

Tradução automática de especificação formal modelada em TLA⁺ para linguagem de programação

Gabriela Moreira Mafra

Universidade do Estado de Santa Catarina

gabrielamoreiramafra@gmail.com

22 de Novembro de 2019

Fundamentos

TLA⁺

Gerador de código

Considerações

Especificação Formal

Especificar Software

- Permite verificações e fornece garantias
- Serve como base para consulta
- Mais fácil de modificar do que a implementação
- Vem antes do desenvolvimento, na etapa de desenho

Sistemas Concorrentes

Muitas ordens (comportamentos) possíveis.

Difícil prever todas as situações e ainda mais relevante especificar.

Sistemas Concorrentes

Um sistema é concorrente quando há mais de uma computação concorrendo pelo mesmo recurso.

O resultado pode depender da ordem em que essas computações conseguem os recursos.

- Muitas ordens possíveis
- Muitos comportamentos possíveis

Especificar pode ser ainda mais relevante.

Geração de código

A implementação - ou a tradução da especificação em linguagem de programação - pode ser feita por um programador ou um tradutor automático.

Problemas da tradução manual:

- Suscetível a erro - causando a perda de propriedades verificadas.
- Custosa

Z, B-Method e ASM (*Abstract State Machine*) tem geradores de código.

Temporal Logic of Actions⁺

Linguagem de especificação baseada na lógica TLA.

- Sintaxe matemática
- Ideal para especificar sistemas concorrentes

Não é disponibilizado um gerador de código.

Temporal Logic of Actions⁺

Linguagem de especificação baseada na lógica TLA.

- Sintaxe matemática
- Ideal para especificar sistemas concorrentes

Não é disponibilizado um gerador de código.

Objetivo

Elaborar um método para gerar código a partir de especificações em TLA⁺

- Encontrar mapeamentos
- Implementar um tradutor, aplicando os mapeamentos e gerando código Elixir

TLA

TLA = Lógica. TLA⁺ = linguagem de especificação.

Ação

Fórmula sobre um passo. Passo = dupla de estados.

Permite definir quais transições são permitidas.

Comportamento

Sequência de passos. Representa uma execução no sistema.

Uma fórmula é verdadeira para um comportamento se ela é verdadeira para cada passo dele.

Tradução

O método de tradução é descrito através de **mapeamentos**. Os primeiros mapeamentos definidos foram:

- Ações \mapsto Funções
- Estado \mapsto Hash mapeando as variáveis aos seus valores
- Ação \vee Ação \mapsto **Disparo de novo processo**

Ação \vee Ação

Proposta inicial para tradução da disjunção

```
def main(variaveis) do
  spawn_link JarrosDeAgua, :main, [grande_para_pequeno(variaveis)]
  spawn_link JarrosDeAgua, :main, [pequeno_para_grande(variaveis)]
  spawn_link JarrosDeAgua, :main, [esvazia_grande(variaveis)]
  spawn_link JarrosDeAgua, :main, [esvazia_pequeno(variaveis)]
  spawn_link JarrosDeAgua, :main, [enche_grande(variaveis)]
  spawn_link JarrosDeAgua, :main, [enche_pequeno(variaveis)]
  spawn_link JarrosDeAgua, :main, [variaveis]
end

JarrosDeAgua.main(%{ grande: 0, pequeno: 0 })
```

Condições e Ações

$$\text{Sacar}(qtd) \triangleq \begin{aligned} &\wedge \text{saldo} \geq qtd \\ &\wedge \text{saldo}' = \text{saldo} - qtd \end{aligned}$$

Condição de ativação: $\text{saldo} \geq qtd$

Ação: $\text{saldo}' = \text{saldo} - qtd$

Condições e Ações

$$\text{Sacar}(qtd) \triangleq \begin{aligned} &\wedge \text{saldo} \geq qtd \\ &\wedge \text{saldo}' = \text{saldo} - qtd \end{aligned}$$

Condição de ativação: $\text{saldo} \geq qtd$

Ação: $\text{saldo}' = \text{saldo} - qtd$

$$\text{Next} \triangleq \begin{aligned} &\vee \text{Sacar}(10) \\ &\vee \text{Sacar}(50) \end{aligned}$$

Se $\text{saldo} = 30$, é possível decidir a próxima ação.

