TypeFox

A Deep Dive into Lookahead Algorithms

Mark Sujew

@mark_sujew @msujew

Introductions

Mark Sujew



- Software Engineer
- Mainly working on Open-Source Software
- Just finished my Master's degree

TypeFox

- Software Consulting Company
- Specialized in building DSLs and IDEs

Lookahead

- Background
- Top-down lookahead (LL – **left**-to-right, **left**-most derivative)
  - $LL(1), LL(k), LL(*), Adaptive\ LL(*)$
  - Packrat Parsers
- Empirical Analysis

Formal Languages

- Building blocks
  - Alphabet: $\Sigma = \{\varepsilon, a, b, c\}$
  - Words: ab, bc, cba, ε
  - ε is really important!
- Languages are a set of **words**

$$L_1 = \{ab, ac, abc\}$$

$$L_2 = \{a, aa, aaa, \dots, a^n\}$$

$$L_3 = \{\varepsilon\}$$

Formal Grammars

- Formal grammars generate languages
- Parsers can deduce grammar rules from words
  - Regular Languages (Type 3) – RegExp, Deterministic finite Automata
  - Context-free Languages (Type 2) – ANTLR, push-down automata
  - Other languages – Humans, Machine Learning

Formal Grammars

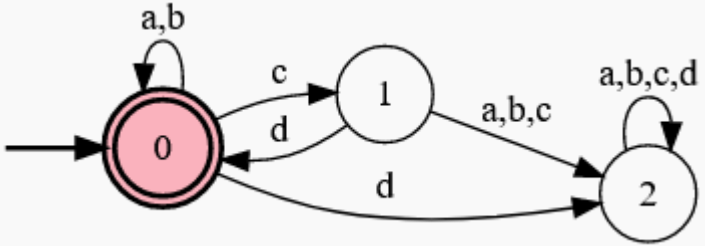
- Building blocks
  - Nonterminals N (Grammar rules)
  - Terminals T (Keywords and terminal rules)
  - Start symbol S (entry rule)
  - Productions $N \cup T$

Formal Grammars

- Rules are written as $A \rightarrow \alpha_1, A \rightarrow \alpha_2$
- Rules can be grouped as $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_p$
- Extended Backus-Naur-Form (EBNF) allows to replace:
  - $P \rightarrow \alpha \mid \varepsilon$ with $P \rightarrow \alpha?$
  - $P \rightarrow \alpha \mid \alpha P$ with $P \rightarrow \alpha^+$
  - $P \rightarrow \alpha \mid \alpha P \mid \varepsilon$ with $P \rightarrow \alpha^*$

