ACH2043
INTRODUÇÃO À TEORIA DA COMPUTAÇÃO

Aula 16

Cap 5 - Redutilibidade
Cap 5.1 – Problemas indecidíveis

Profa. Ariane Machado Lima
ariane.machado@usp.br

Na aula passada...

- A_{MT} é indecidível (usando diagonalização) - insolúvel
- Uma linguagem é Turing-decidível sse ela e seu complemento forem Turing-reconhecíveis
- O complemento de A_{MT} NÃO é Turing-reconhecível (completamente insolúvel)

Na aula de hoje

- Como provar que outros problemas são indecidíveis...
- ... usando a técnica de redutibilidade

Redutibilidade

- Redução: conversão de um problema A em outro problema B de forma que a solução de B seja usada para solucionar A
- Ex:
  - Se você tem amigos morando em Paris, viajar para Paris pode ser reduzido a
  - Comprar uma passagem aérea de São Paulo a Paris, que pode ser reduzido a
  - Ganhar dinheiro para a passagem, que pode ser reduzido a
  - Encontrar um emprego

Redutibilidade

- Exemplos matemáticos:
  - Medir a área de um retângulo pode ser reduzido a medir a altura e a largura do retângulo
  - Resolver um problema de equações lineares pode ser reduzido ao problema de inverter uma matriz

Redutibilidade

- Utilidade:
  - Se A é redutível a B
    - A não pode ser mais difícil do que B
    - Se B for decidível, A também será
    - Se A for indecidível, B também será

Redutibilidade

- Utilidade:
  - Se A é redutível a B
    - A não pode ser mais difícil do que B
    - Se B for decidível, A também será
    - Se A for indecidível, B também será

Chave para provar que certos problemas são indecidíveis (reduzindo um problema conhecidamente indecidível a ele)

Ex: Problema da Parada

- $PARA_{MT}$ = {<M,w> : M é uma MT e M pára sobre a entrada w}
- Que problema indecidível pode ser reduzido a esse?

Ex: Problema da Parada

- $PARA_{MT}$ = {<M,w> : M é uma MT e M pára sobre a entrada w}
- A_{MT} = {<M,w> : M é uma MT e M aceita w}
- A_{MT} (que é indecidível) pode ser reduzido a $PARA_{MT}$
- Logo, $PARA_{MT}$ é indecidível

Ex: Problema da Parada

- Prova (tem que mostrar a redução!): assuma, por contradição, que uma MT R decida $PARA_{MT}$. Então construímos S que usa R para decidir A_{MT} :

-
-

Ex: Problema da Parada

- Prova (tem que mostrar a redução!): assuma, por contradição, que uma MT R decida $PARA_{MT}$. Então construímos S que usa R para decidir A_{MT} :

S = “Sobre a entrada <M, w>, uma codificação de uma MT M e uma cadeia w:

1. Rode a MT R sobre a entrada <M, w>.
2. Se R rejeita, rejeite.
3. Se R aceita, simule M sobre w até ela pare.
2. Se M aceitou, aceite; se M rejeitou, rejeite.”

- Logo A_{MT} pode ser reduzido a $PARA_{MT}$
- Como A_{MT} é indecidível, $PARA_{MT}$ é indecidível

Vacuidade de uma linguagem de uma MT

- Como escrever isso em forma de linguagem?

Vacuidade de uma linguagem de uma MT

- V_{MT} = { <M>: M é uma MT e L(M) = Ø}
- Como podemos usar V_{MT} para resolver A_{MT}?
- Se uma linguagem for vazia, ela não aceita w. Mas e se não for?
-

Vacuidade de uma linguagem de uma MT

- V_{MT} = { <M>: M é uma MT e L(M) = Ø}
- Como podemos usar V_{MT} para resolver A_{MT}?
- Se uma linguagem for vazia, ela não aceita w. Mas e se não for?
- Ideia: construir uma versão de M que apenas teste w

M1 = "Sobre a entrada x:

1. Se x ≠ w *rejeite*
2. Se x = w, rode M sobre a entrada w e *aceite* se M aceita, e *rejeite* se M rejeita"

Vacuidade de uma linguagem de uma MT

- Suponha que R decide V_{MT}, vamos construir S que decide A_{MT}
- S = “Sobre a entrada <M,w>, uma codificação de uma MT M e uma cadeia w:
  1. Use a descrição de M e w para construir M1
  2. Rode R sobre M1
  3. Se R aceita, ? ; se R rejeita, ? ”

Vacuidade de uma linguagem de uma MT

- Suponha que R decide V_{MT}, vamos construir S que decide A_{MT}
- S = "Sobre a entrada <M,w>, uma codificação de uma MT M e uma cadeia w:
  1. Use a descrição de M e w para construir M1
  2. Rode R sobre M1
  3. Se R aceita, *rejeite*; se R rejeita, *aceite*."

