03. Análise de Comportamentos

Princípios de Engenharia de Software (Texto em Elaboração)

Italo S. Vega
italo@pucsp.br

Faculdade de Estudos Interdisciplinares (FACEI)

Pontifícia Universidade Católica de São Paulo

Sumário


D. Knuth — Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

3 Análise de Comportamentos

 — **Programas** de computador implementam modelos de **ações computacionais**.

3.1 Comportamento Válido

A análise de comportamentos baseados em especificações introduz uma importante ferramenta de programação.

 — Especificações lógicas auxiliam na análise de comportamentos.

Capturam-se afirmações a respeito de um comportamento nas regras lógicas de uma especificação. No caso de um código para intercambiar os valores entre duas variáveis, desejamos investigar a possibilidade do seguinte comportamento acontecer: $[a = 5, b = 3] \rightarrow [a = 3, b = 5]$. O predicado inicial "início" e a ação "troca_2" procuram especificar o comportamento desejado de troca:

$$\begin{aligned} \text{início} \triangleq & \wedge a = 5 \\ & \wedge b = 3 \\ \text{troca}_2 \triangleq & \wedge t' = a \\ & \wedge a' = b \\ & \wedge b' = t' \end{aligned}$$

3.1.1 Questão: efeito de um predicado inicial

Contexto Sabe-se que o predicado inicial "início" é verdadeiro em um particular raciocínio:

$$\begin{aligned} \text{início} \triangleq & \wedge a = 5 \\ & \wedge b = 3 \end{aligned}$$

Enunciado Assinale a alternativa contendo um **estado válido**:

1. $[a = 5, b = 3]$
2. $[a = 3, b = 5]$
3. $[a = 3, b = 3]$
4. $[a = 5, b = 5]$

3.1.2 Questão: efeito de uma ação

Contexto Considere a ação "troca_2":

$$\begin{aligned} \text{troca}_2 \triangleq & \wedge t' = a \\ & \wedge a' = b \\ & \wedge b' = t' \end{aligned}$$

Sabe-se que a ação "troca_2" é verdadeira em uma máquina que se encontra no **estado** $[a = 5, b = 3]$.

Enunciado Assinale a alternativa contendo o **estado seguinte** da máquina:

1. $[t = 5, a = 3, b = 3]$
2. $[t = 5, a = 3, b = 5]$
3. $[t = 3, a = 5, b = 3]$
4. $[t = 3, a = 5, b = 5]$

3.1.3 Questão: estados e subestados

Contexto Sabe-se que uma máquina encontra-se no estado $[t = 5, a = 3, b = 5]$.

Enunciado Assinale a alternativa contendo um **subestado verdadeiro**:

1. $[t = 3]$
2. $[a = 3, b = 5]$
3. $[b = 3]$
4. $[a = 5]$

O Profe utiliza um formato parecido com aquele de Fitch[1] para **representar** um raciocínio sobre um comportamento de máquina:

$$\frac{[a = 5, b = 3]}{[a = 3, b = 5]}$$

Nessa representação, afirma-se que o estado $[a = 5, b = 3]$ é verdadeiro. É possível concluirmos o estado $[a = 3, b = 5]$, considerando-se a lógica de intercâmbio contendo "início" e "troca$_2$"? Se sim, teremos uma prova do nosso raciocínio, apoiando o nosso argumento.

Sempre começaremos um raciocínio pelo predicado inicial da lógica-alvo. Especificamente, justificamos que o estado $[a = 5, b = 3]$ é verdadeiro com base em "início". Prosseguindo, quando se considera verdadeira a ação "troca$_2$", também afirma-se a verdade do passo $[a = 5, b = 3] \rightarrow [a = 3, b = 5, t = 5]$. No entanto, o estado $[a = 3, b = 5, t = 5]$ pode ser escrito como $[a = 3, b = 5] \wedge [t = 5]$, por definição do conceito de estado. Donde se conclui $[a = 3, b = 5]$ se aplicarmos a regra lógica de **eliminação de conjunção** (Barker-Plummer et al., 2011, p. 130).

 — O raciocínio foi muito rápido para mim, mas seria essa uma maneira de provar o argumento, não?

 — Acabamos de fazer uma prova rigorosa de argumento sim, embora usando um estilo **informal** (Barker-Plummer et al., 2011, p. 48).

Espec reapresentará o raciocínio em um estilo formal mais adiante. No momento, vale ressaltar que separamos o último estado da prova por uma linha horizontal, destacando-o como estado-conclusão do raciocínio. A lista de afirmações que precedem a linha corresponde à **premissa** e as afirmações seguintes, à **conclusão**. O par premissa-conclusão chama-se **argumento**.

