FPGA上のSATソルバPCMGTPへの前処理の導入

松田, 純一
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 : 修士課程 | 富士通株式会社

越村, 三幸
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

藤田, 博
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

長谷川, 隆三
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

https://doi.org/10.15017/1516864

出版情報：九州大学大学院システム情報科学紀要. 11 (2), pp.109-114, 2006-09-26. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第11巻 第2号 平成18年9月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.11, No.2, September 2006

FPGA上のSATソルバPCMGTPへの前処理の導入

松田純一*・越村三幸**・藤田博**・長谷川隆三**

A Preprocessing Method for the SAT Solver PCMGTP on FPGA

Junichi MATSUDA, Miyuki KOSHIMURA, Hiroshi FUJITA and Ryuzo HASEGAWA

(Received June 16, 2006)

Abstract: This paper describes a preprocessing method for the SAT solver PCMGTP implemented on an FPGA chip. In PCMGTP, each problem is transformed into an HDL code so as to solve the problem directly on an FPGA. It is time consuming to compile an HDL code to a hardware circuit for the FPGA, while its deduction is at speed. Preprocessing SAT problems can sometimes reduce their search space and size considerably. Applying the preprocessing method to SAT problems not only decreases runtime for solving them, but also reduces their circuit size and compilation time. Experimental results show significant performance in solving some benchmark SAT problems.

Keywords: SAT solver, Preprocessing, FPGA, Circuit size

1. はじめに

充足可能性問題（Satisfiability Problem: SAT問題）は多くの実用的応用があり，人工知能分野だけではなく，計算量理論，離散アルゴリズム，設計自動化の分野でも活発に研究されている問題の一つである．これは，与えられたブール式を真にするような変数への値の割当が存在するかどうかを判定する問題として定式化される．近年のSATソルバ（SAT solver）実装技術の進展は著しく，Zchaff[8]やMiniSAT[4]を始めとして，多くの優秀なシステムが発表されている．これらの多くは，Davis-Putnam法（DP法）[2]と呼ばれるアルゴリズムを基に作られている．

一方，FPGA（Field Programmable Gate Array）に代表される再構成可能なハードウェア技術によってより高速な処理を可能とするリコンフィギャラブルコンピューティング[1),13)]と呼ばれる手法が発達してきている．この技術により，アプリケーションごとにハードウェア構成を適応的に再構成することができるようになった．FPGAを利用したSATソルバの開発も行われている[14]．

我々も2002年からSATソルバをFPGA上に構築する試み[9]を始め，幾つかの試行錯誤を経て，PCMGTP (Propositional CMGTP)[10),11)]の開発に至った．PCMGTPは，一階述語論理の定理証明系CMGTP[12]を命題論理に限定して，ハードウェア化したものである．PCMGTPでは，問題ごとにVerilog-HDLで記述した回路を構成し，これをコンパイル（論理合成・配置配線）し，FPGA上にダウンロードして問題を解く．ダウンロード後の実行時間は，ソフトウェアのSATソルバに比べると格段に高速であるが，コンパイルに要する時間が膨大で，問題解決においてここがボトルネックとなっている．

コンパイル時間は，主に問題の記述量に応じて大きくなる．そこで，本研究では，問題を前処理し，できるだけ記述量を減らすことにより，コンパイル時間の短縮を狙う．記述量の減少によって，対応する回路規模が小さくなり，それが論理合成や配置配線に要する時間の短縮につながる．近年，前処理を用いて問題の単純化を行うことでSATソルバの高速化を図る研究[6),3)]が行われており，前処理を用いたいくつかのSATソルバは，SATソルバの世界大会SAT Competition[7]でも優秀な成績を挙げている．このように，前処理はコンパイル時間の短縮だけではなく，推論時間の短縮への寄与も期待できる．

本論文では，PCMGTPへの前処理の導入，そしてその評価を述べる．

2. PCMGTP

モデル生成法に基づいた定理証明器として，MGTP (*Model Generation Theorem Prover*) が開発されている．それを負リテラルに関して拡張し，そして命題論理に特化した定理証明器がPCMGTPである．

