академия
больших
данных

mail.ru
group

Грамматики и парсинг

Шовкопляс Григорий

Введение в алгоритмы и структуры данных

Введение в теорию формальных языков

Основные определения

- Алфавит — конечное непустое множество символов
- Слово — конечная последовательность символов некоторого алфавита
- Язык - множество слов
- Задача распознавания: принадлежит ли слово w языку L?

Классы языков

- Регулярные языки – задаются регулярными выражениями
- Контекстно-свободные языки – задаются КС-грамматиками
- P – языки, распознаваемые за полиномиальное время
- NP, PSPACE, EXP, …
- Разрешимые, перечислимые, ...

Грамматика

- Множество терминалов — символы, из которых состоят слова
- Множество нетерминалов — промежуточные символы, которые помогают строить слово
- Стартовый нетерминал S — то, с чего все начинается
- Правила вида $\boldsymbol{\alpha} \to \boldsymbol{\beta}$ (строку $\boldsymbol{\alpha}$ можно заменить на строку $\boldsymbol{\beta}$)
- Несколько правил можно записывать через черту:
    - $\boldsymbol{\alpha} \to \boldsymbol{\beta}_1 \mid \boldsymbol{\beta}_2 \mid \dots \mid \boldsymbol{\beta}_k$

Дополнительные обозначения

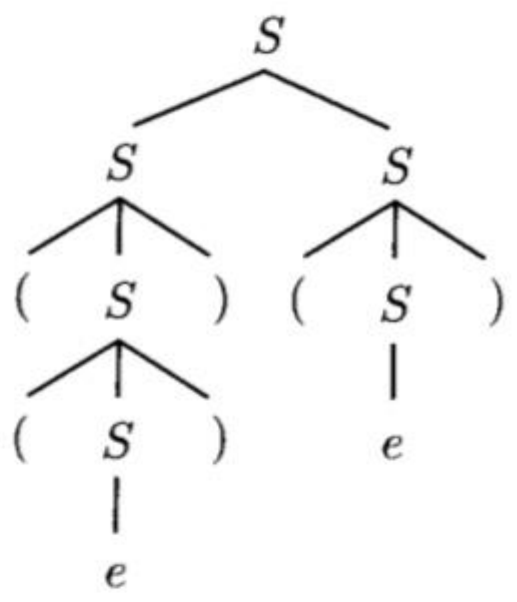
- Нетерминалы обозначаются заглавными буквами латинского алфавита (например: A, B, C)
- Терминалы обозначаются строчными буквами из начала латинского алфавита (например: a, b, c)
- Последовательности из терминалов (слова) обозначают строчными буквами из конца латинского или греческого алфавита (например: ω)
- Последовательности из терминалов и нетерминалов обозначаются строчными буквами греческого алфавита (например: α, β)

