

# Compiladores - Análise Ascendente

---

Fabio Mascarenhas – 2017.2

<http://www.dcc.ufrj.br/~fabiom/comp>

# Análise Descendente vs. Ascendente

---

- As técnicas de análise que vimos até agora (recursiva com retrocesso, recursiva preditiva, LL(1) de tabela) usam a mesma estratégia de análise: a *análise descendente*, ou *top-down*
- Vamos ver agora uma outra estratégia de análise, a *ascendente*, ou *bottom-up*, e as técnicas que a utilizam
- A diferença mais visível entre as duas é a forma de construção da árvore: na análise descendente construímos a árvore de cima para baixo, começando pela raiz, e na ascendente de baixo para cima, começando pelas folhas

# Análise Ascendente

---

- A análise ascendente é mais complicada de implementar, tanto para um analisador escrito à mão (o que é muito raro) quanto para geradores
- Mas é mais geral, o que quer dizer que impõe menos restrições à gramática
- Por exemplo, recursão à esquerda e prefixos em comum não são problemas para as técnicas de análise ascendente
- Vamos usar um exemplo que deixa essas vantagens bem claras

# Gramática de Expressões

---

- Vamos usar como exemplo uma gramática de expressões simplificada:

$S \rightarrow E$   
 $E \rightarrow E + T$   
 $E \rightarrow E - T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow - F$   
 $F \rightarrow \text{num}$   
 $F \rightarrow ( E )$

- Vamos analisar a cadeia num \* num + num

# Reduções

- A análise ascendente analisa uma cadeia através de *reduções*, aplicando as regras da gramática ao contrário:

num	*	num	+	num
F	*	num	+	num
T	*	num	+	num
T	*	F	+	num
T			+	num
E			+	num
E			+	F
E			+	T
E				
S				

- Vamos ler a sequência de reduções de trás para a frente:  $S \rightarrow E \rightarrow E + T \rightarrow E + F \rightarrow E + \text{num} \rightarrow T + \text{num} \rightarrow T * F + \text{num} \rightarrow T * \text{num} + \text{num} \rightarrow F * \text{num} + \text{num} \rightarrow \text{num} * \text{num} + \text{num}$  + à direita

# Reduções vs derivações

---

- A sequência de reduções da análise ascendente equivale a uma *derivação mais à direita*, lida de trás pra frente
- Lembre-se que, para uma gramática não ambígua, cada entrada só pode ter uma única derivação mais à direita
- Isso quer dizer que a sequência de reduções também é única! O trabalho do analisador é então achar qual a próxima redução que tem que ser feita a cada passo

# Exercício

- Qual a sequência de reduções para a cadeia - ( num + num ) - num

$$- ( \sim + \sim ) - \sim$$

$$- ( \text{F} + \sim ) - \sim$$

$$- ( T + \sim ) - \sim$$

$$- ( E + \sim ) - \sim$$

$$- ( E + F ) - \sim$$

$$- ( E + T ) - \sim$$

$$- ( E ) - \sim$$

$$- \text{F} - \sim$$

$$\text{F} - \sim$$

$$T - \sim$$

$$E - \sim$$

$$E - F$$

$$E - T$$


$$E$$

$$\text{F}$$

$$\begin{array}{l} S \rightarrow E \\ E \rightarrow E + T \\ E \rightarrow E - T \\ E \rightarrow T \\ T \rightarrow T * F \\ T \rightarrow F \\ F \rightarrow - F \\ F \rightarrow \text{num} \\ F \rightarrow ( E ) \end{array}$$

# Análise shift-reduce


---

- As reduções da análise ascendente formam uma derivação mais à direita de trás para frente
- Tomemos o passo da análise ascendente que leva a string  $uvw$  para  $uXw$  pela redução usando uma regra  $X \rightarrow v$   

- O pedaço  $w$  da string só tem terminais, pois essa redução corresponde ao passo  $uXw \rightarrow uvw$  de uma derivação mais à direita
- Isso implica que a cada passo da análise temos um *sufixo* que corresponde ao resto da entrada que ainda não foi lido



# Análise shift-reduce

---

- Vamos marcar o foco atual da análise com uma | 
- À direita desse foco temos apenas terminais ainda não lidos
- À esquerda temos uma mistura de terminais e não-terminais
- Imediatamente à esquerda do foco temos um potencial candidato à redução
- O foco começa no início da entrada
- A *análise shift-reduce* funciona tomando uma de duas ações a cada passo: *shift*, que desloca o foco para à direita, e *reduce*, que faz uma redução

