Lógica e Computação

Ano letivo 2022/23

Motivação

Qual é a utilidade de se estudar disciplinas como "Lógica e Computação"?
Num curso de Engenharia as disciplinas focam-se em:

- Tópicos práticos relevantes para o curso
- Tópicos teóricos relevantes para o curso

Ambas os tópicos são importantes para um curso de Engenharia.

Tópicos práticos

Vantagens:

- Ensinam a fazer as coisas "na prática".
- Fornecem ferramentas concretas para resolver os problemas de um cliente.

Desvantagens:

- Muito voláteis e sujeitos a "modas".
- Tempo de vida útil relativamente curto/necessidade de atualização constante.

Tópicos teóricos

Desvantagens:

- São mais abstratas
- São de aplicação não tão óbvia

Vantagens:

- Ao não se focarem numa tecnologia concreta, têm um tempo de vida útil muito mais longo
- Fornecem estratégias e formas de pensar que influenciam de forma positiva a resolução de problemas na prática
- Permitem compreender de forma mais abrangente como as várias ferramentas práticas disponíveis se conjugam umas com as outras, permitindo selecionar e aprender novos paradigmas/tecnologias de forma mais eficaz.

Objetivos

Nesta disciplina pretende-se:

- Melhorar a capacidade de raciocínio lógico.
    - Os computadores são máquinas cujo funcionamento se baseia em princípios lógicos.
    - Melhorar a capacidade de racicocínio lógica melhora a nossa capacidade de comunicar à máquina o que pretendemos o que ela faça.
- Compreender melhor o termo "computação"
    - Conhecer os seus limites e possibilidades
    - Perceber o que é possível fazer através de "computações", independentemente do software/hardware utilizado.

Organização da disciplina

A disciplina de Lógica e Computação é organizada do seguinte modo:

- Lógica
  - **Cálculo Proposicional**: Lógica utilizando operadores simples como *and*, *or*, *not*
  - **Lógica de primeira ordem**: Permite utilizar outros domínios para além do $\{true, false\}$, para além de quantificadores, etc.
- Teoria da Computação
  - **Computabilidade**: permite perceber o que é uma computação, de forma independente do hardware/software, e que problemas é possível resolver com uma computação (com tempo/memória ilimitados)
  - **Complexidade Computacional**: refina a computabilidade para saber que problemas podem ser resolvidos com recursos (memória, tempo, etc.) limitados

Lógica

Cálculo Proposicional

Introdução

- Na lógica (e com computadores) não podemos simplesmente utilizar a linguagem de todos os dias (a *linguagem natural*) uma vez que se presta a ambiguidades que dependem de vários factores externos.
- Apesar disso às vezes utiliza-se a linguagem natural em lógica, como por exemplo os Silogismos que estudaram em Lógica:

Premissa: *Todos os homens são mortais.*
Premissa: *Sócrates é um homem.*
Conclusão: *Logo Sócrates é mortal.*

Mas será isto um silogismo?:

Premissa: *Todo o banco é um móvel.*
Premissa: *A Caixa Geral de Depósitos é um banco.*
Conclusão: *A Caixa Geral de Depósitos é um móvel.*

Conseguimos aperceber-nos que este raciocínio é falso porque a palavra "banco" pode ter dois significados distintos e conseguimos (às vezes) seleccionar o contexto correto.

- Ou seja, temos um problema de *ambiguidade*.
- Mas como resolve um computador este problema?

Outros problemas: paradoxos

Exemplo (paradoxo do mentiroso)

Esta afirmação é falsa.

Exemplo

A afirmação seguinte é verdadeira. A afirmação anterior é falsa.

Exemplo (paradoxo de Berry)

O menor número natural que não pode ser definido com menos de vinte palavras.

Exemplo (Paradoxo de Russell)

Considere o conjunto definido da seguinte forma:

$$X = \{A | A \notin A\}.$$

Por exemplo $\{1\} \in X$ já que $\{1\} \notin \{1\}$. Será que $X \in X$?

Um problema que é evidente aqui é que a nossa de noção de conjunto é intuitiva – isso funciona muitas vezes, mas também falha!

- É necessário ser mais rigoroso na forma como definimos as coisas!
- Por essa razão introduzem-se linguagens formais

Linguagens formais

Definição

Um alfabeto é um conjunto não-vazio.

Definição

Uma palavra sobre um alfabeto Σ *é uma sequência finita de símbolos de* Σ*. O conjunto de todas as palavras sobre o alfabeto* Σ *é designado por* Σ^**.*

Por exemplo, $12+3$ ou $23+$ são palavras sobre o alfabeto $\Sigma = \{1, 2, 3, +\}$ e da mesma forma $pq\wedge \in \{p, q, \wedge\}^*$.

Definição

Uma linguagem formal sobre um alfabeto Σ *é um subconjunto* L *de* Σ^**.*

Noção de fórmula

O conjunto de todas as fórmulas do cálculo proposicional é uma linguagem formal que utiliza um alfabeto constituído pelos seguinte elementos (símbolos):

1. Váriaveis proposicionais $p_1, p_2, p_3, \ldots$ (por conveniência, estes símbolos são frequentemente representados de $p, q, r, \ldots$);
2. Os símbolos $\neg, \wedge, \vee, \Rightarrow$, chamados de conetivos lógicos;
3. Os símbolos (,), (parêntesis) utilizados para agrupar expressões.

Noção de fórmula

Definição

Uma fórmula do cálculo proposicional é toda a expressão que pode ser obtida aplicando as seguintes regras:

(i) *Toda a variável proposicional é uma fórmula;*

(ii) *Se ϕ é uma fórmula, então $(\neg\phi)$ é uma fórmula;*

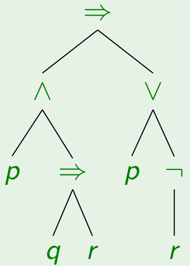
(iii) *Se ϕ e ψ são fórmulas, então $(\phi \wedge \psi)$, $(\phi \vee \psi)$, e $(\phi \Rightarrow \psi)$ são fórmulas.*

As variáveis proposicionais são representadas através de símbolos $p_1, p_2, \ldots, p_n, \ldots$, para salientar que o número de variáveis pode ser infinito. No entanto, podemos também designa-las de forma diferente, como por exemplo: $p, q, r, \ldots$.

Pode-se também representar fórmulas através de árvores.

Exemplo

A árvore associada à fórmula $(p \wedge (q \Rightarrow r)) \Rightarrow (p \vee \neg r)$ é



Em geral:

- Letras minúsculas (com ou sem índice) irão representar variáveis proposicionais.
- Letras gregas minúsculas (por exemplo ϕ, ψ, φ) irão representar fórmulas
- Letras gregas maiúsculas (por exemplo Γ) irão representar conjuntos finitos de fórmulas.

Para evitar fórmulas com demasiados parêntesis, utilizam-se também as seguinte convenções:

- O símbolo de negação tem precedência sobre todos os outros símbolos.
- Omitimos parêntesis desde que isso não torne a fórmula ambígua (i.e. desde que a fórmula não possa ser interpretada de duas maneiras distintas). Por exemplo, não iremos colocar os parêntesis "mais de fora"

Exemplo

$\neg p \vee \neg q = ((\neg p) \vee (\neg q))$

Na lógica há normalmente duas formas alternativas de trabalhar com fórmulas lógicas:

- Utilizando a *semântica*. Neste caso as fórmulas são "interpretadas", e tomam valores, por exemplo V ou F
- Utilizando *sistemas dedutivos*. Neste caso as fórmulas não são interpretadas, mas temos regras, ao estilo do que é feito com o Silogismo, que nos permitem concluir novas fórmulas a partir de fórmulas dadas.

