コンピュータシステムの理論と実装

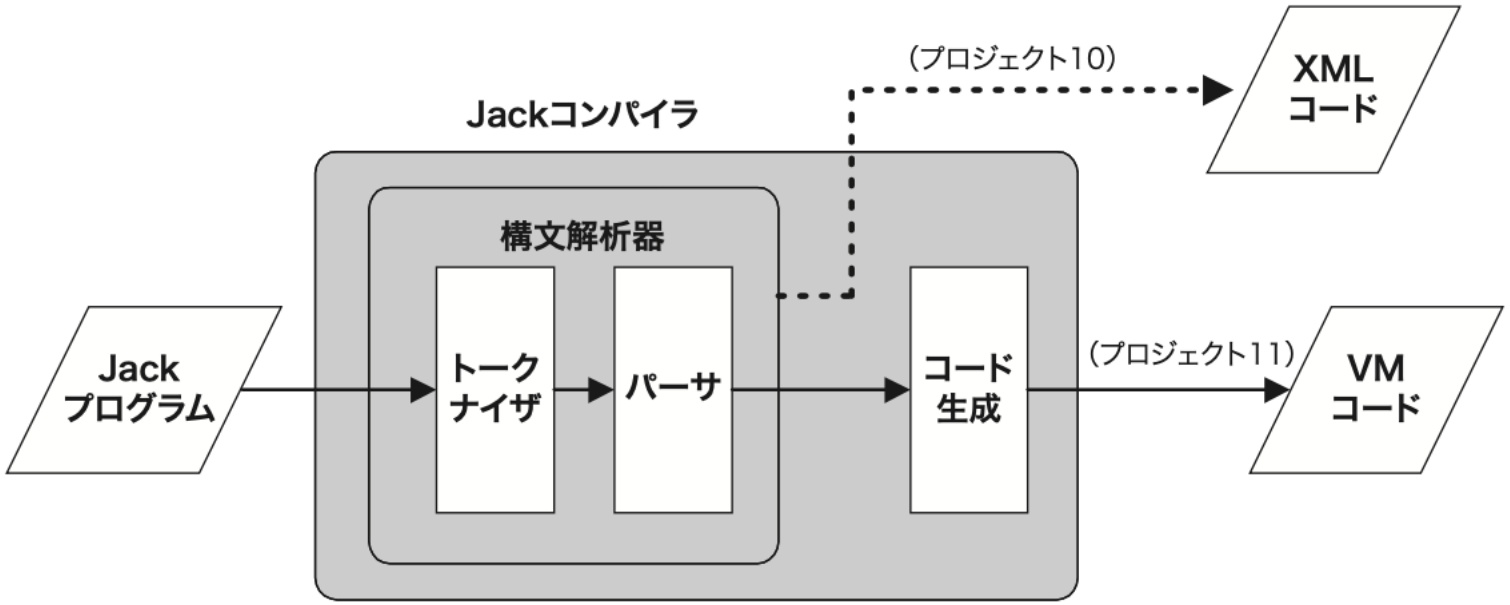
10章 コンパイラ #1: 構文解析

アジェンダ

- コンパイラについて
- パーサについて
- 課題

コンパイラ

- 構文解析
  - トークナイザ
  - パーサ
- コード生成



トークナイザ (復習)

- プログラムの文字をトークンとしてまとめる
- トークン
  - 意味のある最小の単位

```
class Main {
  function void main() {
    var Array a;
    var int length;
    var int i, sum;

    (略)
```

```
<tokens>
<keyword> class </keyword>
<identifier> Main </identifier>
<symbol> { </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> main </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> var </keyword>
<identifier> Array </identifier>
<identifier> a </identifier>
<symbol> ; </symbol>
<keyword> var </keyword>
<keyword> int </keyword>
<identifier> length </identifier>
<symbol> ; </symbol>
略
```

パーサ

- トークンをプログラミング言語の文法に従い言語構造でグループ化する

```
<tokens>
<keyword> class </keyword>
<identifier> Main </identifier>
<symbol> { </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> main </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> var </keyword>
<identifier> Array </identifier>
<identifier> a </identifier>
<symbol> ; </symbol>
<keyword> var </keyword>
<keyword> int </keyword>
<identifier> length </identifier>
<symbol> ; </symbol>
略
```

```
<class>
  <keyword> class </keyword>
  <identifier> Main </identifier>
  <symbol> { </symbol>
  <subroutineDec>
    <keyword> function </keyword>
    <keyword> void </keyword>
    <identifier> main </identifier>
    <symbol> ( </symbol>
    <parameterList>
    </parameterList>
    <symbol> ) </symbol>
    <subroutineBody>
      <symbol> { </symbol>
      <varDec>
        <keyword> var </keyword>
        <identifier> Array </identifier>
        <identifier> a </identifier>
        <symbol> ; </symbol>
      </varDec>
```