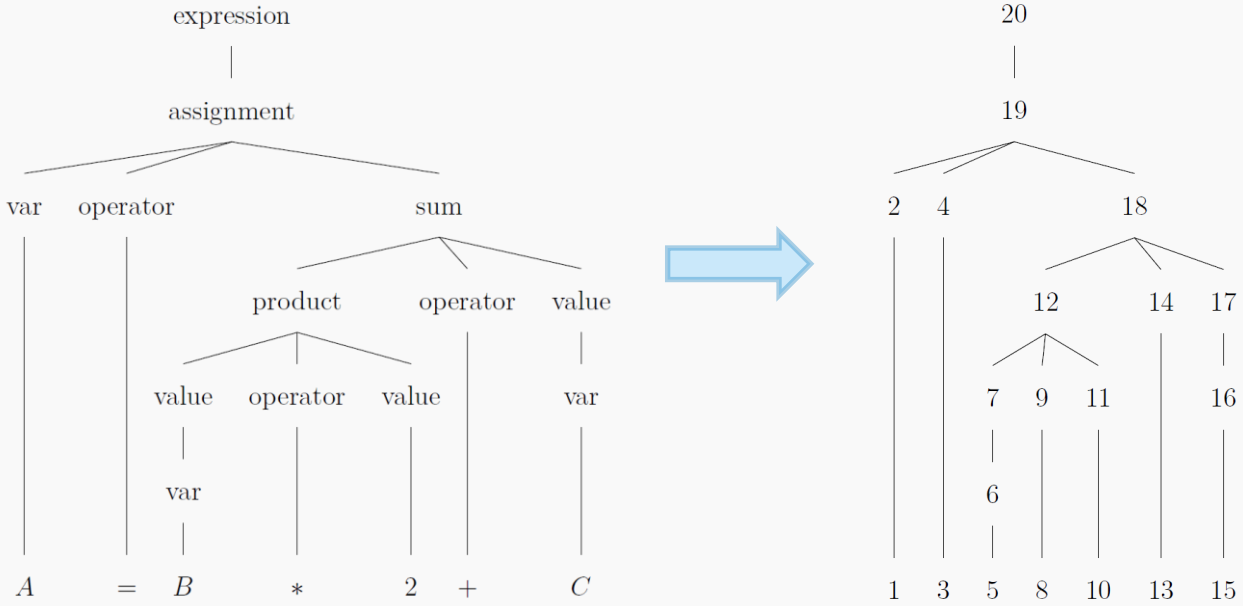
Excursion: Type 3 Automata/RegExp

Production: $P \rightarrow (a^+b^+cd)^*$

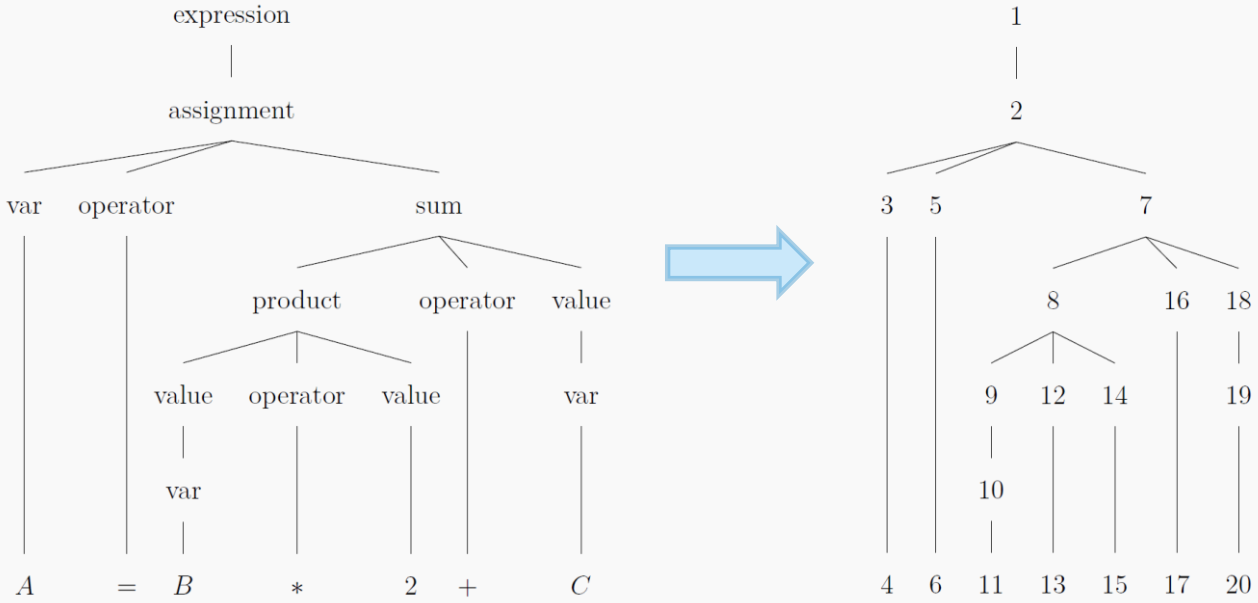


Regex: `^(a+b+cd)*$`

Bottom-Up Parsing



Top-Down Parsing



Top-Down Lookahead

- We need to “decide” on a production/alternative
- We need to do this ***fast***
  - Without parsing the full production
  - Using as few tokens as possible

Top-Down Lookahead: $LL(1)$

- Simple map, storing “jump” instructions
- Generated by identifying the first terminal in each production

