Tese de Church

Diante do fato demonstrado de que a máquina de Turing não tem o seu poder computacional ampliado através de qualquer alteração estrutural (vide resumo anterior). Isto é, tornar a fita duplamente infinita, aumentar o número de fitas, de cabeçotes, criar uma máquina não-determinística, nada aumenta o poder computacional da máquina de Turing conforme foi concebida.

Diante da existência de modelos computacionais distintos, como:

- Máquina de Turing
- Máquina de Post
- Funções recursivas
- Cálculo-λ

Diante da demonstração de que os modelos são equivalentes em seu poder computacional.

Enfim, todos estes fatos nos levam a considerar que todos os modelos de computação são equivalentes (não apenas os estudados até então) e enunciar a **Tese de Church**:
"*As máquinas de Turing são versões formais de algoritmos, e assim sendo, nenhum procedimento computacional será considerado um algoritmo se não puder ser representado como uma máquina de Turing.*"

Quando se utiliza a Tese de Church, e isso é usual em computação, pode-se demonstrar a equivalência entre um modelo computacional qualquer e a máquina de Turing apenas simulando no modelo a máquina de Turing.

Computação por Gramáticas

Pode-se utilizar gramáticas como dispositivos geradores de linguagens, mas também como dispositivos computacionais. Desta maneira, as gramáticas podem ser vistas como computadores de funções, desde que tais funções sejam de e para cadeias. Para isso, pode-se usar a Tese de Church.

Lema: Seja M=(K, Σ, δ, s) uma máquina de Turing qualquer. Então existe uma gramática G tal que para quaisquer configurações (q, u, a, v), (q', u', a', v') de M, $(q, u, a, v) \vdash_M^* (q', u', a', v') \Leftrightarrow [uqav] \Rightarrow_G^* [u'q'a'v']$. Onde, [e] são símbolos não presentes em K ou Σ, e assume-se que K e Σ são disjuntos.

Demonstra-se representando cada configuração como uma cadeia de símbolos, na qual o estado é um símbolo não-terminal de G, e separa o símbolo sob o cabeçote da cadeia à esquerda deste. Os símbolos [e] inseridos servem para indicar o final dos símbolos na fita, isto é, à direita de] só há símbolos "#". Produzindo uma representação em forma de regra de produção para cada movimento da máquina de Turing a ser simulada, obtém-se o efeito desejado. G=(K∪{[,],h,S}, Σ, P, S).

Funções Computáveis por Gramática

A partir do Lema anterior pode-se definir uma função gramaticamente computável.

Def.: Sejam Σ_0 e Σ_1 alfabetos que não contêm #, e seja f uma função de Σ_0^* para Σ_1^*. Então f será computável por gramática se e somente se houver uma gramática G=(V,T,P,S), onde $\Sigma_0, \Sigma_1 \subseteq T$, e houver cadeias x, y, x', y' $\in (V \cup T)^*$, tais que para qualquer u $\in \Sigma_0^*$, v $\in \Sigma_1^*$, $f(u)=v \Leftrightarrow xuy \Rightarrow_G^* x'vy'$. A definição para funções numéricas é semelhante à da Máquina de Turing. Observa-se que x, y, x', y' são marcadores associados à gramática G, assim a computação só terá efeito dentro dos marcadores.

Teorema: Toda função Turing-computável é gramaticalmente-computável.

Demonstra-se a partir de uma máquina de Turing qualquer M=(K, Σ, δ, s), para a qual $f(u)=v \Leftrightarrow (s,\#u\#) \vdash_M^* (h,\#v\#)$. Aplica-se o Lema anterior para obter G, e faz-se x=x'="[#", y="s#]", y'="h#]".

Funções μ-Recursivas

Definem-se as funções recursivas primitivas e, a partir destas e de minimização ilimitada sobre funções regulares de Kleene, definem-se as funções μ-recursivas.

Faz-se um paralelo com cálculo-λ, fundamento das linguagens funcionais como Scheme.

