MESTRADO EM ENGENHARIA ELETRÔNICA E COMPUTAÇÃO
MASTER IN ELECTRONIC AND COMPUTER ENGINEERING
São José dos Campos – São Paulo - Brasil

Earley Parser

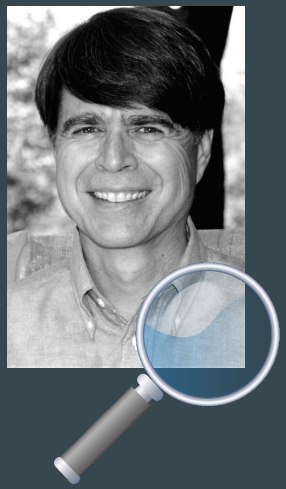
•••

CT200 - Fundamentos de Autômatas e Linguagens Formais

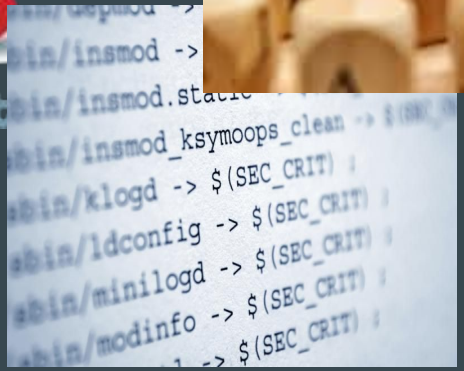
Prof.: Carlos H. Q. Forster

Alunos: Filipe S. de Queiroz
Israel C.S. Rocha

Sumário



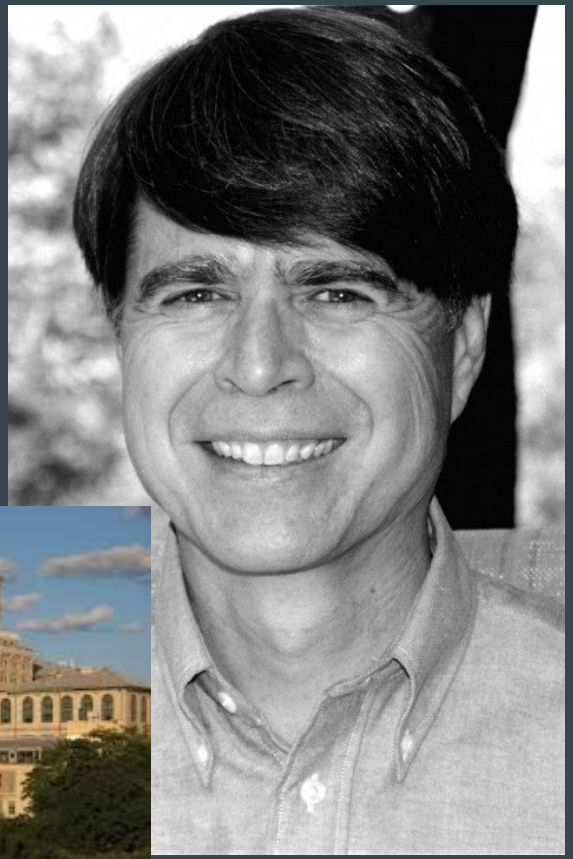
How Does It



Jay Earley

Jay Earley é um americano formado em Ciência da Computação e Psicologia. Desenvolveu seu algoritmo para concluir seu Doutorado em Filosofia na Carnegie-Mellon University, Pensilvânia, EUA, em Agosto de 1968.

Fonte: *Wikipedia, the free encyclopedia*



Earley Parser

- É um algoritmo para utilizado para analisar strings que pertencem à LLC;
- É também chamado de Chart parser;
- É usado principalmente para análise em linguística computacional;

Earley Parser

- Em casos gerais executa em tempo cúbico $O(n^3)$, onde n é o comprimento da sequência analisada.
- Tempo quadrático para gramáticas não ambíguas, e o tempo linear para quase todos os LR gramáticas.
- Funciona particularmente bem quando as regras são escritas à esquerda-recursivamente.

Como funciona?

- Earley Parser é um algoritmo de programação dinâmica e TOP-DOWN;
- Utiliza a notação de dots (.) : dada a produção $X \rightarrow \alpha\beta$, a notação $X \rightarrow \alpha \bullet \beta$ representa uma condição em que α já foi analisado e β é esperado.
- Para cada posição de entrada, o analisador gera um conjunto de estados. Cada estado é uma tupla $(X \rightarrow \alpha \bullet \beta, i)$, consistindo de :
  - A produção atualmente combinada ($X \rightarrow \alpha\ \beta$)
  - Nossa posição atual nessa produção (representada pelo ponto)
  - Índice i da cadeia entrada (*input*) quando o estado foi criado (a posição de origem)

Como funciona?