LL(1) Grammar:

$$A \rightarrow B$$
$$A \rightarrow aA$$
$$B \rightarrow b$$



LL(1) Lookup Table:

	a	b
A	2	1
B	—	3

Top-Down Lookahead: $LL(k)$

- Conceptually similar to $LL(1)$
- Generated by identifying the first k terminals in each production
  - Single terminal → Two dimensional array

$$A \rightarrow B \mid C$$
$$\mathrm{B} \rightarrow abA$$
$$\mathrm{C} \rightarrow abc$$



$$1 \rightarrow [\,[\,a, b, a\,], [\,a, b, c\,]\,]$$
$$2 \rightarrow [\,[\,a, b, a\,]\,]$$
$$3 \rightarrow [\,[\,a, b, c\,]\,]$$

Top-Down Lookahead: $LL(k)$

- Issues with $LL(k)$
  - **No rule recursion** in lookahead
  - **No repetition productions** in lookahead
  - k might not be enough, even without recursion or repetitions
  - Initialization is $O(n^k)$ in worst case, n = amount of productions

Packrat Parsing

- Conceptually similar to other top-down parsers
- Uses **memoization** instead of pre-calculated lookahead
- Can be realized using:
  - Subparser forking
  - Backtracking

Top-Down Lookahead: $LL(*)$

- ANTLR3 introduced the first $LL(*)$ lookahead algorithm
- During generator runtime:
  - Create DFA from production
  - Encode DFA in code
- **Allows repetitions**
- Still, **no rule recursion**

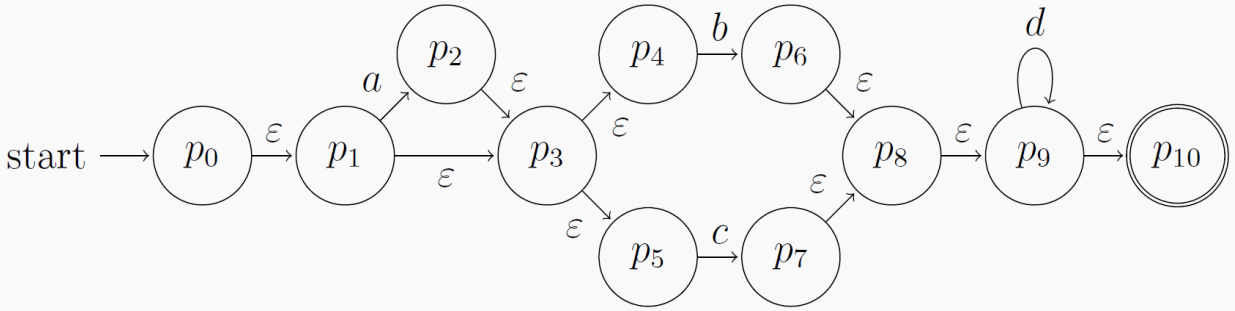
Top-Down Lookahead: Adaptive $LL(*)$

- Lookahead algorithm for ANTLR4
- **Allows repetitions**
- **Allows rule recursion**
- **Enables direct left recursion**