Máquinas de Turing Universais

Pode-se definir uma codificação padrão para um alfabeto infinito contável e um conjunto de estados infinito contável em termos de cadeias de símbolos. Aceita-se, dessa forma, que qualquer máquina de Turing pode ter seus estados e símbolos codificados dentro do padrão. Assim tem-se os seguintes conjuntos:
$K_\infty = \{q_1, q_2, ...\}$ e $\Sigma_\infty = \{a_1, a_2, ...\}$,

De tal maneira que para toda máquina de Turing o conjunto de estados seja um subconjunto finito de K_∞, e o alfabeto da fita seja um subconjunto finito de Σ_∞. Adota-se a seguinte correspondência entre os símbolos componentes de uma máquina de Turing e cadeias sobre o alfabeto {I}.

σ	λ(σ)
q_i	I^{i+1}
h	I
L	I
R	II
a_i	I^{i+2}

Esta escolha elimina a possibilidade de haver dois elementos do mesmo conjunto representados da mesma forma.

Usa-se um outro símbolo ("c") para configurar a completamente a máquina, assim o alfabeto de representação será {c, I}. Portanto, cada símbolo e cada estado de uma máquina de Turing M=(K, Σ, δ, s) podem ser representados, e cada elemento da função de transição de uma máquina de Turing também pode, da seguinte maneira: seja δ(q, a)=(p, b), onde q e p $\in K_\infty \cup$ {h}, a $\in \Sigma_\infty$, e b $\in \Sigma_\infty \cup$ {L, R}, então cada elemento será representado por k.ℓ cadeias S_{pr}="$cw_1cw_2cw_3cw_4c$", ($1 \le p \le k$, $1 \le r \le \ell$) onde $w_1=\lambda(q)$, $w_2=\lambda(a)$, $w_3=\lambda(p)$ e $w_4=\lambda(b)$. Faça-se $S_0=\lambda(s)$ e defina-se a função de codificação $\rho(M)=cS_0cS_{11}S_{12} ... S_{k\ell}c$.

A máquina de Turing Universal opera sobre o alfabeto {c, I, #}, e recebe em sua fita a codificação ρ(M) seguida da codificação ρ(w). A operação da máquina de Turing Universal então é bastante simples, simula a execução de M a partir de s, acompanhando a função de

transição e o símbolo encontrado na fita. Observe-se que mesmo que w contenha símbolos #, ρ(w) não possui #.

Assim define-se a máquina de Turing Universal U como:
$U=(K_U, \Sigma_U, \delta_U, s_U)$, tal que para toda máquina de Turing $M=(K, \Sigma, \delta, s)$ e para toda cadeia $w \in \Sigma^*$,
1. Se (h, u$\underline{a}$v) é uma configuração de parada de M tal que $(s, \#w\underline{\#}) \vdash_M^* (h, u\underline{a}v)$, então $(s_U, \#\rho(M)\rho(w)\underline{\#}) \vdash_U^* (h, \#\rho(uav)\underline{\#})$
2. Se $(s_U, \#\rho(M)\rho(w)\underline{\#}) \vdash_U^* (h, u'\underline{a}'v')$ é uma configuração de parada de U, então a'=#, v'=ε, u'=#ρ(uav), para algum u, a, v tais que (h, u$\underline{a}$v) seja uma configuração de parada de M, e que $(s, \#w\underline{\#}) \vdash_M^* (h, u\underline{a}v)$.

Esta máquina define o padrão de computação usual, qualquer dispositivo computacional pode ser representado por uma máquina de Turing Universal, e cada máquina de Turing construída representa um algoritmo, um programa para a máquina Universal.

Computabilidade e Decidibilidade

Se há problemas não-resolvíveis por máquinas de Turing, então, pela Tese de Church, estes não podem ser resolvidos por algoritmos de qualquer tipo. Esta conclusão motiva o estudo e a identificação dos problemas solucionáveis. Partindo do conceito de decidibilidade para máquinas de Turing pode-se chegar a conclusões importantes.

