FPGA を用いた充足可能性問題の解法における知的バックトラックの導入

岡村 昌彦† 長谷川 隆三‡ 藤田 博‡

†九州大学大学院システム情報科学府 知能システム学専攻
‡九州大学大学院システム情報科学研究院 知能システム学部門

1 はじめに

命題論理式の充足可能性問題 (SAT 問題) は様々な問題を定式化できる一般的な枠組みであり、この問題の効果的な解決法は熱心に模索されている。現在開発されている多くの SAT ソルバは Davis-Putnam 法 (DP 法) に基づいたソフトウェアとして実現されているが、近年ではハードウェアの特性を生かして SAT ソルバの高速化を図る研究も行われてきている [1][2]。

本研究は DP 法に基づいた木探索アルゴリズムを FPGA 上に実装して SAT ソルバーを構成することにより、高速に SAT 問題を解くことを目的している。そのためには探索空間をより刈り込む必要がある。

そこで本稿では、探索時に発生するバックトラックに着目し、FPGA 上に知的バックトラック機構を持つ SAT ソルバーを構成するための手法について提案する。

2 DP 法に基づく SAT 問題の解法

SAT 問題とは、節 (各論理変数のリテラルの和) の集合 $\{C_1, C_2, ..., C_m\}$ と論理変数の集合 $\{v_1, v_2, ..., v_m\}$ に対して、式 $C_1 \cdot C_2 \cdot ... \cdot C_m$ の値を 1 とするような変数の割り当てが存在するかを判定する問題である。式全体が 1 と評価されるためには各節 C_i がそれぞれ 1 と評価されなければならない。

DP 法では、値が未決定の変数を選び、その変数に 1 か 0 の値を割り当てることにより手続きが始まる。もし、いまだ 1 に充足されていない節の中に値が未割り当てのリテラルが 1 つしか存在しない状態が発生すれば、そのリテラルには値 1 が割り当てられる。この状態をインプリケーションと呼ぶ。また、インプリケーションにより割り当てられた値が以前に割り当てられた値と異なる場合を矛盾状態、または衝突と呼び、現在の変数割り当ては解とならないことを示す。

図 1 に DP 法を基にした解の探索アルゴリズムを示す。矛盾状態になることなく全ての変数に値を割り当てることができれば式は充足されることになり、全ての割り当てを行っても式が充足しない場合はその式が充足不能であることを示す。

```
Initialize all variables to "free" value
do {
   Compute implication and check contradiction;
   if (contradiction)
    if (active_variable->assigned_value == 1)
      active_variable->assigned_value = 0;
    else
      backtrack();
    endif
   else
    active_variable = next_free_variable();
    active_variable->assigned_value = 1;
   endif
 } while()
```

図 1: DP 法に基づく木探索アルゴリズム

3 アルゴリズムのハードウェア化

図 1 に示したアルゴリズムをハードウェアへ実装する際、回路は主に以下の 2 つの部分からなる。

1. 各リテラルのインプリケーションの計算用回路
2. 探索中のバックトラックを制御する状態遷移機械

インプリケーション計算用回路の役割は各リテラルにおいてインプリケーションにより割り当てられる値を決めること、及び、値の割り当てにより生じる矛盾状態を検出することである。

バックトラックを起こす探索を管理する制御用ユニットについては、実装を容易にするために分散型の制御システムを設計する。

図 2 に回路のモジュール構成を示す。式中の各変数について個別にインプリケーションの計算用回路と状態遷移機械を実装し、各変数モジュールを直線状に接続する。

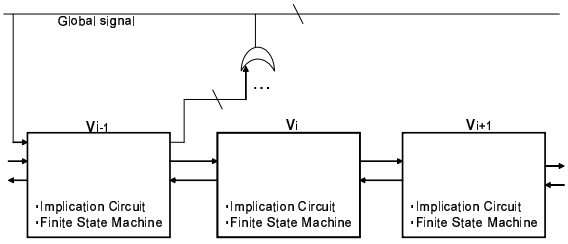

図 2: 回路のモジュール構成

Introducing an Intelligent Backtrack for Solving Boolean Satisfiability Using FPGA
Masahiko Okamura†, Ryuzo Hasegawa‡, Hiroshi Fujita‡
Department of Intelligent Systems, Graduate School of Information Science and Elecrical Engineering, Kyushu University

