Exame de Qualificação 27/02/2014

Leila Ribeiro e Luciana S. Buriol

March 25, 2017

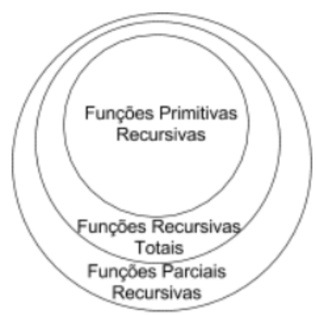
Teoria da Computação

Perguntas

1. **(2.5 Pontos) Teorema: A classe de funções recursivas parciais é idêntica à classe de funções Turing-computáveis.**

   (a) **Defina essa classe de funções e faça um esquema da prova deste teorema explicando o que deve ser provado em cada passo.**

   - A classe de **funções primitivas recursivas** é formada por um conjunto de funções iniciais (**sucessor**, **zero**, **projeção**), duas operações sobre as funções (**composição** e **recursão**), e todas as funções obtidas a partir das funções iniciais mais os dois operadores.
   - A classe de funções primitivas recursivas foi proposta inicialmente para compor todas as funções efetivamente computáveis, porém, Ackermamm propôs uma função cuja comunidade aceitou como sendo efetivamente computável e que não era uma função primitiva recursiva. Assim as funções primitivas recursivas unidas ao operador de **minimização** formam as **funções parciais recursivas**.

   - O Teorema diz que uma função f é Turing-computável, se e somente se, ela forma uma função parcial recursiva. Como o teorema é bidirecional, temos duas partes: (a) PRF $\Rightarrow$ TM e (b) TM $\Rightarrow$ PRF.
     - $PRF \Rightarrow TM$: iremos provar por indução apresentando uma TM que compute cada uma das funções iniciais, os operadores de composição e recursão primitiva, e também o operador de minimização. Uma vez que seja possível criar uma TM para cada uma das PRF, conclui-se então que toda função que pode ser computada a partir das funções recursivas parciais, pode ser computada também a partir de uma TM.
     - $TM \Rightarrow PRF$: precisamos demonstrar que uma função f Turing-computável é também computada por uma PRF. Para isso vamos codificar o comportamento de uma máquina de Turing tal que o seu comportamento é simulado por uma PRF correspondente. A prova consiste de 13 passos onde o conteúdo e o funcionamento da máquina é codificado para permitir que seja simulado através de funções parciais recursivas:

i. Encode the alphabet
ii. Encode the tape contents
iii. Constrcut the predicate *unbr1*
iv. Encode tape positions
v. Encode states
vi. Encode actions
vii. Encode configuration
viii. Define *current-symbol*
ix. Define *next-square*
x. Define *next-tape-contents*
xi. Define *next-configuration*
xii. Define simulation of execution
xiii. Define f

(b) **Explique por que este teorema é importante.**

- Dada a equivalência entre a classe de funções parciais recursivas e as funções Turing-computáveis, podemos então aplicar a grande quantidade de resultados relativos a classe de funções recursivas na classe de funções Turing-computáveis.

2. **(1.5 pontos) Disserte sobre redução de problemas, tanto no contexto de indecidibilidade de problemas quanto no contexto da definição de classes de complexidade.**

- A redução é uma função ou algoritmo que em tempo polinomial realiza a transformação entre uma linguagem aceita por um determinado problema em um outro, tal que: $f : \varepsilon_L \to \varepsilon_{L'}$, $w \in L \Leftrightarrow f(w) \in L'$. Idicando que ambos os problemas ou linguagens podem ser consideradas equivalentes, ou seja, a solução do problema L também representa um solução para o problema L'. É possível expressar que Q é redutivel ao problema R através da seguinte notação: $Q \leqslant_p R$.
- No contexto de **indecidibilidade** de problemas, podemos utilizar a redução polinomial para demonstrar que um problema é indecidível. Basta demonstrar que um problema conhecidamente indecidivel (como o problema da parada) é redútivel em tempo polinomial ao problema desejado P ($halt \leqslant_p P$). Como o problema da parada ($halt$) não é decidível, então P também não possui solução.
- No contexto de **complexidade** podemos utilizar a redução polinomial para determinar a complexidade de um problema. Exemplo: $P \leqslant_p Q$, caso Q apresente uma solução em tempo polinomial, então Q também irá possuir uma solução equivalente. Caso P não seja resolvível em tempo polinomial, então Q também não será.

3. **(1 ponto) Explique intuitivamente o que diz o Teorema de Cook-Levin e por que ele é relevante para a Ciência da Computação? (Não precisa mostrar definições formais nessa questão).**

- Um dos principais resultados da área da complexidade é o **Teorema de Cook-Levin**. Cook e Levin iniciaram de forma independente a discussão sobre a classe de problemas NP-Completos. Nesse sentido, o problema SAT foi o primeiro da história da computação a ser provado como pertencente a classe de problemas NP-Completos. A prova consiste em uma demonstração que SAT está em NP, e a partir disso, de que um problema geral em NP (representando todos os problemas deste grupo) é redutível a SAT em tempo polinomial por uma máquina de Turing determinística. A redução consiste em uma função que pode ser computada por uma MTD em tempo polinomial, tal que transforma todas as soluções válidas de um problema P_1 para uma solução válida de outro problema P_2, representado por $P_1 \leqslant_p P_2$.

- O Teorema de Cook-Levin é de extrema importância para a Ciência da Computação por várias razões. Uma das principais é o fato de ter sido o percursor das discussões sobre a NP-Completude. Outro fator importante foi a realização da prova que identificou o problema raiz da classe de problemas NP-Completos, facilitando a inclusão de novos problemas nesta classe.
- Para provar que um novo problema R é NP-Completo **(essas definições precisam de revisão)**:
  - **Inter-redutibilidade**: escolha um problema NP-Completo Q e execute a redução do problema R (que se desja provar NP-Completo) para Q, assim como de Q para R. (1) $R \leqslant_p Q$ & (2) $Q \leqslant_p R$. Summary: *(1) shows that R cannot be any worse than Q. (2) shows that R can't be any better than Q.*
  - **NP e NP-Hard**: prove que o problema R é NP mostrando que é possível obter um certificado em tempo polinomial. Então, prove que R é NP-Hard, mostrando que todos os problemas em NP podem ser reduzidos ao problema R. Summary: *(1) show that R is NP. (2) show that R is NP-Hard, i.e. $S \leqslant_p R$ for any S in NP.*