9. Синтаксический анализ. Часть 1

Разделы:

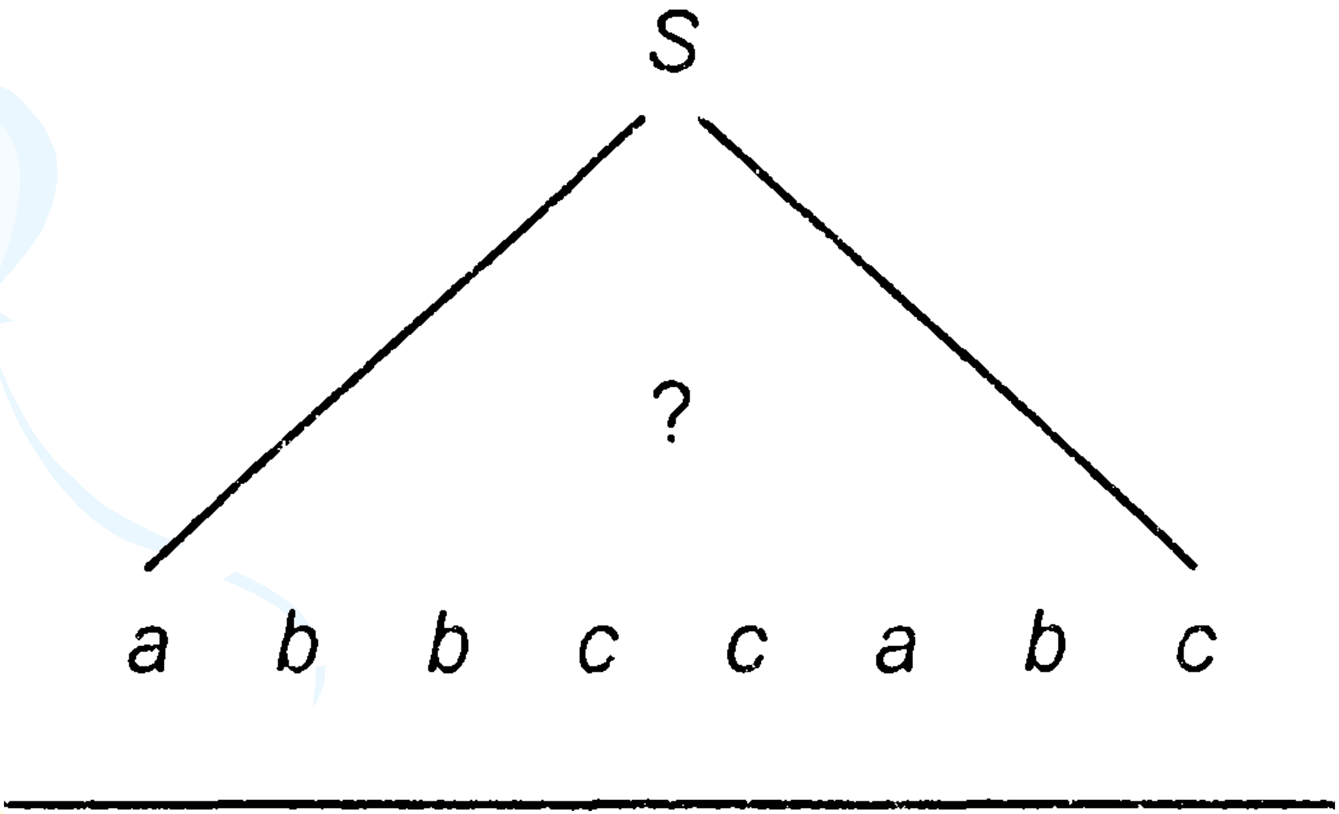
- Общая формулировка задачи СА
- Нисходящий и восходящий СА
- Алгоритм перебора
- Алгоритм Эрли
- Алгоритм Кока-Янгера-Касами

Синтаксический анализ

- Синтаксический анализ (СА)
  - восстановление из аксиомы порождения по терминальной СФ
  - восстановление дерева разбора заданной входной строки
- Для дерева известны листья (входная строка) и корень (аксиома грамматики)
- Рассуждения о восстановлении дерева эквивалентны рассуждениям о восстановлении порождения входной строки из аксиомы
- Восстановление каждого шага порождения соответствует добавлению очередного узла в дерево и обратно