# Shift e reduce

---

- Shift: move o foco uma posição à direita
  - $A B C \mid x y z \Rightarrow A B C x \mid y z$  é uma ação *shift*
- Reduce: reduz o que está imediatamente à esquerda do foco usando uma produção
  - Se  $A \rightarrow x y$  é uma produção, então  $C b x y \mid i j k \Rightarrow C b A \mid i j k$  é uma ação *reduce*  $A \rightarrow x y$
- Acontece um erro sintático quando não se pode tomar nenhuma das duas ações, e reconhecemos a entrada quando o chegamos a  $S \mid$ , onde  $S$  é o símbolo inicial

# Exercício

- Qual a sequência de ações para a cadeia - ( num + num ) - num

Handwritten sequence of actions for the LR(0) item sets and transitions for the grammar:

Grammar rules (written on the right):

- $S \rightarrow E$
- $E \rightarrow E + T$
- $E \rightarrow E - T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow F$
- $F \rightarrow - F$
- $F \rightarrow \text{num}$
- $F \rightarrow ( E )$

Handwritten LR(0) item sets and transitions:

Item Set 1 (Left):

- $| - ( \text{num} + \text{num} ) - \text{num} \triangleright$
- $- | ( \text{num} + \text{num} ) - \text{num} \triangleright$
- $- ( | \text{num} + \text{num} ) - \text{num} \triangleright$
- $- ( \text{num} | + \text{num} ) - \text{num} \triangleright$
- $- ( F | + \text{num} ) - \text{num} \triangleright$
- $- ( T | + \text{num} ) - \text{num} \triangleright$
- $- ( E | + \text{num} ) - \text{num} \triangleright$

Item Set 2 (Middle):

- $- ( E + | \text{num} ) - \text{num} \triangleright$
- $- ( E + \text{num} | ) - \text{num} \triangleright$
- $- ( E + F | ) - \text{num} \triangleright$
- $- ( E + T | ) - \text{num} \triangleright$
- $- ( E | ) - \text{num} \triangleright$
- $- ( E ) | - \text{num} \triangleright$
- $- F | - \text{num} \triangleright$
- $F | - \text{num} \triangleright$

Item Set 3 (Right):

- $T | - \text{num} \triangleright$
- $E | - \text{num} \triangleright$
- $E - | \text{num} \triangleright$
- $E - \text{num} | \triangleright$
- $E - F | \triangleright$
- $E - T | \triangleright$
- $F | \triangleright$

Transitions (indicated by arrows):

- From Item Set 1 to Item Set 2:  $| \rightarrow -$
- From Item Set 1 to Item Set 3:  $| \rightarrow ($
- From Item Set 2 to Item Set 1:  $- \rightarrow |$
- From Item Set 2 to Item Set 3:  $- \rightarrow E$
- From Item Set 3 to Item Set 1:  $- \rightarrow |$
- From Item Set 3 to Item Set 2:  $- \rightarrow E$

# Implementação

---

- O que está à esquerda do foco pode ser implementado usando uma pilha
- O foco é o topo da pilha mais uma posição na entrada
- A ação de *shift* empilha o próximo token e incrementa a posição
- A ação de *reduce*  $A \rightarrow w$ .
  - Desempilha  $|w|$  símbolos (que devem formar  $w$ , ou a redução estaria errada)
  - Empilha  $A$



# Escolhas e conflitos

---

- As técnicas de análise ascendente usam a análise shift-reduce como base, a diferença é a estratégia que elas usam para escolher entre ações de shift e reduce
- Problemas na gramática (como ambiguidade), ou limitações da técnica específica adotada, pode levar a *conflitos*
  - Um conflito *shift-reduce* é quando o analisador não tem como decidir entre uma (ou mais) ações de shift e uma ação reduce, o que normalmente acontece por limitações da técnica escolhida
  - Um conflito *reduce-reduce* é quando o analisador não tem como decidir entre duas ou mais ações de reduce, o que normalmente é um *bug* na gramática

# Handles

---

- Mas como o analisador escolhe qual ação deve tomar?
- Uma escolha errada pode levar a um beco sem saída, mesmo para uma entrada válida
- Tautologia: devemos reduzir se a redução vai nos levar um passo para trás em uma derivação mais à direita, e “shiftar” caso contrário
- Em um passo de uma derivação mais à direita  $uAw \rightarrow uvw$ ,  $A \rightarrow v$  é um *handle* de  $uvw$  (normalmente dizemos apenas que  $v$  é o handle)
- Queremos “shiftar” até o topo da pilha ser um handle