O parser, em seguida, executa repetidamente três operações: ***prediction***, ***scanning*** e ***completing***.

- **Predictor :** Descreve as possibilidades que podem ser aplicadas para a situação corrente.
- **Scanner :** Dados as predições feitas anteriormente, o scanner irá procurar qual produção que casa com a predição.
- **Completer :** Após esse *matching* , o completer vai completando as predições avançando o operador de posição.

Pseudocódigo

```
function EARLEY-PARSE(words, grammar)
        ENQUEUE((γ → •S, 0), chart[0])
        for i ← from 0 to LENGTH(words) do
                for each state in chart[i] do
                        if INCOMPLETE?(state) then
                                if NEXT-CAT(state) is a nonterminal then
                                        PREDICTOR(state, i, grammar) // non-terminal
                                else do
                                        SCANNER(state, i) // terminal
                        else do
                                COMPLETER(state, i)
                end
        end
return chart
```

Pseudocódigo

```
procedure PREDICTOR((A → α•B, i), j, grammar)
        for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
                ADD-TO-SET((B → •γ, j), chart[ j])
         end
end
procedure SCANNER((A → α•aβ, i), j)
        if B ⊂ PARTS-OF-SPEECH(word[j]) then
                ADD-TO-SET((A → αa•β, i), i), chart[j + 1])
        end
end
procedure COMPLETER((B → γ•, j), k)
        for each (A → α•Bβ, i) in chart[j] do
                ADD-TO-SET((A → αB•β, i), chart[k])
        end
end
```

Pseudocódigo deste Trabalho

```
function EARLEY-PARSE(input, grammar)
        ENQUEUE((γ → •S, 0), chart[0])
        for i ← from 0 to LENGTH(input) do
                for each state in chart[i] do
                        if INCOMPLETE?(state) then
                                if NEXT-CHAR(state) is a nonterminal then
                                        PREDICTOR(state, i, grammar) // non-terminal
                                else do
                                        SCANNER(state, i) // terminal
                        else do
                                COMPLETER(state, i)
                        end
                end
return chart
```

|símbolos| = 1

Pseudocódigo deste Trabalho

```
procedure PREDICTOR((A → α•B, i), j, grammar)
        for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
                ADD-TO-SET((B → •γ, j), chart[ j])
         end
end
procedure SCANNER((A → α•aβ, i), j)
        if B == input[j] then
                ADD-TO-SET((A → αa•β, i), i), chart[j + 1])
        end
end
procedure COMPLETER((B → γ•, j), k)
        for each (A → α•Bβ, i) in chart[j] do
                ADD-TO-SET((A → αB•β, i), chart[k])
        end
end
```

|B|=|a| = 1

Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T | T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

Exemplo

Dada a Gramática:

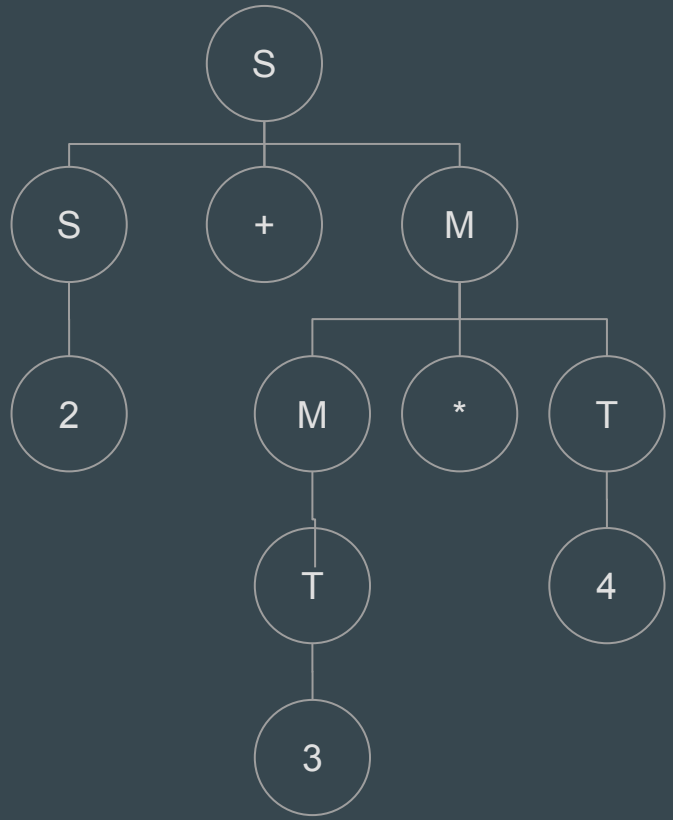
S => S + M | M

M -> M * T | T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4



Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T | T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