Mas como A_{MT} é indecidível, V_{MT} é indecidível

Classe da linguagem gerada por uma MT

- Dada um MT M, a linguagem gerada por ela poderia ser reconhecida por um modelo mais simples?
- Por ex: se a linguagem é regular
- Como escrever esse problema em termos de linguagem?

Determinação de se a linguagem gerada por uma MT é regular

- $REGULAR_{MT}$ = { <M> | M é uma MT e L(M) é regular}

Determinação de se a linguagem gerada por uma MT é regular

- $REGULAR_{MT}$ = { <M> | M é uma MT e L(M) é regular}
- $REGULAR_{MT}$ é indecidível
- Ideia da Prova:

Determinação de se a linguagem gerada por uma MT é regular

- $REGULAR_{MT}$ = { <M> | M é uma MT e L(M) é regular}
- $REGULAR_{MT}$ é indecidível
- Ideia da Prova: Supomos que existe uma MT R que decide $REGULAR_{MT}$ e usamos R em uma MT S para decidir A_{MT}
- Decidir se uma MT M2 é regular, onde M2 reconhece uma linguagem regular (Σ^*) sse M aceita w

Determinação de se a linguagem gerada por uma MT é regular

- S que decide A_{MT} usando R
- S = “Sobre a entrada <M,w>, onde M é uma MT e w é uma cadeia:

1. Construa a MT M2:

   M2 = “Sobre a entrada x:

   1. Se x tem a forma 0^n1^n, *aceite*
   2. senão, rode M sobre a entrada w e *aceite* se M aceita w, *rejeite* se M rejeita”

2. Rode R sobre a entrada <M2>
3. Se R aceita, *aceite*; se R rejeita, *rejeite*”

- Mas A_{MT} é indecidível, então $REGULAR_{MT}$ também é

Determinação de propriedades da linguagem gerada por uma MT

- Da mesma forma, os seguintes problemas são indecidíveis (para uma dada MT M)
  - Determinar se L(M) é livre-de-contexto
  - Determinar se L(M) é sensível ao contexto
  - Determinar se L(M) é decidível (recursiva)
  - ...
  - Na verdade, determinar qualquer propriedade de L(M) (Teorema de Rice)

Equivalência entre MTs

Equivalência entre MTs

- EQ_{MT} = {<M1, M2> | M1 e M2 são MTs e L(M1) = L(M2)}
- Podemos usar EQ_{MT} para resolver V_{MT}!
- Ideia: se uma MT M for equivalente a outra que rejeita qualquer cadeia, então L(M) = Ø
- Assuma que R é uma MT que decide EQ_{MT}
- Vamos construir S que decide V_{MT} usando R

Equivalência entre MTs

- S = “Sobre a entrada <M> onde M é uma MT:
  1. Rode R sobre a entrada <M, M1>, onde M1 é uma MT que rejeita todas as entradas.
  2. Se R aceita, *aceite*; se R rejeita, *rejeite*.”
- Mas V_{MT} é indecidível, então EQ_{MT} também é

Reduções via histórias de computação

- História de computação: sequência de configurações de uma MT, da inicial à de aceitação ou rejeição
- Uma história de computação deve ser finita
- Apenas uma para MTs determinísticas, e possivelmente várias para MTs não-determinísticas
- Aqui consideramos apenas MTs determinísticas

Vacuidade de autômatos linearmente limitados

- Autômatos linearmente limitados (ALL) são os modelos que reconhecem linguagens sensíveis ao contexto
- A_{ALL} = {<M,w> | M é um ALL que aceita a cadeia w}
- A_{ALL} é decidível

Autômatos linearmente limitados

- A_{ALL} é decidível
- Se ALL tem q estados, g símbolos de fita e uma fita de comprimento n, então só existem qng^n configurações distintas (estado atual, posição da cabeça, conteúdo da fita)
- Se uma configuração se repetir, o ALL está em loop.
- Como é uma MT que decide A_{ALL}?