# Reconhecendo um handle

---

- Não existe um algoritmo infalível e eficiente para reconhecer um handle na pilha shift-reduce
- Mas existem boas heurísticas, que sempre encontram os handles corretamente para certas classes de gramáticas
  - LR(0), SLR, LALR, LR(1), LR(k), etc...
- A maioria reconhece (ou não) um handle examinando a pilha e o próximo token da entrada (o *lookahead*)

# Prefixos viáveis

---

- Embora não exista um algoritmo exato e eficiente para reconhecer handles, existe um para *prefixos viáveis*
- Prefixos viáveis são as formas sentenciais que podem aparecer na pilha de um parser shift-reduce
- Enquanto o analisador tem um prefixo viável na pilha, ainda não foi detectado um erro sintático
- O conjunto de prefixos viáveis de uma gramática é uma linguagem regular!
- Isso quer dizer que podemos construir um autômato finito para dizer se o que está na pilha é um prefixo viável ou não



# Itens LR(0)

---

- Para construir um autômato que reconhece prefixos viáveis usamos o conceito de itens LR(0)
- Um item LR(0) de uma gramática é uma produção da gramática com uma marca no seu lado direito
- Por exemplo, os itens para a produção  $F \rightarrow ( E )$  são:

$$\left( \begin{array}{l} F \rightarrow \cdot ( E ) \\ F \rightarrow ( \cdot E ) \\ F \rightarrow ( E \cdot ) \\ F \rightarrow ( E ) \cdot \end{array} \right. \begin{array}{l} \rightarrow \text{item inicial} \\ \\ \\ \rightarrow \text{item de redução} \end{array}$$

- Uma produção vazia tem um único item LR(0), e itens com a marca no final são *itens de redução*

# Construindo o autômato não-determinístico

---

- Para construir o autômato, primeiro adicionamos um novo símbolo inicial  $S'$  e uma produção  $S' \rightarrow S$ , caso necessário
- O item  $S' \rightarrow \cdot S$  é o *item inicial*, e ele dá o estado inicial do autômato
- Cada um dos itens da gramática é um estado do autômato
- Um item  $A \rightarrow u \cdot x w$ , onde  $x$  é um terminal, tem uma transição  $x$  para o estado  $A \rightarrow u x \cdot w$ , lembre que tanto  $u$  quanto  $w$  podem ser vazios!
- Um item  $A \rightarrow u \cdot X w$ , onde  $X$  é um não-terminal, tem transições  $\epsilon$  para todos os itens iniciais do não-terminal  $X$ , e uma transição  $X$  para o estado  $A \rightarrow u X \cdot w$
- Todos os estados do autômato são finais!

