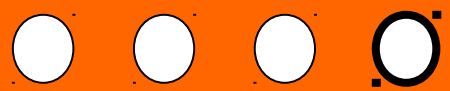
Automata e Linguagens Formais

CTC 34



10

Prof. Carlos H. C. Ribeiro

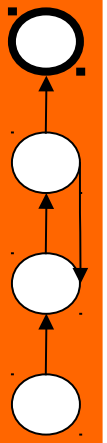
carlos@ita.br

Redução de problemas

O Teorema de Rice

O Problema de Correspondência de Post

O PCP e as LLCs



Relembrando: Problema da Parada

- Seja a linguagem composta pelas cadeias $\langle T,\alpha \rangle$, correspondentes à concatenação das descrições $\langle T \rangle$ de MTs com cadeias α:

$HALT_{MT}$ = { $\langle T,\alpha \rangle$ | T é uma MT e T pára com a entrada α}

- O Problema da Parada corresponde a saber se existe uma MT R que **decide** $HALT_{MT}$:

$R(\langle T,\alpha \rangle)$ {
- R aceita $\langle T,\alpha \rangle$ se T pára com α
- R rejeita $\langle T,\alpha \rangle$ se T não pára com α

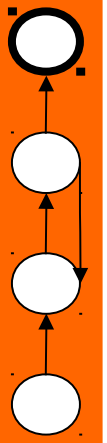
Relembrando: Problema da Parada

- A MT R pode ser usada na construção da MT H que decide a linguagem A_{MT}.

H = "Para a entrada $\langle T,\alpha \rangle$:

1. **Execute R com a entrada $\langle T,\alpha \rangle$.**
2. **Se R rejeitar, rejeite.** *// T não pára com α*
3. **Se R aceitar, simule T com α até T parar** *// T pára com α*
4. **Se T aceitar, aceite. Se T rejeitar, rejeite."**

- Assim, se R decide $HALT_{MT}$, então H decide A_{MT}.
- Como A_{MT} é indecidível (slides 19-21, aula 9), então $HALT_{MT}$ é indecidível.
- Esta demonstração baseia-se em um processo de redução do problema de decisão da linguagem A_{MT} ao problema da parada.



