

Model Checking

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

25 de novembro de 2024

Conteúdo



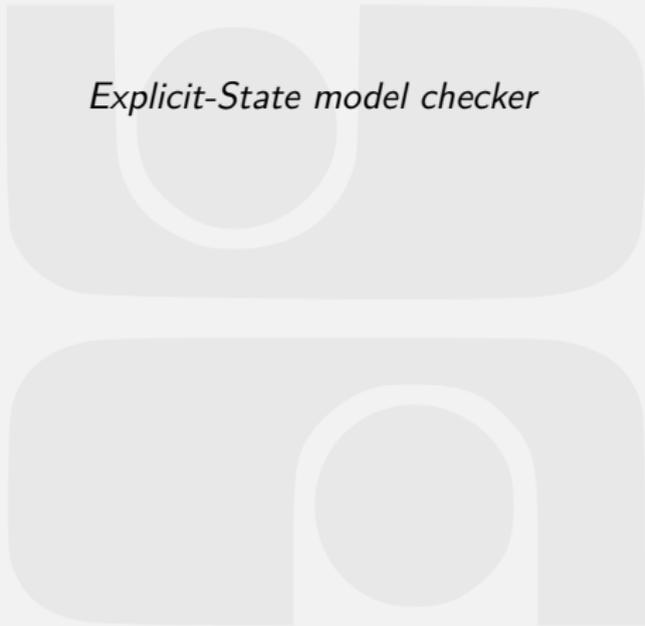
Model Checking

Outline

A large, light gray watermark logo is centered on the slide. It consists of a stylized letter 'M' with a circular element in the center. The text 'Model Checking' is overlaid on the left side of the 'M' in a dark gray, italicized font.

Model Checking

TLC (LAMPORT, 2002)



Explicit-State model checker

TLC (LAMPORT, 2002)

Explicit-State model checker

- Enumera todos os estados

TLC (LAMPORT, 2002)

Explicit-State model checker

- Enumera todos os estados
- Usa um grafo direcionado com estados e transições (*reachability graph*) e uma fila de estados para checar

TLC (LAMPORT, 2002)

Explicit-State model checker

- Enumera todos os estados
- Usa um grafo direcionado com estados e transições (*reachability graph*) e uma fila de estados para checar
- Provavelmente, o tipo de *model checker* que um de nós escreveria se fôssemos tentar (sem pesquisar um monte antes)

Apache (KONNOV; KUKOVEC; TRAN, 2019)

Model checker simbólico



Apache (KONNOV; KUKOVEC; TRAN, 2019)

Model checker simbólico

- Traduz a especificação para um conjunto de restrições (sem quantificação)

Apache (KONNOV; KUKOVEC; TRAN, 2019)

Model checker simbólico

- Traduz a especificação para um conjunto de restrições (sem quantificação)
- Usa um SMT *solver* para verificar a satisfabilidade das restrições

Apache (KONNOV; KUKOVEC; TRAN, 2019)

Model checker simbólico

- Traduz a especificação para um conjunto de restrições (sem quantificação)
- Usa um SMT *solver* para verificar a satisfabilidade das restrições

Bounded model checking

Apache (KONNOV; KUKOVEC; TRAN, 2019)

Model checker simbólico

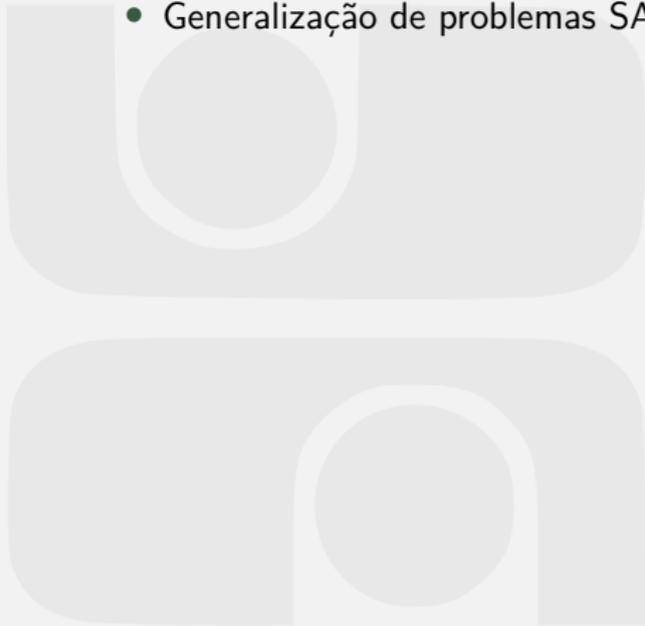
- Traduz a especificação para um conjunto de restrições (sem quantificação)
- Usa um SMT *solver* para verificar a satisfabilidade das restrições

Bounded model checking

- Necessário definir um número máximo de passos (*bound*) para o qual gerar restrições. A verificação se dá dentro desse limite, podendo haver estados atingíveis fora desse limite que não satisfazem a propriedade.