Em termos de código, esse argumento seria escrito como:

```
# CENÁRIO de codificação de um argumento
# PREMISSA  P1: [a=5, b=3]
assert (a == 5 and b == 3)
# CONCLUSÃO C1: [a=3, b=5]
assert (a == 3 and b == 5)
```

 — Esse argumento **não** é executável!

 — Certamente. Precisaremos introduzir funções de estado, predicados de estado e ações—além de variáveis—na tentativa de provar que este argumento pode ser verdadeiro.

[1] "Formato de Fitch", proposto por Frederic Fitch, de acordo com Barker-Plummer, JonBarwise, & Etchemendy (2011), p. 43.

Ou seja, Espec sugere que se faça a **implementação** de alguma prova do argumento. Caso seja possível, a conclusão $[a = 3, b = 5]$ será uma **consequência lógica** das afirmações anteriores. É impossível que $[a = 3, b = 5]$ seja falso, caso $[a = 5, b = 3]$ seja verdadeiro.

E mais. Na existência de uma prova, haverá um comportamento **válido** de máquina: o estado-conclusão será alcançado partindo-se do estado-premissa estabelecido pelo predicado inicial, eventualmente seguido por passos de ação—com base em Barker-Plummer et al. (2011), p.44.

 — Quer dizer que argumentos "resumem" comportamentos válidos da máquina?

 — Somente se você provar que os argumentos são verdadeiros.

 — Humm... a minha **tabela** não serve como **prova** que, se eu admitir $[a = 3, b = 5]$, então o estado $[a = 5, b = 3]$ será verdadeiro?

 — Bem observado. A sua tabela é uma possível **representação** da prova que os valores de a e b foram trocados, mas...

3.2 Prova Lógica e Construção de Programa

Depois de algum tempo, Fubã se lembra que a construção de tabelas-verdade envolvendo quatro ou mais variáveis de estado torna-se bem trabalhoso e, na prática, inviável.

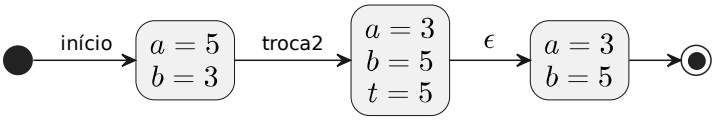
 — ... acho que será difícil montar a tabela no caso de comportamentos mais complicados.

 — Concordo. Em situações mais complicadas, convém usar um **método de prova** (Barker-Plummer et al., 2011, p. 48).

Uma **prova** é uma demonstração passo-a-passo que um estado resulta de **alguns** estados anteriores. Justifica-se cada passo por uma regra pré-existente ou criada com um propósito específico. O formato de apresentação de uma prova é similar ao de um comportamento com as seguintes diferenças. Acrescenta-se o número do passo, uma coluna de justificação e admitimos um ou mais estados-conclusão. Por fim, as afirmações-conclusão encontram-se marcadas com "▶". Assim, o efeito da execução do código até agora desenvolvido, pode ser formalizado na seguinte prova:

	Afirmação	Justificativa
1.	$[a = 5, b = 3]$	início
2.	$[a = 3, b = 5, t = 5]$	$1, \text{troca}_2$
3.	$[a = 3, b = 5] \wedge [t = 5]$	2, def. estado
▶ 4.	$[a = 3, b = 5]$	3, $\wedge$-Elim

Diagramas de máquinas de estado UML (Booch, Jacobson, & Rumbaugh, 1999) ajudam na ilustração de algumas partes da prova (ϵ indica um passo com preservação de atribuições):



 — Uau! Execução de programas vista como provas (Wadler, 2015)?

 — Sim! Vamos construir o código para intercambiar valores de variáveis com a ajuda desta prova.

1. Passo com justificativa "início":

	Afirmação	Justificativa
1.	$[a = 5, b = 3]$	início

No código em Python, coloca-se a máquina no estado inicial, acrescentando-se as variáveis `a` e `b` e atribuindo-se os literais `5` e `3` a elas, respectivamente:

```
# CENÁRIO definição de variáveis e predicado inicial
# VARIÁVEIS e PREDICADO INICIAL
a = 5
b = 3
# O predicado "início" é verdadeiro
# PREMISSA P1: [a=5, b=3]
assert (a == 5 and b == 3)
print (a, b) # observação do estado da máquina
```

2. Passo com justificativa "$1, \text{troca}_2$":

	Afirmação	Justificativa
2.	$[a = 3, b = 5, t = 5]$	$1, \text{troca}_2$

Implementa-se a ação "troca$_2$" com outra variável de código `t` e comandos de atribuição correspondentes:

```
# CENÁRIO de implementação da ação "troca2":
t = a
a = b
b = t
# A ação "troca2" é verdadeira
# [a=3, b=5, t=5]
assert (a == 3 and b == 5 and t == 5)
print (a, b, t) # observação do estado da máquina
```

