Lógica
Aula 20

Renata Wassermann

renata@ime.usp.br

2020

Relação entre $\vdash$ e $\models$

A dedução natural é **correta e completa** em relação à semântica:

$$\varphi_1, \varphi_2, \varphi_3, ..., \varphi_n \vdash \psi \iff \varphi_1, \varphi_2, \varphi_3, ..., \varphi_n \models \psi$$

Problemas de decisão

- Decidir se um número é primo.

Problemas de decisão

- Decidir se um número é primo.
- Decidir se um programa termina.

Problemas de decisão

- Decidir se um número é primo.
- Decidir se um programa termina.
- Decidir se uma fórmula é válida.

Problema da parada

Dado um programa P, decidir se ele termina.

Problema da parada

Dado um programa P, decidir se ele termina.

Máquina de Turing: capaz de simular qualquer computação

Problema da parada

Dado um programa P, decidir se ele termina.

Máquina de Turing: capaz de simular qualquer computação

Turing (1936) - O problema de decidir se, dada uma MT, ela para é indecidível.

Indecidibilidade da LPO

Dada uma fórmula φ, decidir se ela é válida.

Indecidibilidade da LPO

Dada uma fórmula φ, decidir se ela é válida.

Church (1936) - O problema de decidir se, dada uma fórmula φ na lógica de primeira ordem, ela é válida é indecidível.

Verificação Formal de Programas

Por que verificar?

Verificação Formal de Programas

Por que verificar?

Ariane 5 (1996)

- 10 anos de desenvolvimento

Ariane 5 (1996)

- 10 anos de desenvolvimento
- U$ 7 bilhões

Ariane 5 (1996)

- 10 anos de desenvolvimento
- U$ 7 bilhões
- menos de 40 segundos

Ariane 5 (1996)

- 10 anos de desenvolvimento
- U$ 7 bilhões
- menos de 40 segundos
- falha no software de controle do sistema de referência e velocidade.

Ariane 5 (1996)

- 10 anos de desenvolvimento
- U$ 7 bilhões
- menos de 40 segundos
- falha no software de controle do sistema de referência e velocidade.
- "The internal SRI* software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. "

Tipos de verificação

- Prova ou modelo: sistema representado por conjunto de fórmulas $\times$modelo finito

Tipos de verificação

- Prova ou modelo: sistema representado por conjunto de fórmulas $\times$modelo finito
- Grau de automatização

Tipos de verificação

- Prova ou modelo: sistema representado por conjunto de fórmulas $\times$modelo finito
- Grau de automatização
- Verificar propriedade ou sistema completo

Tipos de verificação

- Prova ou modelo: sistema representado por conjunto de fórmulas $\times$modelo finito
- Grau de automatização
- Verificar propriedade ou sistema completo
- Domínio de aplicação: programas reativos ou que terminam

Tipos de verificação

- Prova ou modelo: sistema representado por conjunto de fórmulas $\times$modelo finito
- Grau de automatização
- Verificar propriedade ou sistema completo
- Domínio de aplicação: programas reativos ou que terminam
- Pré ou Pós-desenvolvimento

O método que vamos ver

- Baseado em provas: imposível ter modelo finito com variáveis inteiras.

O método que vamos ver

- Baseado em provas: imposível ter modelo finito com variáveis inteiras.
- Semi-automático: regras de cálculo

O método que vamos ver

- Baseado em provas: imposível ter modelo finito com variáveis inteiras.
- Semi-automático: regras de cálculo
- Verifica propriedade

O método que vamos ver

- Baseado em provas: imposível ter modelo finito com variáveis inteiras.
- Semi-automático: regras de cálculo
- Verifica propriedade
- Domínio de aplicação: programas sequenciais e "transformadores" (input/output)

O método que vamos ver

- Baseado em provas: impossível ter modelo finito com variáveis inteiras.
- Semi-automático: regras de cálculo
- Verifica propriedade
- Domínio de aplicação: programas sequenciais e "transformadores" (input/output)
- Usado durante a codificação para pequenos fragmentos.

Fases

1. Partindo de uma descrição informal D, gerar φ_D.

Fases

1. Partindo de uma descrição informal D, gerar φ_D.
2. Escrever programa P.

Fases

1. Partindo de uma descrição informal D, gerar φ_D.
2. Escrever programa P.
3. Provar que $P \vdash \varphi_D$.

Exemplo - Soma dos Primeiros x Naturais

```
z = 0;
while (x > 0) {
   z = z + x;
   x = x - 1;
}
```

Queremos poder expressar:

Exemplo - Soma dos Primeiros x Naturais

```
z = 0;
while (x > 0) {
   z = z + x;
   x = x - 1;
}
```

Queremos poder expressar:

- Com entrada $x \geq 0$ o programa devolve $z = \sum_{i=1}^{x} i$

Exemplo - Soma dos Primeiros x Naturais

```
z = 0;
while (x > 0) {
   z = z + x;
   x = x - 1;
}
```

Queremos poder expressar:

- Com entrada $x \geq 0$ o programa devolve $z = \sum_{i=1}^{x} i$
- O programa termina.

Triplas de Hoare

Como descrever uma especificação φ_D?

Triplas de Hoare

Como descrever uma especificação φ_D?

Seja D "Calcule um número y cujo quadrado seja menor que a entrada x."

Triplas de Hoare

Como descrever uma especificação φ_D?

Seja D "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_D : y.y < x$$

Triplas de Hoare

Como descrever uma especificação φ_D?