2.1 モデル生成法

本論文では，節$\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$を次のような含意形式の形で表現することにする．

$$A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$$

ここで，$A_i\ (1 \leq i \leq n)$および$B_j\ (1 \leq j \leq m)$は命題変

平成18年6月16日受付

* 知能システム学専攻修士課程（現在，富士通株式会社）
** 知能システム学部門

procedure MGTP(S); /* Input(S): Clause set */
MG(ϕ,S);
procedure MG(Mc, S);
/* Input(Mc): Model candidate */
(1) (Model rejection) If a negative clause ($A_1 \wedge \cdots \wedge A_n \rightarrow$ *false*) $\in S$ is violated under Mc, return.
(2) (Model extension) If a positive or mixed clause ($A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$)$\in S$ is violated under Mc,
for(j=1;$j \leq m$; j++) $MG(Mc \sqcup \{B_j\}, S)$
(3) (Model finding) If none of above rules is applicable, **print** *sat*.

Fig. 1　Model generation procedure.

数である．→の左側を前件部，右側を後件部という．$n=0$のとき，前件部を特にtrueと書き，正節と呼ぶ．一方$m=0$のとき，後件部を特にfalseと書き，負節と呼ぶ．それ以外の節（$m \neq 0$, $n \neq 0$）は混合節と呼ばれる．また，$m \leq 1$なる節をホーン節と呼ぶ．

定義 1（リテラル） 命題変数と命題変数の否定をリテラルと呼ぶ．

定義 2（違反節） 節$A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$が命題変数の集合$M$に違反しているとは，$\forall i(1 \leq i \leq n) A_i \in M \wedge \forall j(1 \leq j \leq m) B_j \notin M$であることをいう．

Fig. 1にモデル生成法による証明手続きを示す．手続きMGは真であると考えられる命題変数の集合Mc（モデル候補）と節集合Sを受け取る．手続きが*sat*と出力すればSは充足可能で，何も出力せずに終了すればSは充足不能である．

2.2　負リテラルに関する拡張

推論効率の改善を図るため，**Fig. 1**のMcに命題変数だけではなく，その否定，つまりリテラルの出現をも許すことにして，**Fig. 1** (2) の$MG(Mc \sqcup \{B_j\}, S)$を$MG(Mc \sqcup \{B_j, \neg B_{j+1}, \ldots, \neg B_m\}, S)$に変更するとともに，(1) と (2) の間に次の (1.5) を挿入して，モデル生成法を拡張する．

(1.5) (Model rejection) If $\exists A$ ($A \in Mc \wedge \neg A \in Mc$), return.

また，S中の各節$A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$について，それと等価な次の$n+m$個の確定節を$S$に加える．

$$\begin{cases} A_1 \wedge \cdots \wedge A_n \wedge \neg B_1 \wedge \cdots \wedge \neg B_{j-1} \\ \qquad \wedge \neg B_{j+1} \wedge \cdots \wedge \neg B_n \rightarrow B_j \ (1 \leq j \leq m) \\ A_1 \wedge \cdots \wedge A_{i-1} \wedge A_{i+1} \wedge \cdots \wedge A_n \\ \qquad \wedge \neg B_1 \wedge \cdots \wedge \neg B_n \rightarrow \neg A_i \ (1 \leq i \leq n) \end{cases}$$

これらの確定節は，前件部に命題変数の否定の出現も許すので，**定義2**のMをリテラルの集合に拡張して，違反節の定義も変更する．

2.3　FPGA上のPCMGTP

PCMGTPは以下の処理を繰り返す．各処理の詳細は文献15）を参照されたい．

(1)　含意に基づく付値：ホーン節によるモデル拡張に対応する処理．
(2)　違反節の選択，投機的付値，モデル候補の拡張：非ホーン節によるモデル拡張に対応する処理．
(3)　リテラルの再選択：バックトラックに対応する処理．
(4)　充足性判定：モデル棄却とモデル発見に関する処理．

