CDD: 006.74

UMA EXTENSÃO DO ALGORITMO DE *PARSER LR(0)*

AN EXTENSION OF *LR(0) PARSER* ALGORITHM

Sandra Mara Guse Scós Venske[1], Silvio Luiz Bragatto Boss[1], Martin Alejandro Musicante[2] , Inali Wisniewiski Soares[1] , Luciane Telinski Wiedermann Agner[1], Angelita Maria de Ré[1], Josiane Michalak Hauagge Dall'Agnol[1]

[1] Autor para contato: Universidade Estadual do Centro-Oeste - UNICENTRO, Departamento de Ciências da Computação, Guarapuava, PR , Brasil; (42) 3621-1069; e-mail: luagner@yahoo.com.br

[2] Universidade Federal do Rio Grande do Norte - UFRN, Departamento de Ciências da Computação, Natal, RN

Recebido para publicação em 05/09/2006
Aceito para publicação em 27/11/2006

RESUMO

A inferência de gramáticas livres de contexto possibilita a geração de gramáticas diferenciadas na validação de cadeias. Este artigo apresenta uma extensão do algoritmo clássico LR(0) de construção de tabelas de análise sintática ascendente, para realizar a inferência de regras de produção de gramáticas livres de contexto. Propõe-se o algoritmo LR(0) Indutivo para adicionar novos elementos LR(0) ao conjunto de itens de uma gramática, sendo que estes são usados para definir novas regras de produção, a serem incluídas na gramática. O objetivo do algoritmo LR(0) Indutivo é o de estender uma gramática já existente, de forma a fazer com que uma dada cadeia não aceita possa ser reconhecida por uma extensão da gramática. Nesse cenário, existe um administrador do sistema o qual é experiente no domínio da aplicação a ser estendida e poderá escolher, dentre as soluções propostas, a gramática (esquema) que melhor se adequar às necessidades dessa aplicação. O algoritmo proposto resulta em um conjunto de soluções (gramáticas) com sugestões de regras de produção, podendo ser aplicado no contexto da extensão de esquemas para XML.

Palavras-chave: inferência gramatical, gramática livre de contexto, algoritmo de Parser LR(0

ABSTRACT

The inference of context-free grammars makes the generation of grammars

that differentiate in the validation of grammar strings possible. This paper presents an extension of the classic algorithm for the construction of tables for ascending syntactic analysis, LR (O), in order to carry through the inference of production rules of context-free grammars. We propose the use of the Inductive LR(0) algorithm in the construction of new LR(0) items to be included in the set of items of a grammar, which are then used to define new production rules to be enclosed in the grammar. Our objective in suggesting the use of the Inductive LR (0) algorithm is to extend an already existing grammar, and thus to make a not accepted string to be recognized by an extension of this grammar. In this context, there is a system administrator who has experience in the domain of the application to be extended and will be able to choose, amongst the proposed options, the grammar (scheme) that best adjusts to the needs of the application. The proposed algorithm results in a set of solutions (grammars) with suggestions for production rules that can be applied to XML projects.

Key words**:** grammatical inference, context-free grammar, LR(0) Parsing algorithm.

1. Introdução

Considera-se uma cadeia *w* pertencente à uma linguagem livre de contexto *L*, gerada por uma gramática do tipo *LR*, denominada *G*. Neste contexto, são permitidas atualizações (inserções ou remoções) de um símbolo na posição *p* de *w*. Uma atualização em *w* produz uma nova cadeia *w'* não pertencente a linguagem *L'*. Desta forma, se *w'* for submetida ao algoritmo de *parser LR(0)*, ocorrerá uma *falha* no seu reconhecimento. O *L*, do nome *LR*, indica que a entrada é processada da esquerda para a direita *(left to right)* e o *R* indica que uma derivação à direita é produzida *(rightmost derivation)*.

Este artigo propõe um algoritmo que produz novas gramáticas *LR* livres de contexto *G'* (extensões de *G*), gerando novas linguagens livres de contexto *L'* a partir da falha no reconhecimento da cadeia *w'* . As extensões de *G* devem respeitar as seguintes condições:

- $L \subseteq L'$;
- devem incluir outras cadeias semanticamente semelhantes a *w'* e não somente adicionar *w'*, garantindo a preservação da consistência de cadeias aceitas anteriormente.

O algoritmo proposto, denominado *LR(0) Indutivo*, é uma extensão do algoritmo ascendente clássico para construção de itens *LR(0)* (Aho *et al*., 1986; Louden, 1997), que realiza a inferência de regras de produção utilizando gramáticas livres de contexto. O *LR(0) Indutivo* gera novas regras de produção para uma dada cadeia inicialmente não aceita. O processo começa quando o algoritmo *LR(0)* falha ao analisar uma cadeia ocorrendo, então, uma chamada ao algoritmo *LR(0) Indutivo*, que gera uma nova regra e adicionando-a ao conjunto de regras de produção iniciais.

O processo de construção de novas gramáticas, feito pelo *LR(0) Indutivo*, trabalha com um conjunto de itens *LR(0)* para *G*, realizando modificações neste conjunto de forma a sugerir as novas gramáticas *G'*, a serem analisadas pelo usuário.