# Gramática de Expressões

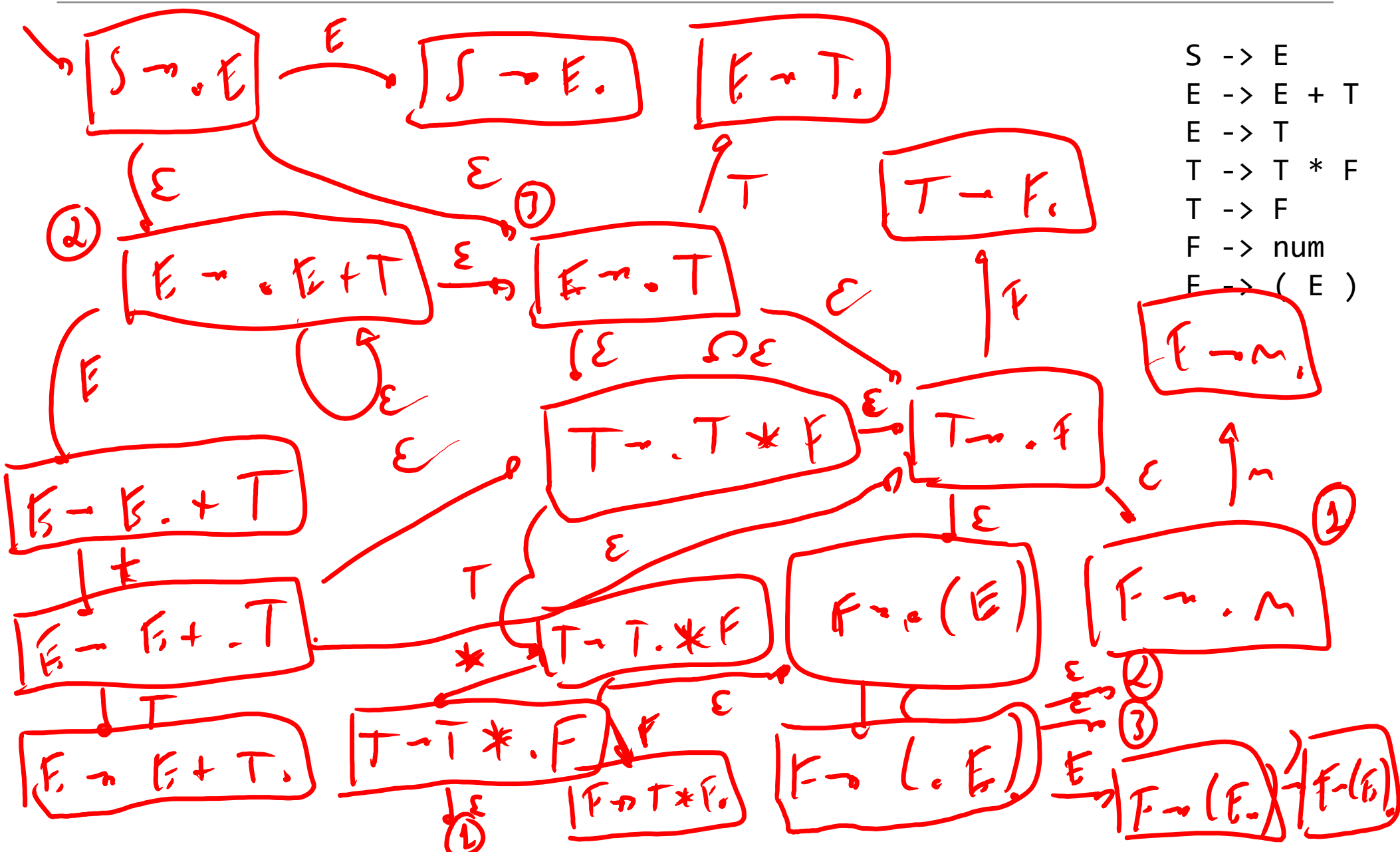
---

- Vamos usar como exemplo uma gramática de expressões simplificada:

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + T \\ E &\rightarrow T \\ T &\rightarrow T * F \\ T &\rightarrow F \\ F &\rightarrow \text{num} \\ F &\rightarrow ( E ) \end{aligned}$$

- Vamos construir o NFA de prefixos viáveis dessa gramática

# NFA de prefixos viáveis



# De NFA para DFA

---

- Podemos converter o NFA para um DFA usando o algoritmo usual
- Isso nos dá um autômato reconhecedor de prefixos viáveis
- Esse autômato é a base da técnica de análise LR(0):
  - Se o autômato leva a um estado com um único item de redução, reduza por aquela produção
  - Senão faça um *shift* e tente de novo
  - Se o autômato chegou em  $S' \rightarrow S \cdot$  e chegamos no final da entrada a entrada foi aceita

# Construção direta do DFA

---

- Na prática construímos diretamente o DFA de itens LR(0) para prefixos viáveis
- Aplicamos a um estado uma operação de *fecho*, que é equivalente ao *fecho- $\epsilon$*  do NFA
  - Se o estado tem um item  $A \rightarrow u \cdot X w$ , onde  $X$  é um não-terminal, então inclua todos os itens iniciais de  $X$
  - Faça isso até nenhum outro item ser incluído
- Sobrarão apenas as transições em terminais e não-terminais, com no máximo uma para cada terminal ou não-terminal saindo de cada estado

