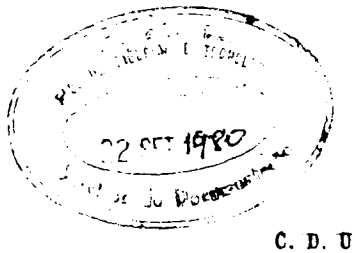
LUÍS MONIZ PEREIRA
Do Centro de Estudos de Cibernética

# INTRODUÇÃO AOS AUTÓMATOS INFINITOS E TEORIA DA COMPUTABILIDADE

«TÉCNICA»
REVISTA DOS ALUNOS DO I. S. T.
Separata do n.º 395 e 396, págs. 265 a 273
e 321 a 328



LISBOA
ASSOCIAÇÃO DOS ESTUDANTES
DO
INSTITUTO SUPERIOR TÉCNICO
1970



C. D. U. 007.52

# INTRODUÇÃO AOS AUTÓMATOS INFINITOS E TEORIA DA COMPUTABILIDADE

Por LUÍS MONIZ PEREIRA
Do Centro de Estudos de Cibernética *

RESUMO

*Procura-se neste artigo equilibrar uma exposição formal com noções intuitivas que motivem a sua leitura e a esclareçam. Começa-se por introduzir as «máquinas de Turing», fornecendo-se em seguida exemplos típicos; depois prossegue-se referindo as importantes noções de «máquinas de Turing Universal» e de problemas irresolúveis. Por fim, conclui-se mencionando as aplicações da teoria exposta.*

## 0 — INTRODUÇÃO

O problema de avaliar exactamente o que podemos ou não esperar deste ou daquele tipo de máquina de calcular já construída ou por construir não é novo. Já alguns anos antes do aparecimento dos primeiros computadores se havia elaborado uma definição de algoritmo e explicitado a sua estreita relação com certo tipo de máquinas conceptuais (Turing 1936). Elas viriam a ser o esquema por detrás dos actuais computadores, que não são mais do que variantes tecnológicas sofisticadas dessas máquinas. Essas máquinas teóricas, chamadas máquinas de Turing, permitem por isso entender melhor os processos realizados por um computador e até demonstrar rigorosamente a existência de problemas insusceptíveis de serem resolvidos nesse computador ou noutro qualquer — isto é, a inexistência de um algoritmo para uma classe de problemas dada.

Este facto não deve ser considerado razão depreciativa dessa classe de problemas: haverá nela problemas cujo enunciado particular é susceptível de um algoritmo, também particular. Não haverá, isso sim, um único algoritmo suficientemente maleável para dar conta da versatilidade de todos os enunciados verosímeis. Note-se que não se exclui a possibilidade de existência, nessa classe de problemas, de um problema irresolúvel que justificasse a inexistência de um algoritmo geral.

A procura de algoritmos, o mesmo é dizer a realização de programas, é a tentativa por um lado de descobrir mais e novos algoritmos, por outro, de encontrar algoritmos cada vez mais amplos, de aplicação cada vez mais geral.

Ninguém até hoje, exceptuando Leibniz, propôs a elaboração de um «algoritmo de todos os algoritmos», o qual se aplicasse a todo e qualquer problema, por mais miscelâneo que fosse. No entanto, existe um algoritmo cujo trabalho é o de imitar qualquer outro algoritmo. À máquina que realiza esse algoritmo chama-se máquina de Turing Universal, ou General Purpose Computer se for uma das suas versões tecnológicas; neste caso, os algoritmos imitados chamam-se programas.

Entre outras coisas, a teoria da computabilidade preocupa-se com o problema da existência de algoritmos (ou procedimentos efectivos de computação) para resolver este ou aquele problema e pretende em última análise um conjunto de instruções a derivar de um enunciado, que conduzam a um procedimento automático garantindo a existência de um resultado.

---

* O C.E.C., Centro de Estudos de Cibernética foi criado no início do ano lectivo de 1968/69 e funciona numa sala-laboratório do Pavilhão de Máquinas do I. S. T. Dos trabalhos publicados constam um artigo na TÉCNICA em Dezembro de 1968, iniciando-se agora uma participação regular. Publicar-se-ão para já os textos relativos ao colóquio «Introdução aos Autómatos Finitos e Infinitos: Aplicações», realizado pelo C.E.C. em Janeiro de 1970.

Como saber porém de antemão se uma dada forma de proceder, que se supõe ser um algortimo, é efectivamente um algoritmo, isto é, se permite obter um resultado final sempre que seja aplicada à classe de problemas que se propõe resolver?

Sabe-se que este problema é irresolúvel. Por outras palavras, prova-se que não existe ou não é possível fornecer um algoritmo que avalie da pertinência ou não pertinência do uso de um modo de proceder *qualquer* para a resolução dos problemas possíveis de uma mesma classe.

Demonstremos que esse algoritmo não existe tomando como exemplo uma classe especial de funções:

Consideremos as funções cujo domínio seja o conjunto dos inteiros não negativos e cujo contradomínio seja um sub-conjunto desses inteiros. Exemplos dessas funções podem ser $k(x) = x^2$, $h(x) = 2^x$, $g(x) = x$ — ésimo dígito no desenvolvimento de $\pi$. Diremos que uma daquelas funções é efectivamente calculável se existir um algoritmo bem definido que nos permita calcular o valor $f(x)$ quando $x$ é dado.

Suponhamos agora que temos uma lista de todas as funções desta classe efectivamente calculáveis e que as ordenamos da seguinte maneira: descrevemos em português cada um dos algoritmos que permite calcular cada função e seguidamente ordenamos alfabèticamente essas descrições. Seja $f_x(x)$ o valor para o argumento $x$, da função cuja ordem é $x$, e definamos a função $h(x) = f_x(x) + 1$. A nossa tese é a de que $h(x)$ não pertence à lista.

Para se ver que isso é verdade suponhamos que era $h(x) = f_i(x)$ ($i$ fixo). Observemos então que para $x = i$, $h(i) = f_i(i) = f_i(i) + 1$, o que é contraditório. Ora qualquer que seja o $i$ escolhido, a contradição aparece sempre que $x$ toma o valor $i$; isto é, $h(x)$ difere de qualquer das funções da lista pelo menos para um valor do argumento. Daqui devemos concluir que $h(x)$ não é efectivamente calculável pois que não pertence à lista. Mas, por outro lado, sabemos que para calcular $h(x)$ nos basta calcular $f_x(x)$, (o que é possível visto $f_x(y)$ ser efectivamente calculável) e juntar-lhe 1. Podemos pois calcular efectivamente $h(x)$. Porque é que então $h(x)$ não pertence à lista? Onde está o nosso erro?

O erro provém de termos assumido em primeiro lugar que era possível construir a lista de todas as funções efectivamente calculáveis. Com efeito, pressupusémos que, para todas as funções tendo como domínio os números inteiros não negativos e contra-domínio um sub-conjunto desses inteiros, existia um algoritmo ou um processo para decidir se cada uma das funções era ou não efectivamente calculável, isto é, se havia um algoritmo que fornecesse o valor da função para cada valor do argumento.

Fica pois provada para esta classe de funções a irresolubilidade do problema levantado. Esta irresolubilidade é aliás equivalente à do «halting-problem»:

Pode demonstrar-se que não existe uma máquina que nos diga, *em todos os casos*, se um dado cálculo levado a cabo por outra máquina terminará eventualmente ou se prosseguirá para sempre sem alcançar um resultado definitivo; este facto não deveria preocupar-nos não fosse ele válido para qualquer máquina de Turing; assim sendo, o que realmente demonstramos é a inexistência de um algortimo geral que permita afiançar se *qualquer* outro algoritmo garante ou não um resultado final definido. Justifiquêmo-nos:

Temos até agora identificado, tàcitamente, processo algorítmico com uma máquina teórica que o realiza; agora, no entanto, fomos mais longe — — impusémos que essa máquina fosse de um tipo restrito de máquina, mais precisamente, uma máquina de Turing. Dissémos até que há uma máquina de Turing Universal suficientemente segura de si para ser capaz de imitar qualquer outra máquina de Turing. Este comportamento é passível de algumas objecções.

Em primeiro lugar, perguntar-se-á, não existirão processos eficientes para alcançar resultados, que não possam ser descritos por nenhuma linguagem formal, por exemplo o pensamento e a fala humanos? A resposta a esta questão terá que ser dada pela Psicologia e pela Linguística; para esse assunto remetemos aos artigos que se seguem e à bibliografia. No entanto, a noção instuitiva que possuimos do que seja um processo eficiente diz-nos que este deve ser *definível*, isto é, temos que encarar a possibilidade de o transmitir a um homem que o consiga executar, por muito desinteressante ou incompreensível que lhe seja o problema. Nesta altura poderíamos ser tentados a definir algoritmo como um conjunto de regras que nos diz, sem ambiguidade, como proceder passo a passo. Esta definição

sujeita-se à crítica de que a interpretação das regras é deixada a cada um, e portanto, que seria até possível inventar-se uma interpretação consistente dessas regras que não fosse sequer suspeitada. Aparecia assim uma ambiguidade e o procedimento eficiente deixava de ser definível. Poderíamos evitar, pelo contrário, os problemas de interpretação ou de compreensão se juntamente com as regras fornecessemos em detalhe o mecanismo que as deve interpretar. É claro que seria ruinoso fazê-lo para cada procedimento eficiente individual; o ideal seria pois, (1) uma linguagem na qual se expressasse as regras, (2) uma única máquina que interpretasse as frases nessa linguagem e desempenhasse todas as etapas prescritas para qualquer procedimento eficiente.

