Linguagens Formais e Autômatos

Máquinas de Turing e Recursividade

Eduardo Furlan Miranda

Baseado em: GARCIA, A. de V.; HAEUSLER, E. H. Linguagens Formais e Autômatos. Londrina: EDA, 2017.

Máquina de Turing (MT)

- 1900: desafio intelectual do programa de Hilbert
  - Provar a consistência e decidibilidade da matemática
    - Necessidade de formalizar operações efetivas sobre objetos concretos (números naturais)
      - Tentativa de encontrar um conjunto completo e consistente de axiomas para toda a matemática
- 1931: prova da indecidibilidade, por Gödel
  - Existência de enunciados matemáticos indecidíveis
- 1936: cálculo-λ, por Church
  - Conceito de funções anônimas (funções sem nome)
    - Equivalente à MT em termos de capacidade computacional
      - Influenciou linguagens de programação funcionais, como Lisp

MT

- 1937: MT, por Turing
  - Modelo matemático (teórico) para estudar números computáveis
  - Conceito de "máquinas automáticas"
    - Uma máquina com uma fita para leitura e escrita de caracteres, e
    - Uma memória de estados finita
  - Equivalência com o conceito de computável de Church

Tese de Church-Turing

- Qualquer função que pode ser calculada por um procedimento efetivo (ou algoritmo), pode ser calculada por uma MT

- A MT se torna um modelo fundamental para a teoria da computação

- Influenciando o desenvolvimento de computadores e linguagens de programação

- Tese = “funciona” . Amplamente aceito, porém não é uma verdade matemática no mesmo sentido de um teorema

Computabilidade

- É natural e não possui uma definição definitiva
  - Similar a outros conceitos fundamentais como tempo, espaço, movimento, força e vida
- “Computação” é tanto “Ciência”, quanto “Física” ou “Biologia”
  - Sujeita a construção de teorias potencialmente refutáveis
- A busca por um modelo formal do que é computável culminou na Tese de Church-Turing
  - Estabelecendo um fundamento sólido para a teoria da computação
    - E mostrando a intersecção entre matemática e ciência empírica

Se baseia em observações, experimentações e experiências diretas

Parada

- Problema da Parada (*Halting Problem*)
  - Turing provou a indecidibilidade do problema da parada
    - Questiona se existe um algoritmo que pode determinar se um programa de computador, dada uma entrada específica
      - Eventualmente irá parar (terminar)
      - Ou continuará a rodar indefinidamente

- Dos trabalhos de Turing e Church:
  - Não existe um processo algorítmico geral para determinar a veracidade de proposições matemáticas
    - Mostra as limitações de sistemas formais e de algoritmos

para = chega a um estado de aceitação

MT Universal (MTU)

- Turing é quem de fato demonstra matematicamente que é possível construir uma máquina programável
  - Que lê programas e os executa
  - Antes dos primeiros computadores
- Cada MT realiza uma só tarefa
- MT Universal (MTU)
  - Capaz de simular qualquer outra MT
    - Ilustra o conceito de um interpretador de linguagem de programação

MT e linguagens

- Cada MT aceita uma única linguagem
  - Existe um mapeamento entre a MT e a linguagem que ela aceita
- É possível definir uma quantidade infinita de MTs que aceitam a mesma linguagem, ao adicionar instruções inúteis que nunca serão executadas
- Quando uma MT nunca para, independentemente da cadeia de entrada, ou seja,
  - não consegue aceitar nenhuma cadeia de símbolos,
    - dizemos que ela reconhece a linguagem vazia porque não possui nenhuma cadeia

para = chega a um estado de aceitação

Linguagens e limitações de uma MT

- Máquina que aceita Σ^* : é a máquina que para em um estado final, para todas as cadeias em Σ^*

- Σ^* representa o conjunto de todas as possíveis cadeias que podem ser formadas usando os símbolos do alfabeto de entrada da máquina Σ , incluindo a cadeia vazia (ε)

- Toda MT define uma linguagem, e existem infinitas máquinas que definem a mesma linguagem
  - O inverso não é verdadeiro

- Existem muito mais linguagens não aceitas por MT,
  - do que linguagens aceitas