Синтаксический анализ

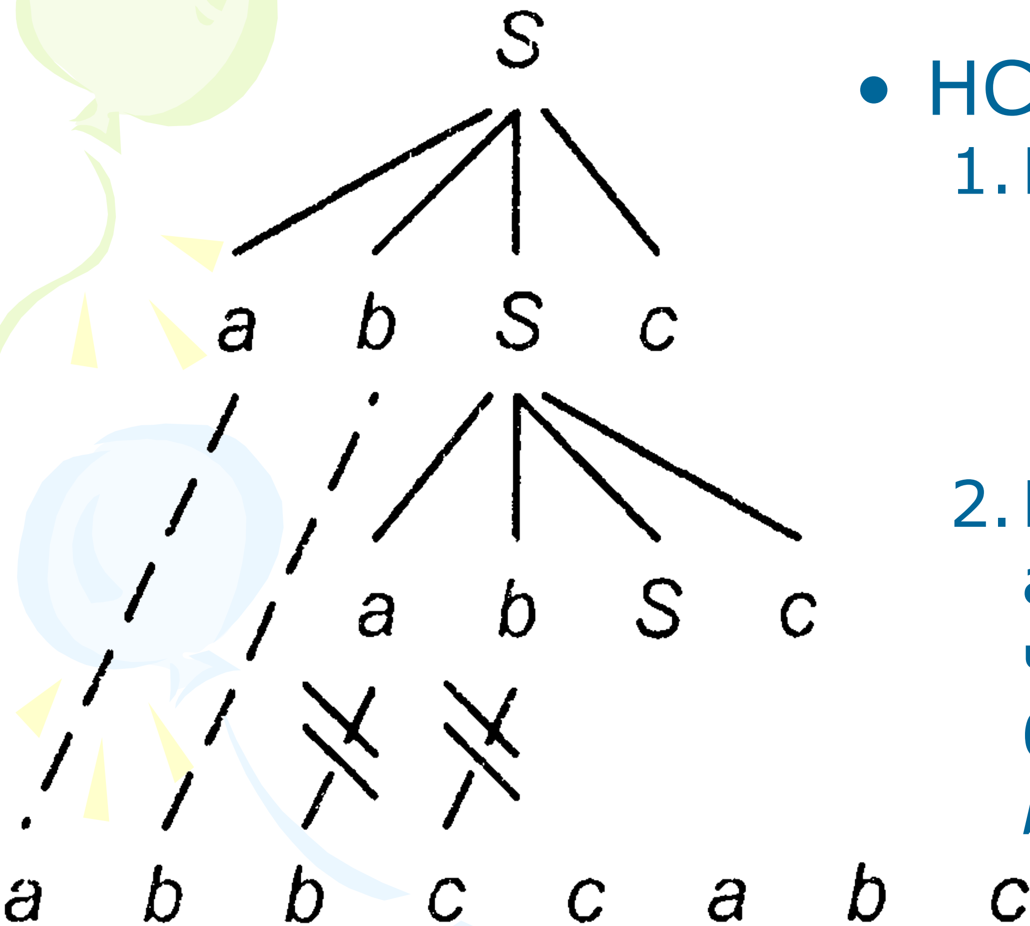
- *S -> abSc | bA, A ->ab | cBA, B -> bBc | c*



Синтаксический анализ

- Можно восстанавливать дерево от корня к листьям
- Эта стратегия СА сверху вниз, или **нисходящий** СА (*top-down parsing*, НСА)
- Можно восстанавливать дерево от листьев к корню
- Это СА снизу вверх, или **восходящий** СА (*bottom-up parsing*, ВСА)

Синтаксический анализ



- НСА *Brute Force*
  1. Выбирается *S -> abSc*
     - СФ – *abSc*
     - Префикс *ab* совпадает с префиксом входной строки
  2. Нужно выбирать из альтернатив для *S* так, чтобы из *Sc* можно было получить строку *bccabc* – суффикс
     - Выбрав *S -> abSc*, получим *abScc*
     - Т.к *ab* ≠ *b*a, выбор неверен, нужен **откат**

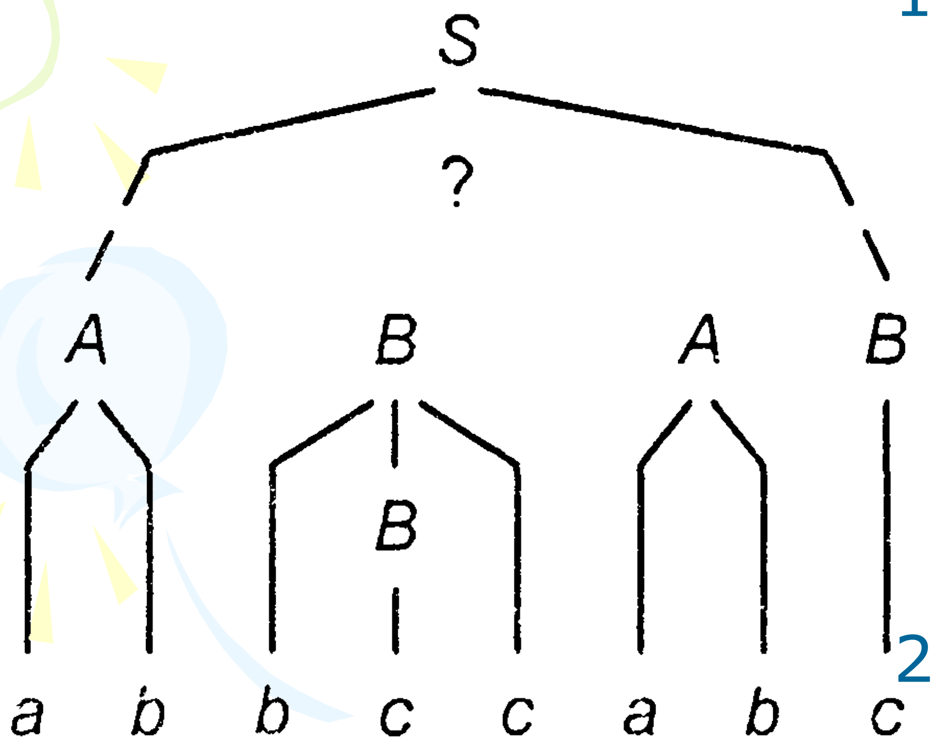
- *S -> abSc | bA, A ->ab | cBA, B -> bBc | c*

Синтаксический анализ

- Откат тратит полиномиальное время
- Чтобы избежать отката при НСА, необходимо сформулировать однозначный критерий выбора альтернативы для каждого нетерминала A в каждом случае, когда он стоит в начале СФ $A\gamma$, из которой следует породить некоторую терминальную СФ β – остаток исходной строки
- В общем случае различных подстрок γ и β, для которых нужны подобные правила, бесконечное количество