Vamos agora estudar o caso da semântica no cálculo proposicional. Mais tarde iremos também estudar sistemas dedutivos para o cálculo proposicional.

Semântica no cálculo proposicional

- Uma fórmula, por si só, é uma expressão sem qualquer significado intrínseco.
- Na semântica pretende-se atribuir valores lógicos a fórmulas.
- No caso concreto do cálculo proposicional, pretende-se atribuir o valor lógico 0 (falso) ou 1 (verdadeiro) a uma fórmula.

Definição

Seja V um conjunto de variáveis proposicionais. Uma valoração de V é uma aplicação $v : V \to \{0, 1\}$.

Obviamente não nos interessa dar valores lógicos apenas às variáveis proposicionais em V, mas sim a qualquer fórmula do cálculo proposicional sobre V.

Definição

Um operador booleano é uma função $f : \{0,1\}^n \to \{0,1\}$, *onde* $n \in \mathbb{N}$.

nome	símbolo	nome	símbolo
conjunção	$\wedge$	*implicação*	$\Rightarrow$
disjunção	$\vee$	*equivalência*	$\Leftrightarrow$
negação	$\neg$		

x	$\neg x$
0	1
1	0

x	y	$x \wedge y$	$x \vee y$	$x \Rightarrow y$	$x \Leftrightarrow y$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Definição

Uma valoração total $\bar{v}$ *(também chamada de interpretação) é uma aplicação* $\bar{v} : \mathcal{F}_V \to \{0,1\}$ *que satisfaz as seguintes propriedades:*

(a) *Existe uma valoração* $v : V \to \{0,1\}$ *tal que* $\bar{v}(p_i) = v(p_i)$ *para todo o* $p_i \in V$*;*

(b) $\bar{v}(\neg\phi) = \neg\bar{v}(\phi)$ *para todo o* $\phi \in \mathcal{F}_V$*;*

(c) $\bar{v}(\phi \wedge \psi) = \bar{v}(\phi) \wedge \bar{v}(\psi)$*;* $\bar{v}(\phi \vee \psi) = \bar{v}(\phi) \vee \bar{v}(\psi)$*, e* $\bar{v}(\phi \Rightarrow \psi) = \bar{v}(\phi) \Rightarrow \bar{v}(\psi)$ *para todos os* $\phi, \psi \in \mathcal{F}_V$*.*

- Nas igualdades da definição acima, os símbolos $\neg, \wedge, \vee, \Rightarrow$ no lado direito designam os respetivos operadores booleanos.
- Para simplificar a notação, iremos simplesmente chamar de valoração a uma valoração total.

Definição

Uma fórmula proposicional ϕ *diz-se:*

- *satisfazível se* $v(\phi) = 1$ *para alguma valoração* v.
- *uma tautologia se* $v(\phi) = 1$ *para toda a valoração* v, *escrevendo-se* $\models \phi$.
- *contraditória se* $v(\phi) = 0$ *para toda a valoração* v.

Se v é uma valoração e ϕ é uma fórmula onde se tenha $v(\phi) = 1$, então diz-se que a valoração v *satisfaz* a fórmula ϕ. Alternativamente, diz-se também que v é um *modelo* de ϕ

Definição

Um conjunto de fórmulas $\Gamma = \{\phi_1, \ldots, \phi_n\}$ *diz-se satisfazível se existe uma valoração* v *tal que* $v(\phi_1) = \ldots = v(\phi_n) = 1$.

Fórmulas logicamente equivalentes

Formalmente falando, as fórmulas $\neg\neg p$ e p não são iguais mas, apesar disso, comportam-se exatamente da mesma forma do ponto de vista lógico. Ou seja, utilizando uma perspetiva lógica, é como se fossem iguais

Definição

Sejam $\phi, \psi \in \mathcal{F}_P$. Se $v(\phi) = v(\psi)$ para todas as valorações v, então diz-se que ϕ e ψ são logicamente equivalentes e escrevemos $\phi \equiv \psi$.

Teorema

$\phi \equiv \psi$ se e só se $\phi \Leftrightarrow \psi$ é uma tautologia.

Teorema

Seja V um conjunto de variáveis proposicionais e ψ uma tautologia (fórmula contraditória) que utiliza somente variáveis $p_1, \ldots, p_n$ que pertencem a V. Sejam $\phi_1, \ldots, \phi_n$ fórmulas do cálculo proposicional e seja ψ' uma fórmula obtida de ψ substituindo cada ocorrência da variável p_i em ψ por ϕ_i, para $i = 1, \ldots, n$. Então ψ' é uma tautologia (fórmula contraditória, respetivamente).

Teorema

Seja V *um conjunto de variáveis proposicionais e sejam* ϕ *e* ψ *fórmulas logicamente equivalentes de* $\mathcal{F}_V$ *que utilizam somente variáveis* $p_1, \ldots, p_n \in V$*. Sejam* $\phi_1, \ldots, \phi_n$ *fórmulas do cálculo proposicional e sejam* ϕ' *e* ψ' *fórmulas obtidas de* ϕ *e* ψ*, respetivamente, substituindo cada ocorrência da variável* p_i *por* ϕ_i*, para* $i = 1, \ldots, n$*. Então* $\phi' \equiv \psi'$*.*

$$\bot = p_1 \wedge \neg p_1$$

$$\top = p_1 \vee \neg p_1$$

Algumas equivalências lógicas

$$\phi \wedge \psi \equiv \psi \wedge \phi \qquad \text{(comutatividade)}$$
$$\phi \vee \psi \equiv \psi \vee \phi$$

$$\phi \vee (\psi \wedge \varphi) \equiv (\phi \vee \psi) \wedge (\phi \vee \varphi) \qquad \text{(distributividade)}$$
$$\phi \wedge (\psi \vee \varphi) \equiv (\phi \wedge \psi) \vee (\phi \wedge \varphi)$$

$$\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi) \qquad \text{(leis de De Morgan)}$$
$$\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$$

$$\neg\neg\phi \equiv \phi \qquad \phi \Rightarrow \psi \equiv \neg\phi \vee \psi$$
$$\phi \Leftrightarrow \psi \equiv \psi \Leftrightarrow \phi \qquad \phi \Rightarrow \psi \equiv \neg\psi \Rightarrow \neg\phi$$

Consequência semântica

Definição

Seja Γ *um conjunto finito de fórmulas e* ϕ *uma fórmula. Diz-se que* ϕ *é uma consequência semântica de* Γ*, e escreve-se* $\Gamma \models \phi$*, sse sempre que uma valoração* v *satisfaz simultaneamente todas as fórmulas de* Γ*, então essa valoração também satisfaz* ϕ*.*

p_1	p_2	$p_1 \vee \neg p_2$	$p_1 \Rightarrow p_2$	$\neg p_1 \vee \neg p_2$	
0	0	1	1	1	
0	1	0	1	1	
1	0	1	0	1	
1	1	1	1	0	$\leftarrow$

Caso importante: se nenhuma valoração satisfaz Γ, então $\Gamma \models \phi$ para qualquer ϕ.

p_1	p_2	$p_1 \land \neg p_2$	$p_1 \Rightarrow p_2$	$p_1 \land \neg p_1$
0	0	0	1	0
0	1	0	1	0
1	0	1	0	0
1	1	0	1	0

$\Gamma = \{p_1 \land \neg p_2, p_1 \Rightarrow p_2\}$ e $\phi = p_1 \land \neg p_1$. Tem-se $\Gamma \models \phi$.

