

# Formal Hardware Verification Methods: A Survey

(Resumo do Artigo)

Jones Oliveira de Albuquerque

DCC - ICEx - UFMG  
Postal Box 702  
30161-970 - Belo Horizonte - MG - Brazil  
joa@dcc.ufmg.br

## Resumo

Este texto apresenta um resumo do artigo *Formal Hardware Verification Methods: A Survey* de Aarti Gupta.

Neste resumo apresentamos a estrutura do artigo e resumimos seu conteúdo mantendo a ordem dos tópicos.

## 1 Estrutura do Artigo

O *survey* está organizado nas seguintes seções:

1. Introdução, verificação formal de hardware - visão geral, resumo do conteúdo do paper;
2. Questões levantadas pelo autor relativas a metodologias de verificação formal;
3. Descrição dos vários modos utilizados para verificação formal de hardware;
4. Visão geral e resumida dos modos descritos na seção 3;
5. Linhas de pesquisa na área e tendências futuras;
6. Conclusões, agradecimentos e bibliografia classificada por tópico.

Este resumo apresenta basicamente a mesma estrutura de tópicos apresentada no artigo original.

## 2 Introdução e Visão Geral

Com o crescimento dos sistemas desenvolvidos, o nível de complexidade dos mesmos também cresceram. Especificamente, a garantia de corretude para sistemas complexos se tornou uma área de pesquisa bastante intensa, haja vista a escassez de modelos que permitam a verificação de corretude e o alto custo envolvido no caso de um erro num sistema de hardware, tanto custo operacional quanto de mercado.

Especificação formal de hardware surgiu como uma alternativa, pois a simulação completa de sistemas complexos não é viável devido basicamente a dois fatores: *i.* quantidade de padrões necessários para testar todas as possibilidades do sistema numa simulação exaustiva, e, como ocorre na prática, *ii.* a simulação de partes do sistema deixa “frestas” que podem dar margem a erros futuros.

Verificação formal de hardware consiste em formalmente estabelecer que uma implementação satisfaz uma especificação, sem entretanto verificar a validação da especificação. Ou seja, se a especificação retrata os requisitos do sistema.

## 3 Questões em Verificação Formal

Nesta seção, o *survey* apresenta algumas questões relacionadas à metodologia de verificação formal de hardware. As questões versam sobre os vários graus de abstração e poder de expressão quanto a implementação, especificação e verificação (relacionamento entre especificação e implementação).

**Especificação.** Especificação é definida como o que se pretende do comportamento do projeto de hardware. Os formalismos mais utilizados são os baseados em Lógica e Teoria de Autômatos. Nos formalismos para verificação formal, as propriedades a serem verificadas podem ser divididas em três classes: propriedades de corretude funcional; propriedade de *liveness* e *safety*; e propriedades de tempo.

Para cada propriedade existe um formalismo que melhor a caracteriza, e.g., a propriedades de *liveness* é melhor caracterizada pela lógica temporal do que por qualquer outra lógica. Este é um dos aspectos considerados na decisão por determinado formalismo: “Que propriedades determinado formalismo pode expressar?”

Outro aspecto é “Que tipo de abstração um formalismo pode alcançar?” Dentre os tipos de abstração, podemos citar: estrutural, comportamental, de dados e temporal.

A utilidade prática de um formalismo está diretamente relacionada com a capacidade de aliar as abstrações e propriedades à especificação do sistema que se deseja verificar a corretude.

**Implementação.** Implementação consiste de uma descrição do projeto de hardware a ser verificado. A linguagem de representação pode modelar o circuito em diferentes níveis de abstração: a nível de circuito, a nível de *switch*, a nível de porta e a nível de transferência de registros; esta sequência está em ordem decrescente de grau de abstração. Então, uma das principais questões é “Qual o nível de abstração escolhido para representar o hardware?”