また，探索ヒューリスティックとしてSDF (Shortest Disjunction First) 戦略を採用している．これは，(2)における違反節の選択の際に後件長の少ない非ホーン節を優先するものである．この戦略は，N王妃パズル，準群問題など，選言要素数の多い節が多く現れる問題に有効であることが知られている．

FPGA上に実装するPCMGTPは，推論エンジンモジュール*Engine*，命題変数モジュールP_i ($0 \leq i \leq \mu$)，節モジュールC_k ($0 \leq k \leq \gamma$)，非ホーン節モジュールN_j ($0 \leq j \leq \nu$)，トーナメントモジュール*Tournament*によって構成される．ここで，μは命題変数の総数，γは節の総本数，νは非ホーン節の本数である．各モジュールの役割を以下に示す．

- 推論エンジンモジュール：他のモジュールを統御し推論を進める．
- 命題変数モジュール：命題変数への付値，および命題変数におけるconflictの検出．
- 節モジュール：当該節における含意の計算．
- 非ホーン節モジュール：当該非ホーン節が違反節であるかの検査，および当該非ホーン節における非ホーン拡張．
- トーナメントモジュール：違反節の選択の際のSDF戦略の実現．

また，一つの非ホーン節は，節モジュール，非ホーン節モジュールの二つのモジュールを要している．

3. 前　処　理

命題変数，非ホーン節，節の各モジュールについて，その構成に必要な論理セル数を**Table 1**に示す（節モジュールについては，リテラル2個のものでは論理セルを検出できなかったため，3個のものを参考として用いている）．**Table 1**からも分かるとおり，非ホーン節モジュールに要する論理セル数は非常に大きく，また，非ホーン節において後件が一つ増えたときに必要な論理セルも，これがリテラル一つに必要なものであることを考えると，大きいものであると言える．このため，非ホーン節またはその後件を削減することが，各問題における必要回路の規模の減少に効果的であると考えられる．また，PCMGTPにおける探索木の各ノードは非ホーン節であり，その枝の数は非ホー

Table 1 Number of logic elements needed to construct each module.

module	No. of logic cells
propositional variable module	14
non-Horn module (without antecedent part)	21 (for a clause with 2-literals in consequent)
	26 (for a clause with 3-literals in consequent)
clause module (with 3-literals)	3

ン節の後件の大きさと等しい．よって，非ホーン節の削減，または非ホーン節の後件の短縮は，探索空間の削減にも繋がる．

以上の理由により，本研究で用いた前処理においては，各非ホーン節の後件長を足し合わせた数，つまり，非ホーン節の後件長の総和を評価の対象とし，この減少を，前処理における問題構造の改良と捉える．これが増加するような処理は行わない．

前処理の単純化手続きをFig. 2に示す．手続きは，節集合が変化しなくなるまで繰り返し適用される．本研究で用いた前処理機能では以下に説明する各機能に加え，DP法でも用いられている，1リテラル規則（one literal rule）と純リテラル規則（pure literal rule）を併用する．

3.1 定義抽出による変数の削除

節集合S中の幾つかの節から命題変数xと等価な論理式Fが得られることがある．例えば，S中に$x \vee \neg a \vee \neg b$，$\neg x \vee a$，$\neg x \vee b$の三つの節があった場合，$x=(a \wedge b)$なる等式が得られる．ここで$F=(a \wedge b)$である．この$F$を$x$の定義（definition）と呼ぶ．変数をその定義で置換することにより，Sからその変数の出現がなくなり，結果として，変数の数を減らすことができる．なお，置換を行っても充足可能性は変わらないが，複数のモデルを同一視する可能性があるため，モデルの数自体は減少することがある．