Teorema

$\Gamma \cup \{\psi\} \models \phi$ se e só se $\Gamma \models \psi \Rightarrow \phi$.

Teorema

Se $\Gamma \models \phi$, então $\Gamma \cup \{\psi\} \models \phi$ para qualquer fórmula ψ.

Teorema

Se $\Gamma \models \phi$ e ψ é uma tautologia, então $\Gamma \backslash \{\psi\} \models \phi$.

Fórmulas normais

Vamos mostrar que toda a fórmula do cálculo proposicional pode ser reescrita noutra que lhe é logicamente equivalente e que tem uma determinada estrutura.

Definição

- *Um literal é uma expressão do tipo p_i ou $\neg p_i$, onde p_i é uma variável proposicional.*
- *Uma conjunção elementar é uma fórmula com o formato $l_1 \wedge \ldots \wedge l_n$, onde $l_1, \ldots, l_n$ são literais.*
- *Uma fórmula normal disjuntiva é uma fórmula do tipo $\phi_1 \vee \ldots \vee \phi_m$, onde $\phi_1, \ldots, \phi_m$ são conjunções elementares.*

Teorema

Toda a fórmula do cálculo proposicional é logicamente equivalente a uma fórmula normal disjuntiva.

Definição

- *Uma disjunção elementar é uma fórmula com o formato* $p_1 \vee \ldots \vee p_n$*, onde* $p_1, \ldots, p_n$ *são literais.*
- *Uma fórmula normal conjuntiva é uma fórmula do tipo* $\phi_1 \wedge \ldots \wedge \phi_m$*, onde* $\phi_1, \ldots, \phi_m$ *são disjunções elementares.*

Teorema

Toda a fórmula do cálculo proposicional é logicamente equivalente a uma fórmula normal conjuntiva.

Definição

Um conjunto de símbolos de operador $\mathcal{O}$ *diz-se adequado quando toda a fórmula do cálculo proposicional é logicamente equivalente a outra fórmula que utiliza apenas símbolos de operador de* $\mathcal{O}$.

Corolário

O conjunto $\{\neg, \wedge, \vee\}$ *é adequado.*

Corolário

Os conjuntos $\{\neg, \wedge\}$, $\{\neg, \vee\}$ *e* $\{\neg, \Rightarrow\}$ *são adequados.*

Tableaux semânticos

- Podemos ver se uma fórmula é satisfazível através da construção de uma tabela de verdade.
- No entanto, apesar de conceptualmente simples, este método é pouco eficiente do ponto de vista computacional, já que o número de valorações (linhas) a testar é exponencial no número de variáveis.
- Em geral, não se conhece nenhum algoritmo que receba como input uma fórmula do cálculo proposicional e que permita decidir, de forma eficiente, se essa fórmula é satisfazível – há fortes razões para pensar que tal não é possível.
- No entanto, há algoritmos (por exemplo baseados na regra da resolução, ou no método dos tableaux semânticos) que em muitos casos são mais eficientes do que o método das tabelas de verdade para decidir se uma fórmula é satisfazível.

- Um *tableau semântico* é uma árvore, em que associamos a cada vértice l um conjunto de fórmulas $U(l)$.
- A raiz da árvore consiste num só vértice, associado ao conjunto formado pela fórmula em estudo.
- Uma folha l será marcada como *fechada* ($\times$) se houver um par complementar de literais contido em $U(l)$, e será marcada como *aberta* ($\odot$) se $U(l)$ for constituído apenas por literais e não contiver um par complementar de literais.
- Diz-se que um tableau é *fechado* se todas as folhas estão marcadas como fechadas. Caso contrário diz-se que o tableau é *aberto*.
- Uma fórmula ϕ será satisfazível se e só se o tableau obtido pela aplicação do algoritmo seguinte esta fórmula é aberto.

Algoritmo (construção de um tableau semântico):

Input: uma fórmula ϕ do cálculo proposicional.

Output: um tableau completo associado à fórmula ϕ.

1. Criar uma árvore que consiste num só vértice (a raiz) que está associado ao conjunto $\{\phi\}$.
2. Para cada folha l não marcada da árvore, verificar se $U(l)$ é um conjunto de literais e, caso o seja, marcar essa folha l como: (i) fechada se $U(l)$ contiver um par complementar de literais ou (ii) aberta, caso contrário.
3. Se todas as folhas da árvore estiverem marcadas, então o algoritmo termina com a seguinte conclusão:
    1. A fórmula ϕ é satisfazível se o tableau obtido é aberto.
    2. A fórmula ϕ é contraditória se o tableau obtido é fechado.
4. Selecionar uma folha l que não está marcada. Vamos expandir a árvore a partir dessa folha da seguinte forma. Escolhemos uma fórmula ϕ de $U(l)$ que não seja um literal.

5. Se ϕ é logicamente equivalente a uma fórmula conjuntiva $\phi \equiv \phi_1 \wedge \phi_2$, então criar um vértice descendente I' a partir de I, com $U(I') = (U(I) \cup \{\alpha_1, \alpha_2\}) - \{\phi\}$, onde as fórmulas α_1, α_2 são dadas pela tabela abaixo (no caso de $\neg\neg A$ não existe fórmula α_2), e depois ir para o passo 2.

ϕ	α_1	α_2
$\neg\neg A$	A	
$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \Rightarrow A_2)$	A_1	$\neg A_2$

6. Se ϕ é logicamente equivalente a uma fórmula disjuntiva $\phi \equiv \phi_1 \vee \phi_2$, então criar dois nós descendentes I' e I'' a partir de I, com $U(I') = (U(I) \cup \{\beta_1\}) - \{\phi\}$ e $U(I'') = (U(I) \cup \{\beta_2\}) - \{\phi\}$, onde as fórmulas β_1, β_2 são dadas pela tabela abaixo, e depois ir para o passo 2.

ϕ	β_1	β_2
$A_1 \vee A_2$	A_1	A_2
$\neg(A_1 \wedge A_2)$	$\neg A_1$	$\neg A_2$
$A_1 \Rightarrow A_2$	$\neg A_1$	A_2

Exemplo

$$
\begin{array}{ccccc}
 & & p \wedge (\neg q \vee \neg p) & & \\
 & & \downarrow & & \\
 & & p, \neg q \vee \neg p & & \\
 & \swarrow & & \searrow & \\
p, \neg q & & & & p, \neg p \\
\odot & & & & \times
\end{array}
$$

Como o tableau é aberto, concluímos que $p \wedge (\neg q \vee \neg p)$ é satisfazível.

Teorema

A aplicação do algoritmo anterior para a construção de um tableau semântico termina sempre.

Teorema (correção e completude)

Seja ϕ uma fórmula e $\mathcal{T}$ um tableau completo que lhe está associado. ϕ é contraditória se e só se $\mathcal{T}$ é fechado.

Corolário

ϕ é satisfazível se e só se $\mathcal{T}$ é aberto.

Corolário

$\neg\phi$ é uma tautologia se e só se $\mathcal{T}$ é fechado.