Escolher uma determinada representação para a implementação pode não só significar que classes de aplicações podemos utilizar, dependendo do grau de abstração; mas também, pode implicar em restrições nos limites de performance. Isto deve-se ao fato da representação poder impôr restrições ao projeto!

**Verificação.** Verificação é o relacionamento entre implementação e especificação, e consiste em provar que uma implementação satisfaz uma especificação. A relação de satisfação pode ser formalizada de diferentes formas:

- Prova de teoremas;
- *Model-Checking*;
- Comparação de equivalências (equivalências de funções, de autômatos, etc...);
- Continência de Linguagens ( $L(\text{Imp})$  está contida em  $L(\text{Esp})$ ).

O método de prova adotado pode ser avaliado por vários critérios:

- Natureza do relacionamento (equivalência, implicação, continência semântica);
- Completude e corretude;
- Grau de automação;
- Complexidade computacional;
- Tratamento de provas (por composição, por hierarquia, por indução, ...)

Dentre as possibilidades, observamos a grande variedade de métodos de verificação formal disponíveis. quando combinamos os vários graus de abstração, os tipos de formalismos de especificação, as propriedades desejadas e os métodos de prova, temos uma grande variedade de possibilidades. Isto dificulta a escolha de um. Contudo, algumas questões práticas devem ser consideradas numa aplicação típica:

- Que tipo de representação (implementação e especificação) fornece um bom compromisso entre expressividade e eficiência?
- É possível introduzir inconsistências num sistema de provas e assim violar as exigências de validação?
- Em que nível um método de prova deveria/pode ser automático?
- O método admite especificações executáveis?
- O que o método fornece como ajuda para correções, *debug*, revisão e modificação de projeto?

## 4 Técnicas de Verificação de Hardware

As técnicas são apresentadas no *survey* segundo a classificação dos formalismos de especificação. Desta forma, são divididas em três categorias: lógica, teoria de linguagens e autômatos e híbridos; conforme como as sentenças são expressas: através de sentenças lógicas, na forma de uma linguagem ou autômato ou usando os dois formalismos, respectivamente.

As técnicas são apresentadas no *survey* com a seguinte estrutura:

- Lógica
  - Lógica de Primeira Ordem
  - Lógica Computacional de Boyer-Moore
  - Lógica de Alta Ordem
  - Lógica Temporal
    - \* LTTL - *Linear Time Temporal Logic*
    - \* BTTL - *Branching Time Temporal Logic*
    - \* CTL - *Computacional Temporal Logic*
    - \* *Model-Checking with Logic*
    - \* ITL - *Interval Temporal Logic*
  - ETL - *Extended Temporal Logic*
  - *Mu-Calculus*
  - Abordagens Funcionais
- Autômatos
  - *Machine Equivalence*
  - *Language Containment*
  - *Trace Conformation*
- Híbridos
  - LTTL e Autômatos de Estados Finitos
  - Lógica Temporal e Autômato de Büchi

### 4.1 Lógica

Na Lógica formal, uma linguagem formal e um sistema formal são usados para representar as sentenças, e métodos lógicos para garantir a veracidade das sentenças.

A linguagem formal é formada por um conjunto de fórmulas bem-formadas sintaticamente por um conjunto de símbolos (alfabeto) e um conjunto de regras de formação. Um sistema formal consiste de uma linguagem formal e um método de dedução; este último,

por sua vez, utiliza-se de axiomas e regras de inferência para checar se uma dada sentença é bem-formada dentro do sistema formal.

A semântica de uma linguagem formal é especificada através de uma interpretação que atribue significado aos símbolos e fórmulas.

Os diferentes tipos de lógica vêm das diferentes regras sintáticas e semânticas associadas à linguagem formal e aos sistemas formais. Tipicamente, uma especificação é representada por uma fórmula na lógica que se está usando, enquanto a implementação é representada ou como uma fórmula ou como um modelo semântica.

Na especificação, a verificação é realizada por um provador de teoremas. Na implementação, tanto provador de teoremas quanto *model-checking* podem ser usados para verificação.