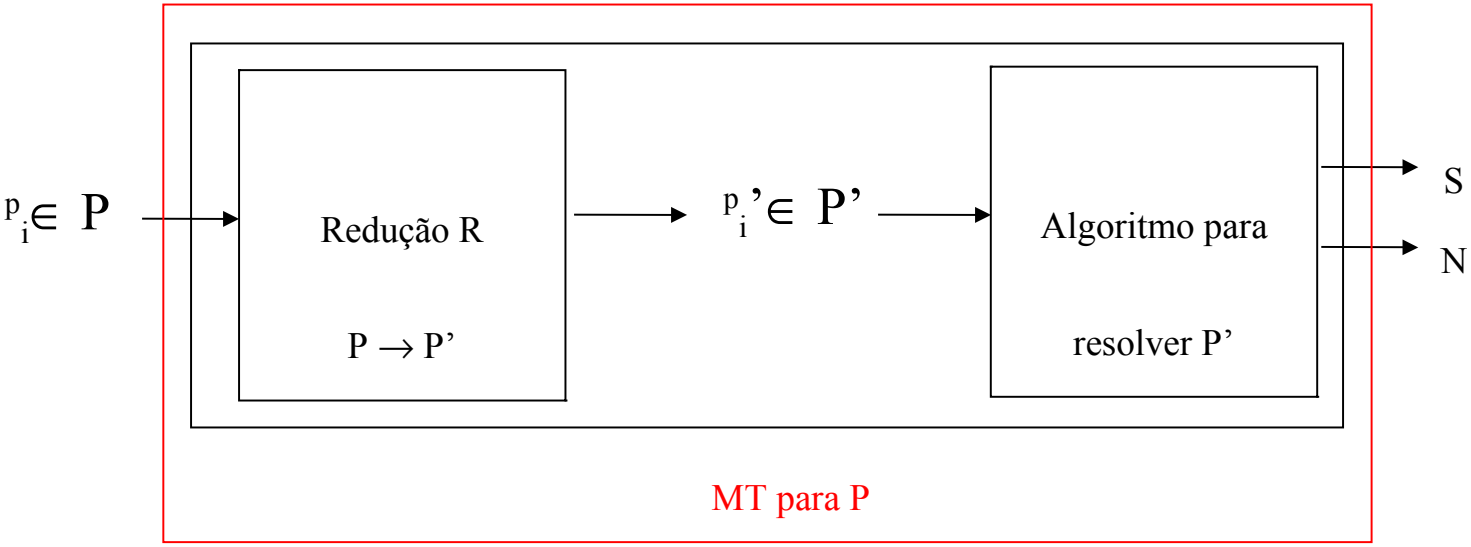
Redução de Problemas

- Problemas “difíceis” freqüentemente podem ser transformados (reduzidos) em outros, para os quais uma solução já foi encontrada.
- Formalmente:

  Um problema de decisão P e redutível a P’ se existir um algoritmo R que, recebendo p_i como entrada, produz um resultado p_i’ tal que a resposta de P para a entrada p_i seja idêntica ou complementar à resposta de P’ para a entrada p_i’, qualquer que seja a entrada p_i.

  Diz-se neste caso que R reduz o problema P ao problema P’.

Redução de Problemas



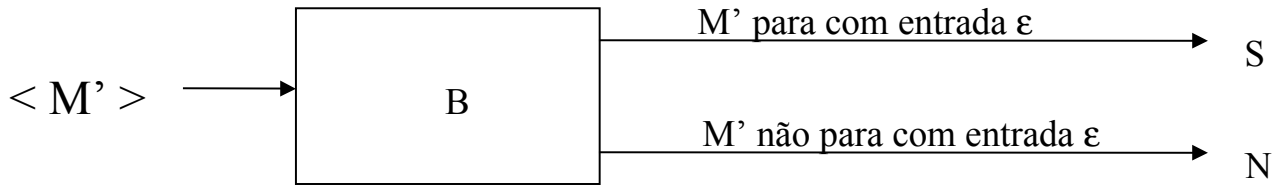
- Se P' é decidível e P é redutível a P', então P é também decidível.
- Se P é não-decidível e P é redutível a P', então P' é não-decidível.

Solução via Redução: Exemplo 1

- Mostrar que não existe algoritmo que determina se uma MT qualquer para quando uma computação é iniciada com a fita em branco.

Idéia: Reduzir o problema da parada em MTs a este problema

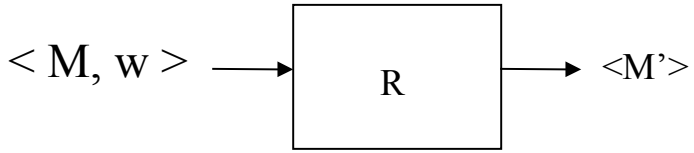
Assuma que exista uma máquina B que resolve o problema, ou seja, que determina se uma dada máquina M' para com entrada em branco:



Solução via Redução: Exemplo 1

Reduzo o problema de parada a este problema usando um algoritmo (máquina de Turing) R que le <M, w> e produz na saída uma representação M' de uma máquina que, a partir de uma fita em branco:

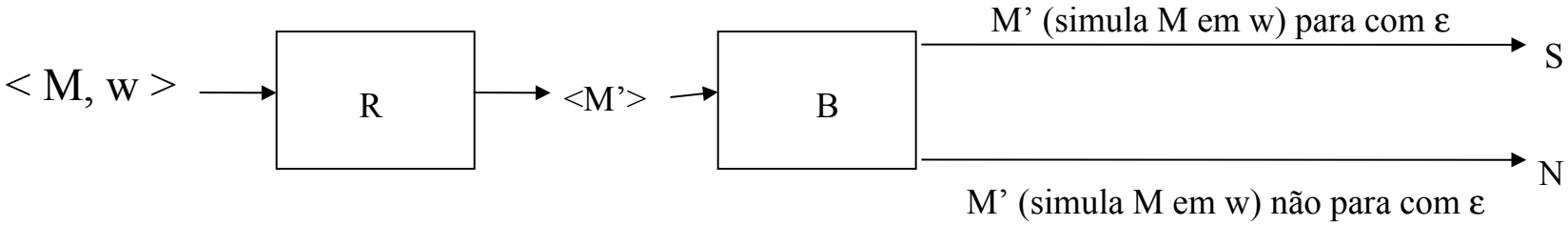
1. Escreve w na fita.
2. Retorna a cabeça para a posição inicial e no estado inicial de M.
3. Simula M.