本研究では，選言のみ，または連言のみで構成される定義のみ抽出を試みる．また，前述の通り，非ホーン節の後件長の総和が増えるような置換は行わない．但し，定義が単一のリテラルからなる場合，無条件に置換を行う．これは，このような定義では置換後に変数が必ず一つ減少し問題全体のリテラル生起回数も必ず減少することに加え，1リテラル規則や純リテラル規則，後述する各機能が働く可能性が増えるとともに，新たな定義の抽出の可能性も増すからである．

3.2 節分配による変数の削除

節C_1とC_2からの導出節（resolvent）を$C_1 \otimes C_2$，節集合Sの中でリテラルxを含む節の集合をS_x，と表記することにする．ある論理変数xに着目し，S_xと$S_{\neg x}$の導出節全体$S'=\{C_1 \otimes C_2 \mid C_1 \in S_x,\ C_2 \in S_{\neg x}\}$を考える．このとき，$S$と$S \sqcup S' \setminus S_x \setminus S_{\neg x}$の充足可能性は等しい．そして，後者は変数$x$を含まないので，前者より変数の数は少ない．このようにSから変数を削除する操作を節分配（clause distribution）[3]と呼ぶ．節分配によって充足可能性は保たれるが，定義による変数の削除と同様にモデル数は減少する可能性がある．

第3.1節と同じく，分配によって非ホーン節の後件長の総和が増える場合は，分配は行わない．加えて，定義を持つ変数xに対しても，節分配による変数の削除は行わない．これは，定義を適用した方が有用な節がより早く導出される[5]からである．

3.3 自己包摂を用いた単純化

節集合中に，次の二つの節が存在したとする．

$\{\ldots, C_1=(x \vee a), C_2=(\neg x \vee a \vee b), \ldots\}$

このとき，C_1とC_2から導出節$C'=(a \vee b)$が得られる．これを節集合に加えると，

$\{\ldots, C_1=(x \vee a), C_2=(\neg x \vee a \vee b), C'=(a \vee b), \ldots\}$

となる．ここで，C_2はC'に包摂されるので削除でき，節集合は

$\{\ldots, C_1=(x \vee a), C'=(a \vee b), \ldots\}$

となる．これはつまり，導出節を考えることでC_2中のリテラル$\neg x$を削除したと見なせる．このように，自身と他の節による導出節を考えることにより一つのリテラルを削除する方法を，自己包摂（self-subsumption）によるリテラル削除，と呼ぶ[3]．

一般的には，節集合中に節C_1と節C_2が存在し，C_1中のリテラルのうち一つを反転させた節C'がC_2を包摂する時，自己包摂が適用できて，C_2からリテラルを一つ削除できる．このとき削除されるリテラルは，“C_1からC'に変換する際に反転させたリテラル”を反転させたものである．

自己包摂による単純化では，必ずリテラルが除去されるので，無条件にこれを適用する．

3.2 リテラルの反転

節集合に現れる変数xの極性を反転，つまりxを$\neg x$に，$\neg x$をxに同時に置換，しても充足可能性は一致する．この操作をリテラルの反転（literal-flip）と呼ぶことにする．本研究では，リテラルの反転によって，非ホーン節の後件長の総和の減少する場合，これを適用する．

```
simplify(){
  while(there exists a newly generated clauses){
    apply self-subsumption.
    apply subsumption.
    apply pure literal rule
    for each v∈variable_set { variable_elimination(v) }
    for each v∈variable_set {
      if(v is not eliminated){
        if((number of v occurence) > (number of ¬v occurence)) apply literal-flip
} } } }

variable_elimination(v) {
  if (there exists unit-definitions) replace variables with their definitions
  if (number of v/¬v occurrences is greater than a threshold) return /* heuristic cut-off */
  if (there are applicable definitions) replace variables with their definitions
  if (at least one definition has been applied) return
  apply clause distribution.
}
```

Fig. 2 Preprocessing algorithm.