Autômatos linearmente limitados

- A_{ALL} é decidível
- L = “Sobre a entrada <M,w>, onde M é um ALL e w é uma cadeia:
  1. Simule M sobre w por qng^n passos ou até que ela pare.
  2. Se M parou, *aceite* se ela aceitou e *rejeite* se ela rejeitou. Se M não parou, *rejeite*.”

Vacuidade de autômatos linearmente limitados

- Mas o problema da vacuidade de ALLs é indecidível
- $V_{ALL} = \{<M> \mid M \text{ é um ALL e } L(M) = \emptyset\}$
- Para provar, vamos usar "histórias da computação" e redução a partir de A_{MT}

Vacuidade de autômatos linearmente limitados

- Podemos construir um ALL B que reconheça apenas histórias de computação de aceitação de uma cadeia w em uma MT M
- Se L(B) for vazia, então M não aceita w, caso contrário aceita
- Como seria esse B?

Vacuidade de autômatos linearmente limitados

- ALL B:
  - Fita contém inicialmente uma história de computação, ou seja, C1 até Cl, cada configuração separada por “#”
  - Deve verificar 3 propriedades:
    1. C1 é a configuração inicial de M sobre w
    2. Cada Ci+1 é originada de Ci
    3. Cl é uma configuração de aceitação para M

Vacuidade de autômatos linearmente limitados

- ALL B:
  - Fita contém inicialmente uma história de computação, ou seja, C1 até Cl, cada configuração separada por “#”
  - Deve verificar 3 propriedades:
    1. C1 é a configuração inicial de M sobre w

       $C1 = q_0w_1w_2...w_n$, onde q_0 é o estado inicial de M
    2. Cada Ci+1 é originada de Ci
    3. Cl é uma configuração de aceitação para M

Vacuidade de autômatos linearmente limitados

- ALL B:
  - Fita contém inicialmente uma história de computação, ou seja, C1 até Cl, cada configuração separada por "#"
  - Deve verificar 3 propriedades:
    1. C1 é a configuração inicial de M sobre w

       C1 = $q_0w_1w_2...w_n$, onde q_0 é o estado inicial de M
    2. Cada Ci+1 é originada de Ci

       Ci e ci+1 são idênticas exceto pelas posições sob e adjacentes à cabeça em Ci (estas, atualizadas conforme a função de transição)
    3. Cl é uma configuração de aceitação para M

Vacuidade de autômatos linearmente limitados

- ALL B:
  - Fita contém inicialmente uma história de computação, ou seja, C1 até Cl, cada configuração separada por "#"
  - Deve verificar 3 propriedades:
    1. C1 é a configuração inicial de M sobre w

       C1 = $q_0w_1w_2...w_n$, onde q_0 é o estado inicial de M
    2. Cada Ci+1 é originada de Ci

       Ci e ci+1 são idênticas exceto pelas posições sob e adjacentes à cabeça em Ci (estas, atualizadas conforme a função de transição)
    3. Cl é uma configuração de aceitação para M

       Cl contém q_{aceita}

Vacuidade de autômatos linearmente limitados

- ALL B:
  - Fita contém inicialmente uma história de computação, ou seja, C1 até Cl, cada configuração separada por “#”
  - Deve verificar 3 propriedades:
    1. C1 é a configuração inicial de M sobre w

       C1 = $q_0w_1w_2...w_n$, onde q_0 é o estado inicial de M
    2. Cada Ci+1 é originada de Ci **COMO?**

       Ci e ci+1 são idênticas exceto pelas posições sob e adjacentes à cabeça em Ci (estas, atualizadas conforme a função de transição)
    3. Cl é uma configuração de aceitação para M

       Cl contém q_{aceita}

Vacuidade de autômatos linearmente limitados

- ALL B:
  - Fita contém inicialmente uma história de computação, ou seja, C1 até Cl, cada configuração separada por “#”
  - Deve verificar 3 propriedades:
    1. C1 é a configuração inicial de M sobre w