É fácil contruir R, tudo o que ela faz é produzir uma representação <M'> de uma máquina com as mesmas transições de M, acrescidas de transições iniciais para preencher a fita em branco com a cadeia w!

Solução via Redução: Exemplo 1

Note agora como resolveríamos o problema da parada:



1. Executo R a partir de < M, w>. Obtenho <M'>, que **a partir da fita em branco escreve w em sua fita e em seguida simula M**.

2.1 Se B parar, significa que M' para com seu conteúdo inicial (cadeia vazia). E portanto, para se escrever w na fita em seguida.

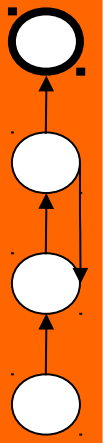
2.2 Se B não parar, significa que M' não para com seu conteúdo inicial (cadeia vazia). E portanto, não para se escrever w na fita em seguida.

R reduz o problema de parada ao problema da fita em branco.

Logo, problema da fita em branco é não-decidível

Solução via Redução: Exemplo 2

- Redução do problema **P: Determinar se uma dada cadeia pertence à linguagem L={uu | u=a^i b^i c^i para algum i ≥ 0}** ao problema **P': Determinar se uma dada cadeia pertence à linguagem L'={u | u =a^i b^i c^i para algum i ≥ 0}** .
  - Construo a maquina M' que aceita L'.
  - Executo o seguinte algoritmo R:
    - Copio cadeia de entrada w (entrada da máquina M) para a fita de uma máquina Z capaz de reconhecer se w=uu, para algum u ∈ {a, b, c}*:
      - Se w = uu, limpo a fita e deixo u na entrada de M'. Se M' agora aceitar u, é porque a entrada de M era uu, com u = a^i b^i c^i .
      - Se w ≠ uu, limpo a fita e deixo um a na entrada de M', que não será aceito por M'.



Redução: Uma Visão Alternativa

- Sejam P $\in \Sigma^*$ e Q $\in \Gamma^*$. Uma redução é uma função computável

  r: $\Sigma^* \rightarrow \Gamma^*$ tal que w $\in$ P sss r(w) $\in$ Q.

- Reduções mapeiam sentenças de uma linguagem em sentenças de outra linguagem:

Se uma sentença é um problema de decisão acompanhado de suas respostas, então a redução mapeia este problema e respostas em outro problema e respostas (numa linguagem diferente).

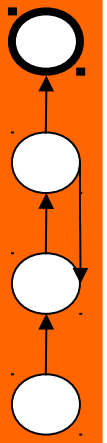
- Reduções não são necessariamente injetoras.

Redução: Uma Visão Alternativa

- Vendo a redução como mapeamento sobre linguagens, podemos reformular os teoremas de decidabilidade:

Se P' é decidível e P é redutível a P', então P é também decidível.
Se P é não-decidível e P é redutível a P', então P' é não-decidível.

Se P' é linguagem recursiva e P é redutível a P', então P é linguagem recursiva.
Se P não é linguagem recursiva e P é redutível a P', então P' é linguagem não-recursiva.

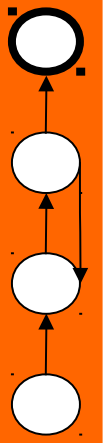


Exemplo 1 revisitado

- Problema da fita em branco: Mostrar que não existe algoritmo que determina se uma MT qualquer para quando uma computação é iniciada com a fita em branco.

Idéia: Reduzir a **linguagem HALT =** {<T,w> | T é MT e para com w} do problema da parada em MTs a **linguagem** BLANK = {<M'> | M' é MT que para com ε}.