4. 比 較 評 価

前節で述べた前処理プログラムをJavaを用いて実装し，その効果を実験により確かめた．回路記述はVerilog-HDL，論理合成はQuartus Ⅱ（Version5.0），FPGAはALTERA EP20K1500EBC652-3（論理セル数 51840個）を使用した．前処理プログラムと論理合成ツールの実行には共に，IBMデスクトップPC（Pentium4 3.2GHz CPU，2GBメモリ，Windows XP）を用いた．問題は，準群問題，およびSATソルバのコンペSAT COMPETITION[7]で使用されたベンチマーク問題を用いた．

Table 2に前処理の結果を示す．ここで，Vars，Cls，NH，NH_sumにおいては，（前処理後の数値）/（前処理前の数値）を示している．非ホーン節の数(NH)が増加している問題もあるが，全ての問題で非ホーン節の後件長の総和（NH_sum）は減少している．また，命題変数の数(Vars)，節の数（Cls）は全ての問題において減少ないし維持している．前処理に要した時間（Pre_time）は約1秒～24秒であった．

次に，FPGA上での実験結果を**Table 3**に示す†1．表中の下段に示した[ ]内の値は，前処理前の問題のPCMGTPでの結果を表しており，*size-over*は，回路規模が大きすぎてFPGAの容量を超えたことを，*time-out*は時間切れを示す．所要クロック数3.0×10^{11}（動作周波数20MHzにおいて，およそ4時間）を制限時間とした．また，*size-over*や*time-out*によって実行，比較ができなかった部分については“—”と表す．実行できなかった問題については，既知のモデルの数，または既知の充足可能性を示している．

全ての問題において，正しく充足可能性が得られている．前処理後の問題においてモデルの数（M）が減少している問題がいくつかあるが，これは前処理によって複数のモデルを同一視したためと考えられる．前処理後の問題でモデルの数が増加したというものも無く，正しく結果が得られたと言える．また，全ての問題において，回路規模（Cells）の削減，およびコンパイル時間（Comp.）の短縮に成功している．回路規模は最大で31%(Cell_ratio)に削減されている．加えて，従来は回路規模の関係でFPGA上にダウンロードできなかった問題についても前処理を通すことでダウンロードさせることができており，前処理による回路削減の効果があったことが分かる．実行時間（Time）は，問題を解くのに費やしたクロック数（Clks）を最大動作周波数（FMAX）で割ることによって求めている．前処理により全ての問題で推論実行速度が向上しており，最大で2300倍（Time_ratio）の高速化に成功している．

次にC言語で開発されたSATソルバMiniSat1.14[4]との比較も行った．これはSAT COMPETITION2005で好成績を収めたシステムである．問題は今回使用したもののうち，充足不能（unsat）の問題のみを解いた（PCMGTPは全解探索，MiniSatは単解探索であるため）．MiniSatの推論実行時間（MiniSat）と比較して，ca016を除く全ての問題において，前処理を用いた方法はMiniSatより高速に問題を解決している．また，homer17，homer18にお

†1　各表中において$>n$は，“少なくともn以上である”ことを示している．

Table 2 Numerical results of preprocessing for some benchmark problems.

Problem	Vars	Cls	NH	NH_sum	Pre_time
qg5_8	245/512	5300/17438	98/70	371/539	9.634
qg5_9	395/729	10365/28141	127/88	546/764	23.433
qg6_8	245/512	5277/17382	99/70	366/539	9.543
qg6_9	396/729	10375/28069	128/88	549/764	23.283
qg7_8	293/512	6773/17430	115/70	419/539	10.465
qg7_9	453/729	12523/28141	138/88	611/764	24.145
homer17	286/286	1742/1742	26/415	264/924	1.793
homer18	307/308	2029/2030	27/592	285/1272	2.293
ca016	121/272	408/780	122/282	306/629	1.392
dp04u03	184/1017	839/2411	265/826	619/1773	7.290
cnt05	116/315	413/1002	137/439	341/963	2.304
ezfact32_1	209/769	1238/4777	602/3034	1458/7577	22.763

