Computabilidade

2012/2013

Sabine Broda

Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

Capítulo 1

Computabilidade

1.1 A noção de computabilidade

Um processo de computação é qualquer método algorítmico que opera sobre zero ou mais elementos de um conjunto de objectos iniciais seguindo para isso um conjunto fixo de regras e produzindo uma sequência de zero ou mais objectos finais. Aos objectos iniciais e finais chamam-se respectivamente o input e o output e ao conjunto de regras o algoritmo do processo de computação. Note-se que um processo de computação pode não parar para um certo input, produzindo por exemplo uma sequência infinita de objectos ou mesmo sem produzir nada.

Um alfabeto A é um conjunto não vazio de símbolos. Uma sequência finita, possivelmente vazia, de símbolos de A chama-se uma palavra de alfabeto A. O comprimento de uma palavra α é o número de símbolos que ocorrem em α e denota-se por $|\alpha|$. A palavra de comprimento zero, i.e. constituída por nenhum símbolo é representada por ϵ. O conjunto de todas as palavras de um alfabeto A denota-se por $A^\star$. No que se segue consideramos somente alfabetos finitos. Para $A = \{a_0, \ldots, a_n\}$ podemos definir uma ordem em $A^\star$ que chamamos ordem lexicográfica (respectivamente à indexação

$a_0, \ldots, a_n$), onde uma palavra α precede a palavra β se e só se

$|\alpha| < |\beta|$ ou
$|\alpha| = |\beta|$ e existem $\gamma, \alpha_1, \beta_1 \in A^\star$ e $a_i, a_j \in A$ tais que $\alpha = \gamma a_i \alpha_1$, $\beta = \gamma a_j \beta_1$ e $i < j$.

Definição 1.1 *Sejam A e B alfabetos. Uma função f de $A^\star$ em $B^\star$ diz-se computável, se existir um processo de computação $\mathcal{P}$, tal que para cada $\alpha \in A^\star$ o processo $\mathcal{P}$, com α como input, pára depois de produzir o elemento correspondente $f(\alpha) \in B^\star$ como output.*

Definição 1.2 *Seja $W \subseteq A^\star$ um conjunto de palavras sobre um alfabeto A e $\mathcal{P}$ um processo de computação.*

1. *$\mathcal{P}$ diz-se um método de decisão para W, se para cada $\alpha \in A^\star$ o processo $\mathcal{P}$, com α como input, pára depois de produzir exactamente uma palavra de $A^\star$ como output, que é ϵ quando $\alpha \in W$ e uma qualquer palavra diferente de ϵ quando $\alpha \notin W$.*
2. *O conjunto W é decidível se existe um método de decisão para W.*

Definição 1.3 *Seja $W \subseteq A^\star$ um conjunto de palavras sobre um alfabeto A e $\mathcal{P}$ um processo de computação.*

1. *$\mathcal{P}$ diz-se um método de enumeração para W, se $\mathcal{P}$ produz exactamente as palavras de W como output, em qualquer ordem e possivelmente com repetições.*
2. *O conjunto W diz-se efectivamente enumerável se existe um método de enumeração para W.*

Exercício 1.1 *Sejam A_1 e A_2 alfabetos tais que $A_1 \subseteq A_2$ e seja $W \subseteq A_1^\star$. Mostre que W é decidível (resp. efectivamente enumerável) relativamente ao alfabeto A_1 se e só se o for relativamente ao alfabeto A_2.*

Em consequência deste resultado podemos deixar de fazer referência aos alfabetos considerados.

Proposição 1.4 *Todo o conjunto decidível é efectivamente enumerável.*

Prova Se $W \subseteq A^\star$ é decidível, então existe um método de decisão $\mathcal{P}$ para W. Basta então enumerar os elementos de $A^\star$, e usar o processo $\mathcal{P}$ para decidir se cada palavra enumerada α pertence ou não a W, e em caso afirmativo juntar a palavra à lista (que está a ser construída) dos elementos de W. •

Proposição 1.5 *Um subconjunto de $W \subseteq A^\star$ é decidível se e só se W e $A^\star \setminus W$ forem efectivamente enumeráveis.*

Prova Se W é decidível, então é efectivamente enumerável pela proposição anterior. A prova de que $A^\star \setminus W$ é efectivamente enumerável é análoga, sendo o output as palavras para as quais o método de decisão para W determinou não pertencerem a W.