Sistemas dedutivos - dedução natural

Podemos inferir novas fórmulas a partir de:

- Um conjunto de fórmulas iniciais (as *premissas*)
- *Regras de inferência*

Definição

Seja Γ *um conjunto de fórmulas. Uma dedução a partir de* Γ *é uma sequência finita de fórmulas* $\phi_1, \ldots, \phi_n$ *em que cada fórmula dessa sequência é um elemento de* Γ *ou pode ser deduzida a partir de fórmulas anteriores utilizando uma das regras de inferência. Se existe uma dedução* $\phi_1, \ldots, \phi_n$ *a partir de* Γ*, dizemos que a fórmula* ϕ_n *pode ser deduzida de* Γ*, e escrevemos* $\Gamma \vdash \phi_n$.

	Introdução	Eliminação
$\land$	$\dfrac{\phi \quad \psi}{\phi \land \psi}\land_i$	$\dfrac{\phi \land \psi}{\phi}\land_{e_1} \qquad \dfrac{\phi \land \psi}{\psi}\land_{e_2}$
$\lor$	$\dfrac{\phi}{\phi \lor \psi}\lor_{i_1} \qquad \dfrac{\psi}{\phi \lor \psi}\lor_{i_2}$	$\dfrac{\phi \lor \psi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}\lor_e$
$\Rightarrow$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \Rightarrow \psi}\Rightarrow_i$	$\dfrac{\phi \quad \phi \Rightarrow \psi}{\psi}\Rightarrow_e$

$\neg$	$\dfrac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \bot\end{array}}}{\neg\phi}\,\neg_i$	$\dfrac{\phi \quad \neg\phi}{\bot}\,\neg_e$
$\bot$	(não há)	$\dfrac{\bot}{\phi}\,\bot_e$
$\neg\neg$	(regra derivada)	$\dfrac{\neg\neg\phi}{\phi}\,\neg\neg_e$

Exemplo

Mostre que $\{p \land q, r\} \vdash q \land r$.

Resolução:

1.	$p \land q$	premissa	
2.	r	premissa	
3.	q	$\land_{e_2}$	1
4.	$q \land r$	$\land_i$	3,2

Regras deriváveis

Podemos criar novas regras de inferência a partir das regras da tabela anterior

$$\frac{\phi \Rightarrow \psi \quad \neg\psi}{\neg\phi}\,\text{MT}$$

$$\frac{\phi}{\neg\neg\phi}\;\neg\neg_i$$

$$\frac{}{\phi \vee \neg\phi}\;\text{LTE}$$

Correção e completude

Teorema (da correção)

Sejam $\psi_1, \ldots, \psi_n, \phi$ *fórmulas do cálculo proposicional e* $\Gamma = \{\psi_1, \ldots, \psi_n\}$.
Se $\Gamma \vdash \phi$, *então tem-se* $\Gamma \models \phi$.

Teorema (da completude)

Sejam $\psi_1, \ldots, \psi_n, \phi$ *fórmulas do cálculo proposicional e* $\Gamma = \{\psi_1, \ldots, \psi_n\}$.
Se $\Gamma \models \phi$, *então tem-se* $\Gamma \vdash \phi$.

- Estes dois teoremas dizem-nos que a abordagem semântica e a abordagem por meio da dedução natural são equivalentes no cálculo proposicional

Lógica

Lógica de Primeira Ordem

Introdução

Apesar de útil, o cálculo proposicional tem as suas limitações:

- Como fazer raciocínios sobre expressões como $x \geq 0$ utilizando o cálculo de proposicional?
- Como lidar com quantificadores sobre outros domínios? Por exemplo, a seguinte afirmação tem um valor lógico?

$$\forall x, y \in \mathbb{R} \quad x + y = y + x$$

- Por esta razão, precisamos de linguagens lógicas que sejam mais expressivas do que o cálculo proposicional.
- Nesta secção iremos estudar a lógica de primeira ordem, também conhecida como *cálculo de predicados* (ou ainda como como *cálculo de predicados de primeira ordem*).

Definição

O alfabeto de uma linguagem de primeira ordem $\mathcal{L}$ *é constituído pelos seguintes elementos:*

1. *Variáveis* $x_1, x_2, \ldots$ *(também podemos designar as variáveis por letras minúsculas* $p, q, r, \ldots$*), sendo geralmente assumido que conjunto das variáveis é infinito;*
2. *Símbolos de constante* $c_1, c_2, \ldots$*;*
3. *Símbolos de função* $f_1^n, f_2^n, \ldots$*, para todo o* $n \in \mathbb{N}$*, onde* n *é a aridade (*$=$ *número de argumentos de uma função);*
4. *Símbolos de predicados:* $p_1^n, p_2^n, \ldots$*, para todo o* $n \in \mathbb{N}$*, onde* n *é a aridade;*
5. *Símbolos de pontuação: "(", ")", e ",";*
6. *Conetivos:* $\neg$ *e* $\Rightarrow$*;*
7. *O quantificador* $\forall$*.*

Linguagem de primeira ordem

Definição

O conjunto $Term(\mathcal{L})$ *dos termos de uma linguagem de primeira ordem* $\mathcal{L}$*, é o conjunto de todas as cadeias de símbolos definidas pelas seguintes regras:*

1. *Variáveis e constantes individuais são termos de* $\mathcal{L}$*;*
2. *Se* f_i^n *é um símbolo de função associado a* $\mathcal{L}$ *e* $t_1, \ldots, t_n$ *são termos de* $\mathcal{L}$*, então* $f_i^n(t_1, \ldots, t_n)$ *é um termo de* $\mathcal{L}$*.*

Definição

Se p_i^n é um símbolo de predicado de $\mathcal{L}$ e $t_1, \ldots, t_n$ são termos de $\mathcal{L}$, então $p_i^n(t_1, \ldots, t_n)$ é uma fórmula atómica de $\mathcal{L}$. As fórmulas de $\mathcal{L}$ são exatamente todas as cadeias de símbolos definidas pelas seguintes regras:

1. *Toda a fórmula atómica de $\mathcal{L}$ é uma fórmula de $\mathcal{L}$;*
2. *Se ϕ e ψ são fórmulas de $\mathcal{L}$, também o são $(\neg\phi)$, $(\phi \Rightarrow \psi)$ e $(\forall x_i \phi)$, onde x_i é uma variável.*

1. $(\exists x_i)\phi$ é uma abreviatura para $\neg(\forall x_i)\neg\phi$;
2. $\phi \wedge \psi$ é uma abreviatura para $\neg(\phi \Rightarrow \neg\psi)$;
3. $\phi \vee \psi$ é uma abreviatura para $\neg\phi \Rightarrow \psi$;
4. $\phi \Leftrightarrow \psi$ é uma abreviatura para $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$

Definição

- *Na fórmula* $\forall x_i\, \phi$, *dizemos que* ϕ *é o âmbito do quantificador. Mais geralmente, se* $(\forall x_i\, \phi)$ *é uma sub-fórmula de* ψ, *então dizemos que o âmbito deste quantificador* $(\forall x_i)$ *é a fórmula* ϕ*.*
- *Se uma ocorrência da variável* x_j *não ocorre no âmbito de um quantificador* $\forall x_j$, *essa ocorrência diz-se livre, e caso contrário ligada ou muda (não são consideradas as variáveis que são utilizadas para definir um quantificador).*
- *Uma fórmula diz-se fechada se todas as ocorrências de variáveis são ligadas.*

Semântica na lógica de primeira ordem

Definição

Uma interpretação (ou modelo) I *de uma linguagem de primeira ordem* $\mathcal{L}$ *é um tuplo* (D_I, C_I, F_I, P_I)*, onde:*