いては，前処理を用いない従来の方法ではMiniSatに比べ膨大な時間を要してしまっているが，前処理を用いることで，コンパイル時間などを加味した，問題解決に要した全ての時間と比べても，MiniSatより高速に問題を解決することができた．

この実験より，前処理を用いることで，PCMGTPは論理合成ツールによるコンパイル時間などを考慮に入れても，MiniSATより高速な問題解決が可能であるということが示された．

5. おわりに

本研究では，FPGA上で走行するSATソルバPCMGTPへの前処理の導入による回路の削減，およびそれに伴う高速化について述べた．前処理により回路規模を約40%に縮小し，FPGAの論理素子使用効率を格段に向上させることができた．これにより，従来はFPGA上に実装不可能であった，命題変数729個，節数約28000本である9次の準群問題を1つのFPGAチップに実装することが可能となった．回路規模の減少に伴い，コンパイル時間についても約30%に短縮できている．

さらに，最大動作周波数の向上や探索空間の削減により，推論実行の速度としても1.5倍以上の向上に成功しており，推論実行の面においても前処理が効果的に作用したと言える．

FPGAへのコンパイル時間については，前処理を導入することでその時間を短縮することができたものの，未だ相当な時間を費やしていることに変わりはない．これは深刻な問題であるが，未解決SAT問題など更に難易度の高い問題の解決がPCMGTPの目標であり，その場合，実行時間が数日～数ヶ月かかることも珍しくない．難易度の高い問題に関しては，論理合成時間を考慮に入れても，十分にPCMGTPのメリットを生かすことができると考えている．実際に，今回使用したいくつかの問題においては，前処理を併用することで，コンパイル時間など問題解決に要した全ての時間を考慮しても，ソフトウェアのSATソルバより高速な問題解決を行うことができた．このように難易度の高い問題においては，ソフトウェアのSATソルバと比べ，前処理を導入したPCMGTPは格段に優位に立てると考えている．

今後の課題として，ソフトウェアのSATソルバに標準的に備わっている補題の自動生成・利用，知的バックトラック等の採用が挙げられる．現在，FPGA上での補題の自動生成・利用についての実装は可能であるものの，実装に伴う回路規模の著しい増加が予想されている．実装方法の一つとして，ソフトとハードのハイブリッド化が挙げられる．本研究では，ソフトウェアで問題の単純化を行い，単純化したものをハードウェアで解くという，いわば静的なハイブリッド化を行った．これに加え，例えば，FPGAをPCとリンクさせることによって，ハードウェアの特長である並列実行可能な部分はハードウェアを用いて実行し，ソフトウェアが得意とする補題の生成，削除や，知的バックトラック機能の部分はソフトウェアで実行する，というような動的なハイブリッド化により，さらなる高速化が可能であると考えている．

参考文献

1) K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, Vol.34, No.2, pp.171-210, 2002.
2) M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of ACM*, Vol.7(3), pp.394-397, 1960.
3) N. Eén and A. Biere, "Effective Preprocessing in SAT through Variable and Clause Elimination," SAT 2005,

Table 3 Numerical results on FPGA and MiniSat for some benchmark problems.