O principal objetivo deste trabalho é gerar um conjunto contendo várias opções de escolha de gramáticas livres de contexto inferidas a partir de uma cadeia não aceita. Estas gramáticas adaptam-se à necessidade de uma aplicação específica. Naturalmente, estas opções são semelhantes à gramática original *G*. Em seguida, o usuário administrador do sistema deve observar este conjunto de sugestões de gramáticas e escolher, dentre as apresentadas, a que mais se adapta ao perfil desejado por ele, de acordo com a semântica

da sua aplicação.

O uso do algoritmo *LR(0)* se justifica devido à sua complexidade computacional que é linear, assim como os outros algoritmos da família *LR* (Aho *et al*., 1986; Louden, 1997), sendo o espaço de armazenamento requerido proporcional ao tamanho da gramática e o tempo de análise proporcional ao tamanho da cadeia a ser testada. Outra vantagem da análise sintática ascendente (tipo da família *LR*) é que gramáticas recursivas à esquerda não são um problema para o *parser*, pois a análise é feita de forma *bottom-up*[1].

Em (Nakamura e Ishiwata, 2000; Nakamura, 2003) são apresentados trabalhos sobre aprendizado de gramáticas livres de contexto utilizando exemplos e contra-exemplos. Neles são propostos o algoritmo *Inductive CYK* (*ICYK*) que gera produções, baseado em um conjunto de cadeias positivas e negativas, sendo que a gramática de entrada é opcional. O *ICYK* é uma adaptação do algoritmo de análise sintática *CYK* que realiza aprendizado incremental e executa buscas em um conjunto de regras. O algoritmo *LR(0) Indutivo*, apresentado neste artigo, entretanto, estende uma linguagem previamente definida, bem como propõe novas regras para as gramáticas que geram as linguagens estendidas.

O resultado desta pesquisa será aplicado em um ambiente de troca de dados baseado em XML. Isto se evidencia pelo fato de que XML *Schema* (Muratta), por exemplo, descreve um padrão para validar documentos XML; analogamente, uma gramática representa um padrão para a validação de cadeias. Portanto, pode-se comparar XML *Schema* com gramáticas, e documentos XML com cadeias. Assim, quando se faz a inferência de gramáticas pode-se, da mesma forma, aplicar tal procedimento a XML *Schema*, já que ambos descrevem uma estrutura a ser seguida por uma cadeia ou documento. Neste sentido, o algoritmo *LR(0) Indutivo* auxilia administradores de sistemas de informação, que são experientes no domínio de uma aplicação, mas que não têm conhecimento em ciência da computação. Alguns trabalhos correlatos são (Bouchou 2003; Bouchou, 2004).

O restante deste artigo está organizado como segue: a Seção 2 apresenta definições formais de gramática livre de contexto e conceitos sobre análise sintática ascendente (particularmente sobre itens *LR(0)*) utilizados neste trabalho. A Seção 3 apresenta detalhadamente o Algoritmo *LR(0) Indutivo* proposto. Um exemplo de teste de execução do algoritmo é mostrado na Seção 4. Na Seção 5 são apresentadas as conclusões, considerações finais e trabalhos futuros.

2. Gramática livre de contexto e análise sintática

Nesta seção são apresentados os conceitos e definições formais para gramáticas livres de contexto, análise sintática e itens de uma gramática *LR(0)*.

2.1. Gramática livre de contexto

Uma Gramática Livre de Contexto é uma quádrupla $G = (N, T, P, S)$, onde (Aho *et al*., 1986; Louden, 1997):

- N é um conjunto finito de símbolos não-terminais;
- T é um conjunto finito de símbolos terminais;
- P é um conjunto finito de regras de produção;
- $S \in N$ é o símbolo inicial.

As regras de produção são da forma $A \rightarrow u$ com $A \in N$, $u \in (N \cup T)$. Adota-se comumente na literatura (Atho *et al*., 1986; Louden, 1997) representar os símbolos não-terminais por caracteres maiúsculos e os símbolos terminais por caracteres minúsculos. Para quaisquer cadeias v e $w \in (N \cup T)+$, escreve-se $v \Rightarrow_G w$, se w pode ser derivada de v substituindo um não-terminal A em v por u, através da aplicação da regra $A \rightarrow u$ em G. A linguagem $L(G)$, gerada pela gramática G é o conjunto

$$L(G) = \{w \in T^* \mid S \Rightarrow_G^* w\}$$

2.2. Análise sintática e itens *LR(0)*

A análise sintática (Grune, 2000), também chamada de *parser*, é um processo que serve para analisar a estrutura gramatical de uma entrada, manipulan-

[1] Termo que define como a árvore sintática de validação da cadeia é construída – das folhas para a raiz.

do os *tokens*, que são segmentos de texto ou símbolos que podem ser analisados, sendo essa avaliação feita com base numa gramática (no caso deste trabalho, uma gramática livre de contexto).