Sejam $\mathcal{P}$ e $\mathcal{P}'$ dois processos que permitem respectivamente enumerar os elementos de W e de $A^\star \setminus W$. Para cada palavra $\alpha \in A^\star$, basta correr os dois processos simultaneamente e esperar que α apareça como output de um deles, o que acaba por acontecer já que α pertence a um dos dois conjuntos. No caso de α aparecer como output de $\mathcal{P}$, conclui-se que $\alpha \in W$. Caso contrário, i.e. se α aparecer como output de $\mathcal{P}'$, tem-se $\alpha \notin W$. •

1.2 Máquinas de registos

Nesta secção vamos definir uma linguagem de programação para escrever programas que serão executados por computadores cuja memória é dividida em unidades $R_0, \ldots, R_m$ e que se chamam registos. Em cada estado da computação cada registo

contém uma palavra de um alfabeto $A^\star$. A partir de agora consideremos um alfabeto fixo $A = \{a_0, \ldots, a_r\}$. Supomos que existem computadores com qualquer número (finito) de registos e que em cada registo se podem guardar palavras arbitrariamente grandes. Chamamos a estes computadores também máquinas de registos.

Um programa (sobre o alfabeto A) consiste num conjunto finito de instruções, cada uma das quais começa por um número natural N, o número da instrução. As instruções podem ser de uma das formas seguintes e têm o significado especificado entre os parenteses:

1. N `LET` $R_i = R_i + a_j$

   com $N, i, j \in \mathbb{N}$ e $j \leq r$ ("adiciona a letra a_j no fim da palavra no registo R_i");

2. N `LET` $R_i = R_i - a_j$

   com $N, i, j \in \mathbb{N}$ e $j \leq r$ ("se a palavra no registo R_i acabar em a_j, tira este a_j; caso contrário não faz nada");

3. N `IF` $R_i = \epsilon$ `THEN` N' `ELSE` N_0 `OR` ... `OR` N_r

   com $N, i, N', N_0, \ldots, N_r \in \mathbb{N}$ ("se a palavra no registo R_i for a palavra vazia ϵ vai para a instrução com número N', se acabar em a_0 vai para N_0,..., se acabar em a_r vai para N_r");

4. N `PRINT`

   com $N \in \mathbb{N}$ ("escreve a palavra guardada em R_0 como output");

5. N `HALT`

   com $N \in \mathbb{N}$ ("pára").

Definição 1.6 *Um programa (para uma máquina de registos) é uma sequência finita $\alpha_0, \ldots, \alpha_k$ de instruções da forma $1. - 5.$ tal que*

- *o número da instrução α_i é i, para $i = 0, \ldots, k$;*

- *cada instrução de salto refere-se a instruções cujo número é menor ou igual a k;*
- *só a ultima instrução α_k é a instrução de paragem.*

Cada programa determina um processo de computação (numa máquina cujo número de registos é igual ou superior ao número de registos referidos no programa) da seguinte forma: Supõe-se que no início da computação o registo R_0 contém o input, enquanto que todos os outros registos contêm a palavra vazia. A seguir é executada uma instrução de cada vez, seguindo a ordem das instruções no programa ou, caso a última instrução tenha sido de salto, saltando para a instrução correspondente. A máquina pára quando chegar à instrução de paragem α_k.

Quando um programa P começado com input $\alpha \in A^\star$ (i.e. tem α em R_0 no início do processo de computação) chega depois de um número finito de passos à instrução α_k e pára escrevemos

$$P : \alpha \to \texttt{halt};$$

caso contrário, i.e. se o programa não parar, escrevemos

$$P : \alpha \to \infty.$$

Se o programa P (com input α) parar depois de ter escrito exactamente uma palavra β como output, então escrevemos ainda

$$P : \alpha \to \beta.$$

Definição 1.7 *Sejam A e B alfabetos. Uma função f de $A^\star$ em $B^\star$ diz-se R-computável, se existir um programa P sobre $A \cup B$, tal que para cada $\alpha \in A^\star$, se tem $P : \alpha \to f(\alpha)$. Neste caso diz-se também que o programa P calcula a função f.*

Exemplo 1.8 *Considere os alfabetos $A = \{\texttt{a}\}$ e $B = \{\bullet\}$. A função $f : A^\star \longrightarrow B^\star$, definida por $f(\alpha) = \underbrace{\bullet \ldots \bullet}_{|\alpha|}$, é R-computável, pois pode ser calculada pelo programa*

seguinte.