Faremos essa redução usando uma máquina de Turing R que mapeará cadeias de HALT em cadeias de BLANK.



Operação de R:

Produzo uma representação <M'> de máquina M' que, a partir de uma fita em branco:

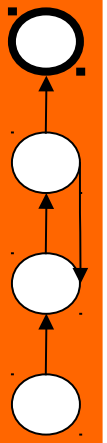
- Escreve uma cadeia x da forma <T,w> na fita.
- Apaga x e escrevo w na fita.
- Simula T.

Agora, verificaremos que R de fato reduz as cadeias <T,w> de HALT a cadeias <M'> de BLANK:

Se T para com w, significa que M' (que começou com fita em branco) também para (pois simula T em w). Ou seja, <T,w> $\in$ HALT implica <M'> $\in$ BLANK.

Se T não para com w, significa que M' (que começou com fita em branco) também não para (pois simula T em w). Ou seja, <T,w> $\notin$ HALT implica <M'> $\notin$ BLANK.

HALT não é linguagem recursiva e HALT é redutível a BLANK, logo BLANK é linguagem não-recursiva e o problema da fita em branco é indecidível!



O Teorema de Rice: Motivação

Idéia: Ao invés de problemas ligados a decidabilidade envolvendo MTs e uma entrada particular

- Máquina M_i pára com entrada w?
- Máquina M_i pára com entrada em branco?

Etc, etc.

Podemos estar interessados em problemas mais gerais, tais como:

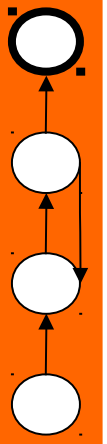
- A cadeia vazia pertence à $L(M_i)$?
- $L(M_i)$ é $\emptyset$?
- $L(M_i)$ é regular?
- $L(M_i) = \Sigma^*$?

Etc, etc.

O Teorema de Rice: Codificação

- Qualquer MT pode ser codificada como uma cadeia sobre {0,1}.
- Um conjunto de MTs portanto define uma linguagem sobre $\Sigma = \{0,1\}$.
- As questões anteriores podem ser reformuladas como:
  - M_i pertence a linguagem $L_\varepsilon = \{ M \mid \varepsilon \in L(M)\}$?
  - M_i pertence a linguagem $L_\emptyset = \{M \mid L(M) = \emptyset\}$?
  - M_i pertence a linguagem $L_{reg} = \{M \mid L(M)$ e regular$\}$?
  - M_i pertence a linguagem $L_\Sigma = \{M \mid L(M) = \Sigma^*\}$?

Etc, etc.



O Teorema de Rice

Def.: Uma propriedade de uma l.r.e. é uma condição que a l.r.e. pode satisfazer. Exemplos: “conter cadeia nula”, “ser regular”, etc.

Def.: Uma linguagem de uma propriedade $\wp$ de uma l.r.e. é o conjunto $L_{\wp} = \{M \mid M \text{ satisfaz } \wp\}$. Exemplo: $L_{\varnothing}$ é a linguagem formada por todas as cadeias que representam as MTs que não aceitam nenhuma cadeia.

Def.: Uma propriedade $\wp$ de uma l.r.e. e dita trivial se a) não existirem linguagens que a satisfaçam ou b) todas as l.r.e.’s a satisfazem.

Teorema (Rice, 1953): Se $\wp$ é uma propriedade não trivial de linguagens recursivamente enumeráveis, então $L_{\wp}$ não é recursiva.

O Teorema de Rice: Algumas Conseqüências

- A propriedade “a linguagem é regular” é não-trivial: existem l.r.e.’s que a satisfazem, e existem l.r.e.’s que não a satisfazem. Pelo Teorema de Rice, então L_{reg} = {M | L(M) é regular} não é recursiva. Logo, não existe MT capaz de ter como entrada qualquer máquina M e parar com aceitação caso esta aceite uma linguagem regular, e parar com rejeição caso esta não aceite uma linguagem regular.
- Não existe MT capaz de ter como entrada qualquer máquina M e parar com aceitação caso esta aceite a cadeia vazia, e parar com rejeição caso esta não aceite a cadeia vazia.
- Não existe MT capaz de ter como entrada qualquer máquina M e parar com aceitação caso esta não aceite nenhuma cadeia, e parar com rejeição caso esta aceite alguma cadeia.