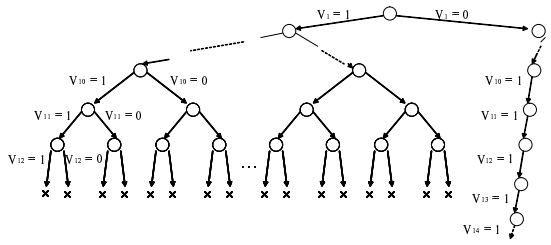

図 3: 冗長な探索の例

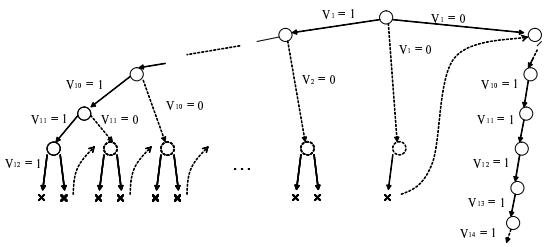

図 5: 知的バックトラックを行う探索の例

4 知的バックトラックの導入

4.1 冗長な探索の例

DP 法では、手続き中に矛盾状態になると 1 つ前に値を割り当てた変数へバックトラックを起こす。しかし、このような順序的なバックトラックでは冗長な探索を行う可能性が高くなる。

図 3 に冗長な探索が起こる例を示す。この図は節 $(\bar{v}_1 + \bar{v}_{12} + v_{13})(\bar{v}_1 + \bar{v}_{12} + \bar{v}_{13})(\bar{v}_1 + v_{12} + v_{14})(\bar{v}_1 + v_{12} + \bar{v}_{14})$ を含む式についての探索木を表す。v_1 から v_{11} まで順に 1 を割り当てた後、v_{12} に 1、0 どちらの値を割り当ててもインプリケーションが生じ、矛盾状態となる。よって、バックトラックを起こし v_{11} に 0 を割り当てるが、v_{12} において同様に矛盾状態になる。この例では v_1 に 1 を割り当てると、v_{12} において必ず矛盾が生じるが、順序的なバックトラックでは v_1 に戻るまで冗長な探索を行ってしまう。

4.2 状態遷移による知的バックトラックの制御

より探索空間を刈り込むためには、値を割り当てた順を飛び越えて、生じた矛盾の本当の原因となる割り当てのなされた変数へ戻る知的バックトラックを行う必要がある。

知的バックトラックはソフトウェアとして実現されている SAT ソルバではよく用いられる手法であるが実行時に動的にデータ構造を変化させる必要があり、ハードウェア上に実装するには不向きである。そこで本稿では、高速なインプリケーションの計算が可能であるというハードウェアの特性を生かし、状態遷移で知的バックトラックを起こす探索を制御する手法を提案する。

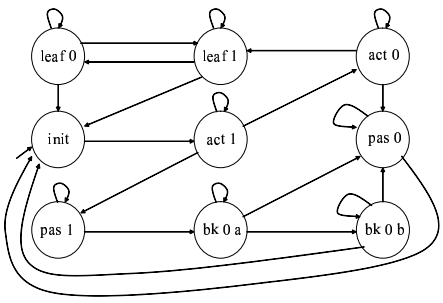

図 4: 各変数の状態遷移図

図 4 に探索を制御するための、変数の状態遷移を示す。各状態では以下のような動作を行う。

init 状態は初期状態であり、変数の値はまだ割り当てられていない。*act 1* 状態では変数に値 1 を割り当て、インプリケーションを計算する。矛盾状態になることなく計算を終えると、値が 1 に決定され *pas 1* 状態へ遷移する。*act 1* 状態で矛盾を検出した場合は、値 0 を割り当てるため *act 0* 状態へ遷移し、同様の計算を行う。*act 0* 状態で矛盾が検出されると 1 と 0 両方の値の割り当てに失敗したことになり、バックトラックを起こす信号を出力して *leaf* 状態へ遷移する。*leaf* 状態は、実際にバックトラックが起きたときの変数の値を初期化せずに記録するための状態である。*pas 1* 状態において、バックトラックが起きたことを示す信号が入力されると、*bk* 状態に遷移する。*bk* 状態では変数に値 0 を割り当ててインプリケーションを計算する。*act 0* 状態との違いは、バックトラックを起こした変数に割り当てられた値も考慮して計算を行う点である。これにより、同じ値の割り当てが原因の矛盾を何度も検出してしまうことなく、矛盾の直接の原因となる割り当てが行われた変数に効率的に戻ることが可能となる。

図 5 は、提案した知的バックトラックを前節で示した例に適用した際の探索木を表す。破線の矢印は刈り込むことのできる探索を示している。

5 おわりに

本稿では、ハードウェア上に構成される SAT ソルバに知的バックトラック機構を導入するための方法について述べた。今後、提案した手法の効果を実験により評価していく予定である。

参考文献

[1] M.Abramovici and J.T.de Sousa, “A SAT Solver Using Reconfigurable Hardware and Virtual Logic,” Journal of Automated Reasoning, 5-36, 2000.

[2] P.Zhong, M.Martonosi, P.Ashar, and S.Malik, “Using Reconfigurable Computing Techniques to Accelerate Problems in the CAD Domain: A Case Study with Boolean Satisfiability,” Design Automation Conf., June 1998.