Problem	M	Cells	Cells_ ratio	Comp. (secs)	FMAX (MHz)	Clks	Time (secs)	Time_ ratio	MiniSat (secs)
qg5_8	1 [1]	16085 [35835]	0.45	5378 [51525]	20.92 [12.88]	161 [175]	7.7×10^{-6} [1.36×10^{-5}]	1.76	—
qg5_9	0 [0]	27276 [*size-over*]	—	12609 [—]	17.20 [—]	404 [—]	2.35×10^{-5} [—]	—	0.11
qg6_8	2 [2]	16019 [34212]	0.47	5579 [22863]	21.70 [14.04]	127 [138]	5.85×10^{-6} [9.83×10^{-6}]	1.68	—
qg6_9	4 [4]	28172 [*size-over*]	—	15605 [—]	19.04 [—]	641 [—]	3.37×10^{-5} [—]	—	—
qg7_8	0 [0]	19750 [34884]	0.57	8563 [47017]	20.47 [13.75]	768 [1041]	3.75×10^{-5} [7.57×10^{-5}]	2.02	0.078
qg7_9	1 [4]	33207 [*size-over*]	—	20364 [—]	13.19 [—]	6526 [—]	4.95×10^{-4} [—]	—	—
homer17	0 [0]	8247 [23117]	0.36	2116 [7772]	22.13 [21.05]	3.7×10^{8} [*time-out*]	13.47 [$>1.4\times10^{4}$]	>1058	4157
homer18	0 [0]	9210 [29998]	0.31	3440 [9576]	21.06 [18.76]	3.7×10^{8} [*time-out*]	17.65 [$>1.6\times10^{4}$]	>906	8900
ca016	0 [0]	7958 [18286]	0.44	1737 [9350]	25.53 [23.44]	1.3×10^{8} [6.5×10^{10}]	4.88 [2754.30]	563.87	0.046
dp04u03	0 [0]	14628 [*size-over*]	—	3490 [—]	22.20 [—]	6729 [—]	3.0×10^{-4} [—]	—	0.031
cnt05	1 [1]	7991 [24971]	0.32	1841 [8929]	27.78 [21.47]	39918 [7.3×10^{7}]	1.4×10^{-3} [3.42]	2381.08	—
ezfact32_1	1 [SAT]	27629 [*size-over*]	—	9517 [—]	18.18 [—]	2.2×10^{5} [—]	0.012 [—]	—	—

LNCS 3569, pp.61-75, 2005.

4) N. Eén and N. Sörensson, "An Extensible SAT-solver," SAT 2003, LNCS 2919, pp.502-518, 2004.
5) É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs, "Automatic extraction of functional dependencies," SAT 2004, LNCS 3542, pp.122-132, 2005.
6) R. Ostrowski, É. Grégoire, B. Mazure, and L. Sais, "Recovering and exploiting structural knowledge from CNF formulas," CP 2002, LNCS 2470, pp.185-199, 2002.
7) "SAT competitions," http://www.satcompetition.org/
8) L. Zhang, C. F. Madigan, M. W. Moskewicz and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," ICCAD 2001, pp.279-285, 2001.
9) 池田 竜馬, 長谷川 隆三, 越村 三幸, "FPGAによる充足可能性検査システムの試作と評価," **火の国情報シンポジウム2002発表論文集**, 情報処理学会九州支部, pp.384-391, 2002.
10) 河野 真史, 藤田 博, 長谷川 隆三, "モデル生成型定理証明器のFPGA上の実装," **情報処理学会 研究報告**, 2004-SLDM-113, pp.119-124, 2004.
11) 木之下 昇平, 松田 純一, 藤田 博, 越村 三幸, 長谷川 隆三, "FPGA上に実装されたPCMGTPを用いたSAT問題の解決," **情報処理学会 研究報告**, 2005-SLDM-118, pp.57-62, 2005.
12) 白井 康之, 長谷川 隆三, "モデル生成型定理証明システムによる制約充足問題の解決とその並列化," **電子情報通信学会論文誌**, Vol.J80-D-II, No.1, pp.224-236, 1997.
13) 末吉 敏則, 天野 秀晴 [編著], "リコンフィギャラブルシステム," オーム社, 2005.
14) 須山 敬之, 横尾 真, 澤田 宏, 名古屋 彰, "再構成可能なハードウェアを用いた充足可能性問題の解法," **電子情報通信学会論文誌**, Vol.J84-D1, No.4, pp.410-420, 2001.
15) 藤田 博, 長谷川 隆三, 越村 三幸, 木之下 昇平, 松田 純一, "FPGA上のSATソルバPCMGTPの改良について," **九州大学大学院システム情報科学紀要**, 第10巻, 第1号, pp.21-26, 2005年3月.