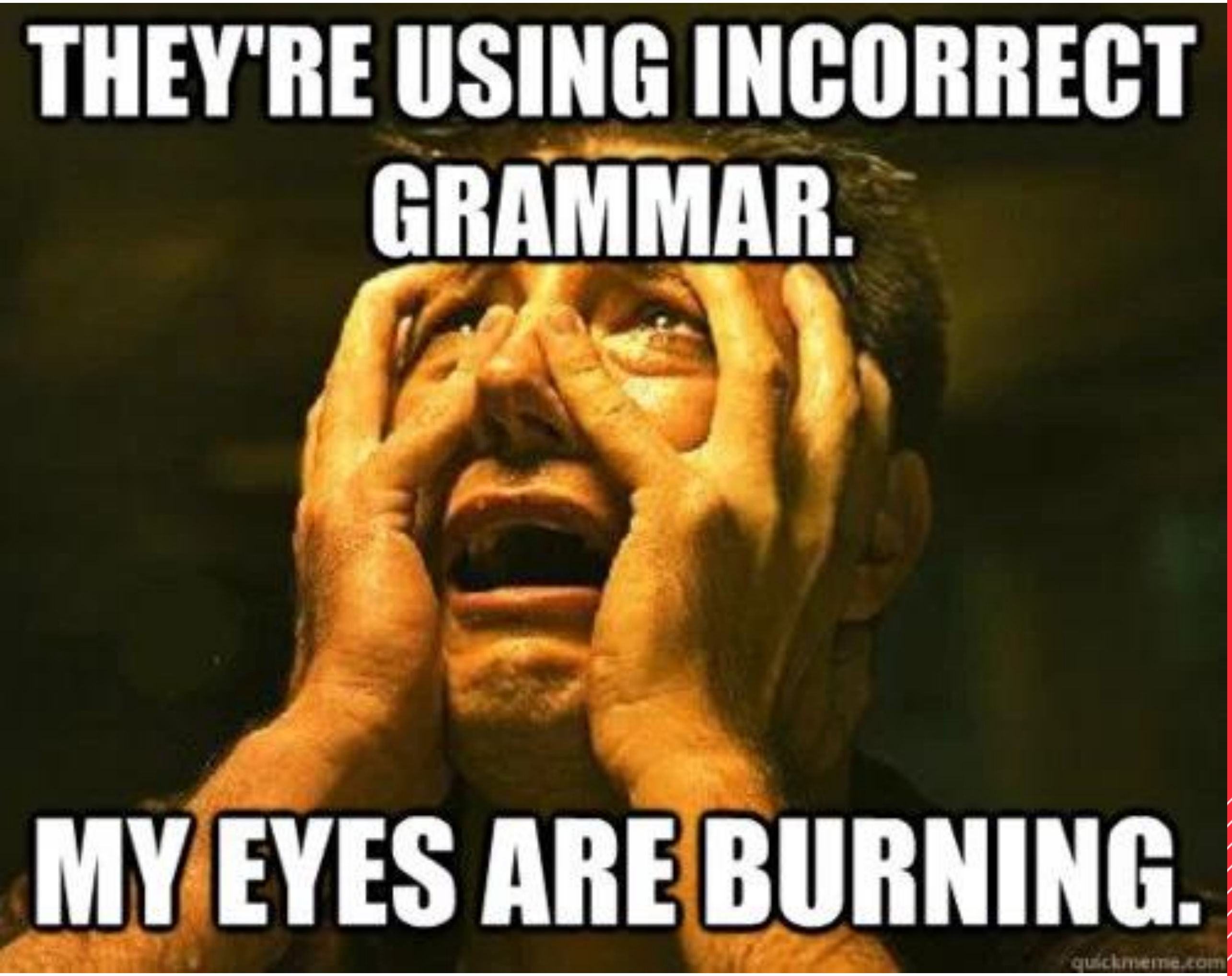
Примеры грамматик

- Язык правильных скобочных последовательностей
  - $\Sigma = \{(,)\}$
  - $S \to \varepsilon|(S)|SS$
- Язык $0^n 1^n 2^n$
  - $\Sigma = \{0,1,2\}$
  - $S \to 012|0TS2$
  - $T0 \to 0T$
  - $T1 \to 11$

Вывод слова в грамматике

- Последовательность примененных правил
- $S \rightarrow SS \rightarrow (S)S \rightarrow ((S))S \rightarrow ((\quad))S \rightarrow ((\quad))(S) \rightarrow (())()$
- Можно визуализировать с помощью дерева разбора





Контекстно-свободные грамматики

Контекстно-свободные грамматики

- Только правила вида $A \to \boldsymbol{\beta}$
  - A – нетерминал
  - $\boldsymbol{\beta}$ – строка из терминалов и нетерминалов
- Пример: правильные скобочные последовательности
  - $S \to \varepsilon|(S)|SS$
- Большинство ЯП – контекстно-свободные