0 IF $R_0 = \epsilon$ THEN 4 ELSE 1 OR 0

1 LET $R_0 = R_0 - \mathtt{a}$

2 LET $R_1 = R_1 + \bullet$

3 IF $R_0 = \epsilon$ THEN 4 ELSE 1 OR 3

4 IF $R_1 = \epsilon$ THEN 8 ELSE 4 OR 5

5 LET $R_1 = R_1 - \bullet$

6 LET $R_0 = R_0 + \bullet$

7 IF $R_1 = \epsilon$ THEN 8 ELSE 4 OR 8

8 PRINT

9 HALT

Definição 1.9 *Seja $W \subseteq A^\star$ e P um programa sobre A.*

1. *Diz-se que P decide W se para cada $\alpha \in A^\star$,*

$$\begin{array}{lll} P : \alpha \to \epsilon & & \text{se } \alpha \in W; \\ P : \alpha \to \beta & \text{com } \beta \neq \epsilon & \text{se } \alpha \notin W. \end{array}$$

2. *W diz-se R-decidível, se existir um programa para uma máquina de registos que decide W.*

Exemplo 1.10 *Considere o alfabeto $A = \{\mathtt{a}\}$ e o conjunto $W = \{\alpha \in A^\star : |\alpha| \text{ é par }\}$. O conjunto W pode ser decidido pelo programa seguinte.*

0 IF $R_0 = \epsilon$ THEN 6 ELSE 1

1 LET $R_0 = R_0 - \mathtt{a}$

2 IF $R_0 = \epsilon$ THEN 5 ELSE 3

3 LET $R_0 = R_0 - \mathtt{a}$

4 IF $R_0 = \epsilon$ THEN 6 ELSE 1

5 LET $R_0 = R_0 + \mathtt{a}$

6 PRINT

7 HALT

Definição 1.11 *Seja $W \subseteq A^\star$ e P um programa sobre A.*

1. *Diz-se que P enumera W se P, começado com input ϵ, escreve como output exactamente as palavras em W, em qualquer ordem e possivelmente com repetições.*

2. *W diz-se R-enumerável, se existir um programa para uma máquina de registos que enumera W.*

Exemplo 1.12 *Considere o alfabeto $A = \{\mathtt{a}, \mathtt{b}\}$ e o conjunto $W = \{\mathtt{a}^n\mathtt{b}^n \in A^\star \mid n \geq 0\}$. O conjunto W pode ser enumerado pelo programa seguinte.*

0 PRINT
1 IF $R_0 = \epsilon$ THEN 5 ELSE 5 OR 2
2 LET $R_0 = R_0 - \mathtt{b}$
3 LET $R_1 = R_1 + \mathtt{b}$
4 IF $R_0 = \epsilon$ THEN 5 ELSE 5 OR 2
5 LET $R_0 = R_0 + \mathtt{a}$
6 LET $R_0 = R_0 + \mathtt{b}$
7 IF $R_1 = \epsilon$ THEN 11 ELSE 7 OR 8

8 LET $R_1 = R_1 - \mathtt{b}$
9 LET $R_0 = R_0 + \mathtt{b}$
10 IF $R_1 = \epsilon$ THEN 11 ELSE 7 OR 8
11 PRINT
12 IF $R_0 = \epsilon$ THEN 1 ELSE 1 OR 1
13 HALT

1.3 Tese de Church

Vimos na secção anterior que cada programa para uma máquina de registos define um algoritmo que certamente corresponde à ideia intuitiva dada de um processo de

computação. De facto, a concepção do conceito de programa para máquinas de registos foi feita (por Shepherdson e Sturgis em 1963) de forma a satisfazer os pontos seguintes:

- desenhar um modelo de computação simples, sobre o qual é fácil de raciocinar;
- cada programa deve corresponder a um algoritmo;
- todo o método algorítmico deve poder ser descrito por um programa para uma máquina de registos.

Enquanto que os primeiros dois objectivos foram alcançados, não é possível verificar o terceiro ponto, uma vez que a nossa ideia do que é um processo de computação (ou método algorítmico) é intuitiva e não corresponde a uma definição (matemática) exacta. A afirmação da veracidade do terceiro ponto é no entanto geralmente aceite e chama-se a tese de Church[1].

Tese de Church: Qualquer processo de computação pode ser descrito por um programa para uma máquina de registos.

Mesmo que não seja possível provar a tese de Church, existem vários argumentos que a suportam:

- Até ao momento conseguiu-se simular qualquer processo de computação por um programa para uma máquina de registos. Em particular, podem-se traduzir os programas escritos em linguagens de programação existentes como o FORTRAN e o C para programas para máquinas de registos equivalentes.
- Desde 1930 foram propostos diversos modelos formais de computação usando os métodos mais variados. Todos estes modelos, entre os quais as máquinas de Turing, as funções recursivas, representações em sistemas de λ-calculus e os programas para máquinas de registos, são equivalentes.

[1]Note-se que Church formulou esta tese em 1935 para um modelo de computação diferente que se provou ser equivalente às máquinas de registos.