Dada uma gramática livre de contexto *G*, um *item LR(0)* para *G* é uma regra de produção com uma posição marcada do lado direito. Por exemplo, a produção $A \rightarrow a\ B\ c$ pode gerar quatro itens:

1. $A \rightarrow \bullet\ a\ B\ c$,
2. $A \rightarrow a \bullet B\ c$
3. $A \rightarrow a\ B \bullet\ c$
4. $A \rightarrow a\ B\ c \bullet$

Quando se encontra um item $A \rightarrow \alpha\bullet\beta$ significa que o processo de *parser* acabou de reconhecer a. O caractere após o marcador é o *símbolo alvo* e representa o próximo símbolo a ser processado pelo *parser*. Uma vez que este símbolo seja reconhecido, o marcador avança uma posição.

Cada estado do *parser LR(0)* é representado por um conjunto de itens que é definido pela gramática da linguagem a ser analisada. No algoritmo *LR(0)*, o conjunto de itens para determinada gramática é construído sem consulta às possíveis cadeias de entrada. Este conjunto consiste na construção de uma tabela de análise sintática (como detalhado na Figura 7 do exemplo da Seção 4) baseada na função de transição da máquina de estados finitos. Cada estado da máquina (como detalhado na Figura 6 do exemplo da Seção 4) corresponde a um conjunto de itens. As transições da máquina de estado são definidas para cada símbolo *x*, de tal forma que, existe uma transição para *x* de qualquer estado contendo o item $[A \rightarrow \alpha\bullet x\beta]$ para um estado contendo $[A \rightarrow \alpha x\bullet\beta]$. Por exemplo, como pode-se observar na Figura 6 existe uma transição para *b* do estado 8 $[A \rightarrow a \bullet b]$ para o estado 9 $[A \rightarrow a\ b\bullet]$.

Para a definição e construção do algoritmo *LR(0) Indutivo* utilizou-se o conceito de item *LR(0)* para identificar a posição na cadeia *w* na qual ocorreu a falha na validação pela gramática *G* (gramática objeto da inferência) durante o processo de análise.

Um analisador sintático *LR* consiste em uma entrada, uma saída, uma pilha, um programa diretor e uma tabela sintática que possui duas partes: ação e desvio. Todos os tipos de analisadores sintáticos *LR* possuem o mesmo programa diretor, sendo que a diferença entre eles está na construção da tabela sintática (Aho *et al*., 1986; Louden, 1997).

Um analisador sintático ascendente realiza quatro possíveis ações: carrega (*shift*), reduz (*reduce*), aceita (*accept*) e erro (*error*):

1. Na ação *carrega*, se o próximo símbolo na entrada é *a* e existe uma transição no autômato finito com *a* do estado do topo da pilha (chamado *qi*) para algum estado *qj*, então empilha-se *qj* na pilha e remove-se *a* da entrada.

2. A ação *reduz* representa a redução por uma dada regra da gramática. Para isto, substitui-se o lado direito da regra (armazenado na pilha) pelo símbolo não-terminal do lado esquerdo da regra de produção.

3. A ação *aceita* ocorre quando o conjunto de itens inicial é reduzido e a entrada foi lida até o final.

4. Finalmente, um *erro* ocorre quando o algoritmo chega a um estado e a uma configuração de entrada de tal forma que não há transições para eles no autômato finito, definido pelo conjunto de itens. Essa ação encerra o processo de análise sintática e produz uma mensagem de erro. Neste artigo, refere-se a *erro* como *falha*.

O algoritmo *LR(0)* pertence à família dos algoritmos de análise sintática *LR* e trabalha sem verificação à frente para tomar decisões durante a análise sintática.

3. Interferência de gramáticas livres de contexto usando o algoritmo *LR(0) indutivo*

Nesta seção são apresentados detalhadamente os algoritmos relacionados a este trabalho, em especial, o *LR(0) Indutivo*.

O algoritmo *LR(0) Indutivo* é usado para gerar regras de produção para uma dada cadeia *w* inicialmente não aceita por uma gramática *G*. O processo inicia-se quando o algoritmo falhar ao analisar *w*, gerando novas regras de produção e adicionando-as ao conjunto de regras de *G*. Esse algoritmo gera uma regra a ser acrescentada à *G* a fim de validar *w*. No entanto, é possível obter-se várias gramáticas sugeridas que podem validar a cadeia *w*.

Considerando uma gramática livre de contexto *G*, seu conjunto de regras de produção P_0 e uma cadeia $w \notin L(G)$, é possível sugerir um conjunto de regras de produção P_1 utilizando o algoritmo *LR(0) Indutivo* e acrescentar cada regra a *G*, de tal forma que *w* seja derivada de $P_0 \cup P_1$. As modificações efetuadas em *G* devem garantir que todas as cadeias pertencentes anteriormente a *L*(*G*) também pertençam à nova gramática e que a nova produção inferida (P_1) seja somente acrescentada a *G*, não ocorrendo mudanças nas produções existentes (P_0).