Mas se $\text{saldo} = 60$, é necessária uma escolha (não determinismo)

Condições e Ações: Tradução

```
def sacar_condition(variaveis, qtd) do
  variaveis[:saldo] >= qtd
end

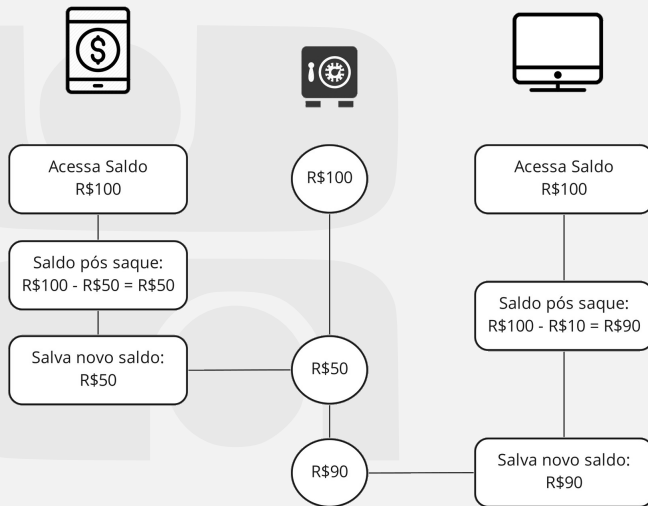
def sacar(variaveis, qtd) do
  %{
    saldo:  variaveis[:saldo] - qtd,
  }
end
```

Determinando a próxima ação

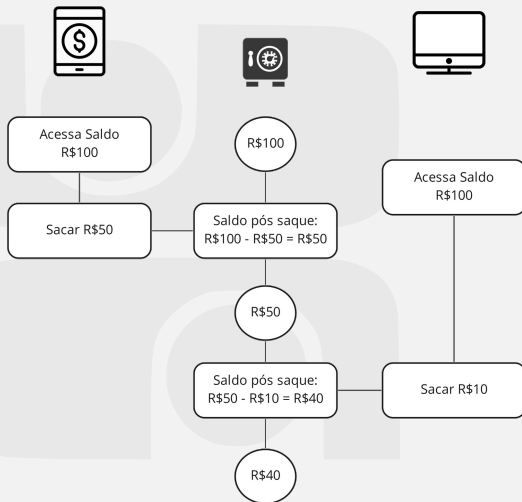
Algorithm 1 Decisão da próxima ação

```
1: procedure DECIDEAÇÃO(ações)
2:   acoes_possiveis  $\leftarrow$  ações com condição satisfeita
3:   estados_distintos  $\leftarrow$  estados únicos resultantes de
     acoes_possiveis
4:   if tamanho de estados_distintos = 1 then  $\triangleright$  Decisão pura
5:     return primeiro estado em estados_distintos
6:   else  $\triangleright$  Influência Externa
7:     envie identificações das acoes_possiveis para o oráculo
8:     acao_escolhida  $\leftarrow$  resposta do oráculo
9:     return estado resultante de acao_escolhida
```

Banco com problemas de concorrência



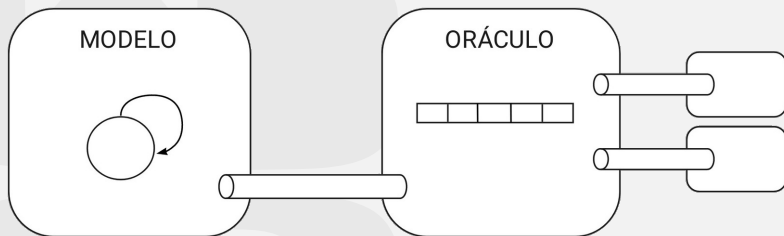
Banco modelado em TLA⁺



Modelos são
sequenciais.

Ordem das ações
pode ser não
determinística

Não determinismo centralizado em um oráculo



Quando o modelo não é capaz de decidir o próximo estado (como no caso de $saldo = 60$), é delegada uma escolha ao oráculo, que centraliza toda a influência externa no modelo.

Tradução

- 1 Estrutura do módulo e declarações
- 2 Para cada definição, suas condições e suas ações
- 3 Função de próximo estado: `main` em loop
- 4 Estado inicial: equações do predicado *Init* como atribuições
- 5 Comentários: documentação ou comentários comuns

Considerações

Principais contribuições

- 1 Levantamento e sumarização dos conceitos de TLA⁺, que apresenta valor didático e potencialmente divulga a linguagem de especificação.
- 2 Gerador de código funcional
 - Protótipos ou ponto de partida para nova aplicação
 - Escopo reduzido - apenas principais elementos foram mapeados

Trabalhos Futuros

- ① Diversas melhorias no tradutor
 - Permitir interrupções ou esperas em qualquer passo
 - Modularizar extensões
 - Ignorar definições não alcançáveis
- ② Permitir que valores não definidos de variáveis sejam escolhidos pelo oráculo
- ③ Traduzir definições de invariantes para testes ou assinaturas para o verificador de tipos.

Obrigada!

Fim :D

Gabriela Moreira Mafra

`gabrielamoreiramafra@gmail.com`

Repositório:

`https://github.com/GabrielaMafra/tla-transmutation`