Na literatura, os conjuntos R-decidíveis ou R-enumeráveis designam-se frequentemente por conjuntos recursivos ou recursivamente enumeráveis. Da mesma forma diz-se ainda função recursiva em vez de R-computável. No que se segue vamos aceitar a tese de Church o que nos permite identificar os conceitos de decidibilidade com R-decidibilidade, etc.

1.4 O problema de paragem para MR's

Nesta secção vamos apresentar um conjunto W que não é R-decidível. Tomando ainda $A = \{a_0, \ldots, a_r\}$, cada palavra do conjunto $W \subseteq A^\star$ vai ser a representação de um programa (para máquinas de registos) sobre A. Consideremos para isso o alfabeto

$$B = A \cup \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}\} \cup \{0, 1, \ldots, 9\} \cup \{=, +, -, \epsilon, \#\} \tag{1.1}$$

e a ordem lexicográfica em $B^\star$ (respectivamente a uma ordem fixada para os símbolos em 1.1). Cada programa sobre A pode ser representado por uma palavra em $B^\star$ como exemplificado pelo programa P

$$\begin{array}{ll} 0 & \texttt{LET } R_1 = R_1 - a_0 \\ 1 & \texttt{PRINT} \\ 2 & \texttt{HALT} \end{array}$$

cuja representação é a palavra

$$\beta_P = 0\texttt{LETR1=R1}-a_0\#1\texttt{PRINT}\#2\texttt{HALT}.$$

Se esta palavra for a n-ésima palavra em ordem lexicográfica em $B^\star$, então representamos o programa P em $A^\star$ por

$$\alpha_P = \underbrace{a_0 \ldots a_0}_{n}$$

ao qual chamamos também número de Gödel de P. Consideremos agora o conjunto de todos os números de Gödel de programas sobre A

$$\Pi = \{\alpha_P \mid P \text{ é um programa sobre } A\}.$$

Existe certamente um algoritmo que dado um programa P calcula a palavra correspondente $\alpha_P \in A^\star$. Por outro lado, podemos determinar se uma palavra $\alpha \in A^\star$ é da forma $a_0 \dots a_0$, neste caso calcular a palavra correspondente em $B^\star$ e decidir se esta corresponde ou não a algum programa P. Em particular, (e usando a tese de Church) obtemos o seguinte.

Lema 1.13 *O conjunto* Π *é R-decidível.* •

Note-se que para os resultados seguintes não vamos usar nenhuma vez a definição específica da numeração de Gödel dada. De facto, qualquer processo efectivo de atribuir, de forma injectiva, palavras de $A^\star$ a programas, tal que o conjunto correspondente Π é ainda decidível, chama-se uma atribuição de números de Gödel a programas de registos e poderia ter sido usado.

Consideremos agora os seguintes subconjuntos (indecidíveis) de Π:

$$\Pi'_{\mathtt{halt}} = \{\alpha_P \mid P \text{ é um programa sobre } A \text{ e } P : \alpha_P \to \mathtt{halt}\}$$

e

$$\Pi_{\mathtt{halt}} = \{\alpha_P \mid P \text{ é um programa sobre } A \text{ e } P : \epsilon \to \mathtt{halt}\}.$$

Teorema 1.14 (Indecidibilidade do problema da paragem)

1. *O conjunto* $\Pi'_{\mathtt{halt}}$ *não é R-decidível;*
2. *O conjunto* $\Pi_{\mathtt{halt}}$ *não é R-decidível.*

Prova

1. Por redução ao absurdo, suponhamos que existe um programa P_0 para uma máquina de registos que decide o conjunto $\Pi'_{\mathtt{halt}}$. Então, para todo o programa

P tem-se

$$(1)\quad \begin{array}{lll} P_0 : \alpha_P \to \epsilon & & \text{se } P : \alpha_P \to \texttt{halt}; \\ P_0 : \alpha_P \to \beta & \text{com } \beta \neq \epsilon & \text{se } P : \alpha_P \to \infty. \end{array}$$

Vamos transformar P_0 de modo que não pare quando imprime a palavra vazia ϵ como output. Para isso basta substituir a instrução de paragem em P_0

$$k \ \texttt{HALT}$$

por

$$\begin{array}{ll} k & \texttt{IF } R_0 = \epsilon \texttt{ THEN } k \texttt{ ELSE } k+1 \texttt{ OR } \ldots \texttt{ OR } k+1 \\ k+1 & \texttt{HALT} \end{array}$$

e todas as instruções da forma

$$N \ \texttt{PRINT}$$

por

$$N \ \texttt{IF } R_0 = \epsilon \texttt{ THEN } k \texttt{ ELSE } k \texttt{ OR } \ldots \texttt{ OR } k$$