Seja D "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_D : y.y < x$$

E se x for -4?

Triplas de Hoare

Como descrever uma especificação φ_D?

Seja D "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_D : y.y < x$$

E se x for -4?

D': "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

Triplas de Hoare

Como descrever uma especificação φ_D?

Seja D "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_D : y.y < x$$

E se x for -4?

D': "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

Ou seja: não basta descrever estado após a execução do programa!

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Ao rodar P num estado que satisfaz φ, chegamos a um estado que satisfaz ψ."

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Ao rodar P num estado que satisfaz φ, chegamos a um estado que satisfaz ψ."

φ é a pré-condição

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Ao rodar P num estado que satisfaz φ, chegamos a um estado que satisfaz ψ."

φ é a pré-condição (pode ser vazia ($|\top|$))

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Ao rodar P num estado que satisfaz φ, chegamos a um estado que satisfaz ψ."

φ é a pré-condição (pode ser vazia ($|\top|$))
ψ é a pós-condição

Triplas de Hoare

"Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

Triplas de Hoare

"Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

Triplas de Hoare

"Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

```
P1:     y = 0
```

Triplas de Hoare

"Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

```
P1:     y = 0
```

```
P2:     y = 0;
        while (y*y<x) {
           y = y+1;
        y = y-1;
```

Correção Parcial

$$\models_{par} (|\varphi|)P(|\psi|)$$

$$\Updownarrow$$

Para qualquer estado satisfazendo φ, se P termina, o estado final satisfaz ψ.

Correção Parcial

$$\models_{par} (|\varphi|)P(|\psi|)$$

$$\Updownarrow$$

Para qualquer estado satisfazendo φ, se P termina, o estado final satisfaz ψ.

Qualquer programa que não para é parcialmente correto.

Correção Total

$$\models_{tot} (|\varphi|)P(|\psi|)$$

$$\Updownarrow$$

$$\models_{par} (|\varphi|)P(|\psi|) \text{ e } P \text{ termina.}$$

Linguagem para Descrição de Programas

Expressões inteiras:

$$E ::= n|x|(-E)|(E + E)|(E * E)$$

Linguagem para Descrição de Programas

Expressões inteiras:

$$E ::= n|x|(-E)|(E+E)|(E*E)$$

Expressões booleanas:

$$B ::= \texttt{true}|\texttt{false}|(!B)|(B\&B)|(B||B)|(E<E)$$

$$E_1 == E_2 \equiv !(E_1 < E_2)\&!(E_2 < E_1)$$

Linguagem para Descrição de Programas

Expressões inteiras:

$$E ::= n|x|(-E)|(E+E)|(E*E)$$

Expressões booleanas:

$$B ::= \texttt{true}|\texttt{false}|(!B)|(B\&B)|(B||B)|(E<E)$$

$$E_1 == E_2 \equiv !(E_1 < E_2)\&!(E_2 < E_1)$$

Comandos:

$$C ::= x = E|C;C|\texttt{if}\ B\{C\}\texttt{else}\ \{C\}|\texttt{while}\ B\{C\}$$

Exemplo - Fatorial 1

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
}
```

Exemplo - Fatorial 1

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
}
```

$$\models_{par} (|x \geq 0|) \text{ Fatorial 1 } (|y = x!|)$$

Exemplo - Fatorial 2

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
}
```

Exemplo - Fatorial 2

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
}
```

$$\models_{par} (|x \geq 0|) \text{ Fatorial 2 } (|y = x!|)???$$

Exemplo - Fatorial 2

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
}
```

$$\models_{par} (|x \geq 0|) \text{ Fatorial 2 } (|y = x!|) \text{ X}$$

$$\models_{par} (|x = x_0 \wedge x \geq 0|) \text{ Fatorial 2 } (|y = x_0!|)$$

Variáveis lógicas

Não aparecem no programa!

```
Soma:     z = 0;
          while (x > 0) {
            z = z + x;
            x = x - 1;
          }
```

Variáveis lógicas

Não aparecem no programa!

```
Soma:     z = 0;
          while (x > 0) {
            z = z + x;
            x = x - 1;
          }
```

$(|x = 3|)$`Soma` $(|z = 6|)$

Variáveis lógicas

Não aparecem no programa!

```
Soma:     z = 0;
          while (x > 0) {
            z = z + x;
            x = x - 1;
          }
```

$(|x = 3|)\texttt{Soma}\ (|z = 6|)$

$(|x = 8|)\texttt{Soma}\ (|z = 36|)$

Variáveis lógicas

Não aparecem no programa!

```
Soma:     z = 0;
          while (x > 0) {
            z = z + x;
            x = x - 1;
          }
```

$(|x = 3|)\texttt{Soma}\ (|z = 6|)$

$(|x = 8|)\texttt{Soma}\ (|z = 36|)$

$(|x = x_0 \wedge x \geq 0|)\texttt{Soma}\ (|z = (x_0.(x_0 + 1))/2|)$

Variáveis lógicas

Não aparecem no programa!

```
Soma:     z = 0;
          while (x > 0) {
            z = z + x;
            x = x - 1;
          }
```

$(|x = 3|)\texttt{Soma}\ (|z = 6|)$

$(|x = 8|)\texttt{Soma}\ (|z = 36|)$

$(|x = x_0 \wedge x \geq 0|)\texttt{Soma}\ (|z = (x_0.(x_0 + 1))/2|)$

Estado dá valor para variáveis do programa, mas não para variáveis lógicas.