Verificação de Propriedades de Circuitos Digitais Combinatórios Utilizando Técnica SAT

Fabrício S. Paranhos Felipe Gemmal Arthur C. Costa

Resumo. *Este meta-artigo descreve como usar a classe que define o modelo para a confecção de Relatórios Técnicos e outros documentos, segundo o padrão adotado pelo Instituto de Informática da UFG. É solicitada a escrita de resumo e abstract. O autor deve tomar cuidado para que o resumo e o abstract não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do relatório.*

Palavras-Chave: Relatório Técnico, LaTeX.

1 Introdução

Na atualidade os sistemas digitais estão crescendo exponencialmente em complexidade para reduzir o tamanho dos componentes. Um problema que isso gera é a garantia do funcionamento de todas as partes e do conjunto delas. Nesse contexto podem-se usar métodos de inteligência artificial (IA) para deixar essa resolução mais eficiênte.

Porêm apenas a aplicação direta de IA não é suficiente, precisa-se modelar o problema e processar as informações da tal forma que o algoritmo possa chegar a uma conclusão de forma eficiênte. Para tal uma alternativa bastante utilizada é o formato de satisfabilidade booleana (SAT) que interage bem com uma busca informada.

O SAT (problema de satisfatibilidade booleana) é um problema clássico de decisão da lógica proposicional, NP completo, que diz respeito ao teste de uma expressão booleana que só aceita AND (e lógico), OR (ou lógico) e NOT (negação), variáveis e parênteses modelados no formado de conjunção de disjunções. Consiste em determinar se existe algum estado das variáveis em que a saída da expressão é positiva (satistativel).

Além do modelo de satisfabilidade também é necessário um formato de interpretação do circuito envolvido no problema, por isso foi elaborado um modelo básico de hardware baseado em *Hardware Description Language* (HDL). O formato de descrição de circuito usado recebe um circuito na forma (porta NAND como exemplo):

```
Nome_Circuito [intput;output] {componentes_do_circuito}
NAND [a b; c] {
  a b AND d;
  d NOT c;
}
```

2 SAT

2.1 Problema de satisfatibilidade booleana

O problema da satisfatibilidade booleana (SAT) é o problema de se determinar, se existe, alguma interpretação que torna uma dada formula booleana verdadeira, ou seja a satisfaça. Por exemplo dada a seguinte formula booleana $F = (A \wedge \neg B)$ é satisfazível pois, se $A = \top$ e $B = \bot$ temos $F = \top$.

O que torna o problema SAT interessante é o fato de estar incluso na categoria de problemas NP-completo (foi o primeiro problema provado ser NP-completo). Isto significa que SAT é um representante de toda classe NP de problemas, sendo mais ou igualmente dificil de resolver quanto qualquer outro problema NP. Mesmo sendo um problema difícil de resolver (exponêncial), existem soluções baseadas em heuristicas e buscas que visam diminuir a complexidade, ou diminuir o caso médio, permitindo resolver grandes instâncias em pouco tempo.

Por definição de NP-completo, qualquer problema na classe NP pode ser reduzido a um problema NP-completo em tempo polinomial[1]. Podemos nos aproveitar deste fato para resolver problemas difíceis e sem solução, reduzindo-o a uma instância SAT e utilizando algorítimos otimizados específicos para SAT. Entre tais algorítimos podemos citar: Davis-Putnam-Logemann-Loveland (DPLL) [2][3], que forma a base de outros algorítimos, e Conflict-Driven Clause Learning (CDCL) [4], que inclui apredizagem de clausulas e *backtracking* não cronológico em cima do DPLL.

2.2 MiniSat

Minisat é um minimalistico, open-source resolvedor de problemas de satisfatibilidade booleana (SAT), desenvolvido para ajudar pesquisadores e desenvolvedores a começarem desenvolvimento em SAT's. Seu funcionamento é basiado em backtracking por conflito de analises e aprendisado.

Os componentes do resolvedor do MiniSat podem ser divididos conceptualmente em três categorias. A primeira é a representação, de alguma forma a instância do SAT deve ser representada por estruturas de dados. A segunda é a inferência, o resolvedor necessita de mecanismos para computar e propagar as implicações do estado atual de informações. A terceira é pesquisa, geralmente combinado com inferência, pesquisa é necessária para encontrar informação.

Para realizar pesquisas, assunções são feitas, atribuindo valores a variáveis selecionadas até que a propagação detecte um conflito. Nesse ponto uma cláusula conflito é construída e adicionada ao problema SAT.

O MiniSat utiliza como base o algoritmo de pesquisa de Davis-Putman-Longemann-Loveland (DPLL), baseado em retrocesso (backtracking), que serve para decidir a satisfatibilidade de formulas da logica proposicional em formula normal conjutiva, ou seja, para resolver problemas SAT.