Essa máquina, é bom dizê-lo, existe. A linguagem que ela utiliza é a linguagem prescrita; note-se, aliás, que ao operar sobre uma sequência de símbolos inicial chamada enunciado, a máquina a transforma numa sequência de símbolos terminal chamada resultado, através de uma sequência de símbolos intermédia chamada resolução; assim um problema goza de solução se existir uma sequência de símbolos permissível entre o enunciado e o resultado. A própria máquina, é no entanto descrita, de maneira precisa, por uma colecção de sequências de símbolos, que fazem a ligação entre as sucessivas sequências de símbolos do cálculo; nas sequências de símbolos da máquina intervém o símbolo que a máquina lê, o símbolo que a máquina escreve, e, como intermédio entre estes dois, o símbolo do estado em que a máquina se encontra; finalmente, aparece ainda o símbolo do estado interno em que a máquina fica depois da operação, como veremos.

As sequências permitidas são precisamente aquelas que a máquina permite que se efectuem visto que a colecção de sequências que a definem é restrita.

Embora seja de esperar que procedimentos eficientes mais complexos obriguem a máquinas mais complexas, o que acontece é que podemos realizar uma máquina do tipo indicado cuja forma permanece constante, independentemente da complexidade do processo em questão, e capaz de o manipular efectivamente — é a máquina de Turing Universal.

Uma outra objecção poderia muito bem ser esta: com que propriedade assimilámos as funções efectivamente calculáveis às funções calculáveis por uma máquina de Turing *(ou funções computáveis)?* Ou esta: quem nos assegura que dado um algoritmo, descrito numa certa linguagem formal, com papel e lápis, exista um dispositivo físico que o realize?

A resposta à segunda questão é imediata: haverá pelo menos um dispositivo físico — aquele que escreva a lápis no papel. Para responder à primeira questão começamos por fazer notar que assimilar o conceito intuitivo de função efectivamente calculável ao conceito rigoroso de função computável, é mais uma tentativa de definição precisa desse conceito intuitivo do que qualquer outra coisa; mas, se por um lado fàcilmente admitimos que uma função computável é por maioria de razão efectivamente calculável, poderão surgir dúvidas quanto à capacidade de a noção de função computável dar conta de todas as funções a que legìtimamente chamaríamos efectivamente calculáveis. De facto, têm-se elaborado variadas definições precisas do que seja calculabilidade efectiva; entre elas contam-se a *definibilidade-λ* de Church, *a recursividade geral* de Herbrand-Gödel-Kleene, os *sistemas canónicos* de Post, os *sistemas formais elementares* de Smullyan e a *computabilidade* de Turing, todas elas formuladas muito diferentemente mas que provaram vir a definir *a mesma classe de funções*, sendo portanto equivalentes. Esta equivalência e o facto de que para máquinas que à primeira vista pareciam ter possibilidades de computação muito maiores do que as máquinas de Turing se ter demonstrado sempre que computavam as mesmas funções, levam a que se identifique ou se defina as funções efectivamente calculáveis como aquelas que são computáveis por uma máquina de Turing.

## Máquinas de Turing

A afirmação de que uma máquina de Turing é um autómato infinito necessita ser tornada mais precisa. Na realidade, o que numa máquina de Turing é infinito é a sua memória porquanto o seu número de estados internos é finito. Podemos fazer a analogia com um computador ao qual fosse sempre possível acrescentar capacidade de memória (acrescentando por exemplo fita magnética) à medida que fosse necessário. Podemos pois referir-nos aos dispositivos físicos que representam uma máquina de Turing como

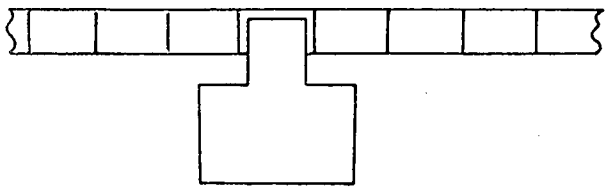
possuindo memória ilimitada no sentido acima dado. Com pretexto no que já foi dito passamos a dar uma.

## 1 — DESCRIÇÃO

Uma máquina de Turing é um autómato com um número finito de estados internos ou configurações ao qual se associa uma memória externa constituída por uma fita ou banda dividida em quadrados ou rectângulos e infinita em ambos os sentidos.

A ligação entre a fita e a máquina pròpriamente dita é assegurada por uma cabeça de leitura-gravação que inspecciona apenas um quadrado de cada vez.

O estado seguinte do autómato é determinado pelo estado actual e pela sua entrada, constituída pelo símbolo inscrito no quadrado sob a cabeça de leitura-gravação. A saída do autómato é ou um símbolo que ele grava no lugar do símbolo de entrada que é apagado, *ou* o deslocamento elementar da fita de um único quadrado, quer para a esquerda quer para a direita. Pode também pensar-se num tipo diferente de saída em que aconteçam ambas as coisas, isto é, inscrição de um símbolo *e* deslocamento; esse tipo de descrição, sendo equivalente à anterior, permite a economia de transições elementares como veremos.



Podemos pois descrever o funcionamento de uma máquina de Turing com três funções

$$Q(t+1) = G(Q(t), S(t))$$
$$R(t+1) = F(Q(t), S(t))$$
$$D(t+1) = D(Q(t), S(t))$$

onde Q (t) é o estado na etapa t.
S (t) é o símbolo lido na etapa t.
R (t) é o símbolo escrito na etapa t.
D (t) é o deslocamento na etapa t.

As três funções são:

G — mudança de estado
F — saída
D — deslocamento

Em cada etapa ou ciclo a máquina encontra-se num estado interno $q_i$, lê o símbolo $s_j$, imprime o símbolo $F(q_i, s_j)$ eventualmente igual ao anterior, desloca-se $D(q_i, s_j)$ e transita para o estado $G(q_i, s_j)$.

Visto que a máquina se pode deslocar em qualquer dos sentidos ao longo da fita, é-lhe sempre possível voltar a um quadrado devidamente assinalado para reaver aí a informação armazenada. Isso torna-lhe possível acumular quantidades de informação arbitràriamente grandes.

Esta possibilidade de voltar «atrás» distingue-a de um autómato finito*, o qual pode considerar-se como uma máquina de Turing que só pode mover-se num único sentido; (ou que se mova nos dois sentidos mas não possa apagar).

Outra diferença importante é que enquanto para os autómatos finitos temos que considerar a entrada como procedendo do meio exterior e portanto que a computação não fica completamente especificada fornecendo-se apenas o estado inicial juntamente com o símbolo de entrada inicial, na máquina de Turing temos um sistema fechado em que a fita serve de ambiente à parte finita da máquina; podemos especificar completamente uma computação dando (1) o estado inicial da máquina, (2) o conteúdo inicial da fita, (3) a posição inicial da fita.

Fazemos aqui uma restrição: inicialmente a fita está vazia excepto num número finito de quadrados. Assim em cada etapa podemos supor a fita como sendo finita, desde que, quando necessário, se acrescente mais um quadrado vazio à fita; mantém-se pois a analogia com os actuais computadores.

## 2 — DESCRIÇÃO FORMAL

*Definição 1* Uma *expressão* é uma sequência finita (eventualmente nula) de símbolos escolhidos da lista: $q_1, q_2, \ldots ; s_0, s_1, \ldots ;$ R, L

---

* Ou máquina sequêncial.

*Definição 2* Um *quádruplo* é uma *expressão* com uma das seguintes formas:

(1) $q_i s_j s_k q_l$ (3) $q_i s_j L q_l$

(2) $q_i s_j R q_l$ (4) $q_i s_j q_k q_l$

Cada quádruplo indica um ciclo de uma máquina de Turing, sendo $q_i$ o estado interno inicial, $s_j$ o símbolo lido e $q_l$ o estado interno no final do ciclo. Em (1), $s_k$ é o símbolo escrito que substitui $s_j$. Em (2) e (3), R e L são deslocamentos da fita tais que a máquina passa a ler o símbolo incrito no quadrado imediatamente à direita ou à esquerda de $s_j$ respectivamente.

Às expressões com a forma (4) permitem que a máquina faça certo tipo de perguntas ao «meio exterior» acerca do conteúdo da sua fita: se a resposta é afirmativa ela assume o estado interno $q_k$; o estado interno $q_l$ caso contrário. Restringe-se o tipo de perguntas às da forma: «será que $\omega \varepsilon A$?» em que A é um conjunto dado e $\omega$ é extraído do conteúdo da fita.