Chamando ao programa modificado P_1 segue então

$$(2)\quad \begin{array}{ll} P_1 : \alpha_P \to \infty & \text{se } P : \alpha_P \to \texttt{halt} \\ P_1 : \alpha_P \to \texttt{halt} & \text{se } P : \alpha_P \to \infty. \end{array}$$

Donde resulta que, para todo o programa P

$$(3)\quad P_1 : \alpha_P \to \infty \quad \text{sse } P : \alpha_P \to \texttt{halt}$$

e em particular tem-se para o programa $P = P_1$

$$(4)\quad P_1 : \alpha_{P_1} \to \infty \quad \text{sse} \quad P_1 : \alpha_{P_1} \to \texttt{halt}$$

o que é absurdo.

2. Para cada programa P pode-se, de modo efectivo, construir um programa P^+ tal que

$$P : \alpha_P \to \texttt{halt} \quad \text{sse} \quad P^+ : \epsilon \to \texttt{halt},$$

donde

$$(*) \quad \alpha_P \in \Pi'_{\texttt{halt}} \quad \text{sse} \quad \alpha_{P^+} \in \Pi_{\texttt{halt}}.$$

De facto, basta para isso calcular $\alpha_P = \underbrace{a_0 \dots a_0}_{n}$ e tomar para P^+ o programa cujas n primeiras linhas são

$$\begin{array}{ll} 0 & \texttt{Let } R_0 = R_0 + a_0 \\ \vdots & \\ n-1 & \texttt{Let } R_0 = R_0 + a_0 \end{array}$$

seguido das instruções do programa P, onde cada número de instrução foi incrementado de n. É então fácil ver que para cada programa P se verifica $(*)$.

Suponhamos agora que o conjunto $\Pi_{\texttt{halt}}$ é R-decidível, i.e. existe um programa P_0 que o decide. Então é possível construir um algoritmo de decisão para $\Pi'_{\texttt{halt}}$. De facto, para $\alpha \in A^\star$, é, pelo lema 1.13, possível decidir se α é o número de Gödel de algum programa P, i.e. se $\alpha \in \Pi$. Se não pertencer podemos concluir que $\alpha \notin \Pi'_{\texttt{halt}}$. Caso contrário, basta determinar o programa P cujo número de Gödel é $\alpha = \alpha_P$ e construir P^+. O programa P_0 decide então se $\alpha_{P^+} \in \Pi_{\texttt{halt}}$, i.e. se $\alpha_P \in \Pi'_{\texttt{halt}}$. •

Existe certamente um grande número de programas para os quais se pode decidir se, começados com a palavra vazia ϵ como input, vão ou não parar. Concluimos no entanto do resultado anterior que não existe nenhum método algoritmico que decide este problema para qualquer programa, i.e. que se possa usar para todos eles.

O lema seguinte mostra que $\Pi_{\texttt{halt}}$ é exemplo de um conjunto efectivamente enumerável que não é decidível.

Lema 1.15 *O conjunto* $\Pi_{\texttt{halt}}$ *é R-enumerável.*

Prova Pela tese de Church, basta-nos descrever um algoritmo de enumeração de $\Pi_{\texttt{halt}}$: para $n = 1, 2, 3, \dots$ geram-se todos os programas cujo número de Gödel é menor

ou igual a n e deixam-se os programas (um de cada vez) correr n passos. Juntam-se os números de Gödel dos programas que pararam à lista dos elementos de $\Pi_{\texttt{halt}}$. •

Corolário 1.16 *O conjunto $A^\star \setminus \Pi_{\texttt{halt}}$ não é R-enumerável.*

1.5 Indecidibilidade da lógica de primeira ordem

Seja $\mathcal{L}_\infty$ uma linguagem de primeira ordem cujo conjunto de símbolos não lógicos é constituído por um conjunto infinito enumerável de constantes $\mathcal{F}_0 = \{c_0, c_1, \ldots\}$ e para cada $n > 0$ por conjuntos infinitos enumeráveis de símbolos funcionais e relacionais n-ários $\mathcal{F}_n = \{f_0^n, f_1^n, \ldots\}$ e $\mathcal{R}_n = \{R_0^n, R_1^n, \ldots\}$.

Nesta secção vamos ver que não existe nenhum algoritmo que dada uma proposição φ de $\mathcal{L}_\infty$ determina se esta é ou não uma fórmula válida de $\mathcal{L}_\infty$ (ou de qualquer outra linguagem $\mathcal{L}$ cujo conjunto de símbolos não lógicos inclui os símbolos que ocorrem em φ).