O algoritimo de backtracking escolhe um literal Φ e lhe da o valor de `TRUE`, simplificando a formula e depois recursivamente checando se a formula simplificada é satisfatível. Caso seja, a

[1] J. van Leeuwen (1998). Handbook of Theoretical Computer Science. Elsevier. p. 84. ISBN 0-262-72014-0.

[2] Davis, M.; Putnam, H. (1960). "A Computing Procedure for Quantification Theory". Journal of the ACM. 7 (3): 201. doi:10.1145/321033.321034

[3] Davis, M.; Logemann, G.; Loveland, D. (1962). "A machine program for theorem-proving". Communications of the ACM. 5 (7): 394–397. doi:10.1145/368273.368557

[4] J.P. Marques-Silva; Karem A. Sakallah (May 1999). "GRASP: A Search Algorithm for Propositional Satisfiability". IEEE Transactions on Computers. 48 (5): 506–521. doi:10.1109/12.769433

formula original também é satisfatível, caso contrario, a mesma checagem recursiva é feita mas dando o valor `FALSE` dado ao literal ϕ. Assim dividindo o problema em dois, simplificados, sub-problemas.

Se uma clausula contem apenas um literal, então essa clausula so pode ser satisfeita se for associado o valor `TRUE` ao literal. Quando uma variável proposicional ocorre com apenas uma polaridade, ela é chamada de pura. Variáveis puras podem ser associadas de um jeito onde toda clausula que as contem resultam em `TRUE`, logo tais clausulas podem ser deletadas da busca.

```
funcao DPLL(Φ, u)
 se todas as clausulas de Y forem verdadeiras
     entao retorne TRUE;

 se alguma clausula de Y for falsa
     entao retorne FALSE;

 se ocorrer uma clausula unitaria c em Y
     entao retorne DPLL(atribuicao(c,Y), (u and c));

 se ocorrer um literal puro c em Y
     entao retorne DPLL(atribuicao(c,Y), (u and c));

 c := escolha_literal(Y);

 retorne DPLL(atribuicao(c,Y), (u and c)) ou
         DPLL(atribuicao(-c,Y), (u and -c));
```

No pseudocódigo acima, `atribuição(c, Φ)` é uma função que retorna uma fórmula obtida pela substituição de cada ocorrência de `c` por `TRUE`, e cada ocorrência do literal oposto por falso na fórmula `Y`, e em seguida, simplificando a fórmula resultante. A função DPLL do pseudocódigo retorna verdadeiro se a atribuição final satisfaz a fórmula ou falso se tal atribuição não satisfaz a fórmula. Em uma implementação real, a atribuição satisfatível também é retornada no caso de sucesso (esta foi omitida para maior clareza).

3 Implementação

3.1 Compilador

O sistema de leitura do circuito e da Forma Normal Conjuntiva (CNF) foram implementados utilizando as ferramentas flex[5] e bison[6].

O flex é um analisador lexicoque analisa o texto e o separa em tokens baseados em expressões regulares (formados que descrevem um conjunto de letras e números em quantidade e ordem específicada). O bison é um parser generator, ou analisador sintático, que recebe esses tokens do analisador lexico, e verifica se estão em concordância com a gramática especificada.

Para o circuito ser analisado ele foi separado em tokens de variaveis representando os fios de entrada e saida e identificadores representando o nome dos circuitos envolvidos, já que um circuito maior pode ser representado por um conjunto de circuitos menores. Abaixo está a gramática utilizada no formato Backus-Naur.

```
circuito    ::=
   ID '[' variaveis ';' variaveis ']' '{' componentes '}'
componentes ::= componente | componentes componente
```

[5]https://github.com/westes/flex

[6]https://www.gnu.org/software/bison/

```
componente  ::= variaveis ID variaveis ';'
variaveis   ::= VAR | variaveis VAR
```

No caso do CNF o analisador léxico separa basicamento o nome que representa o circuito representado e os números do conjunto, e o papel do analisador sintático é de separar os números em suas respectivas cláusulas e converte-las para o formato implementado no sistema SAT. Abaixo está a gramática utilizada.

```
cnf       ::= ID NUM clausulas
clausulas ::= clausula | clausulas clausula
clausula  ::= '(' variaveis ')'
variaveis ::= NUM | variaveis NUM
```

3.2 Transformação de Tseytin

Todo circuito digital combinatório pode ser escrito como uma formula da lógica proposicional, utilizando os conectivos usuais. Entretanto SAT espera uma CNF como entrada, ou seja, contendo apenas os operadores binários $\wedge$ e $\vee$. A transformação é realizada utilizando equivalências lógicas, como as leis de De Morgan, até que se obtenha o formato esperado. Um ponto negativo deste método é a explosão de termos causado pela distributividade da disjunção na conjunção: $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, gerando um aumento exponêncial no tamanho do termo.