#### 4.1.1 Lógica de Predicados de Primeira Ordem

Simplicidade e equivalência com sistemas de hardware são as principais motivações para o uso desta lógica. Como os sistemas de hardware trabalham com lógica Booleana, a lógica de primeira ordem traduz bem tais sistemas. Nesta lógica apenas as variáveis de domínio podem ser quantificadas, seus sub-conjuntos não.

Dentre as técnicas de simulação de hardware que utilizam lógica de primeira ordem, podemos citar:

- *Machine identification*, esta técnica baseia-se em provar que o comportamento de um sistema sequencial é equivalente à de uma máquina de estados finitos conhecida. Este problema é NP-difícil.
- Lógica ternária reduz o número de estados, o que torna a simulação de uma N-bit RAM  $O(N \log N)$ .
- Simulação simbólica.

A principal desvantagem é a sua natureza não-hierarquica; e tem como vantagens: manipulação booleana eficiente e um poderoso modelo para simulação já que possibilita tratar efeitos de baixo nível.

#### 4.1.2 Lógica Computacional de Boyer-Moore

Esta é uma lógica mais restritiva que a de primeira ordem, pois é livre de quantificadores e possui a propriedade de igualdade. Sua sintaxe utiliza a notação infixada, os termos são variáveis e funções.

A lógica é fundamentada no *Principle of Definition* que diz que toda nova função é definida ou não-recursivamente em termos de funções pré-definidas ou recursivamente via regras bem-fundadas (caso base).

As regras de inferência são baseadas no princípio da indução. Com isso, o mecanismo de prova não é completamente automático, necessitando de intervenções do usuário.

Esta lógica apresenta expressividade limitada porque é baseada na lógica de primeira ordem, por exemplo, funções de tempo não podem ser representadas nesta lógica.

### 4.1.3 Lógica de Alta Ordem

Na lógica de alta ordem, os sub-conjuntos das variáveis podem ser quantificados, o que não ocorre nas lógicas de primeira ordem. Daí vem o grande poder de expressão desta lógica.

Uma outra diferença em relação à lógica de primeira ordem é que lógica de alta ordem possibilita que funções podem ser resultados e argumentos de outras funções.

Por outro lado, verificação em tal lógica é mais complicado que na lógica de primeira ordem. Sobretudo, garantir completude e corretude das sentenças lógicas num sistema de provas é bastante difícil pois tal lógica apresenta incompletudes e inconsistências se o sistema semântico não for cuidadosamente bem definido.

### 4.1.4 Lógica Temporal

Esta lógica acrescenta dinâmica às sentenças lógicas, ou seja, são predicados lógicos cujas mudanças temporais são representadas por esta lógica. A lógica temporal tem a capacidade de expressar as sentenças lógicas e como estas variam no tempo.

Especificações em lógica temporal foram primeiramente aplicadas a sistemas concorrentes para a verificação de corretude de tais sistemas. Propriedades como *safety*, *liveness*, exclusão mútua e precedência de eventos são trivialmente representadas na lógica temporal. Por isso sua grande utilização em sistemas para especificação e verificação de hardware.

Contudo, lógica temporal não é bem recomendada para verificação funcional, já que a lógica de predicados é tão expressiva funcionalmente quanto ela e de manipulação mais simples.

O *survey* apresenta uma classificação das lógicas temporais, as quais listaremos abaixo.

**LTTL** - *Linear Time Temporal Logic* .

**BTTL** - *Branching Time Temporal Logic* .

**CTL** - *Computational Temporal Logic* . Sub-linguagem da BTTL

*Model-Checking with Logic* .

**ITL** - *Interval Temporal Logic* .

### 4.1.5 Lógica Temporal Estendida

A extensão da lógica temporal é necessária pois apesar de representar muito bem propriedades de tempo das sentenças não conseguem expressar expressões regulares numa sequência de execução, como por exemplo, o fecho de Kleene de sentenças lógicas.