Etc, etc...

O Problema de Correspondência de Post

- Trata-se de um problema de dominós...
- Cada dominó: primeiro e segunda metades são cadeias de um mesmo alfabeto.
- Assumo um conjunto infinito de dominós de um dado tipo: jogar um dominó não limita o numero de escolhas futuras.
- Dominós são posicionados lado a lado, formando cadeias completas na metade superior e na metade inferior.
- **O problema: posicionar os dominós de modo que a cadeia da metade superior fique igual à cadeia da metade inferior.**

Exemplo 1: Alfabeto {a,b}, dominós [aaa, aa] e [baa, abaaa]

Solução:

aaa	*baa*	*aaa*	aaabaaaaa
ab	*abaaa*	*ab*	aaabaaaaa

O Problema de Correspondência de Post: Exemplo 2

Alfabeto {a,b}, dominós [ab, aba], [bba, aa], [aba,bab]

Solução:

Tenho que começar com:

ab
aba

(única maneira de ter os mesmo prefixos)

Dominó seguinte não pode ser [bba,aa]. Tenho então duas possibilidades:

ab	*ab*
aba	*aba*

Não serve: quarta posição

ab	*aba*
aba	*bab*

Agora, tenho que começar com b na metade de cima...

O Problema de Correspondência de Post: Exemplo 2

ab	*aba*	*bba*
aba	*bab*	*aa*

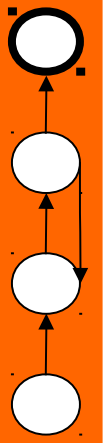
Com problema na sétima posição... Problema sem solução !!!

Vemos portanto que o Problema de Correspondência de Post pode ou não ter solução, dependendo do alfabeto e dominós.

A questão é: existe um algoritmo para determinar se uma dado P.C.P. tem solução?

Em outras palavras: **O P.C.P. e decidivel?**

O P.C.P. é base para vários teoremas envolvendo decidabilidade (especialmente p/ ling. livres de contexto, como veremos...)



Formalização do P.C.P.

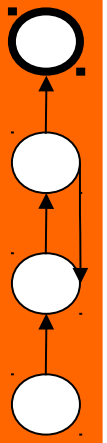
Seja um alfabeto Σ e um conjunto de pares ordenados $[u_i, v_i]$, i=1,2,...,n onde $u_i, v_i \in \Sigma^+$. Uma solução para o sistema de correspondência de Post é uma seqüência i1, i2, ..., ik tal que

$$u_{i1}\, u_{i2} \ldots u_{ik} = v_{i1}\, v_{i2} \ldots v_{ik}$$

O problema de se determinar se um sistema de correspondência de Post tem alguma solução é chamado de

Problema de Correspondência de Post

Teorema (Post, 1946): Não existe algoritmo para se determinar se um dado P.C.P. tem uma solução, ou seja: **o P.C.P. é um problema não-decidível**.



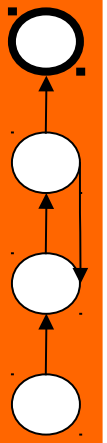
Relembrando: Gramáticas Livres de Contexto

Uma G.L.C. é uma gramática G=(V,T,P,S) em que cada produção é da forma $A \rightarrow w$, em que $A \in V$ e $w \in (V \cup \sum)^*$.

Uma dada G.L.C. (gramática tipo 2 segunda a hierarquia de Chomsky) produz cadeias de uma linguagem livres de contexto, que é por sua vez reconhecida por um autômato *Pushdown*.

Uma derivação de uma sentença em uma gramática em que a variável transformada é sempre aquela mais à esquerda é dita derivação à esquerda. Qualquer cadeia terminal de uma G.L.C. pode ser obtida via derivações à esquerda.