Синтаксический анализ

- ВСА *Brute Force*
  1. Начиная от листьев, ищут **связку** (*handle*)
     - Связка СФ μ - это такая левая подстрока β (возможно, пустая), что существует порождение $S =>^* \alpha A\gamma => \alpha\beta\gamma => \mu$.
     - У нас это *ab* согласно *A -> ab*
     - СФ - *Abccabc*
  2. Через несколько шагов придем к СФ ABAB
     - Нужен **откат**



- *S -> abSc | bA, A ->ab | cBA, B -> bBc | c*

Алгоритм Эрли

- Входная строка $\alpha = a_1 \ldots a_n$ читается слева направо
- Для каждого входного символа a_i строится множество ситуаций M_i, определяющее состояние распознавателя после **анализа** этого символа
- Каждая **ситуация** АЭ – это:
  - Помеченная продукция *Pr*, согласно которой в данный момент считывается **связка** входной строки, выводящаяся в соответствии с *Pr*
  - Место в продукции *Pr*, показывающее, какая доля RHS этой продукции уже распознана
    - Это место отмечается знаком '•'
  - Указатель позиции во входной строке, после которой начался поиск возможности применения этой продукции
- Общий формат ситуации $<A\text{->}\alpha\bullet\beta;\ \gamma>$
  - Один из α, β может быть пустым
  - γ - **правый контекст**

Алгоритм Эрли

- Исходную КСГ с аксиомой *S* пополнена продукцией *S*′ -> *#S#*
  - *S*′ – это новая аксиома, а # – это псевдоскобки, куда будет помещаться каждая терминальная строка, порождаемая исходной КСГ
- В начальном множестве ситуаций (M_0 до анализа первого входного символа) будет содержаться единственная ситуацию <*S*′ -> •*#S#*; 0>
- Метка стоит перед началом единственной RHS, которой может быть заменен *S*′, а указатель равен 0
- После 0-го символа начинается подстрока (в нашем случае – вся анализируемая строка), для которой ищется порождение по продукции *S*′->*#S#*

Алгоритм Эрли

- Множество ситуаций изменяется следующими операторами, предсказания, считывания и завершения
- **Предсказатель**
- Если в M_i есть ситуация $<A\text{->}\alpha\bullet B\beta;\ q>$, то во множество добавляется ситуация $<B\text{->}\bullet\gamma;\ i>$ для всех продукций вида $B\text{->}\gamma$; имеющих B в LHS
- Цель и смысл:
  - После i-го символа входной строки, начиная с a_{i+1}, распознаватель ищет любую подстроку, которую можно породить из B
  - Назовем ситуацию $<A\text{->}\alpha\bullet B\beta;\ q>$ – **родительской**, $<B\text{->}\bullet\gamma\ ;\ i>$ – **порожденной**

Алгоритм Эрли

- Множество ситуаций изменяется следующими операторами, предсказания, считывания и завершения
- **Считыватель**
- Если в M_i есть ситуация $<A\text{->}\alpha \bullet b\beta;\ q>$, и если b – очередной терминал a_{i+1} входной строки, то в следующее множество ситуаций M_{i+1} добавляется ситуация $<A\text{->}\alpha \bullet b\beta;\ q>$