Нормальная форма Хомского

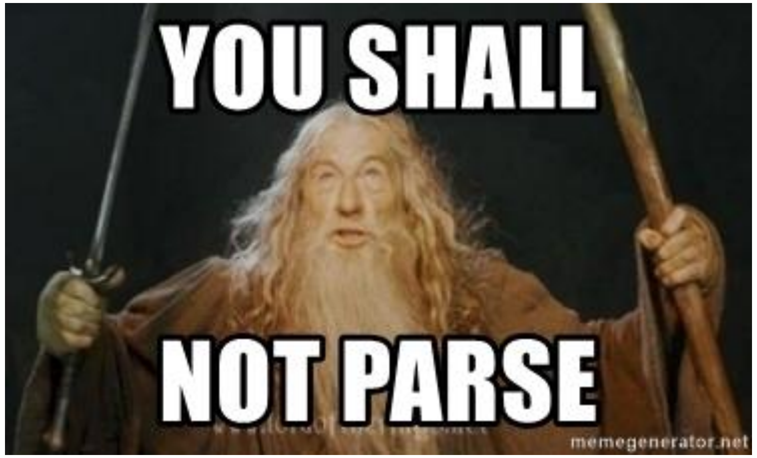
- КС-грамматика с правилами только следующего вида:
  - $A \to BC$
  - $A \to a$
  - $S \to \varepsilon$
- Утверждение:
  - Можно привести любую грамматику в НФХ

Алгоритм Кока-Янгера-Касами (CYK)

- Дана КС-грамматика в НФ Хомского и слово $w \in \Sigma^*$
- Выводится ли это слово в данной грамматике?
- ДП по подотрезкам
  - can[i][j][A] = true, если w[i..j] можно вывести из нетерминала A
- Перебираем место разбиения и правило $A \to \ BC$
- Сложность $O(|w|^3)$
- Есть модификации не для НФХ

Другие алгоритмы

- Алгоритм Эрли
    - $O(|w|^3)$ в худшем случае (константа лучше CYK)
    - $O(|w|^2)$ для однозначных грамматик
        - У каждого слова имеется не более одного дерева разбора в этой грамматике
- Рекурсивный спуск



Рекурсивный спуск

Арифметические выражения

- $Expr \rightarrow Expr + Expr$
- $Expr \rightarrow Expr * Expr$
- $Expr \rightarrow (Expr) | Number | Variable | \ldots$
- Проблемы
  - Неоднозначность построения дерева разбора
  - И как следствие нарушение семантики (в данном случае порядок вычисления)

Арифметические выражения

- $Expr \rightarrow Sum$
- $Sum \rightarrow Sum + Product \mid Product$
- $Product \rightarrow Product * Term \mid Term$
- $Term \rightarrow (Expr) \mid Number \mid Variable \mid \ldots$

Рекурсивный спуск

- Для каждого нетерминала X создадим функцию parseX()
- Перебираем, по какому правилу раскрыть X, запускаемся рекурсивно для этого правила
- Проблемы
  - Можно не угадать правило
  - Можно бесконечно зациклиться

Левая рекурсия

- Правила вида $Sum \to Sum + Product \mid Product$
- $parseSum(\quad) \to parseSum(\quad) \to parseSum(\quad) \to \ldots$
- Решения
    - Переписать правило как $Sum \to Product + Sum$
    - Или так
        - $Sum \to Product\ Sum'$
        - $Sum' \to +Sum \mid \varepsilon$
- Важно, меняет ассоциативность!

Левая рекурсия

- Как вычислять значения?
- 1) Через аккумулятор, аргументом parseSum’(acc)
- 2) Перестроить дерево разбора

- Для каждой грамматики можно устранить левую рекурсию (есть алгоритм)

LL(1) и LL(k) грамматики

LL(1) и LL(k) грамматики

- LL(1) грамматика – такая, что, какое правило применять, можно однозначно понять по первому нерассмотренному *токену*
- LL(k) – аналогично, по первым k *токенам*
- Легко распарить рекурсивно
- Почти все ЯП – LL(1)

Лирическое отступление

- Лексер получает из строки набор токенов
- Парсер строит дерево разбора (или что понадобится) по набору токенов

LL(1) и LL(k) грамматики

- Если грамматика LL(1)
- Тогда в функции parseA() по первому токену можно однозначно понять правило и запуститься рекурсивно
- Парсинг будет работать за $O(n)$

А что в индустрии?