1. D_I *é um conjunto não-vazio, chamado de domínio da interpretação;*
2. $C_I = ([c_1]_I, [c_2]_I, \ldots)$ *é uma coleção de constantes em* D_I*, i.e.* $[c_i]_I \in D_I$ *para todo o* $i \in \mathbb{N}$*;*
3. F_I *é uma coleção de funções* $[f_i^n]_I : D_I^n \to D_I$*, onde* $i, n \in \mathbb{N}$*;*
4. P_I *é uma coleção de predicados* $[p_i^n]_I \subseteq D_I^n$*, onde* $i, n \in \mathbb{N}$ *(nota: se* $x \in D_I^n$ e $x \in [p_i^n]_I$*, então diz-se que* $[p_i^n]_I(x)$ *é verdadeira. Se* $x \notin [p_i^n]_I$*, então diz-se que* $[p_i^n]_I(x)$ *é falsa).*

Definição

Uma valoração em I é uma função $v : Term(\mathcal{L}) \to D_I$ com as seguintes propriedades:

(i) $v(c_i) = [c_i]_I$ *para todo o* $i \in \mathbb{N}$*;*

(ii) $v(f_i^n(t_1, \ldots, t_n)) = [f_i^n]_I(v(t_1), \ldots, v(t_n))$*, para todos os* $i, n \in \mathbb{N}$*, e para quaisquer termos* $t_1, \ldots, t_n$ *de* $\mathcal{L}$*.*

Definição

Dada uma valoração v e um $d \in D_I$, $v[x_i \leftarrow d]$ designa uma valoração satisfazendo

$$v[x_i \leftarrow d](x_j) = \begin{cases} v(x_j) & \text{se } i \neq j \\ d & \text{se } i = j. \end{cases}$$

Por outras palavras $v[x_i \leftarrow d]$ é idêntica a v, exceto em x_i onde retorna o valor d.

Definição

Seja ϕ uma fórmula de $\mathcal{L}$ e seja v uma valoração em I, onde I é uma interpretação. Diz-se que uma valoração v satisfaz ϕ, denotado por $\models_I^v \phi$ (muitas vezes escreveremos simplesmente $\models^v \phi$ pois a uma valoração está sempre subjacente uma interpretação), se (definição por indução na estrutura de ϕ):

a) $\phi = p_i^n(t_1, \ldots, t_n)$ *é uma fórmula atómica e* $[p_i^n]_I(v(t_1), \ldots, v(t_n))$ *é verdadeira;*

b) $\models^v \neg\phi$ *se e só se* $\not\models_I^v \phi$;

c) $\models^v \phi \Rightarrow \psi$ *se e só se* $\not\models^v \phi$ *ou* $\models^v \psi$;

d) $\models^v \forall x_i \phi$ *se e só se* $\models^{v[x_i \leftarrow d]} \phi$ *para todo o* $d \in D_I$.

É fácil verificar da definição anterior e das definições de $\vee, \wedge$ e de $\Leftrightarrow$, que também se tem:

1. $\models^v \phi \vee \psi$ se e só se $\models^v \phi$ ou $\models^v \psi$;
2. $\models^v \phi \wedge \psi$ se e só se$\models^v \phi$ e $\models^v \psi$;
3. $\models^v \phi \Leftrightarrow \psi$ se e só se ($\models^v \phi$ se e só se $\models^v \psi$).

Teorema

$\models^v (\exists x_i)\phi$ *se e só se existe um* $d \in D_I$ *tal que* $\models^{v[x_i \leftarrow d]} \phi$.

Definição

Uma fórmula ϕ diz-se válida numa interpretação I, o que se denota por $\models_I \phi$, se para toda a valoração v em I se tem $\models_I^v \phi$. Neste caso diz-se que I é um modelo para ϕ.

Definição

Uma fórmula ϕ diz-se logicamente válida, escrevendo-se $\models \phi$, se $\models_I \phi$ para toda a interpretação I.

Definição

Uma tautologia em $\mathcal{L}$ é uma fórmula obtida a partir de uma tautologia do cálculo proposicional, substituindo as suas variáveis por elementos de $\mathcal{L}$.

Teorema

Toda a tautologia em $\mathcal{L}$ é logicamente válida.

Fórmulas logicamente equivalentes

Definição

Sejam ϕ e ψ duas fórmulas. Se $\models_I^v \phi$ sse $\models_I^v \psi$ para toda a valoração v em I, onde I é uma interpretação arbitrária, então diz-se que ϕ e ψ são logicamente equivalentes, escrevendo-se $\phi \equiv \psi$.

Alguns exemplos:

$$\forall x \phi(x) \equiv \neg \exists x \neg \phi(x) \quad \text{(relação entre } \forall \text{ e } \exists\text{)}$$
$$\exists x \phi(x) \equiv \neg \forall x \neg \phi(x)$$

Teorema

Para quaisquer fórmulas ϕ e ψ, tem-se

$$\phi \equiv \psi \quad \textit{sse} \quad \models \phi \Leftrightarrow \psi$$

Consequência semântica

Definição

Seja ψ *uma fórmula e* Γ *um conjunto de fórmulas. Diz-se que* ψ *é consequência semântica de* Γ*, escrevendo-se* $\Gamma \models \psi$*, se para toda a interpretação* I *e para toda a valoração* v *em* I *tal que* $\models_I^v \phi$ *para todo o* $\phi \in \Gamma$*, se tem* $\models_I^v \psi$.

Fórmulas prenexas

Definição

Uma fórmula ϕ de uma linguagem de primeira ordem diz-se em forma prenexa se é escrita como

$$Q_1 x_{i_1} \ldots Q_n x_{i_n} \psi,$$

onde cada $Q_1, \ldots, Q_n$ é o quantificador $\forall$ ou $\exists$, e ψ é uma fórmula sem quantificadores, designada de matriz.

Teorema

Para toda a fórmula ϕ de uma linguagem de primeira ordem $\mathcal{L}$, existe uma fórmula prenexa ψ tal que $\phi \equiv \psi$.

Dedução natural

- As regras da dedução natural para a lógica de primeira ordem funcionam de forma semelhante ao caso da dedução natural para o cálculo proposicional.
- Em particular, continuamos a utilizar todas as regras de inferência anteriores (incluindo as derivadas)
- No entanto acrestamos mais algumas regras para lidar com os quantificadores

Definição

Uma variável x_i diz-se livre para um termo t numa dada fórmula ϕ, se toda a ocorrência livre de x_i nunca ocorre no âmbito de um quantificador cuja variável de quantificação é uma variável que aparece na expressão do termo t.

Definição

Dada uma variável x, um termo t, e uma fórmula ϕ, definimos $\phi[t/x]$ como sendo a fórmula obtida substituindo cada ocorrência livre de x em ϕ por t, **desde que a variável x seja livre para o termo t**.

Novas regras de inferência

	Introdução	Eliminação
$\forall$	$\dfrac{\boxed{\begin{array}{l} y \\ \quad\vdots \\ \phi[y/x] \end{array}}}{\forall x\,\phi}\ \forall x_i$	$\dfrac{\forall x\,\phi}{\phi[t/x]}\ \forall x_e$
$\exists$	$\dfrac{\phi[t/x]}{\exists x\,\phi}\ \exists x_i$	$\dfrac{\exists x\,\phi \qquad \boxed{\begin{array}{l} y \quad \phi[y/x] \\ \quad\vdots \\ \quad\chi \end{array}}}{\chi}\ \exists x_e$

Tabela: Novas regras de inferência para quantificadores.