Алгоритм Эрли

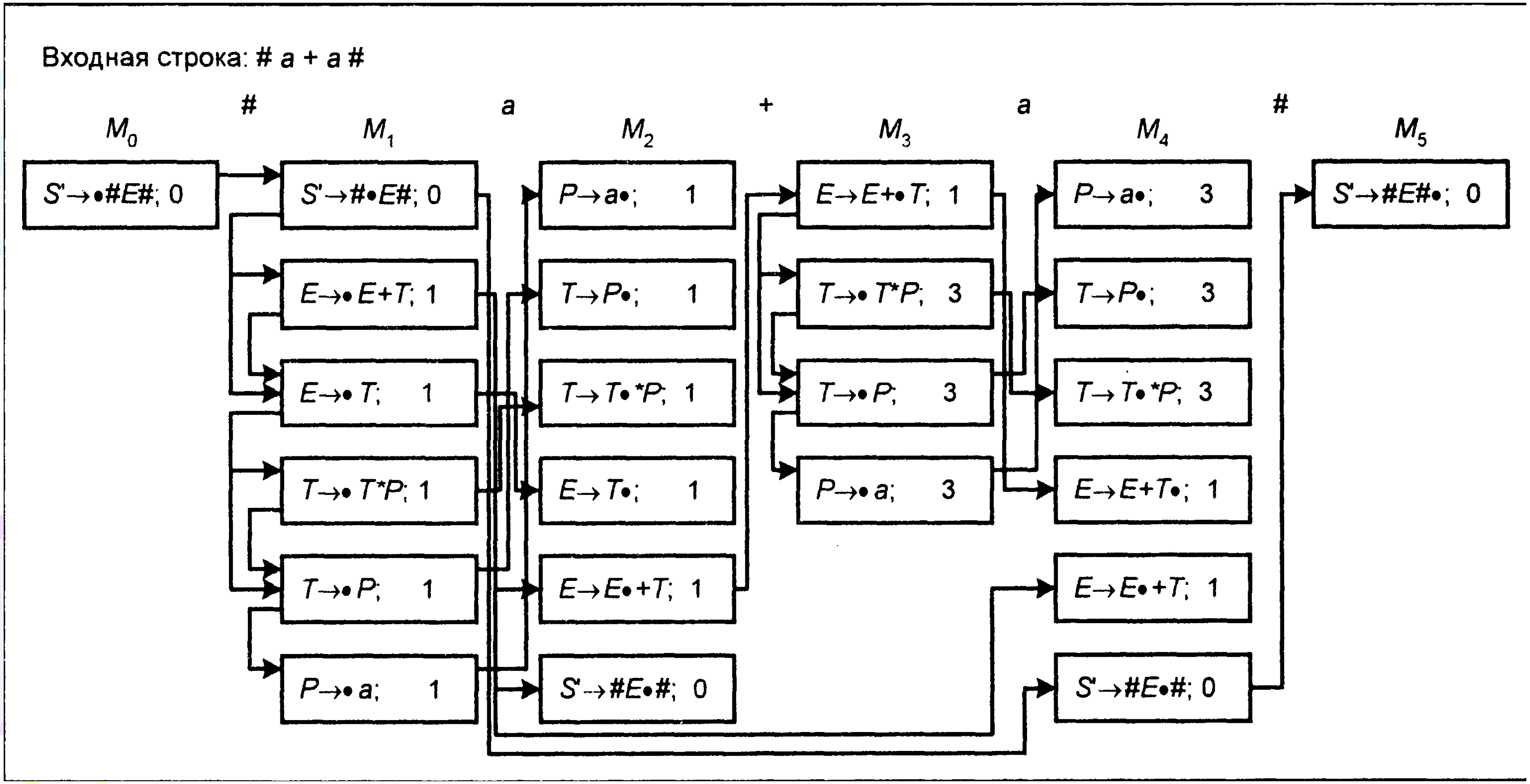
- Множество ситуаций изменяется следующими операторами, предсказания, считывания и завершения
- **Завершатель**
  - Применяется к любой ситуации вида <A->α•; q> в M_i
- Завершение продукции показывает, что она успешно применена, и из нетерминала A, начиная с шага q, порождена строка, совпадающая с $a_q a_{q+1} \ldots a_i$ согласно $A \rightarrow \alpha$
  - Формально, $A \Rightarrow \alpha \Rightarrow^* a_q a_{q+1} \ldots a_i$
- В M_q завершатель ищет ситуацию <A->•α; q>, и для каждой ситуации<B->γA•μ; s>, которая является родительской для <A->•α; q> в M_i он добавляет ситуацию <B->γA•μ; s>
- Значит, во входной строке подстрока $a_s a_{s+1} \ldots a_i$ может быть порождена из начальной части продукции для нетерминала B, а именно $B \Rightarrow \gamma A \mu \Rightarrow^* a_s a_{s+1} \ldots a_i \mu$

Алгоритм Эрли

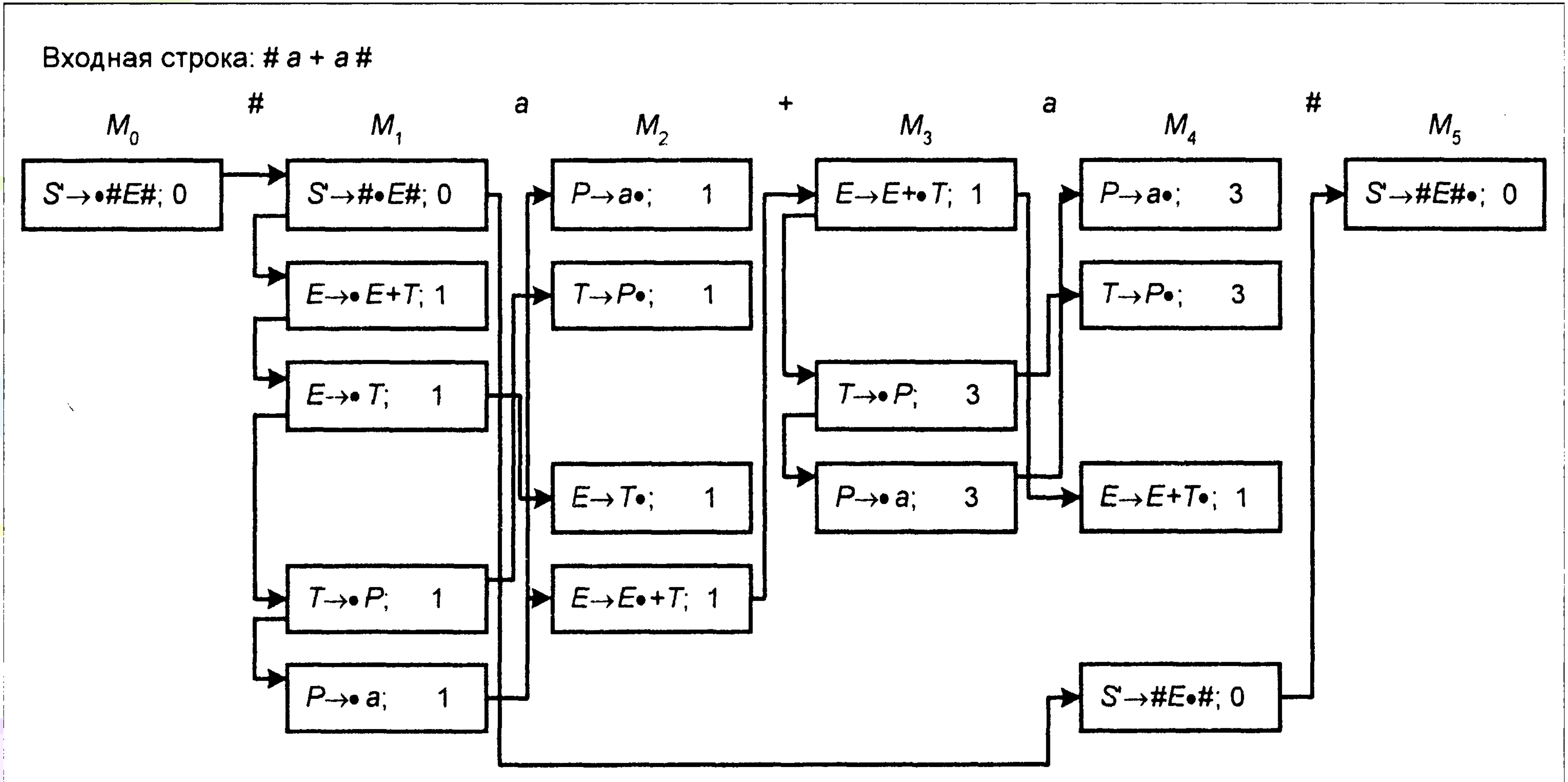
- С каждым множеством ситуаций M_i предсказатель, считыватель и завершатель работают до тех пор, пока в M_i и M_{i+1} не перестанут появляться новые ситуации
- Входная строка принимается, если в заключительном множестве ситуаций, т.е. после последнего символа входной цепочки, встречается ситуация <S'->#S#•; 0>