Obtém-se assim o conceito de computabilidade relativa, ou de função A-computável, quando for necessária a intervenção de quádrulos da forma (4) juntamente com a especificação do conjunto A e o modo de extrair $\omega$ da fita. Esta noção de computabilidade relativa é mais geral que a anterior; de facto, podemos supor que uma função é simplesmente computável quando o conjunto A for vazio; nesse caso a máquina assume sempre, na forma (4), o estado $q_l$, e um quádruplo da forma $q_i s_j q_k q_l$ é equivalente a um outro da forma (1), $q_i s_j s_j q_l$. Há pois toda a vantagem em desenvolver a teoria mais geral de computabilidade relativa ao invés da teoria de computabilidade simples; daí a razão de ser dos quádruplos da forma (4).

*Definição 3* Uma *máquina de Turing* é um conjunto finito (não vazio) de quádruplos, que não contém dois quádruplos diferentes cujos primeiros dois símbolos sejam iguais.

Aos $q_i$'s e aos $s_i$'s que ocorrem nos quádruplos de uma máquina de Turing chama-se estados ou *configurações internas* e *alfabeto* respectivamente. Se nenhum dos quádruplos for da forma (4), a máquina de Turing diz-se *simples*. (Notar que a definição acima dada implica a finitude do número de configurações internas e do número de símbolos do alfabeto de qualquer máquina).

*Definição 4* Uma *descrição instantânea* é uma *expressão* em que há apenas um $q_i$, não há R ou L, e na qual $q_i$ não é o símbolo mais à direita.

Se Z for uma máquina de Turing, $a$ é uma descrição instantânea de Z se o $q_i$ que figura em $a$ for uma configuração interna de Z e se os $s_i$'s que ocorrem em $a$ fizerem parte do alfabeto de Z.

*Definição 5* A uma expressão que consista inteiramente de $s_i$'s chama-se *expressão da fita*. No que se segue, P e Q são expressões da fita.

*Definição 6* Seja Z uma máquina de Turing e $a$ uma descrição instantânea de Z tal que $q_i$ é a configuração interna de Z que figura em $a$ e $s_j$ é o símbolo imediatamente à direita de $q_i$. Diremos então que $s_j$ é o *símbolo lido por Z em a* e que a expressão da fita obtida retirando $q_i$ é a *expressão da fita de Z em a*.

Consideremos a partir de agora que os quadrados vazios da fita são preenchidos por um símbolo $s_0$ ou B que indica precisamente quais os quadrados em branco.

*Definição 7* Seja Z uma máquina de Turing e sejam $a$ e $b$ descrições instantâneas. Escreveremos $a \longrightarrow b$ (Z) ou se não houver ambiguidade apenas $a \longrightarrow b$, se houver um quádruplo de Z da forma (1), (2) ou (3) que transforme $a$ em $b$. Escreveremos $a \xrightarrow[A]{} b$ se houver um quádruplo de Z da forma (4) que transforme $a$ em $b$.

Exemplo: $s_2 q_1 s_0 s_5 s_3$ é transformada em $s_2 s_0 q_1 s_5 s_3$ pelo quádruplo da forma (2), $q_1 s_0 R q_1$.

*Teorema 1* Se $a \longrightarrow b$ (Z) e $a \longrightarrow c$ (Z), então $b = c$. Este teorema é imediato a partir da definição 3.

*Teorema 2* Se $a \longrightarrow b$ (Z) e $Z \subset Z'$, então $a \longrightarrow b$ (Z'). $Z \subset Z'$ significa que todo o quádruplo de de Z é também um quádruplo de Z'. Note-se, aliás, que Z e Z' são dois conjuntos (definição 3).

*Definição 8* Uma descrição instantânea $a$ chama-se *terminal relativamente a* Z se não existir $q$ tal que $a \longrightarrow b$ (Z).

Isto significa que a ausência de uma instrução sob a forma de um quádruplo do tipo (1), (2) ou (3) adequado à descrição instantânea $a$ equivale à paragem da máquina.

*Definição 9* Por *computação de uma máquina de Turing* Z entende-se uma sequência finita $a_1, a_2, \ldots, a_p$ de descrições instantâneas tais que $a_i \rightarrow a_{i+1}$ (Z) para $1 \leqslant i < p$ e tal que $a_p$ é terminal relativamente a Z. Nesse caso escreve-se $a_p = \mathrm{Res}_z(a_1)$ e chama-se a $a_p$ o *resultado* de $a_1$ relativamente a Z.

## 3 EXEMPLOS

Normalmente chama-se $q_1$ à configuração interna presente na descrição instantânea inicial $a$. Assim suponhamos que Z consiste no seguinte conjunto de quádruplos:

$q_1\ s_0\ R\ q_1$
$q_1\ s_2\ R\ q_1$
$q_1\ s_5\ R\ q_1$
$q_1\ s_3\ s_5\ q_2$
$q_2\ s_5\ L\ q_3$
$q_3\ s_0\ s_5\ q_3$
$q_3\ s_2\ s_5\ q_3$
$q_3\ s_3\ s_5\ q_3$

Z tem as configurações internas $q_1$, $q_2$, $q_3$ e o alfabeto $s_0$, $s_2$, $s_3$, $s_5$.

Exemplo de uma computação de Z:

$$\begin{aligned} s_2\ q_1\ s_0\ s_5\ s_3 &\longrightarrow s_2\ s_0\ q_1\ s_5\ s_3 \\ &\longrightarrow s_2\ s_0\ s_5\ q_1\ s_3 \\ &\longrightarrow s_2\ s_0\ s_5\ q_2\ s_5 \\ &\longrightarrow s_2\ s_0\ q_3\ s_5\ s_5 \end{aligned}$$

Aqui $\mathrm{Res}_Z(s_2\ q_1\ s_0\ s_5\ s_3) = s_2\ s_0\ q_3\ s_5\ s_5$.

Portanto o efeito de Z foi o de procurar o símbolo $s_3$ movendo-se para a direita do símbolo inicialmente lido. Localizado $s_3$, substituíu-o por $s_5$ e escreveu outro $s_5$ imediatamente à esquerda do primeiro.

O que aconteceria se não existisse nenhum $s_3$ à direita de $q_1$? Nesse caso a computação prosseguiria eternamente, ou melhor, não existiria computação no sentido dado pela definição nove. Isso porque à direita de $s_5$ existe uma infinidade de quadrados em branco, os quais já suposémos preenchidos com $s_0$. Se $a_1$ fosse $s_2\ q_1\ s_0\ s_5\ s_2$, o $\mathrm{Res}_Z(a_1)$ seria portanto indefinido.

O exemplo seguinte é um *contador de paridade*. Ele diz-nos se o número de $s_1$'s (ou o número de 1's, fazendo $s_1 = 1$) na sua fita é par ou ímpar. Sempre que encontra um 1 ele muda de um estado para o outro. Aqui vamos utilizar uma representação de quíntuplos que permite economizar transições elementares. Os quíntuplos são da forma $(q_i\ s_j\ s_k\ D\ q_l)$ em que D pode ser um dos símbolos R ou L, $q_i$ a configuração interna, $s_j$ o símbolo lido, $s_k$ o símbolo escrito e $q_l$ a configuração interna seguinte. Esta representação é equivalente à anterior, facto que o leitor poderá demonstrar como exercício para uma máquina Z simples.

Os quíntuplos do contador de paridade são:

| $q_i$ | $s_j$ | $s_k$ | D | $q_l$ |
|---|---|---|---|---|
| $q_0$ | 0 | 0 | R | $q_0$ |
| $q_0$ | 1 | 0 | R | $q_1$ |
| $q_1$ | 0 | 0 | R | $q_1$ |
| $q_1$ | 1 | 0 | R | $q_0$ |

Se o estado final é $q_0$ a paridade é par, se é $q_1$, é ímpar.

Vejamos como é processada a descrição instantânea seguinte:

$$a_1 = P\ q_0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ F\ Q$$

em que F é um símbolo que indica o fim da expressão cuja paridade se pretende conhecer e P e Q são como já se disse expressões da fita, possìvelmente apenas constituídas por B's (ver definição 6).

Teremos então:

$$\begin{aligned} P\ q_0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ F\ Q &\longrightarrow P\ 0\ q_1\ 0\ 1\ 1\ 0\ 1\ 1\ F\ Q \\ &\longrightarrow P\ 0\ 0\ q_1\ 1\ 1\ 0\ 1\ 1\ F\ Q \\ &\longrightarrow P\ 0\ 0\ 0\ q_0\ 1\ 0\ 1\ 1\ F\ Q \\ &\longrightarrow P\ 0\ 0\ 0\ 0\ q_1\ 0\ 1\ 1\ F\ Q \\ &\longrightarrow P\ 0\ 0\ 0\ 0\ 0\ q_1\ 1\ 1\ F\ Q \\ &\longrightarrow P\ 0\ 0\ 0\ 0\ 0\ 0\ q_0\ 1\ F\ Q \\ &\longrightarrow P\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ q_1\ F\ Q \end{aligned}$$

que é terminal.

Neste exemplo a máquina move-se sempre para a direita não existindo portanto a possibilidade de reaver mais tarde informação porventura por ela armazenada na fita. Nessa medida, podemos esperar que a computação de paridade efectuada, possa ser levada a cabo por um autómato finito (e portanto de memória finita e entrada sequencial).