Uma breve descrição do algoritmo *LR(0)* é mostrado na Figura 1, onde a entrada para o algoritmo é a gramática *G* e uma cadeia *w* a ser testada. Por sua vez, a saída pode ser de duas formas:

1. Caso $w \in L(G)$, o algoritmo finaliza e responde afirmativamente.

2. Caso $w \notin L(G)$, o algoritmo pára, chama o *LR(0) Indutivo* e este sugere novas regras de produções a serem adicionadas a *G*.

No segundo caso, os passos a serem seguidos definem o algoritmo *LR(0) Indutivo*, que é mostrado na Figura 2. Eles são sumarizados a seguir:

- Verifica-se em que ponto a cadeia *w* não foi reconhecida. Isto é feito detectando-se em que ponto a máquina de estado parou o reconhecimento, sendo que este estado corresponde a um elemento do conjunto C_i de itens *LR(0)* para *G*.
- Baseado no item em C_i detectado é construído um conjunto de teste (*TS – Test Set*, apresentado à frente) com sugestões candidatas sob a forma (α_1, α_2, ... α_n), com $\alpha_i \in (N \cup T)^*$ e $1 \leq i \leq n$. A cadeia a_i representa cada sugestão candidata de *TS*.
- Para cada candidata (α_i) uma nova regra de produção é construída. Esta regra tem o formato $A \rightarrow \alpha_i.$, sendo adicionada a *G*. Então, *w* é resubmetida como entrada para o algoritmo *LR(0)*.
- As sugestões candidatas para as quais o algoritmo *LR(0)* responde afirmativamente para a validação de *w* são agrupadas no conjunto de soluções válidas (*VS – Valid Set*).
- O resultado final é um conjunto de diferentes gramáticas sugeridas, formadas pelas regras de produções compostas pelos elementos de *VS*. Tal conjunto é então submetido à seleção do usuário administrador.

Parser_LR0
Entrada:
Cadeia *w;*
Gramática Livre de Contexto *G;*
Regras de Produção *P;*
Símbolos Não-Terminais *N;*
Símbolos Terminais *T.*
Saída:
Se $w \in L(G)$ a resposta é VERDADEIRO;
senão executar o Procedimento LR0_Indutivo.
Procedimento:
Execução do Algoritmo *LR(0)* clássico.

Figura 1- Algoritmo de *Parser LR(0)*.

LR0_Indutivo
Entrada:
Cadeia *w;*
Gramática Livre de Contexto *G;*
Regras de Produção *P;*
Símbolos Não-Terminais *N;*
Símbolos Terminais *T.*
Saída:
Conjunto de Gramáticas Livres de Contexto
Procedimento:
Considerar-se-a $w = \bullet w_1 w_2$, sendo w_1 a parte de *w* já reconhecida e w_2 a parte de *w* que falta ser lida.
Passo 1:
Encontrar o elemento $C_i[i]$ que corresponde ao elemento do conjunto C_i de itens *LR(0)* da Gramática Livre de Contexto *G* em que ocorreu o erro.
Passo 2:
Executar o Procedimento Construção_*TS*($C_i[i]$, w_2). Este procedimento retornará um conjunto (*TS*) com as sugestões candidatas geradas.
Passo 3:
$P_0 \leftarrow P$;
$P_0 \leftarrow P_0 \cup \{TS[i]\}$, sendo $TS[i]=A \rightarrow \alpha\beta$, representando cada elemento de *TS*;
Executar o Procedimento Parser_LR0(*N, T,* P_0 *, w*);
Se Parser_LR0 retornar VERDADEIRO
então *VS*[j] ← *TS*[i];
Reiniciar o Passo 3.
Passo 4:
Para cada elemento *VS*[j], uma nova Gramática é criada, com novas Regras de Produção;
G[i] ← *G*;
$P \leftarrow P \cup \{VS[j]\}$;
Escrever *G*[i].

Figura 2 - Algoritmo *LR(0) Indutivo*.

O algortimo *LR0_Indutivo* faz a chamada ao procedimento *Construção_TS*, sendo apresentado com maiores detalhes na Figura 3. Os passos para sua construção compreendem desde a identificação do ponto onde ocorre a falha ao reconhecer uma cadeia, até a combinação com os elementos da gramática, tais como:

• Seja $I \rightarrow I_0 \bullet I_1$ o elemento correspondente ao estado em que ocorreu a falha. O início do processo de construção do *TS* se dá utilizando I_0.

• Realizar a concatenação de I_0 com o que falta ser lido da cadeia de entrada a ser validada.

• Realizar a concatenação de I_0 com cada um dos terminais de *G*.

• Realizar a concatenação de I_0 com cada um dos não-terminais de *G*.

• Realizar a concatenação dos não-terminais de *G* uma posição antes do último símbolo de I_0.

Construção_*TS* (Item, Cadeia)

Entrada:

Elemento do conjunto C_i de itens *LR(0)* da Gramática Livre de Contexto *G* em que ocorreu o erro $C_i[i]$ (este item é exatamente o ponto no qual o reconhecimento falhou).

Parte da cadeia que falta ser lida w_2.

Saída:

Conjunto *TS*.

Procedimento

TS deve ser um vetor dinâmico, estando na forma $C_i[i] = I_0 – I_1$.