Teorema: Toda Linguagem Turing-decidível é Turing-aceitável.
Demonstra-se construindo a máquina de Turing de aceitação, a partir de uma máquina de decisão.

Teorema: Se L é uma Linguagem Turing-decidível então o seu complemento $\overline{L}$ também é Turing-decidível.
Demonstra-se construindo a máquina de Turing a partir de uma máquina de decisão para L.

As perguntas que estão sem resposta são:
1. Toda linguagem Turing-aceitável é Turing-decidível?
2. O complemento de uma linguagem Turing-aceitável é Turing-aceitável?

Se houvesse uma máquina de Turing capaz de "descobrir" a saída de uma máquina de Turing M qualquer, então toda linguagem Turing-aceitável seria Turing-decidível. Então pode-se resumir esta constatação na seguinte linguagem $K_0=\{\rho(M)\rho(w): M \text{ aceita } w\}$. Se K_0 for Turing-decidível por alguma máquina M_0, então toda linguagem Turing-aceitável será Turing-decidível.

Se K_0 é Turing-decidível, então $K_1=\{\rho(M): M \text{ aceita } \rho(M)\}$ também o é, e a máquina de Turing M_1 que a decide é composta de uma máquina de codificação, que codifica e copia a cadeia recebida w em ρ(w), e passa o controle para a máquina M_0.
Assim o resultado final de M_0 será **Y** se e somente se:
a) w é ρ(M), e
b) M aceita w, isto é, ρ(M);
que é a definição de K_1. Entretanto se K_1 é Turing-decidível, então seu complemento também o é:
$\overline{K}_1=\{w \in \{I, c\}^*$: w não é a codificação de uma máquina de Turing M, ou w=ρ(M) para alguma máquina de Turing M que não aceita entrada ρ(M)$\}$.
Entretanto $\overline{K}_1$ não é sequer Turing-aceitável, porque se o fosse haveria uma máquina de Turing M^* que a aceita. Pela definição de $\overline{K}_1$, $\rho(M^*) \in \overline{K}_1$ se e somente se M^* não aceita $\rho(M^*)$. Mas M^* deve aceitar $\overline{K}_1$, assim $\rho(M^*) \in \overline{K}_1$ se e somente se M^* aceita $\rho(M^*)$. Portanto M^* aceita $\rho(M^*)$ se e somente se M^* não aceita $\rho(M^*)$, o que é absurdo, logo deve ter havido erro na hipótese sobre M^*, que não deve existir. Logo tem-se:

Teorema: Nem toda linguagem Turing-aceitável é Turing-decidível.

Teorema: Os complementos de algumas linguagens Turing-aceitáveis não são Turing-aceitáveis.

Este é o problema da parada da máquina de Turing (K_0), através dele sabe-se que há problemas que não admitem solução algorítmica. Tais problemas são chamados não-solucionáveis. Por outro lado, um problema é dito solucionável se existe um algoritmo que o resolve, isto é, se há um procedimento de decisão para ele.

Teorema: Uma linguagem é Turing-decidível se e somente se tanto ela quanto o seu complemento são Turing-aceitáveis.

Teorema: Uma linguagem é Turing-aceitável se e somente se ela é a linguagem de saída de alguma máquina de Turing.

Definição: Uma Linguagem é dita Turing-enumerável se e somente se existe uma máquina de Turing que enumera suas cadeias.

Teorema: Uma linguagem é Turing-aceitável se e somente se ela é Turing-enumerável.

Problemas não Resolvíveis sobre MT

Teorema: Os problemas a seguir são não-solucionáveis:
a) Dada uma máquina de Turing M e uma cadeia de entrada w, M pára com a entrada w?
b) Para uma específica máquina M, dada uma cadeia de entrada w, M pára com a entrada w?
c) Dada uma máquina de Turing M, M pára com a fita de entrada vazia?
d) Dada uma máquina de Turing M, há alguma cadeia de entrada com a qual M pára?
e) Dada uma máquina de Turing M, M pára com toda cadeia de entrada?
f) Dadas duas máquinas de Turing M_1 e M_2, elas param com as mesmas cadeias de entrada?
g) Dada uma máquina de Turing M, a linguagem que M aceita é regular? É livre de contexto? É Turing-decidível?