A determinação de satisfabilidade é obtida através de um procedimento em tempo exponencial, assim como corretude e completude são obtidas com o acréscimo de axiomas para os operadores da gramática que representa a lógica.

Através de um procedimento de simulação é mostrado que esta lógica tem o mesmo poder de expressão dos autômatos de Büchi.

#### 4.1.6 Cálculo *Mu*

Cálculo Mu fornece um outro mecanismo para a extensão do poder de expressão da lógica temporal. Definido independentemente da lógica temporal, este cálculo apresenta a vantagem de possibilitar um algoritmo de *model-checking* com complexidade de tempo polinomial para a verificação de um conjunto restrito do cálculo, o qual é suficiente para representar as lógicas CTL.

#### 4.1.7 Abordagens Funcionais

As abordagens apresentadas representam as especificações como linguagens específicas e não como um formalismo lógico, como até então apresentado no *survey*. Tanto as especificações quanto as implementações são representadas como funções na linguagem.

Provas nestas abordagens são realizadas através de um provador de teoremas, como, por exemplo, OBJ. Basicamente, descrições comportamentais (especificação) e estruturais (implementação) em VHDL são convertidas em equações funcionais, e então é verificado se o conjunto de equações da especificação é equivalente ou implica logicamente no conjunto de implementação.

### 4.2 Autômatos

Nesta técnica, autômatos e equivalência entre linguagens são utilizados para representar e verificar uma especificação. Autômatos podem ser caracterizados como reconhecedores e aceitadores de linguagens, e é exatamente neste aspecto que estas técnicas são baseadas. O *survey* apresenta algumas variações da utilização desta técnica para a verificação formal de hardware, as quais citamos a seguir.

#### 4.2.1 *Machine Equivalence*

Nesta abordagem, tanto a especificação quanto a implementação são representadas como autômatos de estado finito. A verificação de que a implementação satisfaz a especificação é realizada mostrando que o comportamento do autômatos obedece a relação de satisfatibilidade, para isso é provado que há uma correspondência entre as máquinas.

Claro que dependendo do grau de correspondência a ser provado (homomorfismo, isomorfismo) a complexidade aumenta. Obviamente, o problema se torna intratável se o número de estados é muito grande, uma vez que a verificação é exaustiva.

Por outro lado, o *survey* apresenta duas abordagens que tornam a solução para o problema mais eficiente, isto usando algoritmos de busca baseados em DFS e BFS com representação simbólica.

#### 4.2.2 *Language Containment*

Observando, mais genericamente, as linguagens ao invés dos autômatos, pode-se mostrar a verificação via uma relação de continência. Ou seja, mostrando que a linguagem que descreve a implementação está propriamente contida na linguagem que descreve a especificação. Esta abordagem torna a representação de especificações mais abstratas, facilitando

a modelagem do sistema que se deseja verificar. Sendo esta uma de suas principais vantagens.

Uma outra vantagem é a possibilidade de utilização de mecanismos como “redução” para tratar o problema da explosão do número de estados, quando da representação equivalente em autômatos. Contudo, o uso desta técnica na prática é limitada devido aos mecanismos de redução nem sempre fornecer a melhor representação.

#### 4.2.3 *Trace Conformation*

Basicamente, a teoria de “traces” utiliza sequências de eventos para representar os circuitos. Para determinadas entradas, sequências de eventos são especificadas. Esta técnica é comumente utilizada em circuitos *speed-independent* por representar os eventos sem a preocupação com delays e outros atrasos.

A verificação formal destes formalismos é baseada no conceito de *safe substitution*, ou seja, uma estrutura “trace” (implementação) está correta se podemos substituí-la por sua especificação e a corretude for preservada.

A principal vantagem desta técnica é a possibilidade de verificação hierárquica. A principal desvantagem é a necessidade operacional de se construir grafos para representar as estruturas “trace”.