       C1 = $q_0w_1w_2...w_n$, onde q_0 é o estado inicial de M
    2. Cada Ci+1 é originada de Ci **COMO? ZIGUE-ZAGUIANDO NA FITA**

       Ci e ci+1 são idênticas exceto pelas posições sob e adjacentes à cabeça em Ci (estas, atualizadas conforme a função de transição)
    3. Cl é uma configuração de aceitação para M

       Cl contém q_{aceita}

Vacuidade de autômatos linearmente limitados

- Redução de A_{MT} a V_{ALL}
- Suponha que existe uma MT R que decida V_{ALL}:
- S decide A_{MT}:

S = “Sobre a entrada <M,w>, onde M é uma MT e w uma cadeia:

1. Construa o ALL B a partir de M e w
2. Rode R sobre <B>
3. Se R rejeita, *aceite*; se R aceita, *rejeite*.”

Mas A_{MT} é indecidível, então V_{ALL} também é

Reduções via histórias de computação

- A mesma estratégia pode ser usada para provar a indecidibilidade de outros problemas
- Por exemplo...

Se uma GLC gera todas as cadeias

- $TODAS_{GLC}$ = {<G>|G é uma GLC e L(G) = Σ*} é indecidível
- Prova por "histórias da computação" e redução a partir de A_{MT}
- Para uma dada MT M e uma cadeia w, construimos uma GLC G que reconheça todas as cadeias que NÃO sejam histórias de computação de M sobre w
  - Se M aceita w, G gera todas as cadeias MENOS a história de computação de M sobre w
  - Se M não aceita w, G gera todas as cadeias
- Se G gerar todas as cadeias, então M aceita w, caso contrário M não aceita w
- Se $TODAS_{GLC}$ fosse decidível, A_{MT} também seria. Logo, $TODAS_{GLC}$ é indecidível

Se uma GLC gera todas as cadeias

- $TODAS_{GLC}$ = {<G>|G é uma GLC e L(G) = Σ*} é indecidível
- Prova por "histórias da computação" e redução a partir de A_{MT}
- Para uma dada MT M e uma cadeia w, construimos uma GLC G que reconheça todas as cadeias que NÃO sejam histórias de computação de M sobre w
  - Se M aceita w, G gera todas as cadeias MENOS a história de computação de M sobre w
  - Se M não aceita w, G gera todas as cadeias
- Se G gerar todas as cadeias, então M aceita w, caso contrário M não aceita w
- Se $TODAS_{GLC}$ fosse decidível, A_{MT} também seria. Logo, $TODAS_{GLC}$ é indecidível

Se uma GLC gera todas as cadeias

- Vamos construir um autômato a pilha equivalente a G (para aceitar cadeias que NÃO sejam histórias de computação de M sobre w), ou seja, cadeias que apresente AO MENOS UMA das seguintes propriedades (testadas não-deterministicamente):

1. **não** começa com C1

   C1 = $q_0 w_1 w_2 ... w_n$, onde q_0 é o estado inicial de M

2. alguma Ci+1 **não** é originada de Ci

3. **não** termina com uma configuração de aceitação

Se uma GLC gera todas as cadeias

- Vamos construir um autômato a pilha equivalente a G (para aceitar cadeias que NÃO sejam histórias de computação de M sobre w), ou seja, cadeias que apresente AO MENOS UMA das seguintes propriedades (testadas não-deterministicamente):

1. **não** começa com C1

   C1 = $q_0w_1w_2...w_n$, onde q_0 é o estado inicial de M

2. alguma Ci+1 **não** é originada de Ci

   como?

3. **não** termina com uma configuração de aceitação

Se uma GLC gera todas as cadeias

- Vamos construir um autômato a pilha equivalente a G (para aceitar cadeias que NÃO sejam histórias de computação de M sobre w), ou seja, cadeias que apresente AO MENOS UMA das seguintes propriedades (testadas não-deterministicamente):

1. **não** começa com C1

   C1 = $q_0 w_1 w_2 ... w_n$, onde q_0 é o estado inicial de M

2. alguma Ci+1 **não** é originada de Ci

   como? Fita: $\#C1\#C2^R\#C3\#C4^R\#...\#Cl\#$

3. **não** termina com uma configuração de aceitação