*Exercício 2* Modificar os quíntuplos de modo a que a expressão da fita inicial se mantenha, conservando assim a informação dada.

Um exemplo elucidativo da utilização da fita ilimitada é o *verificador de parêntesis* que permite decidir se uma sequência de parêntesis é ou não uma frase bem formada de uma certa linguagem. Por exemplo (((( ) ( )) ( ))) é bem formada enquanto que ( ( ))) , ) ( e ((( ))))) não o são.

A descrição instantânea inicial é do tipo $PAq_0$ EAQ, em que A é um símbolo que identifica o início e o fim da expressão da fita E constituída por uma sequência de parêntesis, mal ou bem formada. Os quíntuplos do verificador de parêntesis são:

$q_0$ ) X L $q_1$
$q_0$ ( ( R $q_0$
$q_0$ A A L $q_2$
$q_0$ X X R $q_0$
$q_1$ ) ) L $q_1$
$q_1$ ( X R $q_0$
$q_1$ A N D $q_3$
$q_1$ X X L $q_1$
$q_2$ ) não ocorre
$q_2$ ( N D $q_5$
$q_2$ A S D $q_5$
$q_2$ X X L $q_2$

X é um símbolo utilizado em substituição de um parêntese que está a ser conferido. D é um deslocamento indiferentemente para a esquerda ou para a direita; pode ser substituído por R ou por L. $q_3$ é o estado que figura na descrição instantânea terminal. A máquina imprime S se a expressão é bem formada, N caso contrário.

Examinemos como a máquina procede:

$q_0$ é bàsicamente um estado de deslocação para a direita, que procura, sem nada modificar, um parêntese «)»; substitui-o por X e transforma-se em $q_1$.

$q_1$ é bàsicamente um estado de deslocação para a esquerda, que procura, sem nada modificar, um parêntese «(». Se o encontra substitui-o por X e volta ao estado $q_0$. O cancelamento continua até que aconteça uma das duas coisas: (1) o estado $q_1$ não encontra nenhum «(»; atinge assim o A da esquerda, imprime um N e pára, (2) o estado $q_0$ não encontra mais nenhum «)»; atinge assim o A da direita e passa ao estado $q_2$; alcançado o estado $q_2$ este desloca-se para a esquerda para verificar se persiste ainda algum «(»; se encontra um pára e imprime N; se atinge o A da esquerda sem encontrar nenhum imprime S e pára.

Note-se que esta computação não pode ser realizada por um autómato finito porque a máquina terá que percorrer eventualmente intervalos arbitràriamente longos. Repare-se também numa propriedade interessante desta máquina: ela só muda o conteúdo de cada quadrado uma única vez. Pode mostrar-se que qualquer máquina é num certo sentido equivalente a uma outra restingida a essa condição.

## 4 — COMPUTAÇÕES NUMÉRICAS

Mostrámos já como é que as máquinas de Turing podiam ser utilizadas para realizar computações com símbolos. Para as tornarmos aptas a fazerem computações numéricas é necessário introduzir uma representação simbólica adequada para os números. A maneira mais simples de o fazer é escolher um símbolo $s_1$ como base e simbolizar um número por uma expressão inteiramente formada por ocorrências desse símbolo. Além disso, se m for um inteiro positivo escreveremos $s_i^m$ em vez da expressão $\underbrace{s_i \ldots s_i}_{m}$. A expressão nula representar-se-á $s_i^0$. Utilizaremos doravante 1 em lugar de $s_1$.

*Nota Importante*

Chamamos aqui a atenção para a facto de que apenas trabalharemos no que se segue com funções cujos domínio e contra-domínio são sub-conjuntos dos inteiros não negativos. A mesma teoria pode ser desenvolvida sensìvelmente da mesma maneira para os inteiros negativos ou para os racionais. Quanto aos números reais, há que definir os números reais computáveis. Mas não podemos entrar em detalhes.

*Definição 10* A cada número n inteiro não negativo associamos a expressão da fita

$$\overline{n} = s_1^{n+1} = 1^{n+1}$$

Portanto $\overline{3} = 1\ 1\ 1\ 1$ e, em geral, $\overline{n} = \underbrace{1 \ldots 1}_{n+1}$

*Definição 11* A cada k-tuplo $(n_1, n_2, \ldots, n_k)$ de inteiros não negativos associamos a expressão da fita $\overline{(n_1, n_2, \ldots, n_k)}$, onde

$$\overline{(n_1, n_2, \ldots, n_k)} = \overline{n}_1\ B\ \overline{n}_2\ B \ldots B\ \overline{n}_k$$

Assim, $\overline{(2,3,0)} = 1\ 1\ 1\ B\ 1\ 1\ 1\ 1\ B\ 1$

Esta notação é conveniente para as entradas. Para as saídas utilizaremos a seguinte

*Definição 12* Seja M uma expressão da fita ou uma descrição instantânea. Então $<M>$ é o número de 1,s que ocorrem em M.

Assim, $<1\ 1\ B\ s_4\ q_3> = 2; <q_3\ q_2\ s_5> = 0$.

Note-se que $<\overline{m-1}> = m$ e que $<M\ N> = <M> + <N>$.

## 5 – FUNÇÕES COMPUTÁVEIS

*Definição 13* Seja Z uma máquina de Turing. Então, para cada n, associamos com Z uma função n-ária

$$\Psi_Z^{(n)}(x_1, x_2, \ldots, x_n)$$

do seguinte modo:

Para cada n-tuplo $(m_1, m_2, \ldots, m_n)$, fazemos $x_1 = q_1\ \overline{(m_1, m_2, \ldots, m_n)}$ e distinguimos entre dois casos

(1) existe uma computação de Z, $a_1, \ldots, a_p$. Nesse caso fazemos

$$\Psi_Z^{(n)}(m_1, m_2, \ldots, m_n) = <a_p> = <\mathrm{Res}_Z(a_1)>$$

(2) não existe nenhuma computação $a_1, \ldots, a_p$. isto é, $\mathrm{Res}_Z(a_1)$ é indefinido. Nesse caso deixamos $\Psi_Z^{(n)}(m_1, m_2, \ldots, m_n)$ indefinida.

Escrevemos $\Psi_Z(x)$ em vez de $\Psi_Z^{(1)}(x)$.

*Definição 14* Uma função n-ária $f(x_1, \ldots, x_n)$ é *parcialmente computável* se existir uma máquina de Turing Z tal que

$$f(x_1, \ldots, x_n) = \Psi_Z^{(n)}(x_1, \ldots, x_n)$$

Diremos nesse caso que Z *computa* f. Se, além disso, $f(x_1, \ldots, x_n)$ for uma função *total*, isto é, definida para todos os n-tuplos, diremos que a função f é *computável*.

*EXEMPLOS*

*Adição.* Seja $f(x, y) = x + y$. Vamos construir uma máquina de Turing Z que computa $f(x, y)$; isto é, tal que

$$\Psi_Z^{(2)}(x, y) = x + y$$

Z consiste no conjunto de quádruplos

$$q_1\ 1\ B\ q_1$$
$$q_1\ B\ R\ q_2$$
$$q_2\ 1\ R\ q_2$$
$$q_2\ B\ R\ q_3$$
$$q_3\ 1\ B\ q_3$$

Se $a_1 = q_1\ \overline{(m_1, m_2)} = q_1\ \overline{m}_1\ B\ \overline{m}_2$. Então o leitor verificará que

$$\Psi_Z^{(2)}(m_1, m_2) = <\mathrm{Res}_Z(a_1)>$$
$$= <B\ 1^{m_1}\ B\ q_3\ B\ 1^{m_2}<$$
$$= m_1 + m_2$$

*Sucessão.* Seja $S(x) = x + 1$. Mostremos que $S(x)$ é computável.

Z consiste no quádruplo único $q_1\ B\ B\ q_1$.

Então $\Psi_Z(m) = <q_1\ \overline{m}> m + 1$

*Subtracção.* Definamos a função $x \dot{-} y$ do seguinte modo

$$x \dot{-} y = x - y \quad \text{se } x \geqslant y$$
$$x \dot{-} y = 0 \quad \text{se } x < y$$

Pode construir-se uma máquina de Turing Z tal que $\Psi_Z^{(2)}(m_1, m_2) = m_1 \dot{-} m_2$.

*Função Identidade.* $I(x) = x$

Também é computável. Seja Z constituída por um único quádruplo, $q_1\ 1\ B\ q_1$. Então, $q_1\ \overline{n} = q_1\ 11^n \longrightarrow q_1\ B\ 1^n$, e portanto $\Psi_Z(n) = <q_1\ B\ 1^n> = n$.

*Outras funções identidade.* Sejam as funções n-árias $U_i^n(x_i, \ldots, x_n) = x_i$. Essas funções também são computáveis. Isto é, para cada n e para cada $1 \leqslant i \leqslant n$ há uma máquina Z tal $\Psi_Z^{(n)}(x_1, \ldots, x_n) = x_i$.