Problemas não Resolvíveis sobre Gramáticas

Teorema: Os problemas abaixo são não-solucionáveis:
a) Para uma gramática arbitrária dada G e uma cadeia w, determinar se $w \in L(G)$.
b) Para uma específica gramática G_0 e uma cadeia w, determinar se $w \in L(G_0)$.
c) Dadas duas gramáticas arbitrárias G_1 e G_2, determinar se $L(G_1)=L(G_2)$.
d) Para uma gramática arbitrária G, determinar se $L(G)=\phi$.

Problemas não Resolvíveis para GLC

Teorema: Os problemas a seguir são não-solucionáveis:

a) Dadas duas gramáticas livres de contexto G_1 e G_2, determinar se $L(G_1) \cap L(G_2) = \phi$.
b) Para uma gramática livre de contexto G, determinar se G é ambígua.

Complexidade Computacional

O conceito de complexidade está diretamente associado à realidade objetiva, isto é, à prática da computação em dispositivos reais. Há problemas que, apesar de solucionáveis, têm uma *complexidade* em tempo tão elevada que torna impraticável a sua implementação computacional.

Definição: *Decidibilidade em tempo*. Seja T: $\mathbb{N} \rightarrow \mathbb{N}$ uma função numérica, e $L \subseteq \Sigma_0^*$ uma linguagem, e M=(K, Σ, δ, s) uma máquina de Turing com k fitas e com $\Sigma_0 \subseteq \Sigma$. Diz-se que M decide L em tempo T se sempre que $w \in L$, $(s, \#w\#,\#, ...,\#) \vdash_M^t (h, \#\checkmark\#,\#, ...,\#)$ para algum $t \leq T(|w|)$;
e sempre que $w \notin L$ $(s, \#w\#,\#, ...,\#) \vdash_M^t (h, \#\text{N}\#,\#, ...,\#)$ para algum $t \leq T(|w|)$;

Diz-se que L é decidível em tempo T se há algum $k > 0$ e alguma máquina de Turing com k fitas que decide L em tempo T. A classe de todas as linguagens decidíveis em tempo T é denotada por TIME(T).

Assim adota-se como limite para o número de passos da máquina de Turing por uma função do comprimento da entrada. Assim não há função T tal que $T(n) < 2n + 4$ para algum $n \geq 0$ (já que é necessário percorrer a cadeia de entrada, apagá-la, e escrever ✓ ou N).

Encontrar um limite superior para a função T pode não ser trivial. Entretanto o objetivo da teoria da complexidade computacional é escolher, dentre as várias possíveis máquinas de Turing para decidir uma dada linguagem, aquela capaz de terminar em T passos, onde T é o menor possível, ou, se não for possível, fornecer uma demonstração rigorosa da impossibilidade de uma máquina tão rápida.

Taxa de crescimento de funções

A questão mais relevante a respeito da complexidade computacional é a taxa de crescimento no tempo, os valores constantes podem ser aproximados sempre do menor possível (usando para isso uma máquina de Turing com mais fitas).

Definição: Sejam f e g funções de $\mathbb{N}$ para $\mathbb{N}$. Escreve-se $f=O(g)$ se e somente se há uma constante $c > 0$ e um inteiro $n_0 \in \mathbb{N}$ tal que: $f(n) \leq c.g(n)$, para todo $n \geq n_0$.

Teorema: Seja $f(n) = \sum_{j=0}^{d} a_j n^j$ um polinômio e $r > 1$. Então $f = O(r^n)$.