# Autômato LR(0)

# Um exemplo que funciona

---

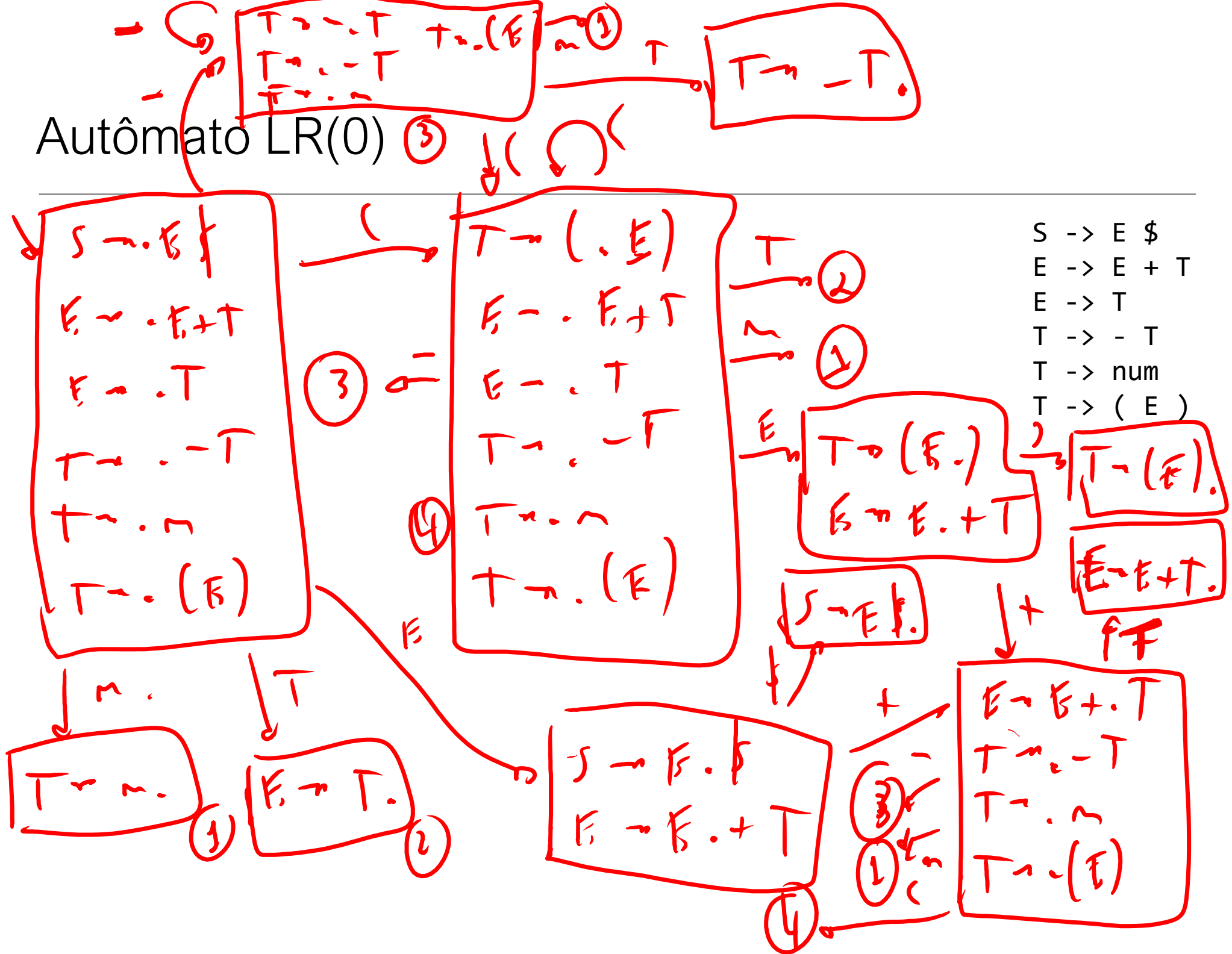
- Todo estado com um item de redução e algum outro item causa conflito LR(0)!
- A técnica LR(0) é bem fraca, mas ainda assim existem gramáticas que ela consegue analisar mas que as técnicas de análise descendente não:

$S \rightarrow E \$$   
 $E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow - T$   
 $T \rightarrow \text{num}$   
 $T \rightarrow ( E )$

- Vamos construir o autômato de itens LR(0) dessa gramática e usá-lo para analisar - ( num + num ) + num \$



# Autômato LR(0) ③



# Analizando uma entrada correta

| - ( num + num ) + num \$

- | ( num + num ) + num \$

- ( | + num ) + num \$

- ( num | + num ) + num \$

- ( T | + num ) + num \$

- ( E | + num ) + num \$

- ( E + | num ) + num \$

- ( E + num | ) + num \$

- ( E + T | ) + num \$

- ( E | ) + num \$

- ( E ) | + num \$

- T | + num \$

T | + num \$

E | + num \$

S → E \$  
E → E + T  
E → T  
T → - T  
T → num  
T → ( E )

E + | num \$

E + num | \$

E + T | \$

E | \$

E \$

5 //

# Analizando uma entrada com erros

- ( num + num + num ) \$

- ( E | + ~ \$

- ( E + | ~ \$

- ( E + ~ | \$

- ( E + T | \$

- ( E | \$

- ( E \$ | X

S  $\rightarrow$  E \$  
E  $\rightarrow$  E + T  
E  $\rightarrow$  T  
T  $\rightarrow$  - T  
T  $\rightarrow$  num  
T  $\rightarrow$  ( E )