Алгоритм Эрли

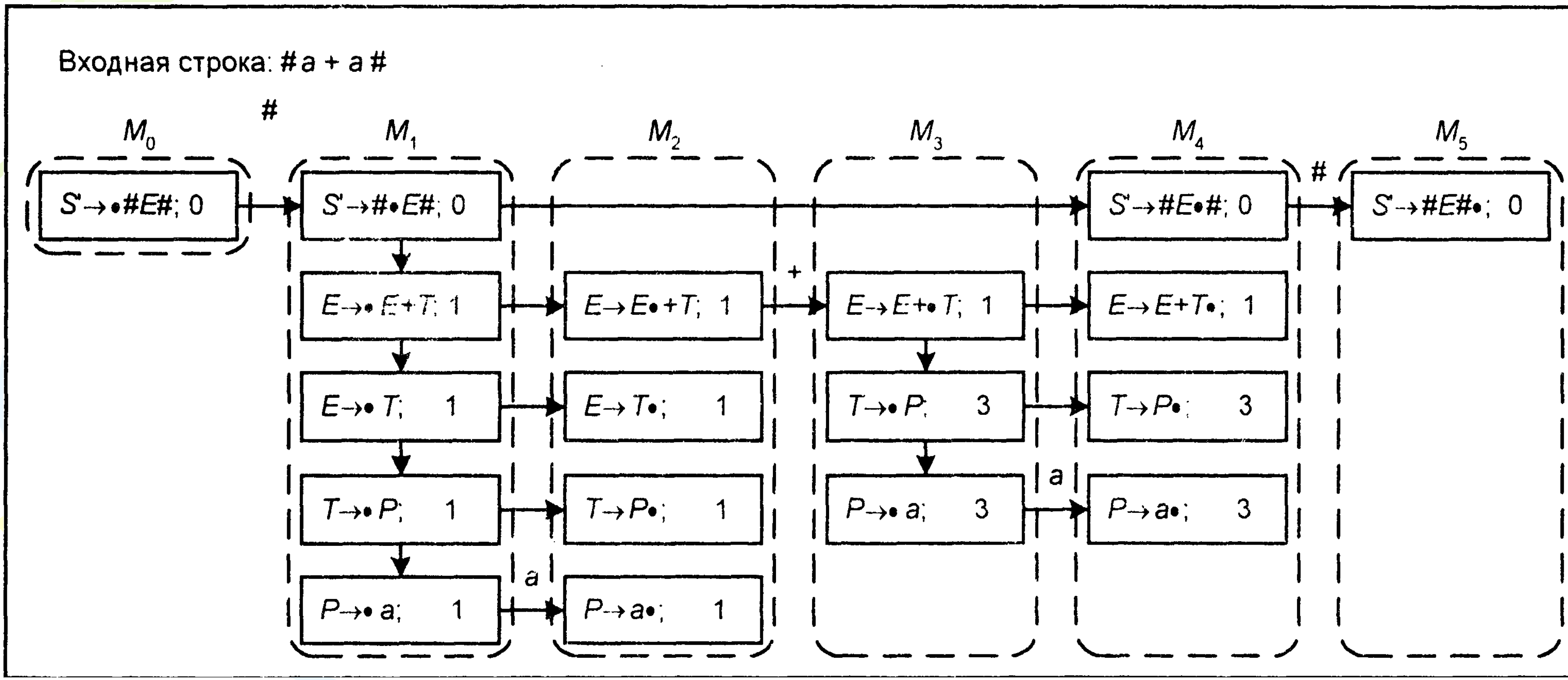
- Дана грамматика *S' -> #E#*, *E->E+T | T*, *T -> T*P | P*, *P -> a*
- Построим множества ситуаций для АЭ на примере анализа строки *# a+a #*



Алгоритм Эрли



Алгоритм Эрли



Алгоритм Эрли

- **Теорема 9.1.** Ситуация <$A \to \alpha \bullet \beta$; i> находится во множестве M_j тогда и только тогда, когда существует такое порождение $S \Rightarrow^* \gamma A \delta$, что:

1. $\gamma \Rightarrow^* a_1 \ldots a_i$
2. $\alpha \Rightarrow^* a_{i+1} \ldots a_j$