Simulações limitadas em tempo

Teorema: Suponha que uma linguagem L é decidida por uma máquina de Turing M_1 com uma fita duplamente infinita em tempo T_1. Então L é decidida por uma máquina de Turing padrão M_2, com uma fita, em tempo T_2, onde para todo $n \in \mathbb{N}$, $T_2(n)=6T_1(n) + 3n + 8$.

Teorema: Suponha que uma linguagem L é decidida por uma máquina de Turing M_1 com k fitas em tempo T_1. Então L é decidida por uma máquina de Turing padrão M_2, com uma fita, em tempo T_2, onde, $T_2(n)= 4T_1(n)^2 + (4n + 4k + 3)T_1(n) + 5n + 15$.
Corolário: Se L é decidida por uma máquina de Turing com k fitas em tempo T, então L é decidida em tempo $T'=O(T^2)$ por uma máquina de Turing com uma fita.

Classes $\mathcal{P}$ e $\mathcal{NP}$

Definição: Define-se $\mathcal{P}$ (decidíveis em tempo polinomial) como a classe de linguagens:
$\mathcal{P} = \cup \{TIME(n^d): d > 0\}$.

A classe $\mathcal{P}$ coincide com a classe de problemas que podem ser resolvidos realisticamente por computadores.

Definição: Seja T: $\mathbb{N} \rightarrow \mathbb{N}$ uma função numérica, e $L \subseteq \Sigma_0^*$ uma linguagem, e M=(K, Σ, Δ, s) uma máquina de Turing não determinística. Diz-se que M aceita L em tempo não determinístico T se para todo $w \in \Sigma_0^*$, $w \in L$ se e somente se $(s, \#w\#) \vdash_M^t (h, v\underline{\sigma}u)$ para algum $v, u \in \Sigma^*$, $\sigma \in \Sigma$, e $t \leq T(|w|)$. Diz-se que L é aceitável em tempo não determinístico T se há uma máquina de Turing não determinística que aceita L em tempo não determinístico T. A classe de linguagens aceitáveis em tempo não determinístico T é denotada por NTIME(T). Define-se $\mathcal{NP} = \cup \{NTIME(n^d): d > 0\}$.

Uma computação é considerada infinita se necessita de mais de $T(|w|)$ passos para uma entrada w.

Teorema: $\mathcal{NP} \subseteq \cup \{TIME(r^{n^d}): r, d > 0\}$.

Classe $\mathcal{NP}$-Completo

Definição: Sejam Σ e Δ alfabetos. Uma função $f: \Sigma^* \rightarrow \Delta^*$ é dita computável em tempo T por uma máquina de Turing determinística com k fitas M=(K, Σ', δ, s) se e somente se para todo $x \in \Sigma^*$,
$(s, \#x\#,\#, ..., \#) \vdash_M^t (h, \#f(x)\#,\#, ..., \#)$, para algum $t \leq T(|x|)$. Diz-se que f é computável em tempo T se existe alguma máquina de Turing M que computa f em tempo T. Diz-se que que *f* é computável em tempo polinomial se existe um polinômio T tal que *f* seja computável em tempo T.

Definição: Sejam as linguagens $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$. Uma função computável em tempo polinomial $\tau: \Sigma_1^* \rightarrow \Sigma_2^*$ é chamada de uma transformação em tempo polinomial de L_1 para L_2 se e somente se para cada $x \in \Sigma_1^*$, é verdadeiro: $x \in L_1$ se e somente se $\tau(x) \in L_2$.

Definição: Uma linguagem L é dita $\mathcal{NP}$-completa se e somente se $L \in \mathcal{NP}$, e para toda linguagem $L' \in \mathcal{NP}$, há uma transformação polinomial de L' para L.

Teorema: Seja L uma linguagem $\mathcal{NP}$-completa. Então $\mathcal{P} = \mathcal{NP}$ se e somente se $L \in \mathcal{P}$.

Problemas $\mathcal{NP}$-Completos:

- Programação Linear Inteira
- Ciclo Hamiltoniano
- Caixeiro Viajante

Lida-se com esses problemas através de algoritmos de aproximação.