Proposição 1.17 *O conjunto dos termos de $\mathcal{L}_\infty$ é decidível.*

Prova Consideremos o alfabeto finito

$$A_t = \{x, \underline{0}, \underline{1}, \ldots, \underline{9}, \overline{0}, \overline{1}, \ldots, \overline{9}, c, f\}.$$

Cada termo de $\mathcal{L}_\infty$ pode ser escrito como uma palavra de $A_t^\star$, tal como $f_{33}^2(x_{41}, f_5^1(c_{28}))$ que pode ser representado por $f\overline{2}\underline{33}x\underline{41}f\overline{1}\underline{5}c\underline{28}$. Vamos agora descrever um algoritmo para decidir se uma palavra $\alpha \in A_t^\star$ é ou não um termo de $\mathcal{L}_\infty$. Primeiro determina-se o comprimento $|\alpha|$ da palavra α. Se o comprimento for 0 ou 1, conclui-se que α não é termo. Para $|\alpha| \geq 2$, verifica-se se a primeira letra de α é c ou x. Neste caso α é um termo se e só se as restantes letras em α são todas da forma $\underline{n}$, com $n = 0, \ldots, 9$, mas a primeira é diferente de $\underline{0}$ se houver mais do que uma. Se a primeira letra de α não for nem c, nem x, nem f, então α não é termo. Finalmente, se $\alpha = f\beta$, então β tem

que ser da forma $\overline{n_k} \ldots \overline{n_0}\underline{m_l} \ldots \underline{m_0}\gamma$, onde $n_0, \ldots, n_k, m_0, \ldots, m_l \in \{0, \ldots, 9\}$, $n_k \neq 0$ se $k \neq 0$, $m_l \neq 0$ se $l \neq 0$, e γ não começa em $\underline{n}$, com $n = 0, \ldots, 9$. Neste caso basta verificar se é possível decompor γ em $\gamma_1 \ldots \gamma_p$, com $p = n_0 + \ldots + n_k \times 10^k$ e tal que $\gamma_1, \ldots, \gamma_p$ são também termos. Este processo vai necessariamente terminar, uma vez que $|\gamma| < |\alpha|$. •

Exercício 1.2 *Mostre que o conjunto das fórmulas de $\mathcal{L}_\infty$ é decidível.*

Corolário 1.18 *Os conjuntos de termos e fórmulas de $\mathcal{L}_\infty$ são efectivamente enumeráveis.*

Exemplo 1.19 *O conjunto das proposições válidas de $\mathcal{L}_\infty$ é efectivamente enumerável.*

Prova Pelo teorema de completude basta descrevermos um algoritmo que enumera todos as proposições de $\mathcal{L}_\infty$ que são teoremas. Podemos escolher um alfabeto finito A_∞ tal que todos as fórmulas de $\mathcal{L}_\infty$ se possam escrever como palavras de $A_\infty^\star$, cf. exercício 1.2. Consequentemente as deduções, que são sequências finitas de fórmulas de $\mathcal{L}_\infty$, são também palavras de A_∞. Podemos construir um algoritmo que para cada palavra α de A_∞ decide se α é uma dedução, i.e. um algoritmo que verifica se α é da forma $\alpha_1 \ldots \alpha_k$ tal que $\alpha_1, \ldots, \alpha_k$ corresponde a uma sequência de fórmulas que é uma dedução de α_k, e em caso afirmativo, determina a fórmula final α_k desta dedução. Por outro lado é certamente decidível se α_k corresponde ou não a uma proposição. Finalmente, note-se que existe um algoritmo de enumeração de $A_\infty^\star$. Aplicando o algoritmo de decisão descrito a cada uma das palavras enumeradas, basta-nos escrever para a lista das proposições válidas de $\mathcal{L}_\infty$ as fórmulas finais daquelas que são deduções de alguma proposição. •

Teorema 1.20 (Indecidibilidade da lógica de primeira ordem)

O conjunto das proposições válidas de $\mathcal{L}_\infty$ não é R-decidível.

