursive learning よりも速いことがある。
- SATの計算時間の大半はランダムシミュレーションで落とせなかった false positive (含意だと思ったけど含意でなかったもの)の検証に費やされている。

Resolution

- (a ∨ b1 ∨ b2 ∨ …)(a’ ∨ c1 ∨ c2 ∨ …)の形の節があるとき、(b1 ∨ b2 ∨ … ∨ c1 ∨ c2 ∨ …)という節を追加することができる。
- 定理 : 充足不能なCNF式は resolution によって空の節が生成される。
- ただし、resolution を繰り返すと節の数は指数爆発する。

Resolution と間接含意

- recursive learning で一つの信号線の値の0と１の含意の結果の共通部分を求める処理と resolution はほぼ同じ。
- なので recursive learning と同様の処理を CNF 式で行うことができる（のではないか？）
- 含意を表す節（2リテラル節）には含意元/含意先の区別はない。⇒初めから対偶の関係を考慮している。ａ ∨ ｂ は a’ ⇒ ｂ と　b’ ⇒ ａ の2つの含意を表す。

ZDDとCNF式

- CNF式 : 節の集合 { a ∨ b, c ∨ d }
- 節 : リテラルの集合 { a, b }
- なので ZDD で CNF 式を表すことが可能。
- 工夫が必要なのは a と a’ が互いに無関係ではないということ（a a’ = ϕ, a ∨ a’ = 1）

ZDDを用いた resolution

- CNF式を表す ZDD からリテラル a を含む部分グラフ f1 とリテラルa'を含む部分グラフf0を求める。
- f1 の各要素と f0 の各要素のユニオンを計算する。

現状

- ZDDを用いたCNF処理系の実装は完了
- ナイーブな実装では処理時間が従来手法に勝てない。
- 今後の方針:
  - アルゴリズムの見直し
  - 近似手法の検討
  - SATベースの全列挙法の高速化