Passo 1:

$TS[i] \leftarrow I_0 \bullet w_2$;

Para cada elemento do Conjunto de Terminais *T*

$TS[i] \leftarrow I_0 \bullet T[j]$;

Para cada elemento do Conjunto de Não-Terminais *N*

$TS[i] \leftarrow I_0 \bullet N[j]$;

Para cada elemento do Conjunto de Não-Terminais *N* uma posição antes do último símbolo de I_0.

$TS[i] \leftarrow N[j] \bullet I_0 \bullet I_1$.

Figura 3 - Algoritmo para Construção do Conjunto *TS*.

4. Execução do algoritmo *LR(0) indutivo*

Nesta seção é apresentado um exemplo de teste de execução do *LR(0) Indutivo*.

Como exemplo de funcionamento do algoritmo, considere a gramática aumentada *G* com as regras de produções apresentadas a seguir e a cadeia *w* = *bbaabb* inicialmente não aceita por *G*:

$$S' \rightarrow S$$
$$S \rightarrow aAb \mid BaAa$$
$$A \rightarrow ab \mid b$$
$$B \rightarrow Bb \mid b$$

O conjunto de itens *LR(0)* para *G* é:

$S' \rightarrow \bullet S$	$S \rightarrow B \bullet aAa$	$A \rightarrow \bullet b$
$S' \rightarrow S \bullet$	$S \rightarrow Ba \bullet Aa$	$A \rightarrow b \bullet$
$S \rightarrow \bullet aAb$	$S \rightarrow BaA \bullet a$	$B \rightarrow \bullet Bb$
$S \rightarrow a \bullet Ab$	$S \rightarrow BaAa \bullet$	$B \rightarrow B \bullet b$
$S \rightarrow aA \bullet b$	$A \rightarrow \bullet ab$	$B \rightarrow Bb \bullet$
$S \rightarrow aAb \bullet$	$A \rightarrow a \bullet b$	$B \rightarrow \bullet b$
$S \rightarrow \bullet BaAa$	$A \rightarrow ab \bullet$	$B \rightarrow b \bullet$

O primeiro autômato finito não-determinístico gerado a partir do conjunto de itens para *G* é apresentado na Figura 4 e o segundo (sem transições vazias) é apresentado na Figura 5. O que é apresentado nestas figuras é o processo base para geração do Autômato Finito Determinístico de Itens, utilizado como ponto de partida para a execução do *LR(0) Indutivo*.