Алгоритм Эрли

- У нас бесконтекстная версия АЭ
- Эрли рекомендовал добавить в каждую ситуацию терминальную строку длиной *k* – **допустимый терминальный контекст**, который может встретиться в строках языка после того, как данная продукция была применена
- Контекст используется завершателем для принятия решения о возможности применения закончившейся продукции
- Приписанный ей контекст сравнивается с последующей подстрокой длины *k*, и продукция применятся только тогда, когда эти строки совпадают

CYK-алгоритм

- CYK восстанавливает снизу вверх (ВСА) все возможные порождения сначала для всех подстрок входной строки длиной 1, затем – длиной 2 и т.д. до тех пор, пока не будет проанализирована вся строка
- Основная идея: результаты СА более коротких подстрок, полученные на предыдущих шагах, используются для определения тех продукций, которые могут быть использованы для порождения более длинных строк
- КСГ в этом случае должна быть приведена в НФХ, что позволяет анализировать разбиение любой подстроки входной строки длиной более 1 на две части, СА каждой из которых уже произведен на предыдущих шагах и запомнен в таблице

CYK-алгоритм

- Создает треугольную ТСА T
- В каждую ячейку t_{ik} ТСА помещается множество нетерминалов, из которых можно вывести отрезок выходной строки длиной k, начиная с a_i

  $t_{ij} = \{A \mid A =>^* a_i \dots a_{i+j+1}\}$
- Содержимое ячейки ТСА вычисляется очень просто по КСГ в НФХ
- Входная строка принадлежит языку, порождаемому грамматикой, если в ячейке t_{1n} есть аксиома

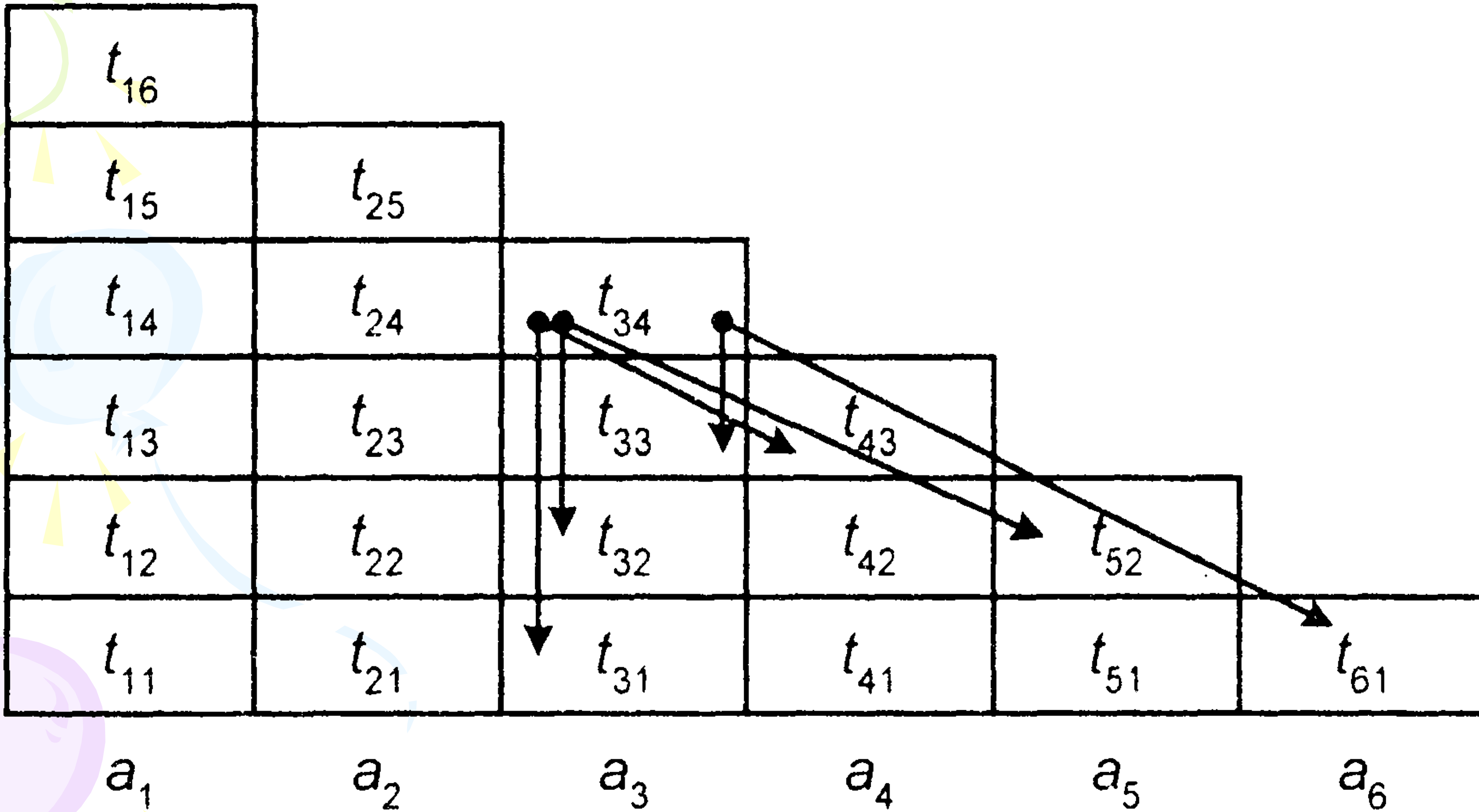
$$t_{i1} = \left\{ A \middle| A \to a_i \in P \right\}$$

$$t_{i1} = \left\{ A \middle| A \to BC \in P \,\&\, \left(\exists k : 1 \le k < j\right) : B \in t_{ik} \,\&\, C \in t_{i+k,\, j-k} \right\}$$