*Multiplicação.* Também é computável. Existe Z tal que $\Psi_Z^{(2)}(x,y) = x.y$

*Definição 15* Seja *a* uma descrição instantânea da forma $P\ q_i\ s_j\ Q$. Então, *a* é *final* em relação a Z se Z não contiver nenhum quádruplo começado por $q_i\ s_j$. (mesmo quádruplos da forma $q_i\ s_j\ q_k\ q_l$).

Corolários:

*Teorema 3* Se Z é simples, então *a* é terminal em relação a Z se e só se for final em relação a Z.

*Teorema 4* *a* é final em relação a Z se e só se simultâneamente se verificarem (1) e (2):

(1) *a* é terminal em relação a Z

(2) qualquer que seja o conjunto A escolhido, não há nenhum *b* tal que $a \underset{A}{\longrightarrow} b$ (Z).

*Definição 16* Por-*A computação* de uma máquina de Turing Z entende-se uma sequência finita $a_1, a_2, \ldots, a_p$ de descrições instantâneas tal que para cada $1 \leqslant i < p$, ou $a_i \longrightarrow a_{i+1}$ (Z) ou $a_i \underset{A}{\longrightarrow} a_{i+1}$ (Z) sendo $a_p$ final. Nesse caso escreveremos $a_p = \text{Res}_Z^A(a_1)$ e chamaremos a $a_p$ o *A-resultado* da $a_1$ em relação a Z.

*Definição 17* Seja Z uma máquina de Turing. Para cada n, associaremos a Z uma função n-ária (que em geral depende de um conjunto A),

$$\Psi_{Z;A}^{(n)}(x_1, \ldots, x_n)$$

do seguinte modo: para cada $(m_1, \ldots, m_n)$ fazemos $a_1 = q_1\ \overline{(m_1, \ldots, m_n)}$ e distinguimos dois casos

(1) existe uma A-computação de $Z, a_1, \ldots, a_p$. Nesse caso fazemos

$$\Psi_{Z;A}^{(n)}(m_1, \ldots, m_n) = <a_p> = <\text{Res}_Z^A(a_1)>$$

(2) não existe uma A-computação de Z; isto é, $\text{Res}_Z^A(a_1)$ é indefinido. Nesse caso deixaremos $\Psi_{Z;A}^{(n)}(m_1, \ldots, m_n)$ indefinida.

Escreveremos $\Psi_Z^A(x)$ em vez de $\Psi_{Z;A}^{(1)}(x)$.

Se Z for simples $\Psi_{Z;A}^{(n)}(x_1, \ldots, x_n) = \Psi_Z^{(n)}(x_1, \ldots, x_n)$ por que $\Psi$ é independente de A.

*Definição 18* Uma função n-ária $f(x_1, \ldots, x_n)$ é *parcialmente A-computável* se existir uma máquina de Turing Z tal que $f(x_1, \ldots, x_n) = \Psi_{Z;A}^{(n)}(x_1, \ldots, x_n)$.

Nesse caso diremos que Z *A-computa* f.

Se além disso $f(x_1, \ldots, x_n)$ for total, então diz-se-á *A-computável.*

Corolário:

*Teorema 5* Se $f(x_1, \ldots, x_n)$ é (parcialmente) computável, então é também (parcialmente) A-computável qualquer que seja A.

Há duas leituras para este teorema; uma com outra sem os parêntesis.

*Teorema 6* A cada máquina de Turing Z corresponde uma máquina de Turing simples Z' tal que $\Psi_{Z'}^{(n)}(x_1, \ldots, x_n) = \Psi_{Z;\phi}(x_1, \ldots, x_n)$ em que $\phi$ é o conjunto vazio. Z' obtém-se a partir de Z substituindo cada um dos seus quádruplos da forma $q_i\ s_j\ q_k\ q_l$ por outro da forma $q_i\ s_j\ s_j\ q_l$.

Corolário:

*Teorema 7* $g(x_1, \ldots, x_n)$ é (parcialmente) $\phi$ — computável se e só se for (parcialmente) computável.

Assim, a cada teorema obtido para a A-computabilidade, fazendo simplesmente $A = \phi$. Isto é, a teoria da computabilidade é um caso particular da teoria da computabilidade relativa.

*Definição 19* Dizemos que um conjunto S é computável (A-computável) se a sua função característica $C_S(x)$ for computável (A-computável).

Por *função característica* de um conjunto S formado por n-tuplos $(x_1, \ldots, x_n)$ entenderemos a função total n-ária $C_S(x_1, \ldots, x_n)$ cujo valor para um dado n-tuplo $(a_1, \ldots, a_n)$ pertence a S e 1 se $(a_1, \ldots, a_n)$ não pertence a S.

*Teorema 8* A é A-computável.

## 6 — OPERAÇÕES SOBRE FUNÇÕES COMPUTÁVEIS

Para se estabelecer a computabilidade de funções mais complicadas podem-se utilizar operações por meio das quais se obtêm novas funções computáveis a partir das anteriores. Essas operações podem ser repetidamente aplicadas, um número finito de vezes, às novas funções obtidas, gerando-se assim uma enorme classe de funções computáveis. Para que essa classe coincida com a de todas as funções computáveis bastam duas operações e um conjunto de partida de seis funções computáveis. É claro que existem outras maneiras de definir essa classe de funções, nomeadamente com outras operações (recursão primitiva, por exemplo) e outros conjuntos de partida. As duas operações que nos vão interessar são a *composição* e a *minimalização*. O conjunto de partida é o das funções já apresentadas como computáveis,

(1) $C_A(x)$

(2) $S(x) = x + 1$

(3) $U_i^n(x_1, \ldots, x_n) = x_i \quad 1 \leqslant i \leqslant n$, para todo o n

(4) $x + y$

(5) $x \dot{-} y$

(6) $x \cdot y$

A operação de composição permite, por exemplo, construir a função $f(g(x))$ a partir das funções $f(y)$ e $g(x)$. O domínio de $f(g(x))$ é definido precisamente pelo conjunto dos números $a$ tais que $a$ pertence ao domínio de $f(x)$ e $g(a)$ pertence ao domínio de $f(y)$.

*Definição 20* A operação de *composição* associa às funções $f(Y^m)$, $g_1(X^n)$, $g_2(X^n)$, $\ldots$, $g_m(X^n)$, a função $h(X^n) = f(g_1(X^n), \ldots, g_m(X_n))$ em que $f(Y^m)$ é uma abreviatura de $f(y_1, y_2, \ldots, y_m)$ e $g_m(X_n)$ é uma abreviatura de $g_m(x_1, \ldots, x_n)$. Essa função é definida precisamente para aqueles n-tuplos a $(_1, \ldots, a_n)$ que pertencem ao domínio de cada uma das funções $g_i(X^n)$, $i = 1, 2, \ldots$, m, e para os quais o m-tuplo $(g_1(a_1, \ldots, a_n), \ldots, g_m(a_1, \ldots, a_n))$ pertence ao domínio de $f(Y^m)$.

*Teorema 9* Sejam $f(Y^m)$, $g_i(X^n)$, $\ldots$, $g_m(X^n)$ (parcialmente) A-computáveis. Seja $h(x^n)$ dada pela expressão da definição anterior. Então $h(X^n)$ é (parcialmente) A-computável.

*Corolário* A classe das funções (parcialmente) A-computáveis é *fechada* debaixo da operação de composição.

Fàcilmente se vê, por consequência de $x \cdot y$ ser computável e pelo teorema anterior, que $x^k$ é computável para cada k e por conseguinte todo o polinómio de coeficientes inteiros é computável.

A função $|x - y|$ é computável visto que $|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

*Definição 21* A operação de *minimalização* associa a cada função total $f(y, X^n)$, a função $h(X^n)$ cujo valor para um dado $X^n$ é o menor valor de y, se esse valor existir, para o qual $f(y, X^n) = 0$. Se esse valor não existir, $h(X^n)$ é indefinida. Escreveremos

$$h(X^n) = \min_y [h(y, X^n) = 0]$$

Por exemplo,

$$x/2 = \min_y [|(y + y) - x| = 0]$$

em que $x/2$ é uma função parcial definida apenas quando x é par. Pelo teorema 10, que se segue, $x/2$ é parcialmente computável.

*Definição 22* Uma função total $f(y, X^n)$ chama-se *regular* se

$$\min_y [j(y, X^n) = 0]$$

for total.

*Teorema 10* Se $f(y, X^n)$ é A-computável, então $h(X^n) = \min_y [f(y, X^n) = 0]$ é parcialmente A-computável. Mais ainda, se $f(y, X^n)$ for regular, $h(X^n)$ é A-computável.

Estes resultados demonstram como é possível estabelecer a computabilidade de funções complexas sem necessidade de recorrer à definição original de computabilidade em termos de máquinas de Turing.

Algumas dessas funções são:

(7) $N(x) = 0$ com $N(x) = U_1^1(x) \dot{-} U_1^1(x)$

(8) $\alpha(x) = 1 \dot{-} x$; isto é, $\alpha(0) = 1$, $\alpha(x) = 0$ para $x > 0$.