BNF

- BNF/EBNF ([extended] Backus–Naur form) – попытка стандартизировать представление грамматик
- Большинство «индустриальных» грамматик
- ISO/IEC 14977

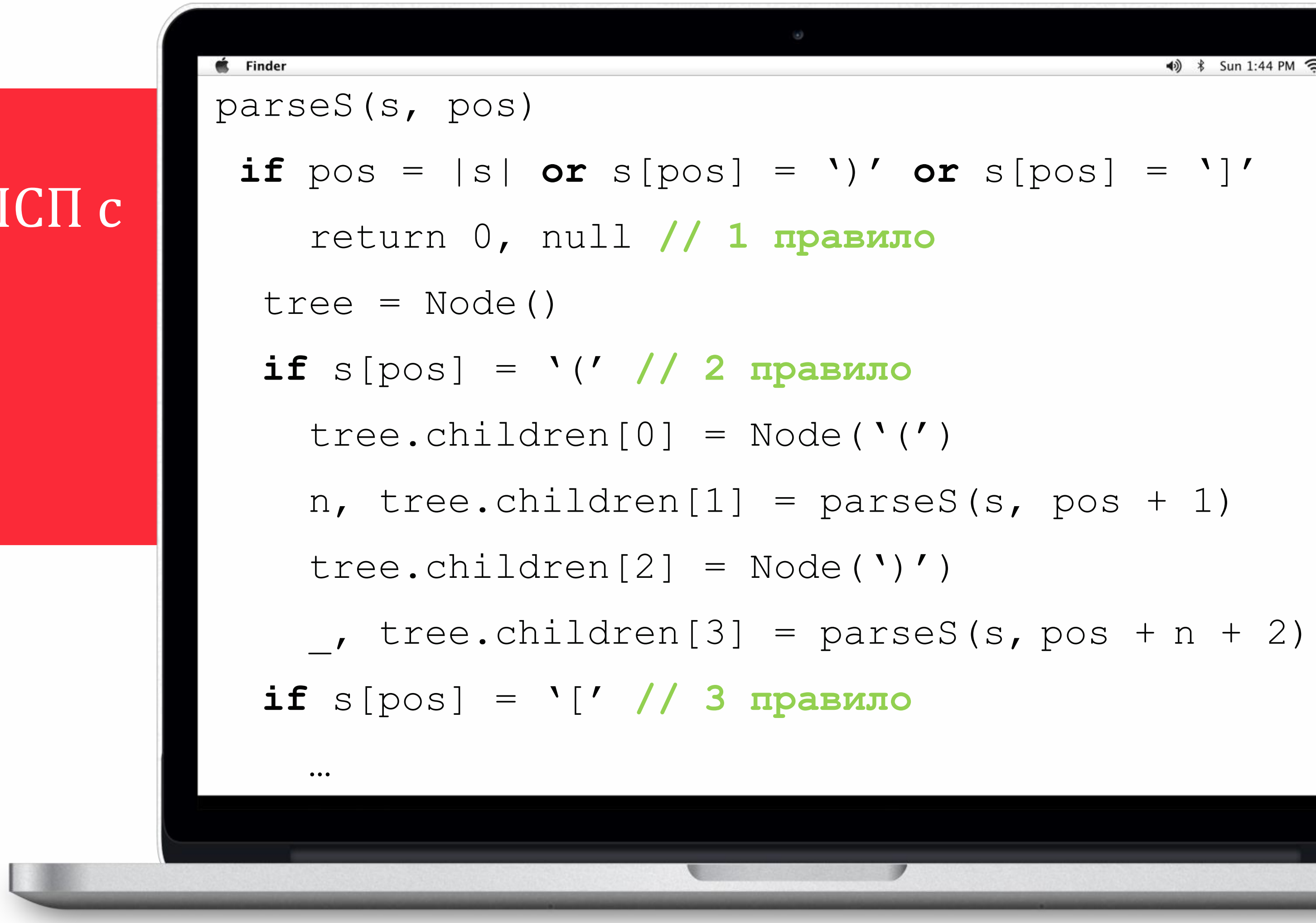
Генераторы парсеров

- Реализовывать с нуля анализ грамматик и разбор для сколько-нибудь сложных грамматик – плохая идея
- Генераторы парсеров
    - ANTLR – для Java
    - Bison – C/C++/Java
    - и десятки других