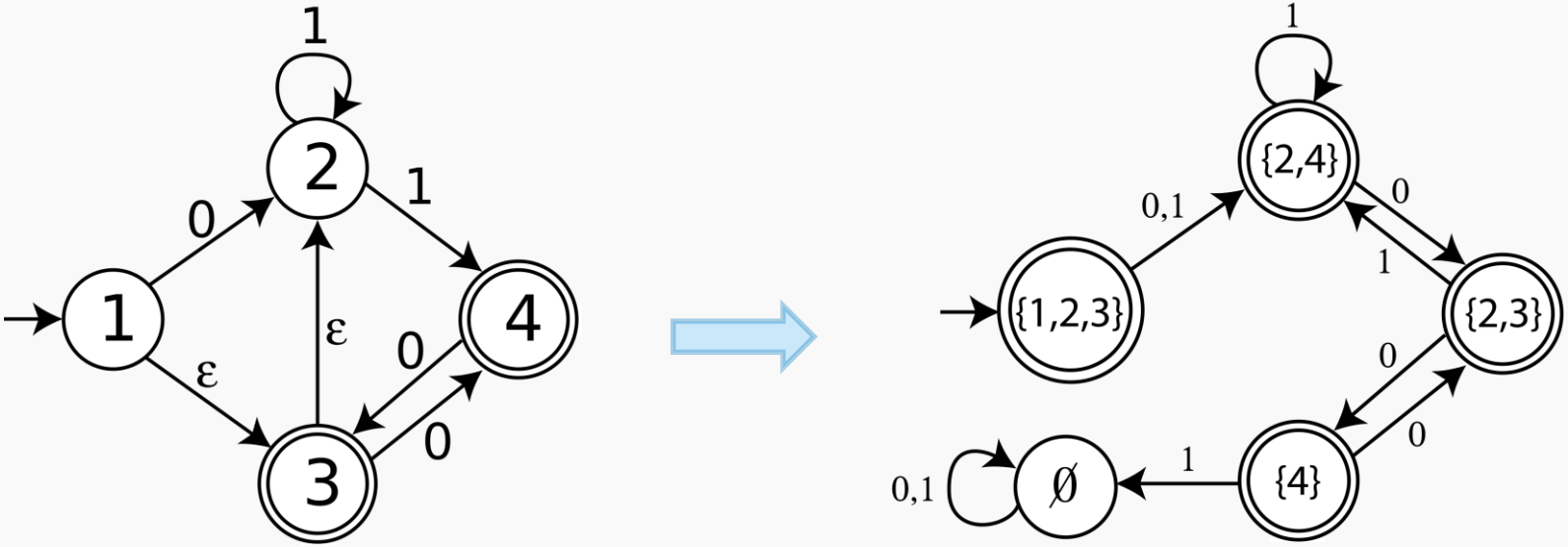
Top-Down Lookahead: Adaptive $LL(*)$

Static analysis creates an Augmented Transition Networks (ATN)

Production: $P \rightarrow a?\,(b|c)d^*$



Excursion: Subset Construction



Top-Down Lookahead: Adaptive $LL(*)$

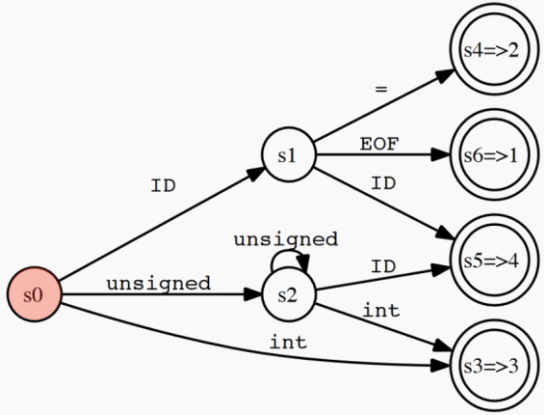
- Dynamic analysis simulates ATN to construct DFA
- When “unknown” token type is encountered:
  - Perform “subset-construction” algorithm on ATN
  - Generate new DFA node
  - Memoize new lookahead path
  - Create accepting state when applicable (unique alternative)

Top-Down Lookahead: Adaptive $LL(*)$

Dynamic analysis runs through an DFA for each decision

Input: ε

```
s : ID
  | ID '=' expr
  | 'unsigned'* 'int' ID
  | 'unsigned'* ID ID
  ;
```

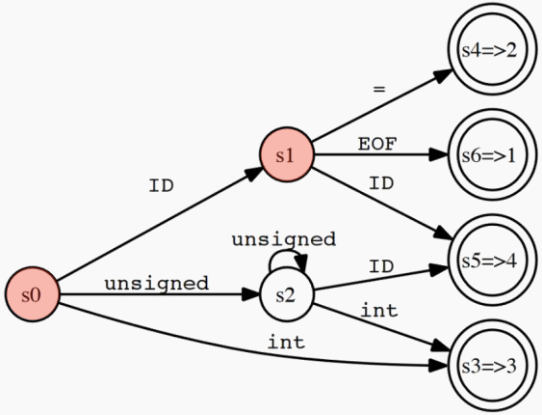


Top-Down Lookahead: Adaptive $LL(*)$

Dynamic analysis runs through an DFA for each decision

Input: `ID`

```
s : ID
  | ID '=' expr
  | 'unsigned'* 'int' ID
  | 'unsigned'* ID ID
  ;
```