CYK-алгоритм

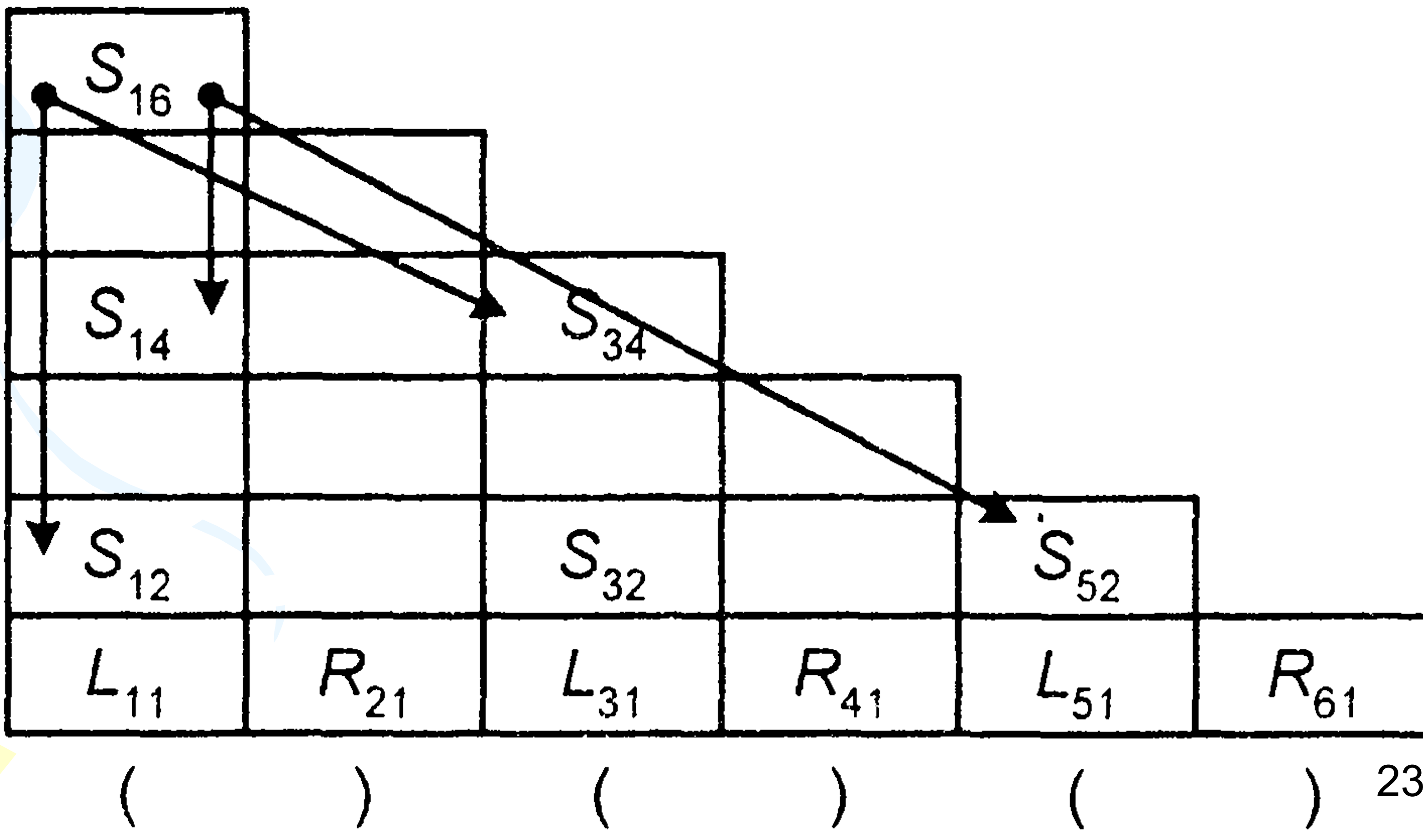
- Нижняя строка ТСА *T* заполняется так
  - в t_{i1} помещаются все нетерминалы A, для которых есть продукция A -> a_i
- Предположим, уже заполнены все строки таблицы *T* от 1 до *j*-1 включительно
- Ячейку t_{ij} соответствует фрагменту $a_i \ldots a_{i+j-1}$
- Этот фрагмент разбивается всеми возможными способами на пары соседних строк $<a_i>$ и $<a_{i+1} \ldots a_{i+j-1}>$ и $<a_{i+2}>$ и $<a_{i+2} \ldots a_{i+j-1}>$ и т.д
- Каждому варианту разбиения соответствует пара ячеек ТСА с нетерминалами, из которых могут быть порождены соответствующие строки
- Пусть эта пара ячеек – (t', t'')
- В ячейку t_{ij} помещается нетерминал *A*, если среди продукций есть *A* -> *BC*, и нетерминал *B* входит в ячейку t', а *C* – в t''

CYK-алгоритм



CYK-алгоритм

- Приведем СА строки ()()() из языка скобок с использованием неоднозначной грамматики с продукциями *S* -> *SS* | *LR*, *L* -> (,. *R* ->)