Пример парсера

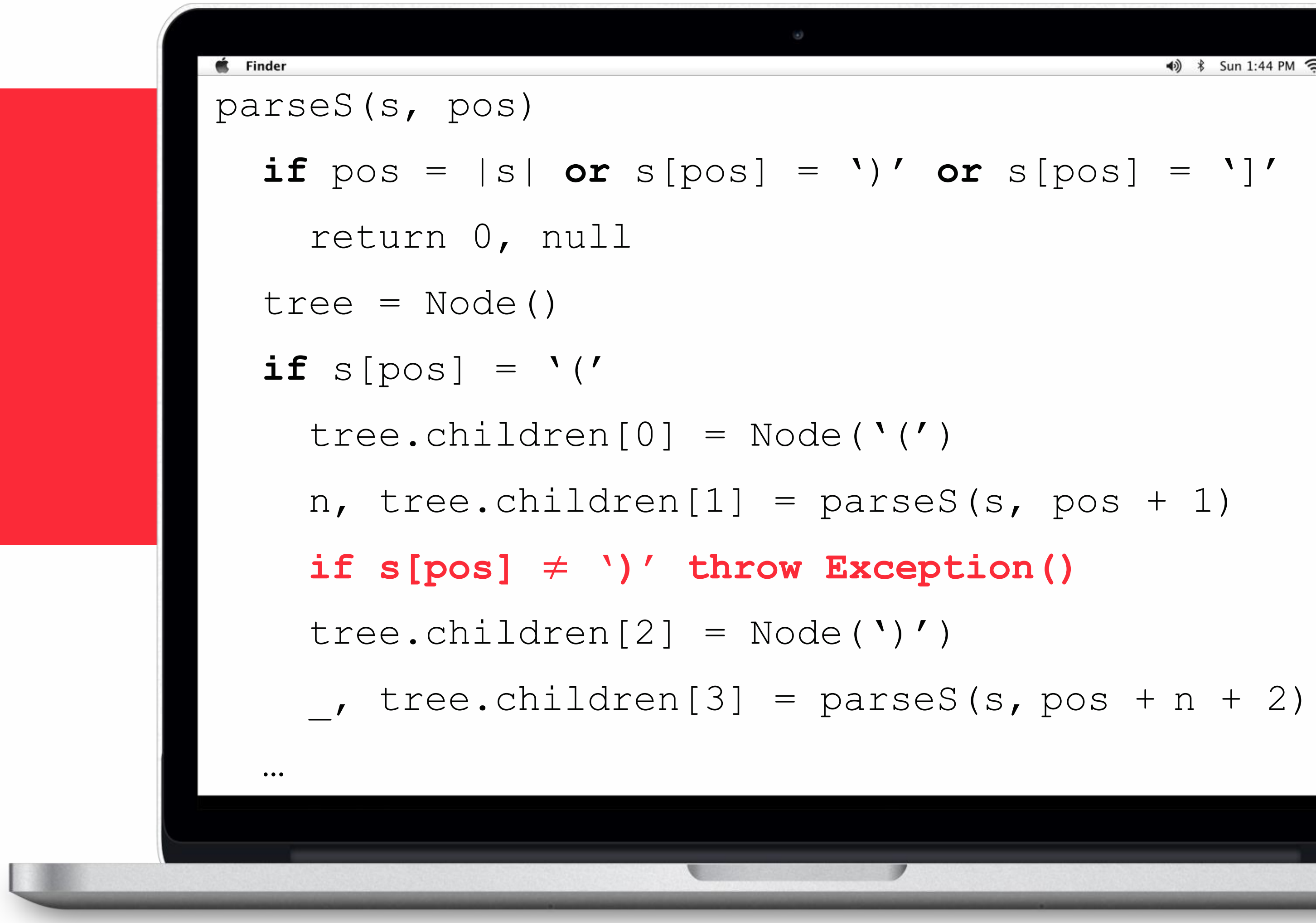
Парсер для ПСП

- LL(1) грамматика для ПСП с двумя типами скобок
- $\Sigma = \{(,), [,]\}$
- $S \to \varepsilon \mid (S)S \mid [S]S$
- Явного лексера нет