Recursividade

para = chega a um estado de aceitação

- Capacidade de uma linguagem ser decidida por uma MT que sempre para
  - retornando "sim" para cadeias que pertencem à linguagem
  - e "não" para as que não pertencem
- Uma linguagem é recursiva se existe uma função característica Turing-computável que mapeia cada cadeia da linguagem em 1 (se pertence) ou 0 (se não pertence)
- Essencialmente, uma linguagem é recursiva se existe um algoritmo que
  - sempre consegue determinar se uma dada cadeia faz parte dela ou não
  - o que garante que o processo de decisão sempre termina com uma resposta

Recursividade

- A recursividade em MTs está relacionada à capacidade de computação e à decidibilidade de problemas, ou seja,
  - a existência de um algoritmo que sempre consegue determinar se uma dada cadeia pertence ou não a uma linguagem
- Em linguagens de programação o conceito de recursividade é aplicado em um nível de abstração diferente
  - O termo "recursividade" é uma técnica utilizada onde uma função chama a si mesma repetidamente
- A recursividade em linguagens de programação é uma técnica prática para implementar algoritmos, enquanto na MT refere-se à capacidade fundamental de computar esses mesmos problemas

Decidível e semidecidível

- Linguagem recursivamente enumerável (RE) (semidecidível)
  - É aquela para a qual existe uma MT que aceita todas as cadeias da linguagem
    - mas pode não rejeitar cadeias fora da linguagem
      - Pode não parar

- Linguagem recursiva (decidível)
  - É um subconjunto das linguagens RE , onde a MT sempre para e decide corretamente se uma cadeia pertence ou não à linguagem
    - Sempre para , e aceita ou rejeita

Recursivamente Enumerável (RE)

sigma estrela

- Uma linguagem $L \subseteq \Sigma^*$ é RE se, e somente se, existe uma MT M, tal que, $\boxed{L(M) = L}$
  - Dada uma L que é RE, então pode-se construir uma MT que aceita exatamente essa linguagem
    - L : conjunto de cadeias formadas por símbolos de um alfabeto
    - L(M) : a linguagem aceita pela MT M
      - Conjunto de todas as cadeias w para as quais a MT M, ao receber w como entrada, para em um estado final
    - $L \subseteq \Sigma^*$: a linguagem L é um subconjunto de Σ^*

a linguagem aceita pela MT M é igual à linguagem L

RE = semidecidível, pode não parar

Codificação de MT

- M0 = 00100200_02_0001021
  - 001: Número de estados (2: q_0, q_{aceita})
  - 002: Número de símbolos do alfabeto de entrada (2: a, b)
  - 000: Número de símbolos do alfabeto da fita (3: a, b, caracter)
  - 000: Estado inicial (q_0)
  - 001: Estado de aceitação (q_{aceita})
  - 020001021: Transição única:
    - 02 = q_0 (estado atual)
    - 00 = caracter (símbolo lido)
    - 01 = q_{aceita} (próximo estado)
    - 02 = caracter (símbolo escrito)
    - 1 = R (direção de movimento)

A ideia é que uma outra MT poderia receber essa cadeia como entrada e simular o comportamento

MT e linguagens RE

- Para demonstrar que uma L é RE ,
  - basta construir uma MT que a reconheça ,
    - que pare em um estado de aceitação para as entradas pertencentes à linguagem
      - não é necessário que a MT sempre pare

- Uma MT pode reconhecer *strings* que não são codificações válidas de MTs ,
  - verificando se seguem a sintaxe definida para codificar MTs
    - Se não obedecer a essa sintaxe (por exemplo, se tiver um formato inválido, símbolos não permitidos ou uma estrutura inconsistente) ,
      - a MT pode imediatamente rejeitá-la (parando em um estado não final), ou entrar em um estado de erro

transita para um estado específico que representa a detecção de uma entrada inválida

Linguagem Paradoxal (LP)

- Definida como o conjunto de MTs que não aceitam a si mesmas

- Formalmente pode ser expressa como

  LP = { M | M é uma MT, e M não aceita M}

- Não é RE , devido a um argumento de contradição
  - Mostra que a existência de uma MT que reconhece a LP leva a uma situação impossível,
    - onde a MT teria que aceitar e não aceitar sua própria codificação ao mesmo tempo
  - Isso prova que uma tal MT não pode existir e, consequentemente, que a LP não é recursivamente enumerável

    - RE : semidecidível, pode não parar
    - Não é RE : não pertence à classe de linguagens que podem ser reconhecidas por uma MT