Igualdade na lógica de primeira ordem

	Introdução	Eliminação
$=$	$\dfrac{}{t = t} =_i$	$\dfrac{t_1 = t_2 \quad \phi[t_1/x]}{\phi[t_2/x]} =_e$

Tabela: Regras de inferência para a igualdade na lógica de primeira ordem.

Correção e completude

Teorema (correção e completude)

Sejam $\psi_1, \ldots, \psi_n, \phi$ *fórmulas de uma linguagem de primeira ordem e* $\Gamma = \{\psi_1, \ldots, \psi_n\}$. *Tem-se que* $\Gamma \vdash \phi$ *se e só se* $\Gamma \models \phi$.

Teoria da Computação

Computabilidade

- Com a teoria da computação pretende-se compreender melhor o que é o termo "computação" e, em particular, pretende-se definir este termo formalmente.
- Com a *computabilidade* pretende-se perceber o que pode ser calculado/computado através de algoritmos/modelos de computação, sem prestarmos atenção à quantidade de recursos utilizados

Preliminares

Definição

- *Um alfabeto é um conjunto finito e não-vazio de símbolos*
- *Uma palavra sobre um alfabeto* Σ *é uma sequência finita de símbolos de* Σ
- *O comprimento de uma palavra* w *sobre* Σ *é o número de símbolos presentes em* w*, denotado por* $|w|$
- *A palavra vazia, denotada de* ε*, é a palavra com zero ocorrências de símbolos*

$$\Sigma^k = \{a_1 \dots a_k | a_i \in \Sigma, \text{ para } i = 1, \dots, k\}.$$

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

$$w^k = \underbrace{ww\ldots w}_{k \text{ vezes}}.$$

Definição

Dadas duas palavras $x = x_1 \ldots x_k$, $y = y_1 \ldots y_m$ *sobre* Σ, *definimos a sua concatenação* $x \circ y$ *(também representada simplesmente como* xy*) como sendo a palavra*

$$x_1 \ldots x_k y_1 \ldots y_m.$$

Definição

Dada uma palavra $x = x_1 \ldots x_k$, *a sua palavra reversa é* x *escrita ao contrário, i.e. É* $x^{\mathcal{R}} = x_k \ldots x_1$. *Se* $x = x^{\mathcal{R}}$, x *diz-se um palíndromo.*

Linguagens

Definição

Uma linguagem sobre um alfabeto Σ *é um subconjunto* L *de* Σ^**.*

Definição

Sejam A *e* B *linguagens sobre o alfabeto* Σ*. Definimos as seguintes operações cujo resultado são também linguagens sobre* Σ*.*

- **União**: $A \cup B = \{x \in \Sigma^* | x \in A \text{ ou } x \in B\}$.
- **Concatenação**: $A \circ B = \{xy \in \Sigma^* | x \in A \text{ e } y \in B\}$.
- **Operador de fecho**: $A^* = \{x_1 x_2 \dots x_k \in \Sigma^* | k \geq 0 \text{ e } x_i \in A\}$.

Máquinas de Turing

- As máquinas de Turing fornecem o modelo teórico para os computadores

Definição

Uma máquina de Turing (MT) é um 7-tuplo $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *onde:*

1. Q *é um conjunto finito (de estados),*
2. Σ *é um alfabeto (dos inputs),*
3. Γ *é um alfabeto (da fita), contendo o símbolo especial* B *(o símbolo branco), e satisfazendo* $\Sigma \subseteq \Gamma$*,*
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{D, E, P\}$ *é a função de transição,*
5. $q_0 \in Q$ *é o estado inicial,*
6. $F \subseteq Q$ *é o conjunto dos estados finais.*

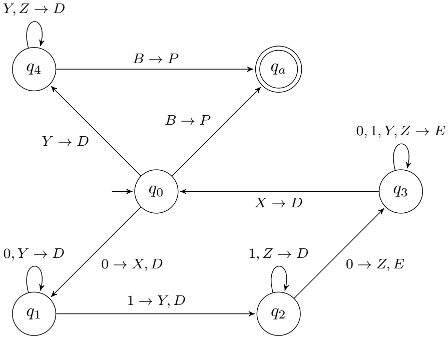
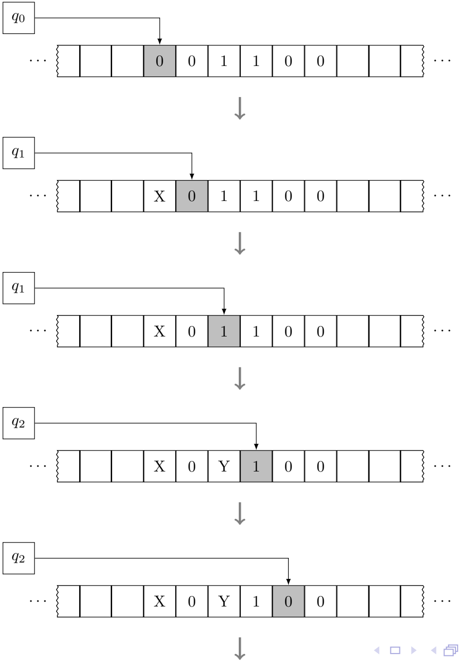


Figura: Um exemplo de máquina de Turing.


q_0
… 0 0 1 1 0 0 …
q_1
… X 0 1 1 0 0 …
q_1
… X 0 1 1 0 0 …
q_2
… X 0 Y 1 0 0 …
q_2
… X 0 Y 1 0 0 …

Configurações de máquinas de Turing

Definição

Uma configuração de uma MT é um triplo (v, q_i, w), *onde:*

- q_i *é o estado atual da computação*
- v *é a palavra que vai do primeiro símbolo não branco da fita até ao símbolo que está na célula imediatamente à esquerda da cabeça de leitura*
- w *é a palavra que vai da célula atualmente lida pela cabeça de leitura até à célula mais à direita da fita que não seja branca.*

Definição

Uma configuração (v, q_i, w) *diz-se*

- *Aceitadora se* q_i *é um estado final*
- *Rejeitadora, se a esta configuração não se lhe segue mais nenhuma*
- *De paragem, se é uma configuração aceitadora ou rejeitadora*

Definição

Uma máquina de Turing aceita a palavra $w \in \Sigma^*$ *se existe uma sequência de configurações* $C_1, \ldots, C_k$ *tal que:*

1. C_1 *é a configuração inicial de* M *para a palavra* w,
2. *A regra de transição de* M *determina que a cada configuração* C_i *se siga a configuração* C_{i+1},
3. C_k *é uma configuração aceitadora.*

A linguagem de uma máquina de Turing M *(ou linguagem aceite por* M*) é a classe*

$$L(M) = \{w \in \Sigma^* | M \text{ aceita } w\}.$$

Definição

Uma linguagem diz-se recursivamente enumerável (r.e.) se é a linguagem de alguma máquina de Turing.

Definição

Uma linguagem L *diz-se recursiva se é a linguagem de alguma máquina de Turing* M*, com a propriedade adicional que, para qualquer* $w \in \Sigma^*$*, a computação de* M *com input* w *tem sempre de acabar numa configuração de paragem. Neste caso dizemos também que* M *decide* L*.*

	O que faz a MT M com o *input* w?	
	Se M aceita L	Se M decide L
$w \in L$	M com *input* w acaba por parar e aceitar w	M com *input* w acaba por parar e aceitar w
$w \notin L$	M com *input* w: • Acaba por parar e rejeitar w ou • Nunca para	M com *input* w acaba por parar e rejeitar w

Tabela: Quadro que resume as diferenças entre *aceitar* e *decidir* uma linguagem L.