Prova Consideremos $A = \{|\}$ e identificamos cada palavra $\alpha \in A^\star$ com o número natural $|\alpha|$. Do teorema 1.14 sabemos que o conjunto

$$\Pi_{\mathtt{halt}} = \{\alpha_P \in A^\star \mid P \text{ é um programa sobre } A \text{ e } P : \epsilon \to \mathtt{halt}\}$$

não é R-decidível. Vamos construir de modo efectivo para cada programa P sobre A uma proposição φ_P de $\mathcal{L}_\infty$ tal que

$$\models \varphi_P \qquad \text{sse} \qquad P : \epsilon \to \mathtt{halt}.$$

Como $\Pi_{\mathtt{halt}}$ não é R-decidível, concluimos o mesmo (justificar!) para o conjunto

$$\Phi_P = \{\varphi_P \mid P \text{ é um programa sobre } A \text{ e } \models \varphi_P\}$$

e consequentemente também para o conjunto das proposições válidas de $\mathcal{L}_\infty$

$$\Phi = \{\varphi \mid \varphi \text{ é uma proposição de } \mathcal{L}_\infty \text{ e } \models \varphi\}$$

pois $\Phi_P \subseteq \Phi$ e para $\varphi \in \Phi$ podemos decidir se $\varphi \in \Phi_P$ ou não.

Seja então P um programa para uma máquina de registos com instruções $\alpha_0, \ldots, \alpha_k$ e seja n o menor número natural tal que todos os registos referidos em P estão entre $R_0, \ldots, R_n$. Um $(n+2)$-tuplo de números naturais $(N, m_0, \ldots, m_n)$, com $N \leq k$, diz-se uma configuração de P. Diz-se que $(N, m_0, \ldots, m_n)$ é *a configuração de P ao fim de s passos*, se o programa P, começado com ϵ como input, executar pelo menos s passos e se depois de s passos a instrução N é a próxima a ser executada, tendo os registos $R_0, \ldots, R_n$ respectivamente os números $m_0, \ldots, m_n$. Em particular temos $(0, \ldots, 0)$ como configuração inicial de P e como α_k é a única instrução de paragem, tem-se

$$P : \epsilon \to \mathtt{halt} \quad \text{sse} \quad \begin{array}{l}\text{existem } s, m_0, \ldots, m_n \in \mathbb{N} \text{ tal que} \\ \qquad (k, m_0, \ldots, m_n) \text{ é a configuração de } P \text{ ao fim de s passos.}\end{array} \tag{1.5}$$

Se $P : \epsilon \to \mathtt{halt}$ seja ainda s_P o número de passos executados por P até chegar à instrução de paragem α_k.

Vamos agora especificar a construção de φ_P. Para isso escolhemos dos símbolos não lógicos de $\mathcal{L}_\infty$ dois símbolos relacionais R e $<$, respectivamente $(n+3)$-ário e binário,

um símbolo funcional unário f e uma constante c (por exemplo R_0^{n+3}, R_0^2, f_0^1 e c_0). Seja $\mathcal{L}$ a linguagem cujos símbolos não lógicos são R, $<$, f e c.

Vamos definir uma estrutura $\mathcal{A}_P$ de $\mathcal{L}$ em que podemos descrever o comportamento do programa P. Distinguimos entre dois casos:

1. Se $P : \epsilon \to \infty$, então tomamos $A_P = \mathbb{N}$ como domínio de $\mathcal{A}_P$, interpretamos $<$ como a ordem usual em $\mathbb{N}$, c como 0, f como a função sucessor e

   $$R^{\mathcal{A}_P} = \{(s, N, m_0, \ldots, m_n) \mid (N, m_0, \ldots, m_n) \text{ é a configuração de } P \text{ ao fim de } s \text{ passos}\}.$$

2. Se $P : \epsilon \to \texttt{halt}$, então sejam $e = max\{k, s_P\}$ e $A_P = \{0, \ldots, e\}$. Interpretamos $<$ como a ordem usual em $\{0, \ldots, e\}$, c como 0 e tomamos ainda

   $$f^{\mathcal{A}_P}(m) = \begin{cases} m+1 & \text{se } m < e \\ m & \text{se } m = e \end{cases}$$

   e

   $$R^{\mathcal{A}_P} = \{(s, N, m_0, \ldots, m_n) \mid (N, m_0, \ldots, m_n) \text{ é a configuração de } P \text{ ao fim de } s \text{ passos}\}.$$

Agora descrevemos uma proposição ψ_P de $\mathcal{L}$ que descreve o comportamento de P, quando começado com ϵ como input. Para simplificar a notação abreviamos os termos $c, f(c), f(f(c)), \ldots$ por $\overline{0}, \overline{1}, \overline{2}, \ldots$. Seja então

$$\psi_P = \psi_0 \wedge R(\overline{0}, \ldots, \overline{0}) \wedge \psi_{\alpha_0} \wedge \ldots \wedge \psi_{\alpha_k},$$

onde ψ_0 diz que $<$ é uma ordem total com primeiro elemento c, que se tem $x \leq f(x)$ para todo o x e que $f(x)$ é o sucessor de x excepto quando x for o último elemento:

$$\begin{aligned} \psi_0 = \quad & \forall x x \not< x \wedge \forall x \forall y \forall z((x < y \wedge y < z) \to x < z) \wedge \forall x \forall y(x < y \vee x = y \vee y < x) \wedge \\ & \forall x(c < x \vee x = c) \wedge \forall x(x < f(x) \vee x = f(x)) \wedge \\ & \forall x(\exists y x < y \to (x < f(x) \wedge \forall z(x < z \to (f(x) < z \vee f(x) = z)))). \end{aligned}$$