3. Passo com justificativa "2, def. estado":

Afirmação	Justificativa
$3.[a = 3, b = 5] \wedge [t = 5]$	2, def. estado

Aplica-se a definição de estado na situação em que a máquina se encontra. Nenhum código efetivo em Python precisa ser acrescentado:

```
# CENÁRIO de definição de estado por conjunção lógica
# [a=3,   b=5,        t=5]
assert (a == 3 and b == 5 and t == 5)
# (a=3 /\ b=5)   /\   t=5
assert (a == 3 and b == 5)
assert (t == 5)
print (a, b, t) # observação do estado da máquina
```

4. Passo com justificativa "3, $\wedge$-Elim":

Afirmação	Justificativa
$\blacktriangleright$ $4.[a = 3, b = 5]$	3, $\wedge$-Elim

Aplica-se a regra de eliminação de conjunção sobre o estado da máquina. Nenhum código efetivo precisa ser acrescentado:

```
# CENÁRIO de eliminação de conjunção lógica:
# (a=3 /\ b=5)   /\   t=5
assert a == 3 and b == 5
assert t == 5
# (a=3 /\ b=5)
# CONCLUSÃO C1: [a=3, b=5]
assert a == 3 and b == 5
print (a, b) # visão de usuário...
```

Portanto, com base na prova apresentada, o estado-conclusão $[a = 3, b = 5]$ é uma **consequência lógica** de estados anteriores. Ou seja, o **argumento faz sentido**[2]. Segundo a prova, é impossível que a conclusão $[a = 3, b = 5]$ seja falsa se a afirmação $[a = 5, b = 3]$ for admitida como verdadeira. Neste caso, provamos que a conclusão segue da premissa estabelecida no argumento.

 — Nunca tinha enxergado desta maneira a construção de um programa:

```
# CENÁRIO do código de intercâmio de valores
a = 5
b = 3
t = a
a = b
b = t
print (a, b)
```

 — E se eu quiser fazer o intercâmbio de outros valores numéricos? Por exemplo, $[a = 4, b = 3]$ como estado inicial da máquina?

3.3 Lógica Interativa

Fubã levantou a preocupação de tornar a sua lógica de intercâmbio de valores mais geral, inspirado por uma descoberta.

 — Descobri que a plataforma Python oferece o método `input` para interagir com o usuário.

A chamada do método `input` produz um literal do tipo `int` correspondente ao número informado pelo usuário. O argumento da chamada sugere a espécie de número solicitado. No código a seguir, utiliza-se um predicado inicial "início$_2$" não formalizado ainda.

 — Posso informar que desejo o estado inicial $[a = 4, b = 3]$, por exemplo:

```
# CENÁRIO de lógica interativa (caso especial)
# VARIÁVEIS e PREDICADO INICIAL
a = input ("Digite o valor numérico de 'a' [4]")
b = input ("Digite o valor numérico de 'b' [3]")
# predicado inicial "início2" é verdadeiro
# a máquina se encontra no estado [a=4, b=3]
print (a, b)
```

[2]Um argumento logicamente válido onde todas as premissas são verdadeiras é conhecido por "argumento *sound*", traduzido como "argumento faz sentido".

 — Quero conhecer o comportamento da máquina quando eu pedir uma prova do argumento:

$$\frac{[a = 4, b = 3]}{[a = 3, b = 5]}$$

```
# CENÁRIO de passo de prova
t = a
a = b
b = t
# ação "troca2" é verdadeira
# a máquina muda para o estado [a=4, b=5, t=5]
assert (a == 3 and b == 5 and t == 5)
# eliminação de conjunção lógica
# a máquina encontra-se no subestado [a=3, b=5]
assert (a == 3 and b == 5)
# a máquina produz absurdo para este argumento
print (a, b, t)
```

 — Ops! A máquina produziu um absurdo!