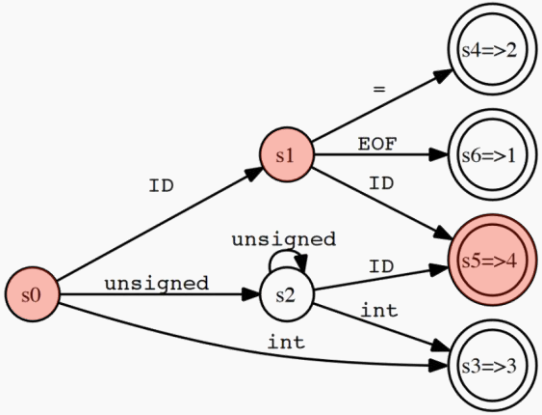


Top-Down Lookahead: Adaptive $LL(*)$

Dynamic analysis runs through an DFA for each decision

Input: `ID ID`

```
s : ID
  | ID '=' expr
  | 'unsigned'* 'int' ID
  | 'unsigned'* ID ID
  ;
```

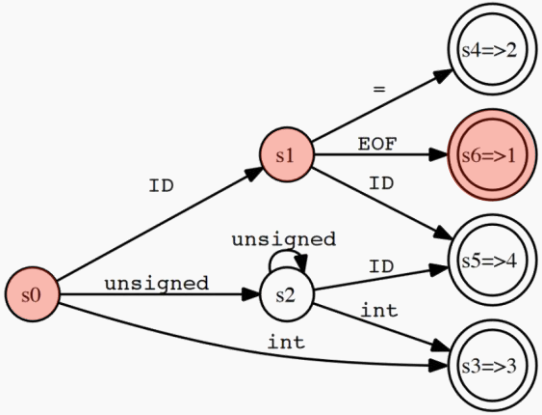


Top-Down Lookahead: Adaptive $LL(*)$

Dynamic analysis runs through an DFA for each decision

Input: `ID EOF`

```
s : ID
  | ID '=' expr
  | 'unsigned'* 'int' ID
  | 'unsigned'* ID ID
  ;
```

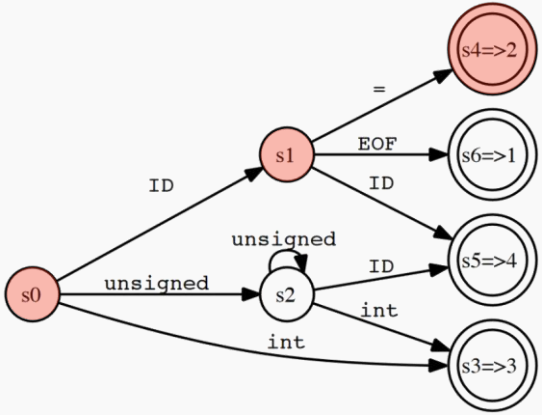


Top-Down Lookahead: Adaptive $LL(*)$

Dynamic analysis runs through an DFA for each decision

Input: `ID '='`

```
s : ID
  | ID '=' expr
  | 'unsigned'* 'int' ID
  | 'unsigned'* ID ID
  ;
```



