TLA+ avançado

Aula para disciplina de Métodos Formais

Gabriela Moreira

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

02 de julho de 2025

Conteúdo

Outline

Model values

Segundo a documentação em (CENTRE, [s.d.]):

- Podem ser usados quando vamos definir os valores de uma constante

Model values

Segundo a documentação em (CENTRE, [s.d.]):

- Podem ser usados quando vamos definir os valores de uma constante
- Ao invés de indentificar processos, semáforos, etc por identificadores com `{1, 2, 3}` ou `{"p1", "p2", "p3"}`, podemos model values: `{p1, p2, p3}` (sem aspas).

Model values

Segundo a documentação em (CENTRE, [s.d.]):

- Podem ser usados quando vamos definir os valores de uma constante
- Ao invés de indentificar processos, semáforos, etc por identificadores com `{1, 2, 3}` ou `{"p1", "p2", "p3"}`, podemos model values: `{p1, p2, p3}` (sem aspas).

Um *model value* e um valor não especificado que o *model checker* considera diferente de qualquer outro valor que possa ser expressado:

- `p1 = p1`
- `p1 # p2`
- `p1 # 1`
- `p1 # "p1"`

Model values - pra que serve?

Evitar erros:

Model values - pra que serve?

Evitar erros:

- operações não intencionais sobre os identificadores, i.e. `p + 1`

Model values - pra que serve?

Evitar erros:

- operações não intencionais sobre os identificadores, i.e. `p + 1`
- evitar erros de digitação em strings (assim como tipos soma em quint)

Model values - pra que serve?

Evitar erros:

- operações não intencionais sobre os identificadores, i.e. `p + 1`
- evitar erros de digitação em strings (assim como tipos soma em quint)

Mesmo no TLC, existe possibilidade de dar tipos para esses valores adicionando uma letra e `_` no começo:

- `P_1`, `P_2`, ...
- `a_1`, `a_2`, ...

Model values - pra que serve?

Evitar erros:

- operações não intencionais sobre os identificadores, i.e. `p + 1`
- evitar erros de digitação em strings (assim como tipos soma em quint)

Mesmo no TLC, existe possibilidade de dar tipos para esses valores adicionando uma letra e `_` no começo:

- `P_1`, `P_2`, ...
- `a_1`, `a_2`, ...

Dessa forma, o TLC reporta um erro quando comparamos *model values* de tipos diferentes (como se estivéssemos comparando `1` e `"a"`):

- `P_1 = P_2` resulta em `FALSE`
- `P_1 = a_1` resulta em erro em runtime

Outline

Conjuntos de Simetria (*symmetry sets*)

Explicação em (WAYNE, 2022)

Podemos dizer para o *model checker* que um conjunto é simétrico, fazendo com que ele precise considerar menos estados.

Conjuntos de Simetria (*symmetry sets*)

Explicação em (WAYNE, 2022)

Podemos dizer para o *model checker* que um conjunto é simétrico, fazendo com que ele precise considerar menos estados.

Suponha que nossa especificação constrói uma sequência a partir do conjunto de *model values* `{s1, s2, s3, s4}`. Imagine alguns dos valores que podem ser construídos:

```
1 (1) <<s1, s2, s3>>
2 (2) <<s2, s1, s3>>
3 (3) <<s1, s2, s2>>
4 (4) <<s2, s3, s3>>
```

Conjuntos de Simetria (*symmetry sets*)

Explicação em (WAYNE, 2022)

Podemos dizer para o *model checker* que um conjunto é simétrico, fazendo com que ele precise considerar menos estados.

Suponha que nossa especificação constrói uma sequência a partir do conjunto de *model values* `{s1, s2, s3, s4}`. Imagine alguns dos valores que podem ser construídos:

```
(1) <<s1, s2, s3>>
(2) <<s2, s1, s3>>
(3) <<s1, s2, s2>>
(4) <<s2, s3, s3>>
```

- A única diferença do (1) pro (2) é que trocamos a ordem de `s1` e `s2`.
- A única diferença do (3) pro (4) é que trocamos os `s1` por `s2` e os `s2` por `s3`.

Conjuntos de Simetria para otimização

Conjuntos de Simetria para otimização

Se essas diferenças não forem relevantes para nosso modelo, podemos definir que `{s1, s2, s3, s4}` é um conjunto de simetria, e assim o *model checker* vai precisar verificar bem menos estados.

Conjuntos de Simetria para otimização

Se essas diferenças não forem relevantes para nosso modelo, podemos definir que `{s1, s2, s3, s4}` é um conjunto de simetria, e assim o *model checker* vai precisar verificar bem menos estados.

- Para isso, o TLC vai primeiro computar todas as permutações possíveis desse conjunto, o que pode levar um tempo para conjuntos grandes. Nesses casos, pode valer a pena usar um conjunto normal (sem simetria).

Conjuntos de Simetria para otimização

Se essas diferenças não forem relevantes para nosso modelo, podemos definir que `{s1, s2, s3, s4}` é um conjunto de simetria, e assim o *model checker* vai precisar verificar bem menos estados.

- Para isso, o TLC vai primeiro computar todas as permutações possíveis desse conjunto, o que pode levar um tempo para conjuntos grandes. Nesses casos, pode valer a pena usar um conjunto normal (sem simetria).

Exemplo no trabalho de RPG: dois monstros são simétricos

Conjuntos de Simetria para otimização

Se essas diferenças não forem relevantes para nosso modelo, podemos definir que `{s1, s2, s3, s4}` é um conjunto de simetria, e assim o *model checker* vai precisar verificar bem menos estados.

- Para isso, o TLC vai primeiro computar todas as permutações possíveis desse conjunto, o que pode levar um tempo para conjuntos grandes. Nesses casos, pode valer a pena usar um conjunto normal (sem simetria).

Exemplo no trabalho de RPG: dois monstros são simétricos

Exemplo no trabalho do banco: Usuários são simétricos

Outline

Refinamento

Vamos ver dois exemplos:

1. Transaction commit de Leslie Lamport (GRAY; LAMPORT, 2004)
2. Threads de Hillel Wayne (WAYNE, 2021)

FIM

TLA+ avançado

Aula para disciplina de Métodos Formais

Gabriela Moreira

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

02 de julho de 2025