Temos que $\alpha(x) = S(N(x)) \dot{-} U_1^1(x)$

(9) $x^2 = U_1^1(x) \cdot U_1^1(x)$

(10) $[\sqrt{x}]$ (o maior inteiro $\leqslant \sqrt{x}$)

$$[\sqrt{x}] = \min_y [((y+1)^2 \dot{-} x) \neq 0]$$
$$= \min_y [\alpha((S(U_2^2(x,y)))^2 \dot{-} U_1^2(x,y)) = 0]$$

(11) $|x - y| = (x \dot{-} y) + (y \dot{-} x)$

(12) $[x/y]$ (se $y \neq 0$, $[x/y]$ é o maior inteiro $\leqslant x/y$. Se $y = 0$, $[x/y] = 0$)

Teremos

$$[x/y] = \min_z [y = 0 \ V \ y(z+1) > x]$$
$$= \min_z [y = 0 \ V \ y(z+1) \dot{-} x \neq 0]$$
$$= \min_z [y = 0 \ V \ \alpha(y(z+1) \dot{-} x) = 0]$$
$$= \min_z [y \cdot \alpha(y(z+1) \dot{-} x) = 0]$$

o símbolo «V» significa «ou um ou outro ou ambos». É equivalente a dizer que o produto de ambos os termos (de um e outro lado do «V») é nulo.

(13) $R(x,y)$ (se $y \neq 0$, $R(x,y)$ é o resto da divisão de x por y). Isto é,

$$x/y = [x/y] + \frac{R(x,y)}{y}$$
$$R(x,y) = x \dot{-} y[x/y]$$

Tal que $R(x.0) = x$. Notar que $x \geqslant y[x/y]$.

## 7 — MÁQUINAS DE TURING ESPECIAIS

Vamos considerar nesta secção máquinas que copiam, máquinas que transferem, máquinas que apresentam os seus resultados de forma a que estes possam dar imediatamente entrada noutra máquina, e máquinas que fazem a computação simultânea das computações de várias máquinas.

*Definição 23* Se Z for uma máquina de Turing, chamamos $\theta(Z)$ ao maior i tal que $q_i$ é uma confirmação interna de Z.

*Definição 24* Uma máquina de Turing é *n-regular* ($n > 0$) se se verificar (1) e (2):

(1) existe um $s > 0$ tal que, sempre que $\text{Res}_Z^A [q_1 \overline{(m_1, \ldots, m_n)}]$ é definido, ele tem a forma $q_{\theta(z)}(r_1, \ldots, r_s)$ com $r_1, \ldots, r_s$ apropriados.

(2) nenhum quádruplo de Z começa por $q_{\theta(z)}$.

Uma máquina n-regular apresenta pois o seu resultado apto a ser o início de uma nova computação, de uma outra máquina. Nota: permite-se a ocorrência à direita de $r_s$ de um número indefinido de B's, mas $q_{\theta(z)}$ tem que ser o símbolo imediatamente à esquerda dos símbolos diferentes de B.

*Teorema 11* Para toda a máquina de Turing Z podemos construir uma máquina Z' tal que, para cada n, Z' é n-regular e

$$\text{Res}_{Z'}^A [q_1 \overline{(m_1, \ldots, m_n)}] =$$
$$= q_{\theta(z')} \overline{\Psi_{Z;A}^{(n)}(m_1, \ldots, m_n)}^*$$

* A igualdade entre funções parciais implica a dos seus domínios.

Para isso, Z' faz a computação que Z faria até parar, conta o número de 1' s na fita, e reune-os num único bloco, apagando tudo o resto. Para evitar que Z' percorra a fita indefinidamente à procura de mais 1' s, é necessário fazer com que Z' leve a cabo a computação de Z sempre entre dois símbolos que servem de marcas. Essas marcas são deslocadas sempre que é necessário mais espaço para a computação.

*Teorema 12* Para cada máquina de Turing n-regular e para cada $p > o$, existe uma máquina de Turing $Z_p$, $(n + p)$-regular, tal que, quando

$$\mathrm{Res}^{A}_{Z}\left[\overline{q_1\,(m_1, \ldots, m_n)}\right] = q_{\theta(z)}\ \overline{(r_1, \ldots, r_s)}$$

também é

$$\mathrm{Res}^{A}_{Z_p}\left[q_1\,(\overline{k_1, \ldots, k_p, m_1, \ldots, m_n})\right] =$$

$$= q_{\theta(z)}\ \overline{(k_1, \ldots, k_p, r_1, \ldots, r_s)},$$ e quando $\mathrm{Res}^{A}_{Z}\left[q_1\,(\overline{m_1, \ldots, m_n})\right]$ é indefinido, também $\mathrm{Res}^{A}_{Z_p}\left[q_1\,(\overline{k_1, \ldots, k_p, m_1, \ldots, m_n})\right]$ é indefinido.

Por outras palavras, $Z_p$ actua sobre $m_1, \ldots, m_n$ como Z o faria, mas deixa $k_1, \ldots, k_p$ incólume.

$Z_p$ procede do seguinte modo: substitui os 1's dos argumentos $k_1, \ldots, k_p$ por um símbolo especial $\varepsilon$, depois executa a computação de Z de tal modo que sempre que encontra um $\varepsilon$ desloca todo o bloco de $\varepsilon$'s uma casa para a esquerda. No fim, volta a substituir os $\varepsilon$'s por 1's.

*Teorema 13 As máquinas $C_p$ que copiam.*

Para cada $n > 0$ e $p \geqslant 0$, define-se uma máquina de Turing $C_p$, $(n + p)$ - regular, tal que

$$\mathrm{Res}_{C_p}\,[q_1\ \overline{(k_1, \ldots, k_p, m_1, \ldots, m_n)}] =$$

$$= q_{p+16}\ \overline{(m_1, \ldots, m_n, k_1, \ldots, k_p, m_1, \ldots, m_n)}.$$

Isto é, os argumentos $(m_1, \ldots, m_n)$ são copiados à esquerda dos $(k_1, \ldots, k_p)$.

*Teorema 14 As máquinas $R_p$ de transferir.*

Para cada $n > 0$ e $p > 0$, define-se uma máquina de Turing $R_p$, $(p + n)$-regular, tal que

$$\mathrm{Res}_{R_p}\,[q_1\ \overline{(k_1, \ldots, k_p, m_1, \ldots, m_n)}] =$$

$$= q_{p+16}\ \overline{(m_1, \ldots, m_n, k_1, \ldots, k_p)}.$$

Isto é, os argumentos $(k_1, \ldots, k_p)$ trocam de lugar com os argumentos $(m_1, \ldots, m_n)$.

Estas máquinas podem fàcilmente obter-se das máquinas copiadoras $C_p$, apagando na sua expressão final os argumentos $(m_1, \ldots, m_n)$ da extrema direita.

*Teorema 15* Para cada máquina de Turing Z, n-regular, há uma máquina de Turing Z', n-regular, tal que, quando

$$\mathrm{Res}^{A}_{Z}\left[q_1\,(\overline{m_1, \ldots, m_n})\right] = q_{\theta(Z)}\,\overline{(r_1, \ldots, r_s)}$$

também é

$$\mathrm{Res}^{A}_{Z'}\left[q_1\,(\overline{m_1, \ldots, m_n})\right]$$

$$= q_{\theta(z')}\ \overline{(r_1, \ldots, r_s, m_1, \ldots, m_n)}$$ e quando

$\mathrm{Res}^{A}_{Z}\left[q_1\,(\overline{m_1, \ldots, m_n})\right]$ é indefinido, também

$\mathrm{Res}^{A}_{Z'}\left[q_1\,(\overline{m_1, \ldots, m_n})\right]$ é indefinido.

Assim, os argumentos $(m_1, \ldots, m_n)$ são preservados por Z'.

*Teorema 16* Sejam $Z_1, \ldots, Z_p$ máquinas de Turing. Seja $n > o$. Então existe uma máquina de Turing Z', n-regular, tal que

$$\mathrm{Res}^{A}_{Z'}\left[q_1\,(\overline{m_1, \ldots, m_n})\right] =$$

$$= q_{\theta(z')}\ \overline{\left(\overline{\psi^{(n)}_{Z_1;\,A}(m_1, \ldots, m_n)}, \ldots, \overline{\psi^{(n)}_{Z_p;\,A}(m_1, \ldots, m_n)}\right)}$$

Para a demonstração dos teoremas desta secção pode consultar-se Davis (1958).

## 8 — MEMÓRIA ASSOCIATIVA

Suponhamos que temos um certo número de quádruplos designados por $U_i$, e cada um deles associamos um nome $N_i$.

Codifiquemos, em binário por exemplo, cada $U_i$ e cada $N_i$. Disponhamo-los ao longo da fita de uma máquina de Turing, cada $U_i$ precedido pelo $N_i$ correspondente, e os vários pares $N_i U_i$ separados entre si por um X.

Pretendemos agora que a máquina identifique o quádruplo U que responde pelo nome N. A descrição instantânea inicial da máquina será a seguinte:

$$Y\,N\,q_1\;X\;N_1\;U_1\;X\;N_2\;U_2\;X\ldots X\;N_n\;U_n\;Y$$

Os Y's delimitam a extensão do arquivo.