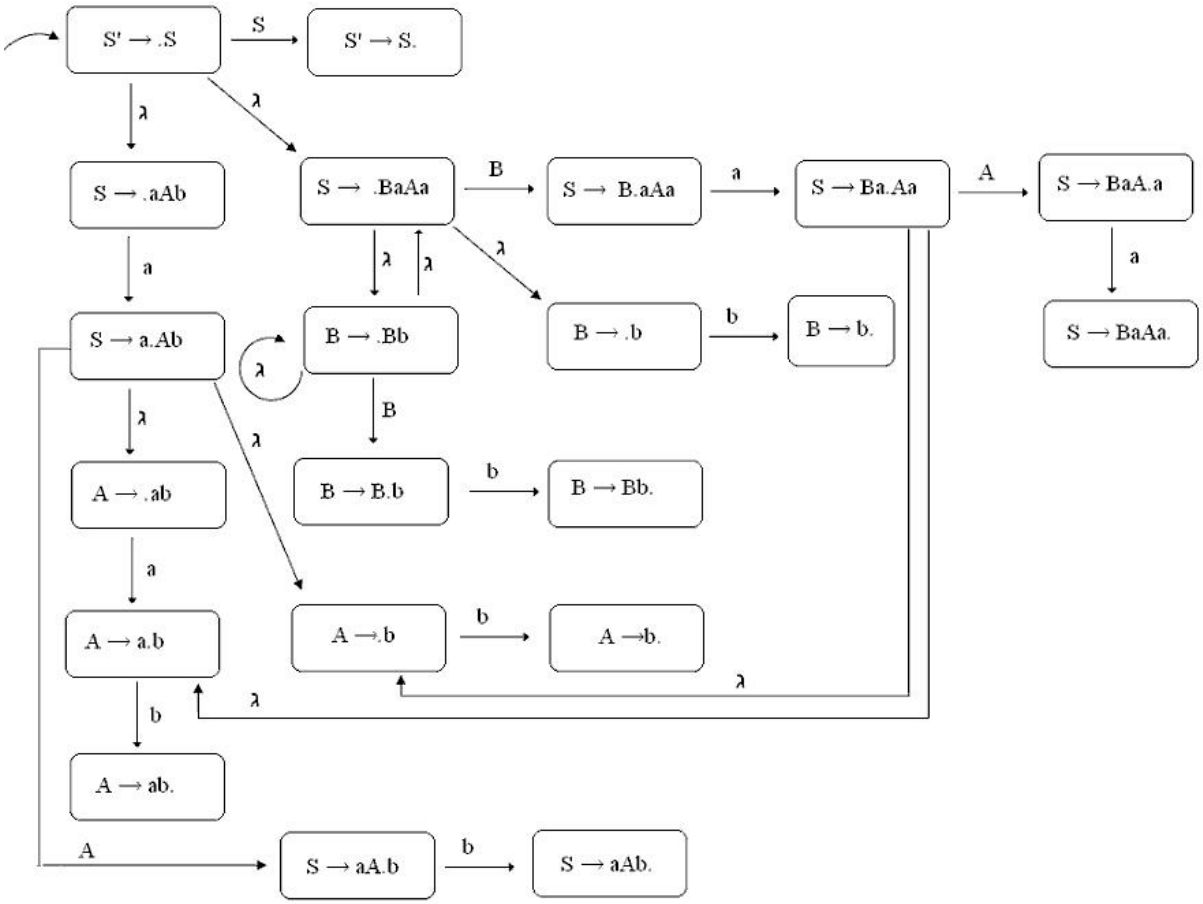

Figura 4 - Autômato Finito Não-Determinístico com transições vazias.

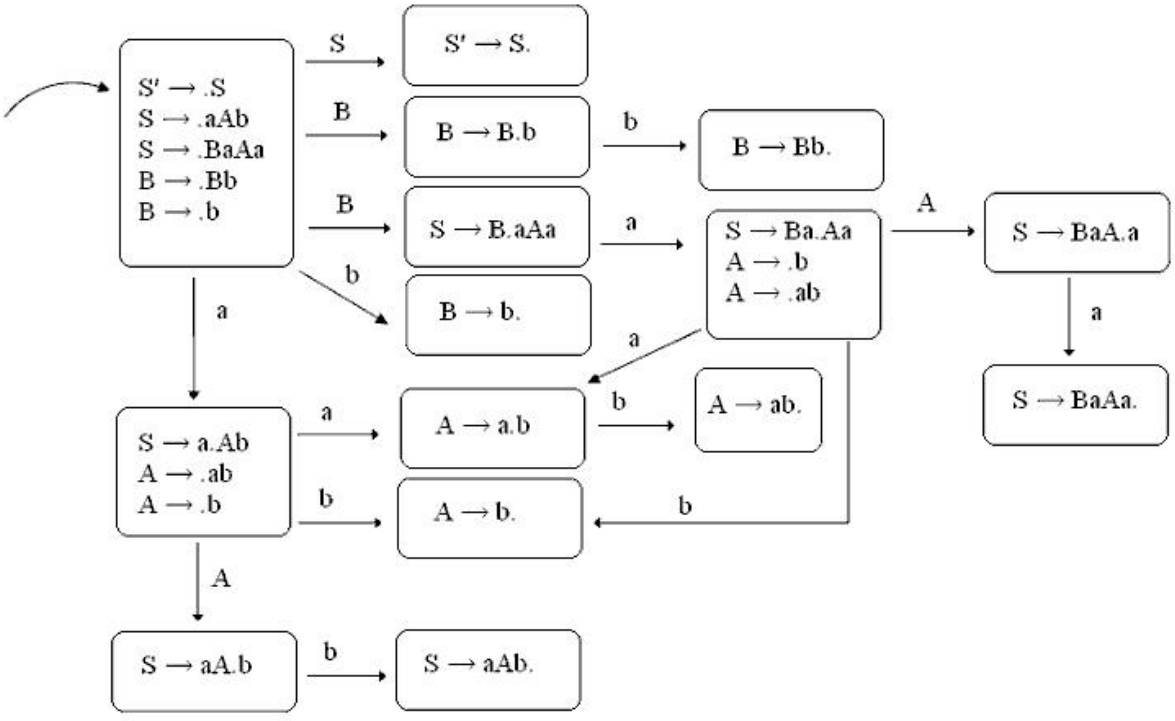

Figura 5 - Autômato Finito Não-Determinístico.

O autômato finito determinístico correspondente é apresentado na Figura 5, onde todos os estados com duas ou mais transições rotuladas com o mesmo símbolo são agrupados em um único estado (isto elimina o não-determinismo do autômato). Mais detalhes sobre Autômatos Finitos podem ser encontrados em Mcnaughton (1993).

A Figura 7 apresenta a tabela de análise sintática para a gramática *G*; sendo que para cada linha há somente uma ação a ser tomada (carregar – *shift* ou reduzir – *reduce*).