Autoaceitação

- Uma MT aceita a si mesma se, ao executar com sua própria descrição codificada como entrada, ela atinge o estado final
  - MTs podem ser representadas como strings finitas
    - permitindo que sejam usadas como entrada para outras MTs

- $P = \{M \mid M \in \Sigma_{\text{Turing}}$ e M aceita $M\}$
  - P : linguagem das MT que aceitam a si mesmas
  - M : uma Máquina de Turing
  - Σ_{Turing} : conjunto de todas as codificações de MTs
  - M aceita M : a MT M , ao receber sua própria codificação como entrada, para em um estado final

Definições de P e ¬P

¬ : complemento

- P = {M | M aceita M}
  - Conjunto que contém todas as MTs que aceitam sua própria codificação como entrada

- ¬P = {M | M não aceita M}
  - O complemento de P é o conjunto de todas as MT que não aceitam sua própria codificação como entrada

complemento do conjunto P

O ¬P é recursivamente enumerável

- ¬P é um conjunto recursivamente enumerável
  - Isso significa que existe uma MT que pode listar todos os elementos de ¬P
    - Em outras palavras, existe uma MT que aceita uma máquina M se esta não aceita sua própria codificação
- Implicação
  - Se uma MT M rejeita a si mesma como entrada, isso significa que M pertence ao conjunto ¬P
    - A MT que reconhece ¬P aceitaria então M
  - O conjunto das cadeias que não são MT é facilmente aceito por uma MT
    - Significando que é relativamente simples construir uma MT que reconheça todas as cadeias de símbolos que não representam a codificação de uma MT válida

RE = semidecidível, pode não parar

Reconhecer e decidir

- Para alguns problemas
  - conseguimos reconhecer soluções (verificar uma entrada)
  - mas não decidir (sempre determinar se existe solução)

(decidir está no próximo slide)

- Verificar (reconhecer)
  - Uma MT pode ser programada para reconhecer soluções
    - Isso significa que, se fornecermos uma cadeia (uma solução) à MT,
      - ela pode executar seu conjunto de instruções e determinar se essa cadeia é uma solução válida para o problema
      - Se for, a MT aceitará essa cadeia, entrando em um estado de aceitação

chegou a um estado final,
indicando que a entrada foi
reconhecida como válida

Reconhecer e decidir

- Determinar (decidir)
  - Significa determinar se existe uma solução para qualquer entrada fornecida
- Para alguns problemas, não há uma maneira garantida de uma MT descobrir se uma solução existe ou não
  - A MT pode não parar nunca
  - Pode entrar em um ciclo infinito sem encontrar a solução

Função característica F_L

- Uma linguagem L é recursiva (decidível) se existe uma MT que sempre para e responde "sim" ou "não" para qualquer entrada
  - Construção da MT: suponha que temos 2 MTs
    - M_1: reconhece L (aceita cadeias de L, pode não parar nas demais)
    - M_2: reconhece o complemento ¬L (aceita cadeias fora de L, pode não parar nas demais)
- Execução em paralelo
  - Simulamos M_1 e M_2 na mesma entrada w, alternando passos entre elas
  - Se M_1 aceitar w → resposta é "sim"
  - Se M_2 aceitar w → resposta é "não"
  - Garantimos que uma delas vai parar e aceitar, pois ambas são RE e cobrem todas as possibilidades
- Conclusão
  - L é recursiva ⇔ L e ¬L são RE
  - Execução em paralelo é uma técnica teórica importante para mostrar decidibilidade

RE = aceita todas as cadeias pertencentes à L, e pode não parar para cadeias fora da L

Recursividade e complemento

- Se L é recursiva, então $\overline{L}$ também é recursiva
  - Isso porque a MT que decide L pode ser adaptada para decidir $\overline{L}$ simplesmente invertendo suas respostas (aceitar quando L rejeita e vice-versa)
- A execução em paralelo é essencial nesse contexto porque garante que a MT principal sempre pare,
  - mesmo que uma das MTs individuais (MT para L ou $\overline{L}$) não pare para certas entradas
- Dessa forma, a MT principal consegue decidir L de maneira eficiente e completa

