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

Especificação Formal

Especificar Software é como fazer a planta de um edifício

- Permite verificações
- Serve como base para consulta
- Mais fácil de modificar do que o produto final
- Vem antes da produção

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**

$$\text{Ação} \vee \text{Ação}$$

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

$$Sacar(qtd) \triangleq \begin{array}{l} \land\ saldo \geq qtd \\ \land\ saldo' = saldo - qtd \end{array}$$

Condição de ativação: $saldo \geq qtd$
Ação: $saldo' = saldo - qtd$

Condições e Ações

$$Sacar(qtd) \triangleq \land\ saldo \geq qtd \\ \land\ saldo' = saldo - qtd$$

Condição de ativação: $saldo \geq qtd$
Ação: $saldo' = saldo - qtd$

$$Next \triangleq \lor\ Sacar(10) \\ \lor\ Sacar(50)$$

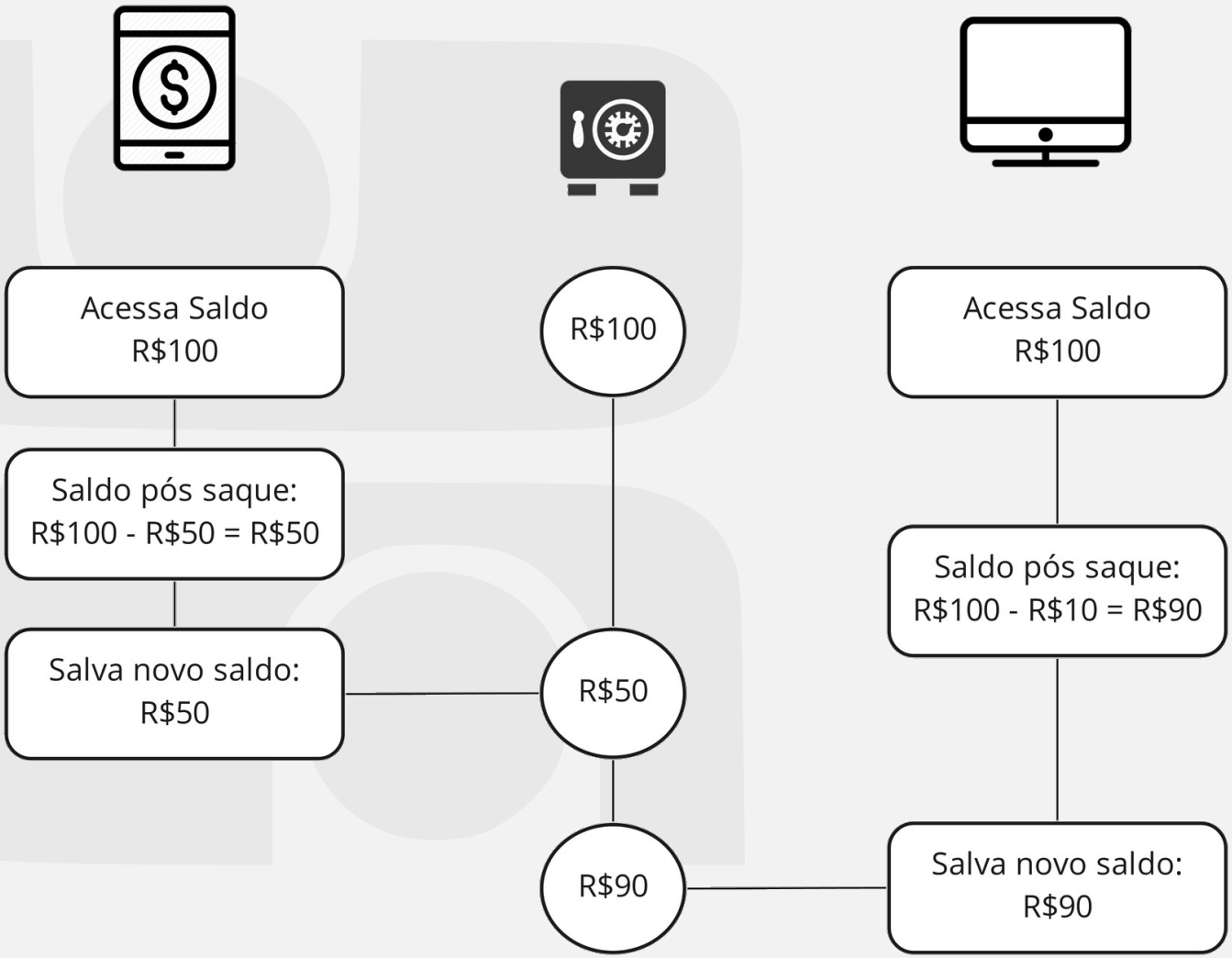
Se $saldo = 30$, é possível decidir a próxima ação.
Mas se $saldo = 60$, é necessária uma escolha (não determinismo)