Top-Down Lookahead: Adaptive $LL(*)$

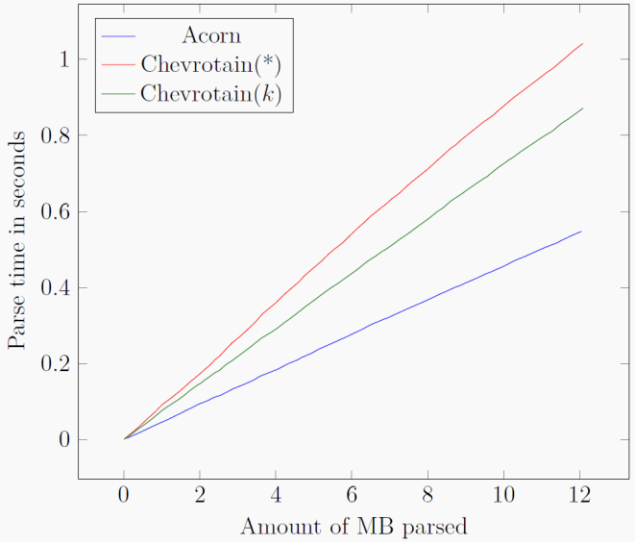
- Deals with rule recursion by “jumping” to the next ATN
- Deals with direct left recursion by cleverly restructuring the grammar
- Memoization makes this approach really fast
    - Each DFA node only needs to be computed once

Empirical Analysis

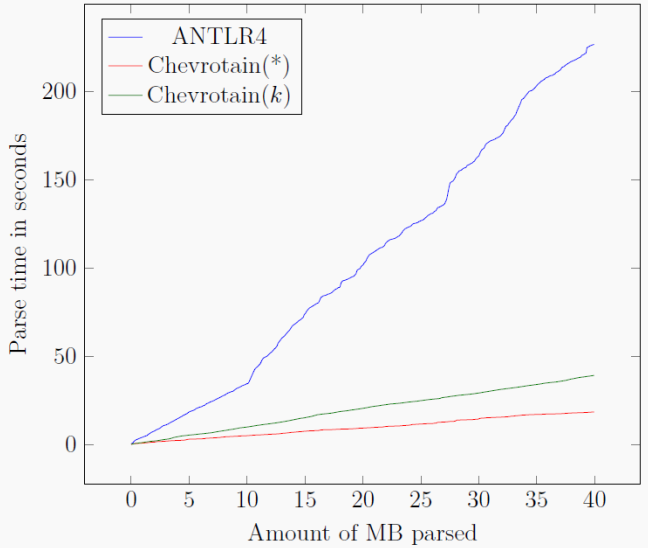
	Chevrotain(*)	Chevrotain(k)	ANTLR4	Acorn
JSON	9184	9527	3175	—
CSS	3092	3127	8.22	—
ECMA5	471	496	4.65	621
Java	20.05	2.03	1.22	—