Uma G.L.C. é dita ambígua se existir alguma cadeia $w \in L(G)$ que pode ser obtida por duas derivações à esquerda distintas.



Gramáticas Livres de Contexto Ambíguas

Uma G.L.C. é dita ambígua se existir alguma cadeia w ∈ L(G) que pode ser obtida por duas derivações à esquerda distintas.

Exemplo:

G_1: S → aS | Sa | a (ambígua, pois aa tem duas derivações à esquerda).

G_2: S → aS | a (não ambígua, e equivalente a G_1).

A ambigüidade é uma propriedade da gramática, e não da linguagem. Porem , podemos definir:

Uma linguagem livre de contexto é dita ser inerentemente ambígua se toda G.L.C. que a gera for ambígua.

P.C.P. e Gramáticas Livres de Contexto

Seja (Σ, [u_1, v_1], [u_2, v_2],..., [u_n, v_n]) um sistema de correspondência de Post.
Considere as seguintes gramáticas livres de contexto:

G_U: $V_U = \{S_U\}$

$\Sigma_U = \Sigma \cup \{1, 2, ..., n\}$

$P_U = \{S_U \longrightarrow u_i\ S_U\ i,\ S_U \longrightarrow u_i\ i\}$ (i = 1, 2, ..., n)

G_V: $V_V = \{S_V\}$

$\Sigma_V = \Sigma \cup \{1, 2, ..., n\}$

$P_V = \{S_V \longrightarrow v_i\ S_V\ i,\ S_V \longrightarrow v_i\ i\}$ (i = 1, 2, ..., n)

G_U gera as cadeias da metade superior dos dominós.

G_V gera as cadeias da metade inferior dos dominós.

Os dígitos (i) registram a seqüência de dominós que gera a seqüência.

P.C.P. e Gramáticas Livres de Contexto

Uma solução do problema de correspondência de Post corresponde a uma seqüência $i_1\, i_2 \ldots i_{k-1}\, i_k$ tal que $u_{i1}\, u_{i2} \ldots u_{ik-1}\, u_{ik} = v_{i1}\, v_{i2} \ldots v_{ik-1}\, v_{ik}$ Nesse caso, as gramáticas produzem as derivações:

$S_U \Rightarrow u_{i1}\, u_{i2} \ldots u_{ik-1}\, u_{ik}\, i_k\, i_{k-1} \ldots i_2\, i_1$

$S_V \Rightarrow v_{i1}\, v_{i2} \ldots v_{ik-1}\, v_{ik}\, i_k\, i_{k-1} \ldots i_2\, i_1$

com $u_{i1}\, u_{i2} \ldots u_{ik-1}\, u_{ik}\, i_k\, i_{k-1} \ldots i_2\, i_1 = v_{i1}\, v_{i2} \ldots v_{ik-1}\, v_{ik}\, i_k\, i_{k-1} \ldots i_2\, i_1$

Ou seja: $L(G_U) \cap L(G_V) \neq \emptyset$. Por outro lado, se $w \in L(G_U) \cap L(G_V)$ então $w = w'\, i_k\, i_{k-1} \ldots i_2\, i_1$. A cadeia $w' = u_{i1}\, u_{i2} \ldots u_{ik-1}\, u_{ik} = v_{i1}\, v_{i2} \ldots v_{ik-1}\, v_{i\,k}$ é uma solução para o problema de correspondência de Post.

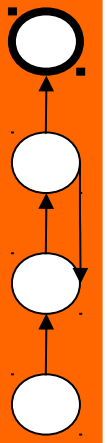
P.C.P. e Gramáticas Livres de Contexto

Teorema: Determinar se duas linguagens livres de contexto são disjuntas é um problema não-decidivel.

Prova: Assuma que o problema seja decidivel. Então existe um algoritmo para resolvê-lo, e o problema de correspondência de Post poderia ser resolvido da seguinte maneira:

1. Construa as gramáticas G_U e G_V como indicado no slide anterior;
2. Use o algoritmo para determinar se $L(G_U) \cap L(G_V) = \varnothing$.
3. Se $L(G_U) \cap L(G_V) \neq \varnothing$, tenho a solução (w') para o P.C.P. em questão. Se $L(G_U) \cap L(G_V) = \varnothing$, não tenho a solução (w') para o P.C.P. em questão.