 — Sim, em um caso mais geral pressupõe-se a existência do tipo `int` na lógica de especificação:

$$\begin{aligned} \text{início}_2 \triangleq & \wedge k_1, k_2 : \text{int} \\ & \wedge a = k_1 \\ & \wedge b = k_2 \end{aligned}$$

$$\begin{aligned} \text{troca}_2 \triangleq & \wedge t' = a \\ & \wedge a' = b \\ & \wedge b' = t' \end{aligned}$$

A provas tornam-se mais gerais:

Afirmação	Justificativa
1. $[a = k_1, b = k_2]$	início_2
2. $[a = k_2, b = k_1, t = k_1]$	$1, \text{troca}_2$
3. $[a = k_2, b = k_1] \wedge [t = k_1]$	2, def. estado
▶ 4. $[a = k_2, b = k_1]$	3, $\wedge$-Elim

E os programas, também:

```
# CENÁRIO de lógica interativa 2 (caso especial)
# VARIÁVEIS e PREDICADO INICIAL
k1 = input ("Digite o valor numérico de 'a'")
k2 = input ("Digite o valor numérico de 'b'")
a = k1
b = k2
# predicado inicial "início2" é verdadeiro
# a máquina se encontra no estado [a=k1, b=k2]
assert (a == k1 and b == k2)
t = a
a = b
b = t
# ação "troca2" é verdadeira
# a máquina muda para o estado [a=k2, b=k1, t=k1]
assert (a == k2 and b == k1 and t == k1)
# eliminação de conjunção lógica
# a máquina encontra-se no subestado [a=k2, b=k1]
assert (a == k2 and b == k1)
```

 — Enxergo um código mais simples, embora só consiga conferir a conclusão inspecionando os valores:

```
# CENÁRIO do código interativo para intercâmbio de valores numéricos
# (caso: literais do tipo 'int')
a = input ("Digite o valor numérico de 'a'")
b = input ("Digite o valor numérico de 'b'")
print (a, b)
t = a
a = b
b = t
print (a, b)
```

3.4 Resumo

1. **Argumento** é constituído por uma **premissa** (estado de partida) e por uma **conclusão** (estado de chegada).
2. **Prova** é uma demonstração passo-a-passo que um estado resulta de **alguns** estados anteriores.
3. Provas **rigorosas** podem ser escritas em um estilo informal ou em um estilo formal (usando uma linguagem lógica ou uma linguagem de programação).
4. O **comportamento** de uma máquina é **válido** se existir uma prova da sua sequência de estados.

3.5 Exercícios

3.5.1 Raciocínio sobre Comportamentos

Contexto Considere uma especificação lógica constituída pelas seguintes regras de estado:

$$\begin{aligned} \text{início} \triangleq & \wedge a = 5 \\ & \wedge b = 3 \end{aligned}$$

$$\begin{aligned} \text{outraTroca} \triangleq & \wedge t' > b \\ & \wedge a' = b \\ & \wedge b' = t' \end{aligned}$$

Um código em Python que implementa tal especificação tem a forma:

```
# Raciocínio sobre Comportamentos
a = 5
b = 3
t = b
a = b
b = t
print (a, b, t)
```

Pretende-se investigar os possíveis estados-conclusão x do seguinte comportamento decorrente de um passo, pelo menos, afirmado pela ação "outraTroca":

$$\frac{[a = 5, b = 3]}{x}$$

Enunciado Assinale a alternativa contendo um x **falso**:

1. $[a = 3]$
2. $[a = 5]$
3. $[b = 3]$
4. $[t = 3]$

3.5.2 Índice de Massa Corporal [*]

Contexto O índice de massa corporal (IMC) é um "valor derivado da massa [m, em metros] e do peso [p, em quilogramas] de uma pessoa"[3] conforme a seguinte lógica:

$$\begin{aligned} \text{início} \triangleq & \wedge p = 70 \\ & \wedge m = 1.7 \\ & \wedge k = \bot \end{aligned}$$

$$\text{imc} \triangleq \wedge k' = \frac{p}{m^2}$$

O valor numérico atribuído à variável k possui precisão de um dígito, com arredondamento direto do tipo "rounding up"[4].

Enunciado

1. Prove que o seguinte argumento produz um absurdo usando um estilo formal:

$$\frac{[p = 70, m = 1.7]}{[k = 10]}$$

2. Implemente a prova do item anterior em Python.
3. Desenvolva uma lógica interativa para se calcular o IMC.
4. Com base na lógica do item anterior, elabore um argumento verdadeiro a respeito do seu IMC.
5. Prove que argumento do item anterior é verdadeiro.
6. Implemente a prova do item anterior em Python.

[3] URL: https://en.wikipedia.org/wiki/Body_mass_index. Acesso em 11/03/2022.
[4] URL: https://en.wikipedia.org/wiki/Rounding. Acesso em 11/03/2022.

Referências

Barker-Plummer, D., JonBarwise, & Etchemendy, J. (2011). *Language, proof, and logic* (2º ed, p. 620). CSLI Publications.

Booch, G., Jacobson, I., & Rumbaugh, J. (1999). *The Unified Modeling Language User Guide*. Addison Wesley.

Wadler, P. (2015). Propositions as Types. *Communications of the ACM*, *58*(12), p. 75–84. doi:10.1145/2699407