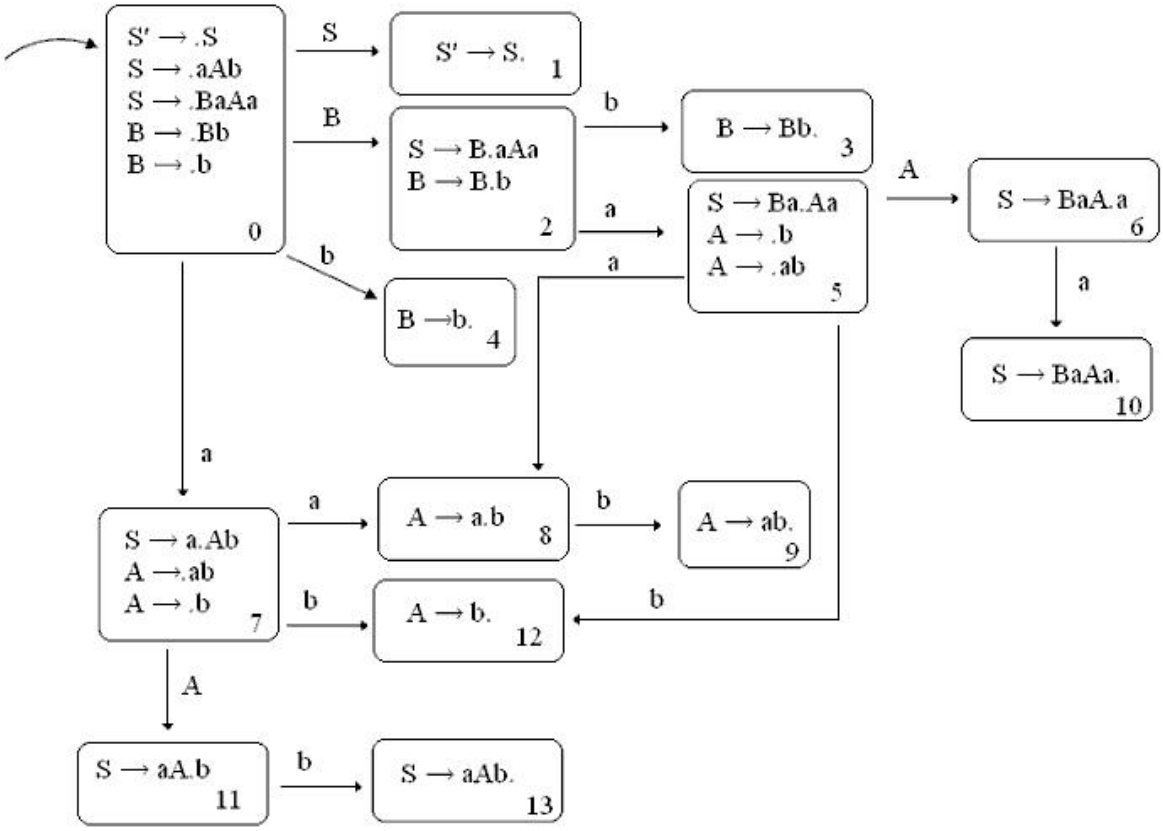

Figura 6 - Autômato Finito Determinístico sem transições vazias.

State	Action	Rule	Imput		Goto		
			a	b	S	A	B
0	Shift		7	4	1		2
1	Reduce	S → S					
2	Shift		5	3			
3	Reduce	B → Bb					
4	Reduce	B → b					
5	Shift		8	12		6	
6	Shift		10				
7	Shift		8	12		11	
8	Shift			9			
9	Reduce	A → ab					
10	Reduce	S → BaAa					
11	Shift			13			
12	Reduce	A → b					
13	Reduce	S → a AB					

Figura 7 - Tabela de Análise Sintática.

Neste exemplo, quando se submete a cadeia *w = bbaabb* para reconhecimento pelo *LR(0)*, a validação ocorre normalmente até o quinto símbolo de *w* (***bbaab****b*). A falha ocorre no reconhecimento do estado 6 do autômato finito determinístico (Figura 6 e Figura 7), pois o referido estado espera ler um *a* para mudar para o estado 10 (de acordo com seu item $S \rightarrow BaA\bullet a$) e a cadeia apresenta um *b*. Desta forma, o algoritmo *LR(0) Indutivo* é chamado para criar o *TS* tendo como ponto de partida o item *LR(0)* onde houve a falha no reconhecimento da cadeia ($S \rightarrow BaA\bullet a$). As combinações realizadas neste caso são:

• De *BaA* com o que falta ler da cadeia de entrada a ser validada, resultando em {*BaAb*}.

• De *BaA* com cada um dos terminais de *G*, resultando em {*BaAa, BaAb*}.

• De *BaA* com cada um dos não-terminais de *G*, resultando em {*BaAS, BaAA, BaAB*}.

• Dos não-terminais de *G* uma posição antes do último símbolo de *BaA*, resultando em {*BaSA, BaAA, BaBA*}

Com o uso do algoritmo *LR(0) Indutivo*, portanto, obtém-se o seguinte *TS* contendo as combinações de sugestões de produções para esta gramática (desconsiderando-se as sugestões geradas repetidas):

TS = {*BaAb, BaAS, BaAA, BaAB, BaSA, BaBA*}

Na seqüência, um a um individualmente, os elementos de *TS* são adicionados como regra de produção de *G* ao não-terminal S, e então a gramática é resubmetida à execução do *LR(0)*, que retornará afirmativamente para as soluções válidas (que reconhecem a cadeia *bbaabb)*. Cada nova regra é adicionada como produção de S porque foi este o não-terminal presente no estado onde ocorreu a falha. Assim, estas soluções são armazenadas no conjunto de soluções válidas *VS*:

VS = {*BaAb, BaAA, BaAB, BaBA*}

Portanto, para que *G* aceite *w*, as produções sugeridas poderiam ser um dos seguintes conjuntos (as produções inferidas pelo algoritmo estão destacadas):