Para contornar este problema utilizamos uma transformação proposta por Tseytin [7]. Neste método o tamanho da formula final é linear com o tamanho do circuito. A transformação não gera formulas equivalentes, pois há introdução de variáveis novas, mas mantem a satisfatibilidade, ou seja, a formula final é satistatível se, e somente se, a formula inicial for satisfatível. A transformação consiste em três partes (exemplo abaixo): idêntificação de todas subformulas, introduzir uma nova variável para cada subformula e conjunção de todas subformulas. Formula original,

$$\phi := ((p \vee q) \wedge r) \leftarrow (\neg s)$$

idêntificação das subformulas,

$$\begin{gathered} \neg s \\ p \vee q \\ (p \vee q) \wedge r \\ ((p \vee q) \wedge r) \leftarrow (\neg s) \end{gathered}$$

introdução de variáveis novas,

$$\begin{aligned} x_1 &\leftrightarrow & \neg s \\ x_2 &\leftrightarrow & p \vee q \\ x_3 &\leftrightarrow & (p \vee q) \wedge r \\ x_4 &\leftrightarrow & ((p \vee q) \wedge r) \leftarrow (\neg s) \end{aligned}$$

conjunção de todas as formulas e substituição em ϕ,

$$T(\phi) := x_4 \wedge (x_4 \leftrightarrow x_3 \leftarrow x_1) \wedge (x_3 \leftrightarrow x_2 \wedge r) \wedge (x_2 \leftrightarrow p \vee q) \wedge (x_1 \leftrightarrow \neg s)$$

deste ponto em diante utiliza-se as transformações usuais da lógica proposicional.

[7] G.S. Tseytin: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics, pp. 115–125. Steklov Mathematical Institute (1970).

4 Resultados

No programa já estão incluidas as portas básicas AND, OR e NOT. Circuitos mais complexos devem ser construidos a partir de tais portas, montando uma biblioteca de circuitos disponíveis para uso posterior.

4.1 Compilação

O processo de compilação é relativamente simples. O programa aplica as transformações de Tseytin para portas já definidas, emitindo, no final a CNF do circuito descrito na entrada. Para ilustrar os resultados iremos utilizar a porta NAND como exemplo.

```
cat nand.txt
NAND [a b; c] {
  a b AND d;
  d NOT c;
}
```

A estrutura do input é formada pelo nome do circuito (deve começar com uma letra maiúscula), seguido da descrição das entradas e saidas (aqui a b são portas de entrada enquanto c saida) e sucessivamente uma lista de componentes do circuito (portas AND e NOT).

Para a obtenção da CNF executa-se o comando de compilação e o resultado é emitido para saida padrão do sistema.

```
satcirc c nand.txt > NAND.cnf
NAND 3 (-1 -2 4)(1 -4)(2 -4)(-4 -3)(4 3)
```

Como visto na descrição do NAND acima o usuário pode introduzir variáveis para descrever saidas de subcircuitos para posterior utilização interna.

4.2 Especificação

A verificação de propriedades é feita utilizando um arquivo auxiliar. Como exemplo, queremos verificar se nossa especficação de porta NAND está correta. Vamos analisar em quais condições a saida é verdade(1):

```
cat nand_ver.txt
c
satcirc v nand.txt nand_ver.txt
SAT a(0) b(0) c(1)
```

Para saida c(1) temos que é satisfatível (SAT) para as entradas a(0) e b(0), como de acordo com a especificação da porta NAND. Entretanto está não é a única possibilidade de c(1), para encontrar as outras possibilidades devemos introduzir o resultado negado no arquivo de verificação:

```
cat nand_ver.txt
c
a b -c
satcirc v nand.txt nand_ver.txt
SAT a(0) b(1) c(1)
```

Para cada resultado, uma nova linha negada é introduziada no arquivo de verificação:

```
cat nand_ver.txt
c
a b -c
a -b -c
```

```
satcirc v nand.txt nand_ver.txt
SAT a(1) b(0) c(1)
```

e finalmente:

```
cat nand_ver.txt
c
a b -c
a -b -c
-a b -c
satcirc v nand.txt nand_ver.txt
UNSAT
```

Temos que este problema não é satisfatível (UNSAT), ou seja, não existem mais possibilidades da saida c(1) de acordo com as especificações de verificação. Abaixo temos o análogo para saida c(0).

```
cat nand_ver.txt
-c
satcirc v nand.txt nand_ver.txt
SAT a(1) b(1) c(0)

cat nand_ver.txt
-c
-a -b c
satcirc v nand.txt nand_ver.txt
UNSAT
```

5 Considerações finais

Em casos reais de verificação de circuitos, a quantidade de variáveis chegaria a casa de milhares ou centenas de milhares, assim, SAT solvers são exemplos de que, mesmo com uma complexidade muito alta, podemos resolver problemas de tamanho expressívos utilizando poucos recursos, em pouco tempo.

Conseguimos verificar propriedades de circuitos combinatórios digitais simples. Utilizando um compilador de lógica proposicional para CNF poderiamos verificar propriedades mais avançadas, como a releção entre propriedade de input e output com o uso de implicações, com uma possível continuação do desenvolvimento deste software.