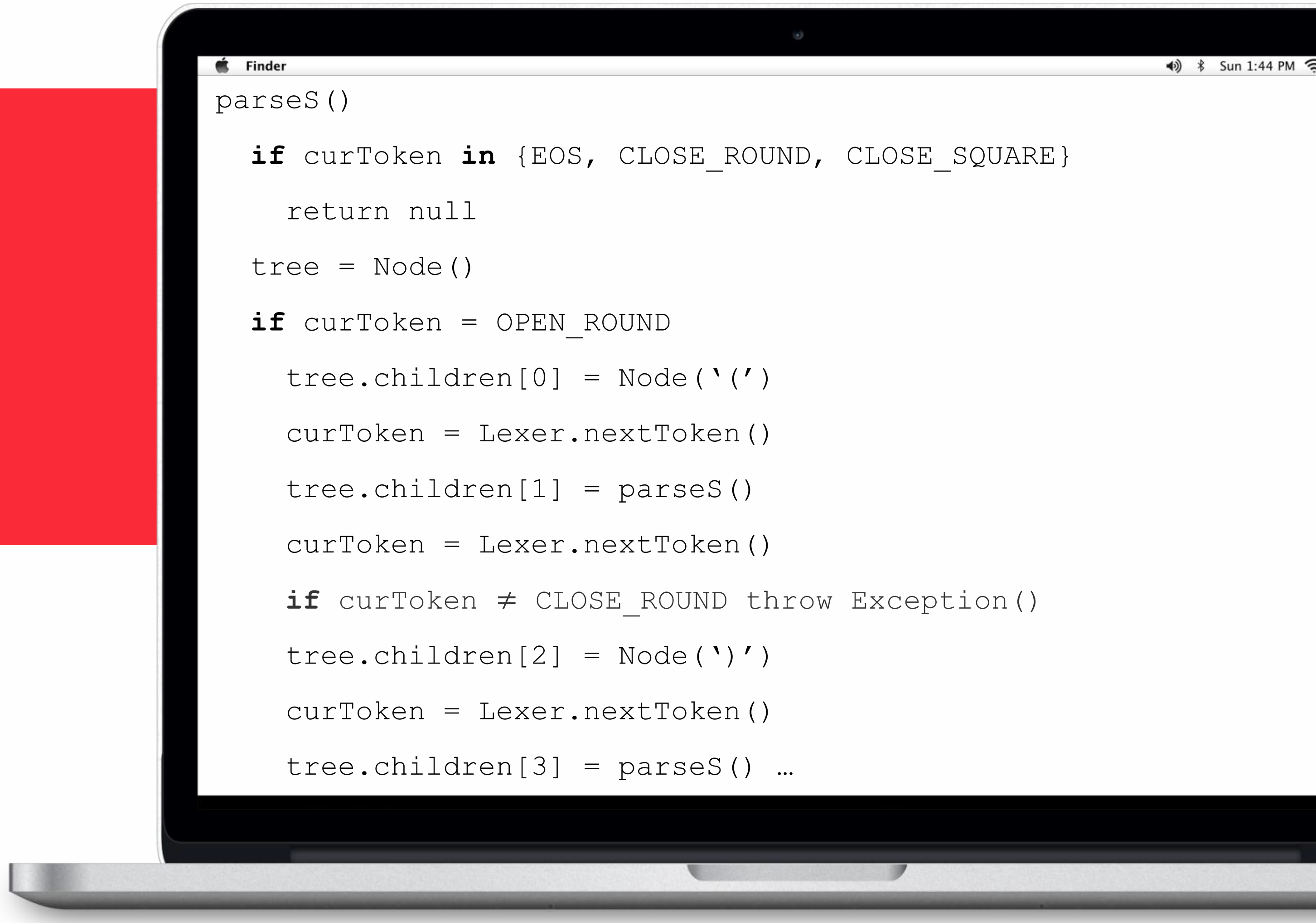
Парсер для ПСП

- $S \to \varepsilon \mid (S)S \mid [S]S$
- Обработка ошибок



Парсер для ПСП

- $S \to \varepsilon \mid (S)S \mid [S]S$
- Если есть лексер



Все!