A máquina procederá comparando N com cada um dos $N_i$'s, até encontrar uma correspondência perfeita entre os dois nomes. Depois de localizar N na sua memória, ela transferirá o quádruplo U para a posição ocupada primeiro por N, imediatamente à direita de Y. Isto é,

$$Y\;U\;q\;\;X\;N_1\;U_1\;X\ldots X\;N_n\;U_n\;Y$$

Num computador de série, as memórias utilizadas não são deste tipo embora possa haver vantagens na utilização de memórias associativas. O acesso à memória faz-se por um processo mais indirecto, nomeadamente através da localização numa matriz, do elemento de memória que contém a informação pretendida.

Exercício 4 Construir a máquina referida.

## 9 — EFICIÊNCIA DE UMA MÁQUINA DE TURING

Há dois aspectos complementares dessa eficiência. Por um lado, podem-se fazer computações que não seriam possíveis num autómato finito; utilizando apenas algumas operações simples pode, surpreendentemente, calcular-se funções bastante complexas. Por outro, esse carácter elementar das operações obriga a cálculos extremamente longos e fastidiosos. Apesar da lentidão, as máquinas de Turing não fazem necessàriamente um uso ineficiente da sua fita. Algumas modificações, no entanto, podem aproximá-las mais do computador convencional (entre elas o uso de várias fitas). Mas que fique bem explícito que elas são apenas modelos, mais pedagógicos e mais formalizados do que os computadores em uso. Ainda se não fez até hoje nenhuma máquina de Turing para fins práticos.

## 10 — OUTRAS MÁQUINAS DE TURING

Não podendo aqui entrar em detalhe, apenas referiremos

(1) a possibilidade de construção de uma máquina de Turing equivalente a qualquer outra dada, mas com um alfabeto de apenas dois símbolos. O problema aqui é mais o da codificação do que qualquer outro.

(2) a equivalência de uma fita infinita nos dois sentidos com uma fita infinita apenas num sentido (dobrando a fita e intercalando os quadrados).

(3) a possibilidade de utilização simultânea de várias fitas, controladas por uma saída constituída por um k-tuplo de saídas.
Essa possibilidade, entre outras coisas permite simular uma memória disposta em matriz n-dimensional.

Poderia parecer que as máquinas que utilizam várias fitas tivessem a possibilidade de computar funções que não estivessem ao alcance das convencionais máquinas só com uma fita. Para vermos que isso não é assim, basta pensarmos que podemos fazer uma correspondência entre os quadrados das K fitas, com os quadrados da fita de uma máquina convencional do seguinte modo:

Atribuindo aos quadrados de cada uma das K fitas (em número finito), quadrados da fita única espaçados de K em K.

(4) a possibilidade de construção de uma máquina de Turing equivalente a qualquer outra dada, mas apenas com dois estados internos. Isso consegue-se ampliando enormemente o alfabeto da máquina.

## 11 — EXERCÍCIOS

5 — Construir uma máquina de Turing que enuncia todos os números binários, começando com uma fita em branco.

6 — Construir uma máquina de Turing que gere todas as sequências de parêntesis bem formadas.

7 — Diz-se que uma máquina de Turing «aceita uma sequência finita de símbolos», se começando

a máquina numa fita cuja expresão inicial seja essa, e só para essa sequência, a máquina pára com a fita em branco.

Construir uma máquina que aceite a seguinte sequência:

$$10,\ 1100,\ 111000,\ \ldots,\ 1^n\ 0^n$$

## 12 — MÁQUINA DE TURING UNIVERSAL

Dada uma função computável f e uma máquina de Turing T que a computa, quando um número x é escrito, em notação simbólica, na fita de outro modo vazia de T, se T iniciar o seu funcionamento no estado $q_1$, à esquerda do último símbolo de x, então, quando a máquina pára, o número f(x) está escrito na sua fita.

Turing mostrou como construir uma máquina fixa U, com a seguinte propriedade: para toda e qualquer máquina de Turing T, existe uma sequência de símbolos $d_T$ tal que, se o número x é escrito, em notação simbólica, e seguido da sequência $d_T$, na fita de outro modo vazia de U, e se U for começada imediatamente à esquerda do último símbolo de $d_T$, então, quando U parar, o número f(x) estará escrito na sua fita; f(x) é o número que seria computado por T se começada apenas com x na sua fita.

Por outras palavras, por muito complexa que seja a estrutura de uma máquina de Turing T, o seu comportamento pode ser imitado por uma máquina única e fixa U, independentemente do número de estados de T e da extensão do seu alfabeto. Basta que seja fornecida a U uma sequência de símbolos $d_T$, para que ela compute a mesma função que T.

O que vem a ser $d_T$? Òbviamente, um modo de descrever a máquina T à máquina U.

A máquina U simulará T etapa por etapa. Saberá em que ponto da sua fita T começa, e conservará a cada momento uma descrição completa da fita de T. Para saber o que fará T na etapa seguinte, basta-lhe consultar a descrição $d_T$ dos quádruplos de T, memorizar o que T faz para um dado por $q_i\ s_j$ correspondente à descrição instantânea de T nessa etapa, mover-se até à descrição instantânea de T que consta da sua fita, e executar na sua fita o que T faria. Mover-se-á em seguida para o novo quádruplo de T correspondente à nova descrição instantânea, e recomeçará o ciclo.

Estas operações envolvem várias máquinas tipo como sejam as que considerámos na secção 7 (copiadoras, transferidoras, etc.), e as de memória associativa que localizam, extraem e arquivam informação. Envolvem ainda a existência de uma inter-ligação dessas máquinas parciais, que delimite as suas intervenções e sancione da oportunidade das mesmas; enfim, de uma coordenação.

Não faremos aqui a descrição formal da máquina de Turing Universal, porquanto a sua apresentação arrastar-nos-ia para o cálculo dos predicados, e obrigar-nos-ia a considerar a aritmetização das máquinas de Turing através do chamado «número de Gödel de uma máquina de Turing». Pode consultar-se para isso Davis (1958).

A sua descrição intuitiva completa, também não será feita porque, extensa que seria, pouco faria a acrescentar à descrição sumária que fizemos. Note-se no entanto que há muitas maneiras de construir uma máquina de Turing Universal.

## 13 — LIMITAÇÕES DA COMPUTABILIDADE

Vamos referir nesta secção os problemas que se demonstram serem algorìtmicamente irresolúveis. Isso equivale a afirmar que não pode existir uma máquina de Turing que os resolva.

Seja dada uma descrição instantânea inicial de uma máquina de Turing; há dois casos possíveis:

(1) a máquina é *aplicável* a essa descrição instantânea, isto é, após um número finito de etapas ou ciclos a máquina pára com uma descrição instantânea final* na sua fita.

(2) a máquina é *não aplicável* a essa descrição instantânea, isto é, não surge nunca uma descrição instantânea final, e a máquina envereda por uma via interminável.

É portanto pertinente o seguinte problema geral:

**Problema de decisão da aplicabilidade (Halting Problem)**

Sejam dados os quádruplos de uma máquina de Turing e uma descrição instantânea dessa

* No sentido da definição 15.

máquina. Pretendemos decidir se a máquina é ou não aplicável a essa descrição instantânea. Este é um problema típico de resolução por algoritmo. De facto, para o resolvermos, devemos encontrar um método geral (algoritmo ou máquina) que permita resolver automàticamente o problema da aplicabilidade de uma máquina dada, arbitrária, a uma configuração instantânea dada, mas também arbitrária.

Suponhamos agora que na fita de uma máquina de Turing estão representados os seus próprios quádruplos, escritos com o alfabeto da máquina. Se a máquina for aplicável à descrição formada por essa expressão da fita e pela situação inicial, diremos que ela é *auto-aplicável;* caso contrário, *não auto-aplicável.* Põe-se portanto o seguinte problema geral:

### Problema de decisão da auto-aplicabilidade

Para um conjunto de quádruplos arbitrários, decidir a que classe pertence a máquina; se à classe das máquinas auto-aplicáveis, se à classe das máquinas não auto-aplicáveis.

*Teorema* Este problema é algorìtmicamente irresolúvel (no sentido dado no início desta secção).

É claro que, deste teorema, resulta imediatamente que o problema mais geral da aplicabilidade é também algorìtmicamente irresolúvel.

### Problema da decisão da tradutabilidade

Dada uma máquina de Turing arbitrária, e duas descrições instantâneas dessa máquina *a* e *b* pretende-se saber se dada *a* inicialmente, a máquina a transforma em *b* após um número finito de etapas.

Visto que o problema da aplicabilidade se reduz ao problema da tradutabilidade (considerando a colecção das descrições instantâneas finais $b_1, \ldots, b_n$), temos que:

*Teorema* O problema da tradutabilidade é algorìtmicamente irresolúvel.

### Problema da impressão

Dada uma máquina de Turing, arbitrária, podemos ou não saber se a máquina imprimirá eventualmente o símbolo $s_i$ partindo de uma descrição instantânea *a* também arbitrária?