### 4.3 Híbridos

Os formalismos híbridos utilizam-se das vantagens de ambas as técnicas (lógica e autômatos) para representar especificações e implementações. Como a lógica é naturalmente mais versátil para representar especificações e a teoria de linguagens/autômatos possui algoritmos consolidados para dedução e verificação, os formalismos híbridos unem estas duas características e utilizam lógica para especificar e teoria de autômatos para análise e verificação.

Dentre os formalismos híbridos o *survey* apresenta dois.

- LTTL e Autômato de Estado Finito.
- Lógica Temporal e Autômato de Büchi.

No primeiro, a especificação lógica é convertida num autômato equivalente e então a verificação segue no domínio da teoria de linguagens/autômatos. No segundo, na direção oposta: transforma o autômato implementação numa representação lógica, e a verificação segue no domínio da lógica e dos provadores de teorema.

## 5 Overview das Técnicas de Verificação

As técnicas apresentadas estão dispostas em três eixos, representados graficamente na Figura 1.

- No eixo da implementação temos os seguintes formalismos para descrever o hardware:



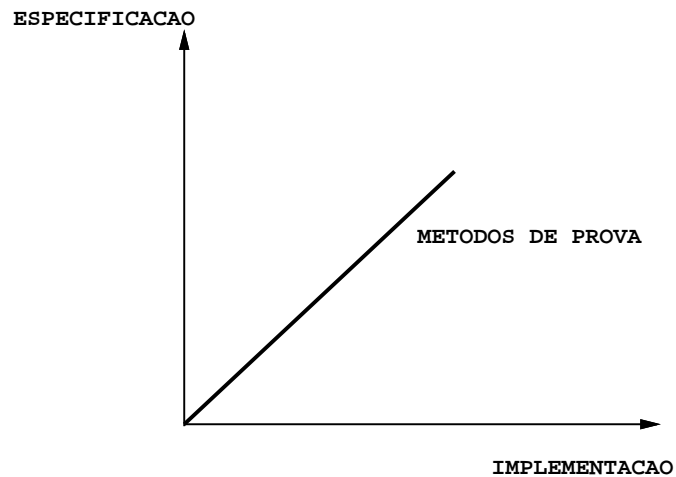


Figura 1: Três Eixos da Verificação Formal

- Lógica
  - Grafos
  - Autômatos
  - Traces
- No eixo da especificação temos os seguintes formalismos:
    - Lógica
    - Lógica Temporal
    - Grafos
    - Autômatos
    - traces
  - E finalmente, no eixo dos métodos de prova temos:
    - Provedores de Teorema
    - *Model-Checking*
    - *Machine Equivalence*
    - *Language Containment/Equivalence*
    - *Trace Conformation*

Contudo, nem todas as 100 combinações foram exploradas. O autor do *survey* ressalta que algumas apontam para uma direção não muito frutíferas e que outras simplesmente não foram pesquisadas.

## 6 Conclusões

Observamos pelo *survey* que a área de verificação formal de hardware tem muito o que contribuir, já que os sistemas estão cada vez mais complexos e quanto mais tarde os erros são descobertos mais oneroso se torna o processo de correção. Desta forma, a verificação formal durante o projeto, especificação e implementação de sistemas de hardware se tornará um fato consolidado.

Observamos também, que apesar de algumas das técnicas serem utilizadas já na indústria intensamente, isso não inviabiliza as demais, já que cada técnica diferencia-se das demais por fatores como expressividade, facilidade de prova, facilidade de automação, grau de abstração alcançado, entre outros. Estes fatores são decisivos durante a escolha do formalismo a ser utilizado e o que se deseja deste formalismo, pois determinados sistemas exigem características específicas que só um determinado formalismo fornece.

## 7 Bibliografia

[Gupta] Gupta, Aarti.

Formal Hardware Verification Methods: A Survey

School of Computer Science. CMU. Pittsburgh, Pennsylvania 15213.