Mas já vimos (teorema anterior) que o P.C.P. é não-decidivel! Contradição...



P.C.P. e Gramáticas Livres de Contexto

Exemplo: Gramáticas G_U e G_V para o sistema de Post [aaa,aa], [baa,abaaa]:

G_U: $S_U \rightarrow aaa\ S_U\ 1 \mid aaa1 \mid baaS_U\ 2 \mid baa2$

G_V: $S_V \rightarrow aa\ S_V\ 1 \mid aa1 \mid abaaaS_V\ 2 \mid abaaa2$

Derivações que produzem solução para o problema de correspondência:

$S_U \Rightarrow aaa\ S_U\ 1 \Rightarrow aaabaa\ S_U 21 \Rightarrow aaabaaaaa\ 121$

$S_V \Rightarrow aa\ S_V\ 1 \Rightarrow aaabaaa\ S_V 21 \Rightarrow aaabaaaaa\ 121$

Outros teoremas provados a partir da construção das gramáticas G_U e G_V :

Um Teorema...

- Não existe algoritmo para determinar se a linguagem de uma G.L.C. é Σ^*.

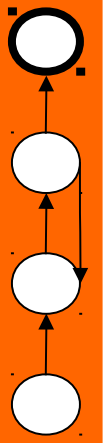
Prova: Um algoritmo para $L = \Sigma^*$ *equivale a um algoritmo para* $\overline{L} = \varnothing$
Seja portanto C um PCP. Como $\overline{L(G_V)}$ *e* $\overline{L(G_U)}$ *são GLCs, então*

$L = \overline{L(G_U)} \cup \overline{L(G_V)}$ *também é GLC. Pela Lei de De Morgan,*

$$\overline{L} = \overline{\overline{L(G_U)} \cup \overline{L(G_V)}} = L(G_U) \cap L(G_V)$$

Assim, ter um algoritmo para determinar se $\overline{L} = \varnothing$ *equivale a ter um algoritmo para determinar se* $L(G_U) \cap L(G_V)$

Mas este último é um problema não-decidível. Assim, não existe algoritmo para determinar se $\overline{L} = \varnothing$ *(ou* $L = \Sigma^*$*).*



Observação

Mas há algum tempo vimos que existe um algoritmo de decisão para linguagens regulares:

"Seja M um AFD com k estados. Então, L(M) é não-vazio $\Leftrightarrow$ M aceita uma cadeia z com length(z) < k."

O que mudou agora?

- Não existe mais um AFD de k estados que aceita L, pois L é GLC.
- Logo, não tenho como limitar a inspeção de cadeias z àquelas com length(z)<k.
- Assim, teria que verificar todas as infinitas cadeias em Σ^* ao APD que reconhece a linguagem L, para determinar se a linguagem de L é mesmo $\varnothing$. Mas um algoritmo deve terminar, e tentar operar sobre um número infinito de possíveis argumentos é naturalmente um processo que não termina!

Mais um Teorema...

- Não existe algoritmo para determinar se uma G.L.C. é ambígua.

Prova: Sejam G_U e G_V as GLCs associadas a um PCP C. Suponha que combinemos estas gramáticas para gerar uma terceira gramática da forma:

$$G: V = \{S, S_U, S_V\}; \Sigma = \Sigma_C; P = P_U \cup P_V \cup \{S \rightarrow S_U , S \rightarrow S_V\}$$

Note que:

1. *Todas as derivações de G são á esquerda, já que qualquer forma sentencial contém no máximo um símbolo não-terminal.*
2. *G_U e G_V são não-ambíguas, pois derivações distintas produzem diferentes sequências de inteiros.*

Assim, G será ambígua apenas se e somente se existir alguma cadeia em $L(G_U) \cap L(G_V)$. Assim, um algoritmo para determinar se G é ambígua corresponde a uma algoritmo para determinar se $L(G_U) \cap L(G_V)$. Mas já vimos que determinar se duas LLC são disjuntas é não-decídivel!