Para $\alpha = \alpha_0, \ldots, \alpha_{k-1}$, definimos ψ_α de forma a corresponder à instrução α:

Se α é N `LET` $R_i = R_i + |$, toma-se

$$\psi_\alpha = \forall x \forall y_0 \ldots \forall y_n (R(x, \overline{N}, y_0, \ldots, y_n) \rightarrow (x < f(x) \wedge R(f(x), \overline{N+1}, y_0, \ldots, y_{i-1}, f(y_i), y_{i+1}, \ldots, y_n)).$$

Se α é N `LET` $R_i = R_i - |$, toma-se

$$\begin{aligned}\psi_\alpha = \forall x \forall y_0 \ldots \forall y_n (R(x, \overline{N}, y_0, \ldots, y_n) \rightarrow \\ (x < f(x) \wedge ((y_i = \overline{0} \wedge R(f(x), \overline{N+1}, y_0, \ldots, y_n) \vee \\ (y_i \neq \overline{0} \wedge \exists u (f(u) = y_i \wedge R(f(x), \overline{N+1}, y_0, \ldots, y_{i-1}, u, y_{i+1}, \ldots, y_n)).\end{aligned}$$

Se α é N `IF` $R_i = \epsilon$ `THEN` N' `ELSE` N_0, toma-se

$$\begin{aligned}\psi_\alpha = \forall x \forall y_0 \ldots \forall y_n (R(x, \overline{N}, y_0, \ldots, y_n) \rightarrow \\ (x < f(x) \wedge ((y_i = \overline{0} \wedge R(f(x), \overline{N'}, y_0, \ldots, y_n)) \vee \\ (y_i \neq \overline{0} \wedge R(f(x), \overline{N_0}, y_0, \ldots, y_n))))).\end{aligned}$$

Se α é N `PRINT`, toma-se

$$\psi_\alpha = \forall x \forall y_0 \ldots \forall y_n (R(x, \overline{N}, y_0, \ldots, y_n) \rightarrow (x < f(x) \wedge R(f(x), \overline{N+1}, y_0, \ldots, y_n))).$$

Tem-se então o seguinte (verificar!):

(i) $\mathcal{A}_P \models \psi_P$;

(ii) se $\mathcal{A}$ é uma estrutura de $\mathcal{L}$, tal que $\mathcal{A} \models \psi_P$ e tal que $(N, m_0, \ldots, m_n)$ é a configuração de P ao fim de s passos, então $\mathcal{A} \models R(\overline{s}, \overline{N}, \overline{m_0}, \ldots, \overline{m_n})$.

Finalmente definimos φ_P por

$$\varphi_P = \psi_P \rightarrow \exists x \exists y_0 \ldots \exists y_n R(x, \overline{k}, y_0, \ldots, y_n)$$

e vejamos que

$$\models \varphi_P \qquad \text{sse} \qquad P : \epsilon \rightarrow \texttt{halt}.$$

De facto, de $\models \varphi_P$, resulta em particular $\mathcal{A}_P \models \varphi_P$. Por outro lado, sabemos que $\mathcal{A}_P \models \psi_P$, por (i). Donde $\mathcal{A}_P \models \exists x \exists y_0 \ldots \exists y_n R(x, \overline{k}, y_0, \ldots, y_n)$. Logo, existem

$s, m_0, \ldots, m_n \in \mathbb{N}$ tal que $(k, m_0, \ldots, m_n)$ é a configuração de P ao fim de s passos. De (1.5) concluimos então $P : \epsilon \to \texttt{halt}$.

Por outro lado, se $P : \epsilon \to \texttt{halt}$, então existem $s, m_0, \ldots, m_n \in \mathbb{N}$ tal que $(k, m_0, \ldots, m_n)$ é a configuração de P ao fim de s passos. Então tem-se $\mathcal{A} \models R(\overline{s}, \overline{k}, \overline{m_0}, \ldots, \overline{m_n})$ sempre que $\mathcal{A} \models \psi_P$. Concluimos que φ_P é uma fórmula válida. •