SMT

- Generalização de problemas SAT



SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana

SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana
 - SMT: Satisfabilidade “*modulo*” teorias

SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana
 - SMT: Satisfabilidade “*modulo*” teorias
 - “*modulo*” no sentido de “dentro de”

SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana
 - SMT: Satisfabilidade “*modulo*” teorias
 - “*modulo*” no sentido de “dentro de”
 - Permite mais elementos: inteiros, reais, arrays, etc

SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana
 - SMT: Satisfabilidade “*modulo*” teorias
 - “*modulo*” no sentido de “dentro de”
 - Permite mais elementos: inteiros, reais, arrays, etc

SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana
 - SMT: Satisfabilidade “*modulo*” teorias
 - “*modulo*” no sentido de “dentro de”
 - Permite mais elementos: inteiros, reais, arrays, etc

Z3 (DE MOURA; BJØRNER, 2008):

- Leonardo de Moura e Nikolaj Bjørner
 - Podcast: Type Theory Forall #33 Z3 and Lean, the Spiritual Journey

SMT

- Generalização de problemas SAT
 - SAT: Problema da satisfabilidade Booleana
 - SMT: Satisfabilidade “*modulo*” teorias
 - “*modulo*” no sentido de “dentro de”
 - Permite mais elementos: inteiros, reais, arrays, etc

Z3 (DE MOURA; BJØRNER, 2008):

- Leonardo de Moura e Nikolaj Bjørner
 - Podcast: Type Theory Forall #33 Z3 and Lean, the Spiritual Journey
- Versão em WebAssembly
 - <https://people.csail.mit.edu/cpitcla/z3.wasm/z3.html>

Resolvendo SMT com Z3

```
1 (declare-fun p () Bool)
2 (declare-fun q () Bool)
3 (assert (implies p q))
4 (assert p)
5 (assert (not q))
6 (check-sat)
```

Resultado: **unsat**

Resolvendo SMT com Z3

```
1 (declare-fun p () Bool)
2 (declare-fun q () Bool)
3 (assert (implies p q))
4 (assert p)
5 (assert (not q))
6 (check-sat)
```

Resultado: **unsat**

```
1 (declare-fun p () Bool)
2 (declare-fun q () Bool)
3 (assert (implies p q))
4 (assert p)
5 (assert q)
6 (check-sat)
```

Resultado: **sat**

Resolvendo SMT com inteiros

```
1 (declare-const a Int)
2 (declare-const b Int)
3 (assert (= (+ a b) 20))
4 (assert (= (+ a (* 2 b)) 10))
5 (check-sat)
6 (get-model)
```

Resultado: **sat**

Simulação simbólica

Podemos classificar não-determinismo em dois tipos:



Simulação simbólica

Podemos classificar não-determinismo em dois tipos:

- Não-determinismo de dados (*data non-determinism*)
 - `1.to(10).oneOf()`

Simulação simbólica

Podemos classificar não-determinismo em dois tipos:

- Não-determinismo de dados (*data non-determinism*)
 - `1.to(10).oneOf()`
- Não-determinismo de controle (*control-flow non-determinism*)
 - `any { A, B }`

Simulação simbólica

Podemos classificar não-determinismo em dois tipos:

- Não-determinismo de dados (*data non-determinism*)
 - `1.to(10).oneOf()`
- Não-determinismo de controle (*control-flow non-determinism*)
 - `any { A, B }`

Simulação simbólica:

Simulação simbólica

Podemos classificar não-determinismo em dois tipos:

- Não-determinismo de dados (*data non-determinism*)
 - `1.to(10).oneOf()`
- Não-determinismo de controle (*control-flow non-determinism*)
 - `any { A, B }`

Simulação simbólica:

- *Model Checkers*: exaustivamente checam todos os dados e fluxos possíveis

Simulação simbólica

Podemos classificar não-determinismo em dois tipos:

- Não-determinismo de dados (*data non-determinism*)
 - `1.to(10).oneOf()`
- Não-determinismo de controle (*control-flow non-determinism*)
 - `any { A, B }`

Simulação simbólica:

- *Model Checkers*: exaustivamente checam todos os dados e fluxos possíveis
- Simuladores aleatórios: aleatoriamente checam alguns dados e fluxos

Simulação simbólica

Podemos classificar não-determinismo em dois tipos:

- Não-determinismo de dados (*data non-determinism*)
 - `1.to(10).oneOf()`
- Não-determinismo de controle (*control-flow non-determinism*)
 - `any { A, B }`

Simulação simbólica:

- *Model Checkers*: exaustivamente checam todos os dados e fluxos possíveis
- Simuladores aleatórios: aleatoriamente checam alguns dados e fluxos
- Simulador simbólico: aleatoriamente checa alguns fluxos, mas considerando todos os dados possíveis

Referências

DE MOURA, L.; BJØRNER, N. **Z3: an efficient smt solver** Proceedings of the theory and practice of software, 14th international conference on tools and algorithms for the construction and analysis of systems.

Anais.: Tacas'08/etaps'08. Budapest, Hungary: Springer-Verlag, 2008

KONNOV, I.; KUKOVEC, J.; TRAN, T.-H. Tla+ model checking made symbolic. **Proc. acm program. lang.**, v. 3, n. OOPSLA, Oct. 2019.

LAMPORT, L. **Specifying systems: The tla+ language and tools for hardware and software engineers.** Boston: Addison-Wesley, 2002.

Model Checking

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

25 de novembro de 2024