Definição

Uma função $f : \Sigma^* \to \Sigma^*$ *é computada por uma máquina de Turing* M *se, para cada input* $w \in \Sigma^*$, M *retorna o valor de* $f(w)$ *do seguinte modo:*

1. *Inicialmente a cabeça de leitura está a ler o símbolo mais à esquerda do input* w. *Para além do input, a fita só tem brancos.*
2. *Depois a computação é iniciada, e continua até parar com uma configuração aceitadora.*
3. *Quando a máquina para, a fita conterá somente o resultado* $f(w)$ *(o resto da fita só terá símbolos brancos), estando a cabeça de leitura a ler o símbolo mais à esquerda de* $f(w)$.

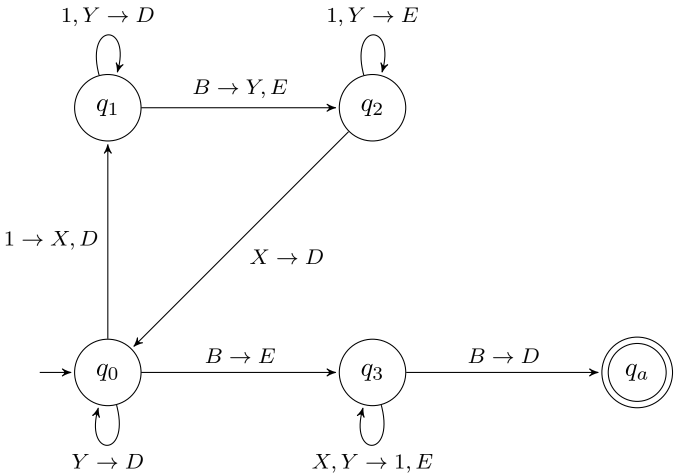


Figura: Máquina de Turing que duplica o tamanho do *input*, quando este é constituído só por 1's.

Tese de Church-Turing

Tese de Church-Turing: *Uma função é "efectivamente computável" se e só se pode ser computada através de uma máquina de Turing.*

Por uma função "efectivamente computável" entende-se qualquer função que seja "naturalmente considerada computável", no sentido que essa função pode ser calculada por um algoritmo que tenha as seguintes propriedades:

- O algoritmo consiste num número finito de instruções simples e precisas, que são descritas com um número finito de símbolos;
- O algoritmo pode, em princípio, ser executado por um ser humano com apenas papel e lápis:
- A execução do algoritmo não requer inteligência do ser humano além do necessário para entender e executar as instruções.

Outros tipos de máquinas de Turing:

- Máquinas de Turing com 2 ou mais fitas
- Máquinas de Tuing com fitas bidimensionais, tridimensionais, etc.
- Máquinas de Turing não-determinísticas

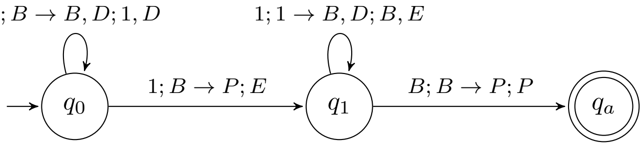


Figura: Uma máquina de Turing com duas fitas.

Definição

Uma máquina de Turing não-determinística é um 7-tuplo $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *definido como no caso da máquina de Turing, com a exceção que* $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q \times \Sigma \times \{D, E, P\})$*, onde* $\mathcal{P}(A) = \{X | X \subseteq A\}$.

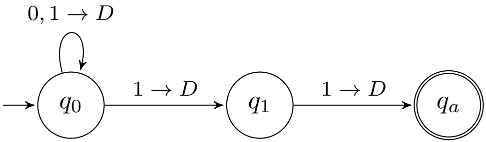


Figura: Exemplo de máquina de Turing não-determinística.

Estas máquinas são todas equivalentes, do ponto de vista computacional, à máquina de Turing com 1 fita. Por exemplo:

Teorema

Se $L(M)$ é a linguagem aceite (decidida) por uma máquina de Turing não-determinística M, então existe uma máquina de Turing que também aceita (decide) $L(M)$.

O jogo da vida

Este jogo pretende simular vida artificial. Consiste em utilizar uma grelha bidimensional infinita formada por células, em que cada célula pode estar viva ou morta. Fixa-se um estado incial para cada célula, e depois a evolução do sistema é feita passo a passo, de acordo com as seguintes regras:

- Qualquer célula viva com menos de dois vizinhos vivos morre de solidão.
- Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.
- Qualquer célula morta com exatamente três vizinhos vivos torna-se uma célula viva por reprodução.
- Qualquer célula viva com dois ou três vizinhos vivos continua no mesmo estado na próxima geração.

Link para simulador online

Problema

Será que é possível criar um programa *JogodaVida* que recebe como *input* uma configuração inicial e a posição de uma célula, e retorna:

- `Verdadeiro`, se esta célula irá ficar viva nalgum momento;
- `Falso`, se esta célula estará morta em todos os instantes

- A resposta a esta questão é negativa: não existe nenhum programa *JogodaVida* que satisfaz estas condições. Diz-se que é um exemplo de um problema *indecidível*.
- Nesta secção vamos analisar este tipo de questão com mais cuidado.

Indecibilidade do Problema da Paragem

Teorema (O problema da paragem é indecidível)

A linguagem definida por:

$$H_{paragem} = \{\langle M, w\rangle \mid \text{a máquina de Turing } M \text{ para com o input } w\}$$

é recursivamente enumerável, mas não é recursiva.

Algoritmo 1 Programa P

Require: String binária w

1: **function** $\mathrm{P}(w)$
2:     **if** $ProblemaParagem(w, w) = 0$ **then**
3:         **return** 1 ▷ Aceita o *input*
4:     **else**
5:         **while** $0 \neq 1$ **do** ▷ Entra num ciclo infinito
6:             loop=1
7:         **end while**
8:     **end if**
9: **end function**

Comentários:

- **Linha 2:** O programa (rotina) que computa a função *ProblemaParagem* é-nos fornecida previamente. Este programa recebe duas palavras binárias u e w como *input*. Se u codifica uma máquina de Turing M, então $ProblemaParagem(u, w)$ retorna 1 se $\langle M, w\rangle \in H_{paragem}$, e retorna 0 caso contrário.

Indecibilidade na lógica de primeira ordem

Teorema (Church)

Não é possível determinar através de um algoritmo (mais concretamente, através de uma máquina de Turing) se uma fórmula da lógica de primeira ordem é logicamente válida ou não (e, por dualidade, também não é possível decidir se a fórmula é satisfazível). Formalmente este problema diz-se indecidível.