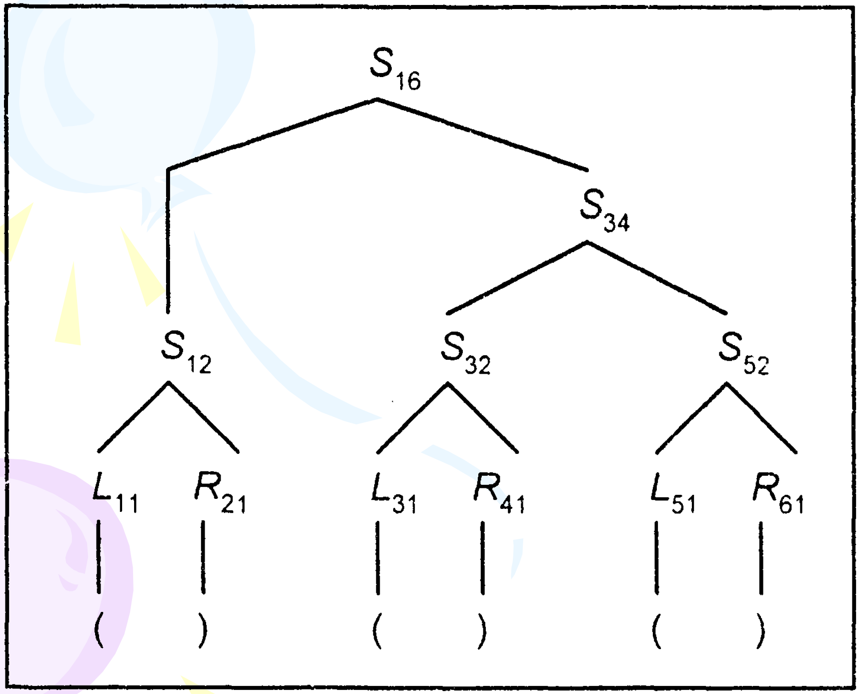


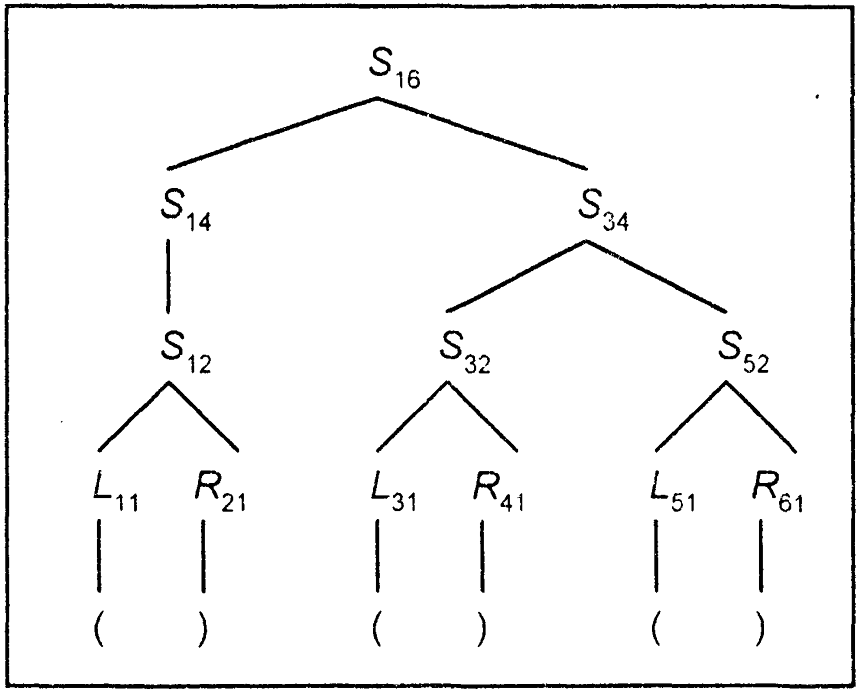
CYK-алгоритм

- Второй шаг алгоритма – это восстановление дерева разбора строки
- Оно осуществляется с помощью рекурсивной процедуры *REDUCE*(i, j, A), которая дает левое порождение $A =>^* a_i a_{i+1} \dots a_{j-1}$
  1. Если $j = 1$ и $A \text{ -> } a_i$ – это продукция с номером m, то выдать m
  2. Пусть $j > 1$
     – выберем k – наименьшее из чисел, для которых существуют B из t_{ik}, C из $t_{i+k,j-k}$ и продукция $A \text{ -> } BC$ с номером m
     – Выберем эту продукцию, выдаем m и выполняем *REDUCE*(i, k, B) и *REDUCE*(i+1, j-k, C).
- Начало - *REDUCE*(1, n, S)

CYK-алгоритм

- Два возможных варианта включения нетерминала S_{16} в ячейку t_{16}
  - либо по продукции S_{16} -> $S_{12}S_{34}$
  - либо по продукции S_{16} -> $S_{14}S_{34}$





Дополнительные источники

- Контекстно-свободная грамматика - http://ru.wikipedia.org/wiki/Контекстно-свободная_грамматика
- Серебряков В. А., Галочкин М. П., Гончар Д. Р., Фуругян М. Г. Теория и реализация языков программирования — М.: МЗ-Пресс, 2006 г., 2-е изд. - http://trpl7.ru/t-books/TRYAP_BOOK_Details.htm
- Контекстно-свободные грамматики - http://www.math.spbu.ru/user/mbk/PDF/Ch-
- J. Earley, "An efficient context-free parsing algorithm", Communications of the Association for Computing Machinery, 13:2:94-102, 1970

Дополнительные источники

- Earley parser - http://en.wikipedia.org/wiki/Earley_parser
- CYK algorithm - http://en.wikipedia.org/wiki/CYK_algorithm
- John Cocke and Jacob T. Schwartz (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.
- T. Kasami (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.
- Daniel H. Younger (1967). Recognition and parsing of context-free languages in time n3. Information and Control 10(2): 189–208.