Condições e Ações: Tradução

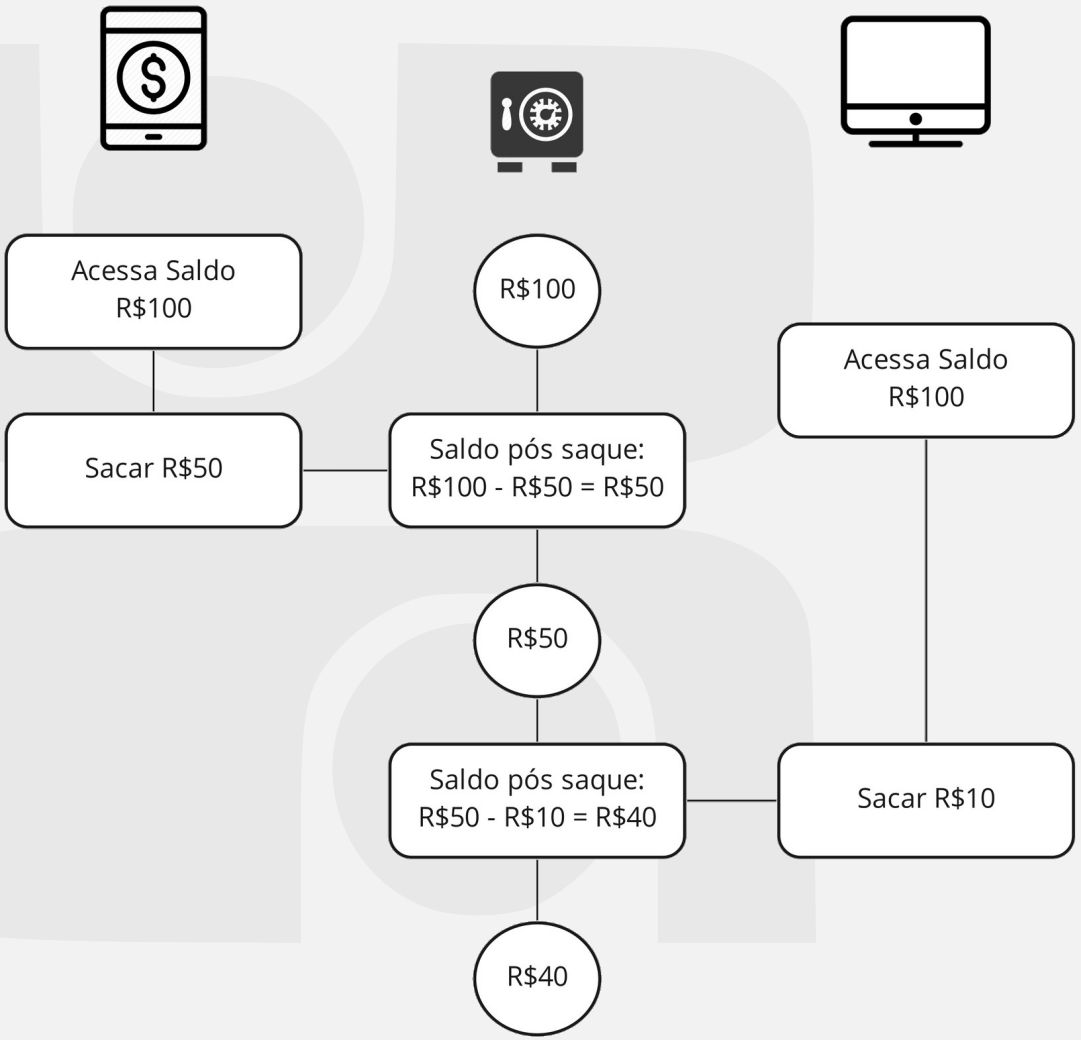
```
def sacar_condition(variaveis, qtd) do
  variaveis[:saldo] >= qtd
end

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

Banco com problemas de concorrência



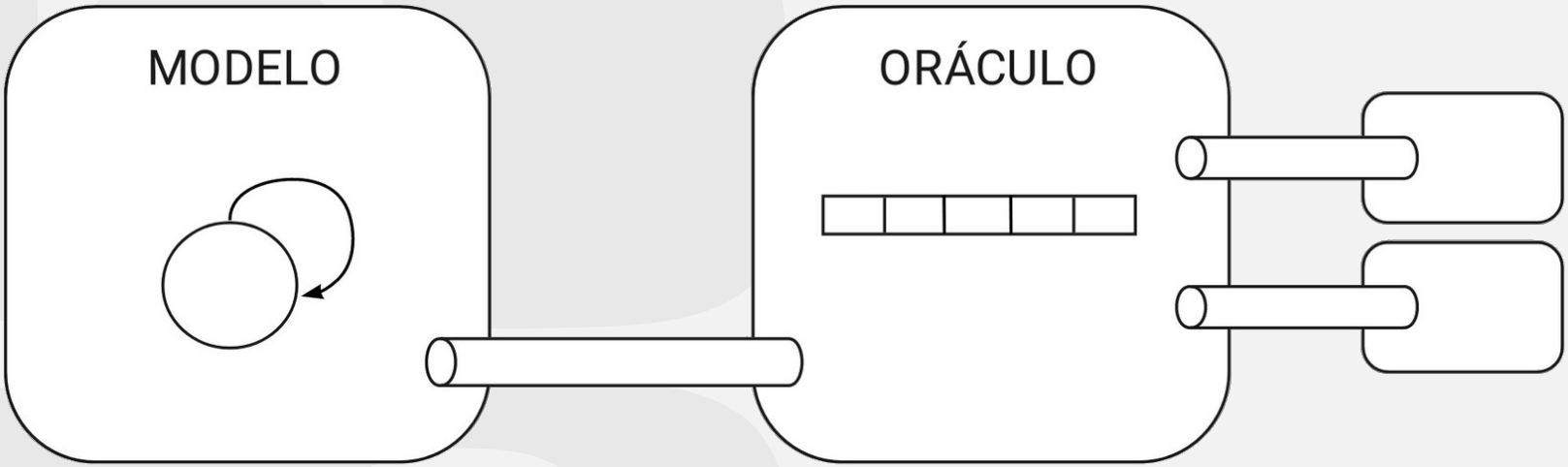
Banco modelado em TLA+



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.

Por que Elixir?

1. Concorrência: gera *bytecode* da máquina virtual do Erlang (BEAM).
2. Paradigma funcional: se aproxima de definições matemáticas, proporcionando um complexidade menor para as traduções.
3. Expressividade: código entendível, permitindo alta manutenabilidade.
4. Transparência de plataforma
5. Open Source

Considerações

Até aqui:

- Estudo dos construtores de TLA$^+$ e entendimento da lógica.
- Validação da estrutura inicial do código traduzido.
- Início da listagem de mapeamentos.

Próximos passos

A continuidade do trabalho será feita em duas frentes paralelas:

- Busca por mais mapeamentos
- Implementação do tradutor (em Haskell)

Próximos passos

A continuidade do trabalho será feita em duas frentes paralelas:

- Busca por mais mapeamentos
- Implementação do tradutor (em Haskell)

Exploração

- Fornecimento de garantias para os mapeamentos
- Gerar complementos para melhorar o ambiente de desenvolvimento (e.g. testes unitários)

Obrigada!

Fim :D

Gabriela Moreira Mafra

gabrielamoreiramafra@gmail.com