Exemplos de outras linguagens indecidíveis

Teorema

As seguintes linguagens são indecidíveis:

- *A linguagem*

$$A_{TM} = \{\langle M, w\rangle |\ a\ máquina\ de\ Turing\ M\ aceita\ o\ input\ w\}$$

- *A linguagem*

$$NOTEMPTY_{TM} = \{\langle M\rangle : M\ é\ uma\ MT\ e\ L(M) \neq \varnothing\}$$

- *A linguagem*

$$EQ_{TM} = \{\langle M_1, M_2\rangle : M_1\ e\ M_2\ são\ MTs\ e\ L(M_1) = L(M_2)\}$$

Teoria da Computação

Complexidade Computacional

- O objetivo da complexidade computacional é refinar a teoria da computabilidade, e perceber o que é possível calcular com recursos (por exemplo, tempo de computação, memória, número de linhas de um programa) limitados
- Para isso necessitamos de medidas para medir a complexidade de uma linguagem

Definição

Sejam $f, g : \mathbb{N} \to \mathbb{N}$ *duas funções. Dizemos que* g *é um limite superior (assimptótico) para* f*, e escreve-se* $f \in O(g)$*, se existem inteiros* c, n_0 *tais que para todo o inteiro* $n \geq n_0$ *se tem*

$$f(n) \leq cg(n).$$

Definição

Seja M *uma máquina de Turing que para em todos os inputs. O tempo de execução de* M *é uma função* $f : \mathbb{N} \to \mathbb{N}$*, onde* $f(n)$ *é o maior número de passos que* M *utiliza para concluir uma computação com um input de tamanho* n*. Definimos também a seguinte classe*

$$TIME(f) = \{L | L \text{ é uma linguagem decidida em tempo } O(f) \text{ por alguma } MT\}$$

Definição

Seja M *uma máquina de Turing não-determinística que para em todos os ramos de computação para qualquer input. O tempo de execução de* M *é uma função* $f : \mathbb{N} \to \mathbb{N}$, *onde* $f(n)$ *é o maior número de passos que* M *utiliza para concluir um ramo de computação para um input de tamanho* n. *Definimos também a seguinte classe:*

$$NTIME(f) = \{L | L \text{ é uma linguagem decidida em tempo } O(f) \text{ por uma } MTND\}.$$

Teorema

Seja $f : \mathbb{N} \to \mathbb{N}$ *uma função satisfazendo* $f(n) \geq n$ *para todo o* $n \in \mathbb{N}$. *Então* $NTIME(f) \subseteq TIME(2^{O(f)})$.

Teorema

Seja $f : \mathbb{N} \to \mathbb{N}$ *uma função satisfazendo* $f(n) \geq n$ *para todo o* $n \in \mathbb{N}$*. Se uma linguagem* L *é decidida em tempo* $f(n)$ *por uma máquina de Turing com* k *fitas, então* L *é decidida em tempo* $O(f^2(n))$ *por uma máquina de Turing com uma só fita.*

As classes P e NP

Definição

$$P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$$

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

$$EXPTIME = \bigcup_{k \in \mathbb{N}} TIME(2^{n^k})$$

$$P \subseteq NP \subseteq EXPTIME.$$

$$UNARY - RELPRIME = \{1^x \# 1^y : x, y \in \mathbb{N} \text{ são primos entre si}\} \in P.$$

Algoritmo 2 Programa para calcular $mdc(x, y)$

Require: $x, y \in \mathbb{N}$

function $\mathrm{MDC}(x, y)$
    while $y \neq 0$ **do**
        $x = x \mod y$
        $aux = x$
        $x = y$
        $y = aux$
    end while
    return x
end function

Caracterização alternativa de NP

Teorema

$L \in NP$ *se e só se existe uma máquina de Turing* M, *cujo tempo de execução é polinomial (em* w*), tal que*

$$L = \{w \in \Sigma^* | M \text{ aceita } \langle w, c \rangle \text{ para algum } c \in \Sigma^*\}.$$

$$NOTPRIMES = \{\langle x \rangle : x \in \mathbb{N} \text{ não é um número primo}\} \in NP.$$

Algoritmo 3 Programa para verificar se x não é primo

Require: $x \in \mathbb{N}$
Require: $c \in \{0,1\}^*$ ▷ Certificado

```
function NotPrime(x, c)
    if c não codifica um número inteiro k then
        return 0                                  ▷ Rejeita
    else
        if n mod k = 0 then
            return 1                              ▷ Aceita
        else
            return 0                              ▷ Rejeita
        end if
    end if
end function
```

Require: $\langle G \rangle$, onde $G = (V, A)$ e $V = \{v_1, \dots, v_k\}$ é o conjunto dos vértices; s, t, onde $s, t \in V$; $c \in \{0, 1\}^*$ (certificado)

function TREE($\langle G, s, t \rangle$)
    if c codifica k vétices distintos $w_1, w_2, \dots, w_k$ de G **then**
        if $w_1 \neq s$ ou $w_k \neq t$ **then**
            return 0 ▷ Rejeita
        end if
        for $i = 1$ to $k - 1$ **do**
            if $\{w_i, w_{i+1}\}$ não é aresta de G **then**
                return 0 ▷ Rejeita
            end if
        end for
        return 1 ▷ Aceita
    else
        return 0 ▷ Rejeita
    end if
end function

Problemas NP-completos

- Os problemas NP-completos pretendem ser os problemas mais "difíceis" de NP
- Pensa-se que $P \neq NP$ e que portanto os problemas NP-completos não estão em P

Definição

Uma função $f : \Sigma^* \to \Sigma^*$ *diz-se computável em tempo polinomial se existe uma máquina de Turing que computa* f *em tempo polinomial (i.e. o tempo de execução da MT é* $O(n^k)$ *para algum* $k \in \mathbb{N}$*).*

Definição

Uma linguagem A *é redutível em tempo polinomial a uma linguagem* B*, escrito como* $A \leq_P B$*, se existe uma função* $f : \Sigma^* \to \Sigma^*$ *computável em tempo polinomial, tal que para todo o* $w \in \Sigma^*$*,*

$$w \in A \quad \text{se e só se} \quad f(w) \in B.$$

Teorema

Se $A \leq_P B$ e $B \in P$*, então* $A \in P$*.*

Definição

Uma linguagem B é NP-completa se satisfaz as seguintes condições:

1. $B \in NP$;
2. *Todo o $A \in NP$ é redutível em tempo polinomial para B.*

Teorema

Se B é NP-completo e $B \in P$, então $P = NP$.

Teorema

Seja $\mathcal{F}_V$ o conjunto das fórmulas do cálculo proposicional. Então a seguinte linguagem é NP-completa

$$SAT = \{\langle\phi\rangle \text{ onde } \phi \in \mathcal{F}_V \mid \phi \text{ é satisfazível}\}.$$

Teorema

Se B é NP-completo e $B \leq_P C$, onde $C \in NP$, então C é NP-completo.

Definição

Uma linguagem L diz-se NP-difícil (NP-hard em inglês) se todo o $A \in NP$ é redutível em tempo polinomial para L (a diferença em relação a linguagens NP-completas é que L não tem necessariamente de pertencer a NP).

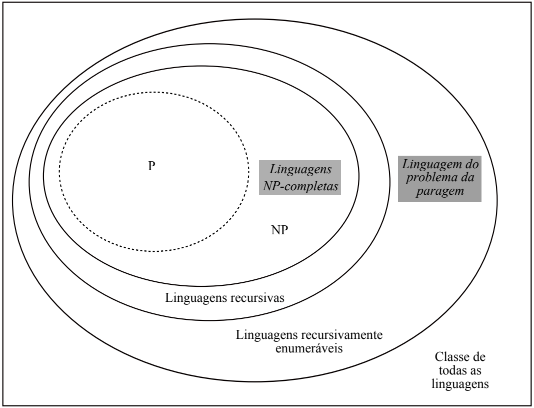


Figura: Relação entre as principais linguagens estudadas nesta disciplina.