*Teorema* O problema da impressão é algorìtmicamente irresolúvel porque uma máquina que não use normalmente o símbolo $s_i$, pode ser alterada de forma a que o imprima antes de parar*; portanto o problema da impressão na máquina alterada é equivalente ao problema da aplicabilidade (halting problem) na máquina original. Visto que uma máquina que usa normalmente $s_i$ pode ver o seu alfabeto modificado pela substituição de $s_i$ por outro símbolo qualquer, e ser subsequentemente alterada da maneira referida, então a solubilidade do problema da impressão arrastaria a solubilidade do problema da aplicabilidade em todos os casos (máquinas que usam $s_i$, e máquinas que não usam $s_i$), o que é contrário aos teoremas anteriores.

### Problema da aplicabilidade da fita em branco

Será que podemos ter uma máquina que decida se uma máquina de Turing qualquer pára ou não, se iniciada numa fita em branco?

*Teorema* Este problema é algorìtmicamente irresolúvel.

### Problema da aplicabilidade uniforme

Será que podemos ter uma máquina que nos diga se uma outra máquina de Turing arbitrária pára se iniciada em toda e qualquer descrição instantânea inicial?

*Teorema* Este problema é algorìtmicamente irresolúvel.

### Considerações gerais

A irresolubilidade destes problemas tem um carácter construtivo, porque a demonstração que se efectua para cada problema fornece o método de construção da máquina que faz a contraprova. É de notar, aliás, que a existência de problemas algorìtmicamente irresolúveis, faz com que se ponha com acuidade a possibilidade de um algoritmo desejado não existir. Portanto, paralelamente aos esforços dispendidos na tentativa de construir esse algoritmo, é de ensaiar também a demonstração da impossibilidade do mesmo.

---

* O leitor fàcilmente imaginará essa modificação

O sucesso poderá assim ocorrer numa das duas direcções.

Não podemos deixar de sublinhar que a irresolubilidade destes problemas, postos desta forma geral, não é de modo algum pretexto da irresolubilidade de cada um dos problemas particulares que fazem parte dessa classe geral como já dissemos na introdução, logo de início.

Essa irresolubilidade quer apenas dizer que a classe de problemas em causa é tão vasta que não existe um único algoritmo capaz de dar conta de todos os problemas particulares nela incluídos.

Outro ponto essencial, é que a existência de um algoritmo pode não ser garantia da sua aplicação. Com efeito, algoritmos há que, ou por demasiado morosos e extensos, ou porque implicam a existência de uma quantidade de memória e de uma quantidade de tempo que não estão à disposição, não são na prática utilizados. Assim, a impossibilidade prática de usar um algoritmo sistemático para o jogo de xadrês embora ele exista, obriga a considerar os chamados métodos heurísticos.

Esperamos que em breve possamos dedicar a esses métodos toda a atenção que merecem, em próximas realizações do CEC.

## 14 — APLICAÇÕES

A teoria da computabilidade fornece, antes de tudo o mais, um modelo formal.

É a tentativa de adequação desse modelo às várias estruturas que lhe são submetidas que está na origem das suas aplicações. Referiremos três.

Assim, demonstrada a equivalência, que já mencionámos, entre as funções recursivas gerais e as funções computáveis, ou melhor, entre as funções A-parcialmente recursivas gerais e as funções A-parcialmente computáveis (e entre os seus diversos sub-tipos), o aparato das funções recursivas, indispensável à metamatemática, passou a ter como opção o formalismo mais intuitivo das máquinas de Turing. Essa opção, permite por outro lado uma justificação mais directa, por parte da teoria da computabilidade, dos processos a empregar na realização das máquinas de computar. Entre essas justificações figuram (1) o número de funções, quais, e que operações sobre funções são necessárias para obter toda a classe de funções computáveis, (2) a permissibilidade de utilização de índices e de ordens recursivas nas linguagens de programação, (3) a própria estrutura das linguagens de programação.

É de notar ainda que, ao comparar-se os computadores actuais à máquina de Turing Universal, se verifica que tanto nesses computadores como na máquina o programa é inserido na memória juntamente com os dados; isso significa que as operações do programa vão sendo escolhidas de acordo com o valor do operando em cada etapa.

Outro campo de aplicação da teoria da computabilidade que as máquinas de Turing vieram suscitar, é o estudo das linguagens formais, que parte do conceito de frase aceite por uma máquina de Turing já aqui referido num dos exercícios (7). A possibilidade ou impossibilidade de essas linguagens formais darem conta da *competência* de um indivíduo que conhece uma linguagem natural, é o objecto de uma outra destas conferências. Nela se mencionará o paralelo entre os diversos tipos de linguagens formais e os correspondentes *autómatos intermédios*, isto é, os diferentes tipos de autómatos que tendo um comportamento que não pode ser imitado por nenhum autómato finito, são no entanto inferiores às máquinas de Turing em versatilidade (não computam portanto todas as funções computáveis)

Por último, referiremos a importância epistemológica das máquinas de Turing, em particular da máquina de Turing Universal.

Mc Culloch e Pitts mostraram que os cérebros humanos, juntamente com a memória externa ilimitada a que têm acesso, são máquinas de Turing Universais e que qualquer máquina de Turing pode ser realizada com neurónios.

À pergunta: «podem as máquinas evoluir?» John von Neumann reponde que dada uma máquina de Turing conveniente, ligada a um duplicador da sua fita e a um armazém de partes, ela pode imprimir a descrição de si própria na sua fita e comandar a execução de uma máquina que lhe é igual. Depois dessa primeira reprodução seguir-se-ão outras, e um sistema de mutações e selecções ditará quais sobrevivem. É de notar que é indispensável um mínimo de complexidade para a reprodução ser possível e que as moléculas de ADN alcançam esse limiar.

A quarta conferência terá por encargo dar toda a oportunidade ao assunto dos modelos neuronais.

## 15 – BIBLIOGRAFIA


1) CELLERIER, G. *Cibernétique et Épistemologie* collection Études d'épistèmologie génètique P.U.F. 1968 ~ 135 págs. ~ 130$00
pode encontrar-se um resumo em 6).

2) DAVIS, Martin *Computability and Unsolvability* McGraw-Hill
1958 210 págs. ~ 300$00
Fornece a teoria formalizada e estudos mais avançados. Não requer conhecimentos especiais excepto nos últimos capítulos. Tem aplicações aos Problemas Combinatórios, às Equações Diofânticas e à Lógica Matemática. Trata das funções Recursivas.

3) HOPCROFT; ULLMAN *Formal Languages and their relation to Automata.*
Addison — Wesley London
1969 234 págs. ~ 400$00

4) Mc CULLOCH, Warren S. *Embodiments of Mind*
M. I. T. Press 1965 402 págs. 526$50
Reúne as principais contribuições de um dos pioneiros da cibernética às teorias dos modelos neuronais.

5) MINSKY, Marvin L *Computation: Finite and Infinite Machines*
Prentice-Hall 1967 317 págs. 544$50
Obra bastante pedagógica que não exigindo conhecimentos iniciais faz um apanhado estimulante dos Autómatos Finitos (redes Neuronais), Autómatos Infinitos, Funções Recursivas e suas relações com as máquinas de Turing, Problemas irresolúveis, Modelos de Computadores digitais, Sistemas de manipulação de símbolos de Post Bases simples para a computabilidade, Exercícios com soluções e uma Bibliografia detalhada e orientada em assuntos. Trata os problemas de forma rigorosa e atraente.

6) MONIZ PEREIRA, Luís *Cibernética — Epistemologia* publicado na Técnica de Dezembro 1968 (N.º 383).

7) SIMÕES DA FONSECA, J. L. *Bases Neuronais da Vida Psiquica.* Clínica Psiquiátrica da Faculdade de Medicina de Lisboa. Centro de Estudos Egas Moniz. 1969, 335 págs.

8) TRAHTENBROT, B .A. *Algoritme Machiness et à calculer.* Dunod 1963, 149 págs. ~ 100$00. Dá a noção de algoritmo e expõe a teoria das máquinas de Turing e limitações.

9) SIMÕES DA FONSECA, J.L. (editor) *Signification and Intention.* Clínica Psiquiátrica da Faculdade de Medicina de Lisboa. Centro de Estudos Egas Moniz. 1970, 97 págs

10) ARBIB, M. *Theories of Abstract Antomata.*
1969 Prentice-Hall 412 págs. 675$00
Livro avançado que requer conhecimentos prévios.

11) MONIZ PEREIRA, LUÍS e MONTEIRO, LUÍS F. *Aspectos Cibernéticos da Epistemologia*
in Actas do Colóquio «Epistemologia das Ciências do homem», realizado no I. S. T. pelo C.E.C. em 1970, de colaboração com o G.E.L.T. e o G.E.A. 1970 Editora Presença ~ 40$00.

12) PASK, GORDON
*Uma introdução à cibernética*
Tradução portuguesa com bibliografia e glossário de L. Moniz Pereira, a sair brevemente, publicado pela Editora Arménio Amado. Original inglês:
*An Approach to Cybernetics*
1961 Hutchinson-Radius Books.