Parsing performance in Operations/Second

Empirical Analysis

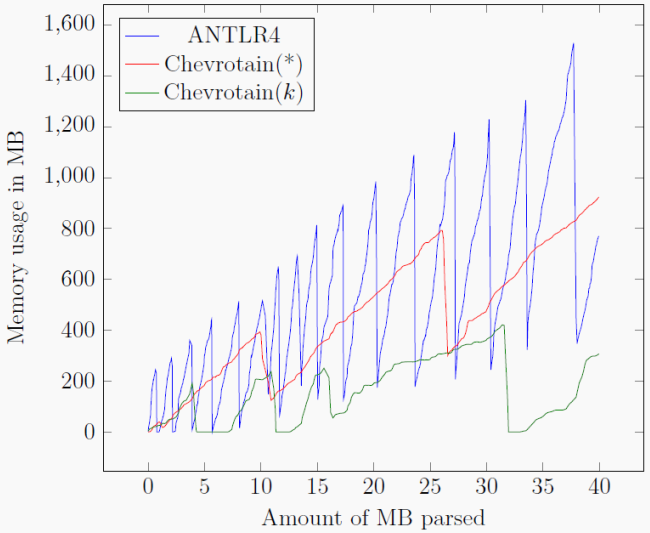


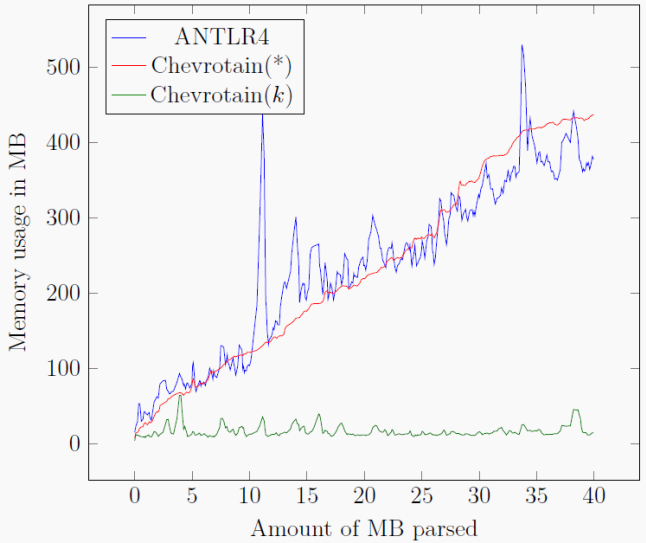
Parsing JavaScript code



Parsing Java code

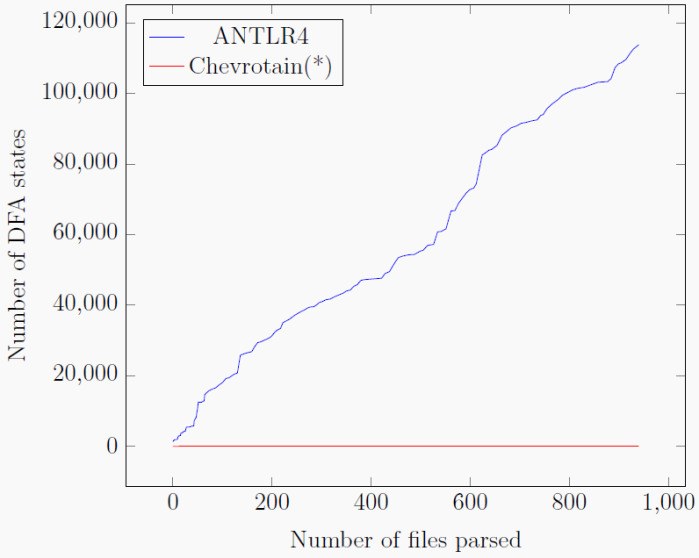
Empirical Analysis



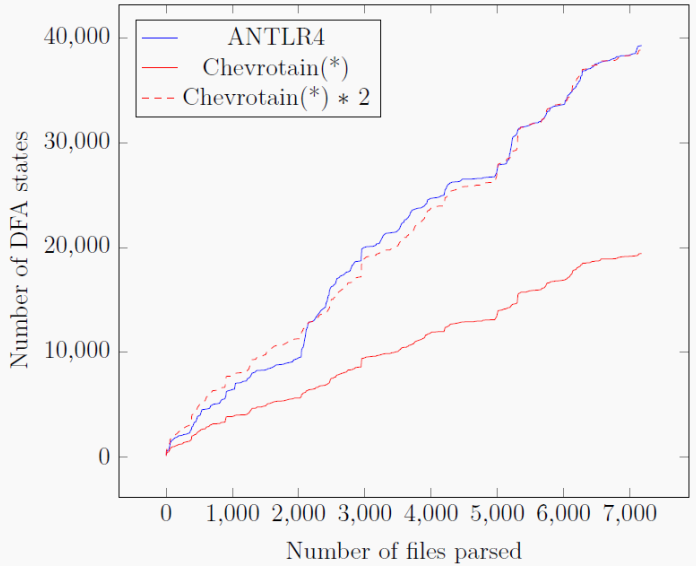


Memory consumption parsing Java Code (Without/With manual GC)

Empirical Analysis



Parsing JavaScript code



Parsing Java code

Conclusion

- Implementing ALL(*) allows for unbounded lookahead
- It has a slightly negative impact on parsing performance in general
- It removes the necessity for most backtracking in parsers
- Parses in $O(n)$ time, with a theoretical complexity of $O(n^4)$

Discussion