.2 + 3 * 4

Chart 0:

Index 0: @ -> .S	(Origin Pos: 0 - STARTING at 0)
Index 1: S -> .S+M	(Origin Pos: 0 - PREDICTION at 0)
Index 2: S -> .M	(Origin Pos: 0 - PREDICTION at 0)
Index 3: M -> .M*T	(Origin Pos: 0 - PREDICTION at 2)
Index 4: M -> .T	(Origin Pos: 0 - PREDICTION at 2)
Index 5: T -> .1	(Origin Pos: 0 - PREDICTION at 4)
Index 6: T -> .2	(Origin Pos: 0 - PREDICTION at 4)
Index 7: T -> .3	(Origin Pos: 0 - PREDICTION at 4)
Index 8: T -> .4	(Origin Pos: 0 - PREDICTION at 4)

Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T | T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

2. + 3 * 4

Chart 1:

Index 0: T -> 2.	(Origin Pos: 0 - SCANNING at 6)
Index 1: M -> T.	(Origin Pos: 0 - COMPLETION at 0)
Index 2: M -> M.*T	(Origin Pos: 0 - COMPLETION at 1)
Index 3: S -> M.	(Origin Pos: 0 - COMPLETION at 1)
Index 4: S -> S.+M	(Origin Pos: 0 - COMPLETION at 3)
Index 5: @ -> S.	(Origin Pos: 0 - COMPLETION at 3)

Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T | T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

2 + .3 * 4

Chart 2:

Index 0: S -> S+.M (Origin Pos: 0 - SCANNING at 4)

Index 1: M -> .M*T (Origin Pos: 2 - PREDICTION at 0)

Index 2: M -> .T (Origin Pos: 2 - PREDICTION at 0)

Index 3: T -> .1 (Origin Pos: 2 - PREDICTION at 2)

Index 4: T -> .2 (Origin Pos: 2 - PREDICTION at 2)

Index 5: T -> .3 (Origin Pos: 2 - PREDICTION at 2)

Index 6: T -> .4 (Origin Pos: 2 - PREDICTION at 2)

Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

2 + 3 .* 4

Chart 3:

Index 0: T -> 3.	(Origin Pos: 2 - SCANNING at 5)
Index 1: M -> T.	(Origin Pos: 2 - COMPLETION at 0)
Index 2: M -> M.*T	(Origin Pos: 2 - COMPLETION at 1)
Index 3: S -> S+M.	(Origin Pos: 0 - COMPLETION at 1)
Index 4: S -> S.+M	(Origin Pos: 0 - COMPLETION at 3)
Index 5: @ -> S.	(Origin Pos: 0 - COMPLETION at 3)

Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

2 + 3 * .4

Chart 4:

Index 0: M -> M*.T (Origin Pos: 2 - SCANNING at 2)

Index 1: T -> .1 (Origin Pos: 4 - PREDICTION at 0)

Index 2: T -> .2 (Origin Pos: 4 - PREDICTION at 0)

Index 3: T -> .3 (Origin Pos: 4 - PREDICTION at 0)

Index 4: T -> .4 (Origin Pos: 4 - PREDICTION at 0)

Exemplo

Dada a Gramática:

S => S + M | M

M -> M * T

T = 1 | 2 | 3 | 4

Sequência de Entrada:

2 + 3 * 4

2 + 3 * 4.

Chart 5:

Index 0: T -> 4.	(Origin Pos: 4 - SCANNING at 4)
Index 1: M -> M*T.	(Origin Pos: 2 - COMPLETION at 0)
Index 2: M -> M.*T	(Origin Pos: 2 - COMPLETION at 1)
Index 3: S -> S+M.	(Origin Pos: 0 - COMPLETION at 1)
Index 4: S -> S.+M	(Origin Pos: 0 - COMPLETION at 3)
Index 5: @ -> S.	(Origin Pos: 0 - COMPLETION at 3)

Demonstração

Conclusão

Em geral Earley Parser é um algoritmo que pode ser aplicado de forma eficiente em Linguagens Livre de Contexto e também fornece base para implementação de Linguagens de programação.

Referências

- Aycock, John; Horspool, R. Nigel (2002). "Practical Earley Parsing". The Computer Journal 45 (6). pp. 620–630. (http://dx.doi.org/10.1093/comjnl/45.6.620).

- Leo, Joop M. I. M. (1991), "A general context-free parsing algorithm running in linear time on every LR(k) grammar without using lookahead", Theoretical Computer Science 82 (1): 165–176, (http://www.ams.org/mathscinet-getitem?mr=1112117).

- Tomita, Masaru (1984). "LR parsers for natural languages". COLING. 10th International Conference on Computational Linguistics. pp. 354–357.

FIM