1: $S' \rightarrow S$, $S \rightarrow aAb \mid BaAa \mid \mathbf{BaAb}$, $A \rightarrow ab \mid b$, $B \rightarrow Bb \mid b$

2: $S' \rightarrow S$, $S \rightarrow aAb \mid BaAa \mid \mathbf{BaAA}$, $A \rightarrow ab \mid b$, $B \rightarrow Bb \mid b$

3: $S' \rightarrow S$, $S \rightarrow aAb \mid BaAa \mid \mathbf{BaAB}$, $A \rightarrow ab \mid b$, $B \rightarrow Bb \mid b$

4: $S' \rightarrow S$, $S \rightarrow aAb \mid BaAa \mid \mathbf{BaBA}$, $A \rightarrow ab \mid b$, $B \rightarrow Bb \mid b$

É importante salientar que cabe ao usuário do sistema definir qual dentre as sugestões fornecidas melhor se adapta à semântica da aplicação.

Conclusões, Considerações finais e trabalhos futuros

Com este estudo foi realizada uma investigação quanto a utilização do algoritmo de análise sintática ascendente *LR(0)* combinado à inferência de gramáticas livres de contexto. Assim, foram realizadas modificações no *LR(0)* resultando no algoritmo *LR(0) Indutivo*, que retorna novas gramáticas *G´*, as quais são uma extensão de uma dada linguagem *L*, representada por uma gramática *G*. A nova linguagem gerada *L´* é, via de regra, mais geral que $L \cup \{w'\}$, prevendo as possíveis necessidades da aplicação desejada pelo administrador do sistema.

O resultado desta pesquisa tem aplicação em um ambiente de troca de dados baseado em XML, já que pode-se fazer uma analogia entre XML *Schema* e gramáticas e, conseqüentemente, entre documentos XML e cadeias. Neste sentido, o algoritmo *LR(0) Indutivo* pode ser utilizado por administradores de sistemas de informação que são experientes no domínio de uma aplicação, mas que não são experientes em ciência da computação, possibilitando que o mesmo escolha a solução que melhor se adequar à sua aplicação.

A aplicação do algoritmo *LR(0)* neste estudo permitiu a verificação de sua efetiva utilização na inferência de gramáticas livres de contexto. Além disso, os outros algoritmos da família *LR* podem ser utilizados e testados da mesma forma.

A pesquisa aponta como próximas atividades relacionadas a este projeto:

• Utilização de algoritmos genéticos sobre o conjunto de teste para verificar a possibilidade de aumento do número das possíveis soluções candidatas.

• Aplicação de técnicas relacionadas a algoritmos de busca para percorrer e gerar o conjunto de teste e determinar caminhos alternativos otimizados para o uso das soluções candidatas.

• Fazer o tratamento dos conflitos e ambigüidades presentes em algumas gramáticas livres de contexto com o algoritmo *LR(0) Indutivo*.

• Utilização de outros algoritmos da família *LR* para inferência com gramáticas livres de contexto.

Outro ponto a ser considerado (e alvo de futuros estudos) é a definição do *TS*, pois o modo como as combinações das possíveis soluções candidatas são obtidas é fundamental para o processo de inferência.

Agradecimentos

Este trabalho foi parcialmente apoiado pelo CNPq.

REFERÊNCIAS

1. AHO, A.; SETHI, R.; ULMAN, J.D. **Compilers: Principles, Techniques, and Tools**. Addison-Wesley, 1986.

2. GRUNE, D. **Modern Compiler Design.** John Wiley and Sons, Inc, 2000.

3. LOUDEN, K. C. **Compiler Construction: Principles and Practice.** PWS Kent, 1997.

4. MCNAUGHTON, R.**Elementary Computability, Formal Languages, and Automata.** Z B Publishing Industries, 2nd edition, 1993.

5. NAKAMURA, K. **Incremental learning of context free grammars by extended inductive *CYK* algorithm.** In *Workshop and Tutorial at ECML/PKDD: Learning Context-Free Grammars*, 2003.

6. NAKAMURA, K.; ISHIWATA, T. **Synthesizing context free grammars from sample strings based on inductive *CYK* algorithm.** In *ICGI '00: Proceedings of the 5th International Colloquium on Grammatical Inference*, pages 186.195, London, UK. Springer-Verlag, 2000.

7. MURATA, M.; LEE, D.; MANI, M. **Taxonomy of XML Schema Languages Using Formal Language Theory**. In Proceedings of the *Extreme Markup Languages Conference*, pages 12-17, Montreal, Quebec, Canada, 2001.

8. BOUCHOU, B.; DUARTE, D.; ALVES, M. H. F.; LAURENT, D.; MUSICANTE, M. A. **Schema Evolution for XML: A Consistency-Preserving Approach.** In Proceedings of the *Mathematical Foundations of Computer Science MFCS*, pages 876-888, Springer Verlag Lecture Notes in Computer Science, 2004.

9. BOUCHOU, B.; ALVES, M. H. F. **Updates and Incremental Validation of XML Documents.** In DBPL, pages 216-232, 2003.