O Problema da Parada Paradoxal

- Questiona se uma MT para quando recebe seu próprio código como entrada : o problema é indecidível
  - Implica que a linguagem das MTs que não se aceitam (e consequentemente, as que se aceitam) não é recursiva
  - Para uma linguagem ser recursiva, deve existir uma MT que sempre decide (para e dá uma resposta) se uma cadeia pertence ou não à linguagem
- A tentativa de criar um algoritmo para determinar se uma MT para com seu próprio código leva a uma contradição lógica,
  - demonstrando a indecidibilidade desse problema
  - e a não recursividade das linguagens relacionadas

Indecidibilidade

- Indecidibilidade e impossibilidade de detecção de comportamentos em programas
  - Um problema relevante com parada de máquinas é o problema da parada L_{Para} que é a linguagem das cadeias M**$**d , onde M é uma MT , d uma cadeia qualquer, e M para (aceitando portanto) quando tem d por entrada
  - É indecidível determinar se M$d pertence a L_{Para}
    - Não existe um algoritmo geral que determine se uma MT M para para uma entrada arbitrária d

L_{Para} é um conjunto (ou linguagem) que contém todas as possíveis combinações de uma MT e uma entrada d para a qual essa MT para

$ = símbolo separador
M = MT M
d = entrada
É uma notação que indica uma máquina de Turing específica (M) e os dados (d) que ela processará

Teorema de Rice

- Demonstra que não é possível criar um algoritmo geral para verificar se uma MT aceita uma propriedade não trivial
  - O que implica que diversos problemas relacionados ao comportamento das MTs são indecidíveis

- Propriedades não triviais são características de cadeias sobre um alfabeto que não são universais (satisfeitas por todas as cadeias) nem impossíveis (não satisfeitas por nenhuma cadeia)
  - Elas se aplicam a algumas cadeias, mas não a todas
    - Exemplos incluem ter comprimento par, ser balanceada, ou começar e terminar com símbolos específicos

- Propriedades triviais são aquelas que são satisfeitas por todas as cadeias (como a linguagem Σ^*) ou por nenhuma cadeia (como a linguagem vazia $\emptyset$)

GI e linguagens RE

- Gramáticas Irrestritas (GI) geram exatamente as linguagens recursivamente enumeráveis (RE)
- Uma máquina de Turing (MT) pode simular as derivações de uma GI, aceitando uma cadeia se ela for gerada pela gramática
- Essa simulação demonstra a equivalência entre GI e linguagens RE

RE = semidecidível, pode não parar

GI e linguagens RE

- A MT aceita uma cadeia se, e somente se, ela puder ser derivada pela gramática,
  - caso contrário a MT entra em um laço infinito
- Essa característica define as linguagens recursivamente enumeráveis:
  - uma linguagem é RE se existe uma MT que a aceita dessa forma

GI e MTs equivalentes

- A recíproca também é verdadeira:
  - dada uma MT, podemos construir uma GI que gera exatamente as cadeias aceitas pela MT
- A ideia é simular as configurações da MT usando as regras da GI
- Codificamos as configurações como cadeias $\alpha Q \beta$, onde:
  - α é a parte da fita à esquerda da cabeça de leitura/escrita
  - Q é o estado atual da MT
  - β é a parte da fita à direita da cabeça de leitura/escrita (incluindo o símbolo sob a cabeça)

(continua)

GI e MT

- Para simular as transições da MT, a GI possui regras da forma $Q\sigma \rightarrow \sigma'Q'$, correspondendo às quíntuplas de transição da MT $\langle Q, \sigma, Q', \sigma', \rightarrow \rangle$

- A configuração inicial é gerada por regras que produzem cadeias da forma $\alpha\$Q_0\alpha$, onde α é qualquer cadeia do alfabeto da MT e Q_0 é o estado inicial

- Assim, a GI gera uma cadeia se, e somente se, a MT pode alcançar a configuração correspondente

GI e MT

- A gramática simula a aceitação da MT através de regras como $Q_f\sigma \rightarrow Q_f$, $\sigma Q_f \rightarrow Q_f$ e **$**$Q_f \rightarrow \varepsilon$, garantindo que a derivação termina quando a MT alcança um estado final

- Assim, uma cadeia é gerada pela GI se, e somente se, a MT a aceita

- Essa correspondência biunívoca demonstra a equivalência entre MTs e GIs
  - Para toda gramática irrestrita G, existe uma MT M tal que L(M) = L(G)
  - Para toda MT M, existe uma gramática irrestrita G tal que L(M) = L(G)