文法

- P232 - プログラム構造

keyword	**'class'** | **'constructor'** | **'function'** | **'method'** | **'field'** | **'static'** | **'var'** | **'int'** | **'char'** | **'boolean'** | **'void'** | **'true'** | **'false'** | **'null'** | **'this'** | **'let'** | **'do'** | **'if'** | **'else'** | **'while'** | **'return'**
symbol	**'{'** | **'}'** | **'('** | **')'** | **'['** | **']'** | **'.'** | **','** | **';'** | **'+'** | **'-'** | **'*'** | **'/'** | **'&'** | **'|'** | **'<'** | **'>'** | **'='** | **'~'**
integerConstant	0 から 32767 までの 10 進数の数字
stringConstant	ダブルクォートと改行文字を含まないユニコードの文字列
identifier	アルファベット、数字、アンダースコア（_）の文字列。ただし数字から始まる文字列は除く

class	**'class'** className **'{'** classVarDec* subroutineDec* **'}'**
classVarDec	(**'static'** | **'field'**) type varName (**','** varName)* **';'**
type	**'int'** | **'char'** | **'boolean'** | className
subroutineDec	(**'constructor'** | **'function'** | **'method'**) (**'void'** | type) subroutineName **'('** parameterList **')'** subroutineBody
parameterList	((type varName) (**','** type varName)*)?
subroutineBody	**'{'** varDec* statements **'}'**
varDec	**'var'** type varName (**','** varName)* **';'**
className	identifier
subroutineName	identifier
varName	identifier

文法

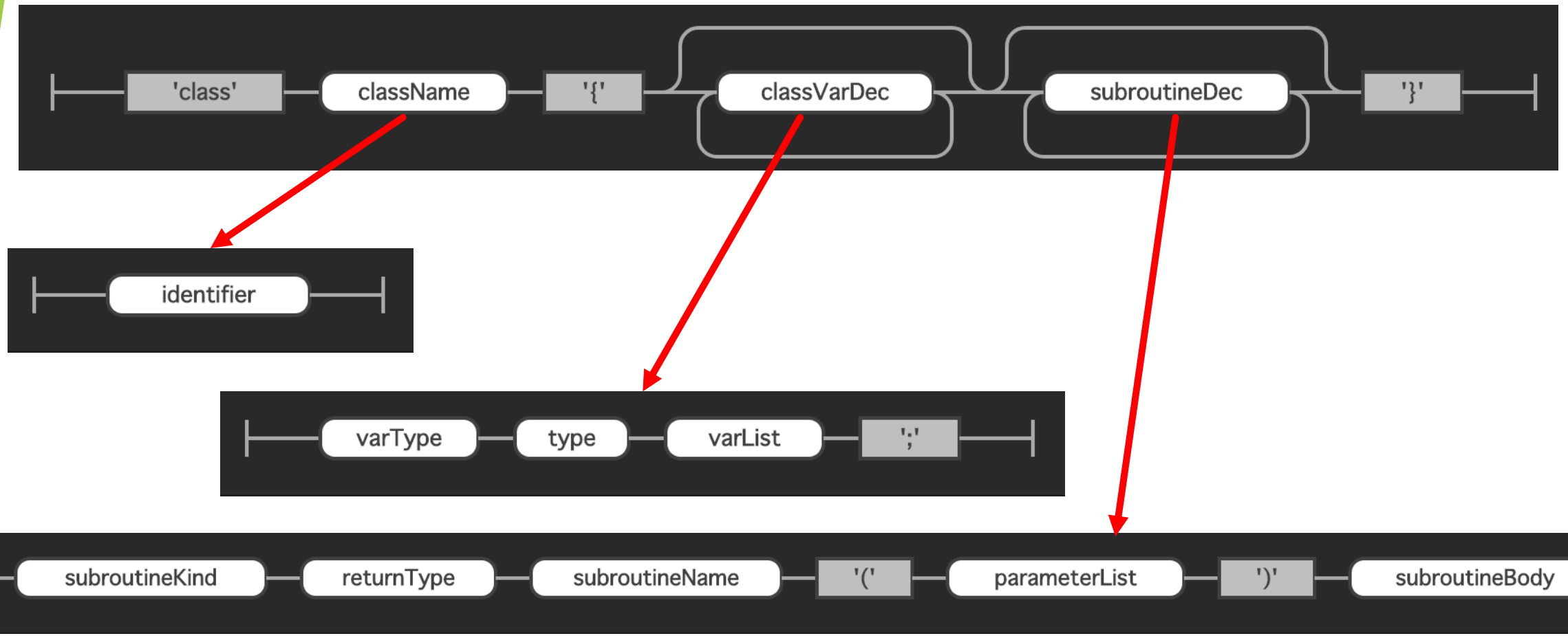
- P232 - プログラム構造

statements	statement*
statement	letStatement | ifStatement | whileStatement | doStatement | returnStatement
letStatement	'let' varName ('[' expression ']')? '=' expression ';'
ifStatement	'if' '(' expression ')' '{' statements '}' ('else' '{' statements '}')?
whileStatement	'while' '(' expression ')' '{' statements '}'
doStatement	'do' subroutineCall ';'
returnStatement	'return' expression? ';'

expression	term (op term)*
term	integerConstant | stringConstant | keywordConstant | varName | varName '[' expression ']' | subroutineCall | '(' expression ')' | unaryOp term
subroutineCall	subroutineName '(' expressionList ')' | (className | varName) '.' subroutineName '(' expressionList ')'
expressionList	(expression (',' expression)*)?
op	'+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '='
unaryOp	'-' | '~'
KeywordConstant	'true' | 'false' | 'null' | 'this'

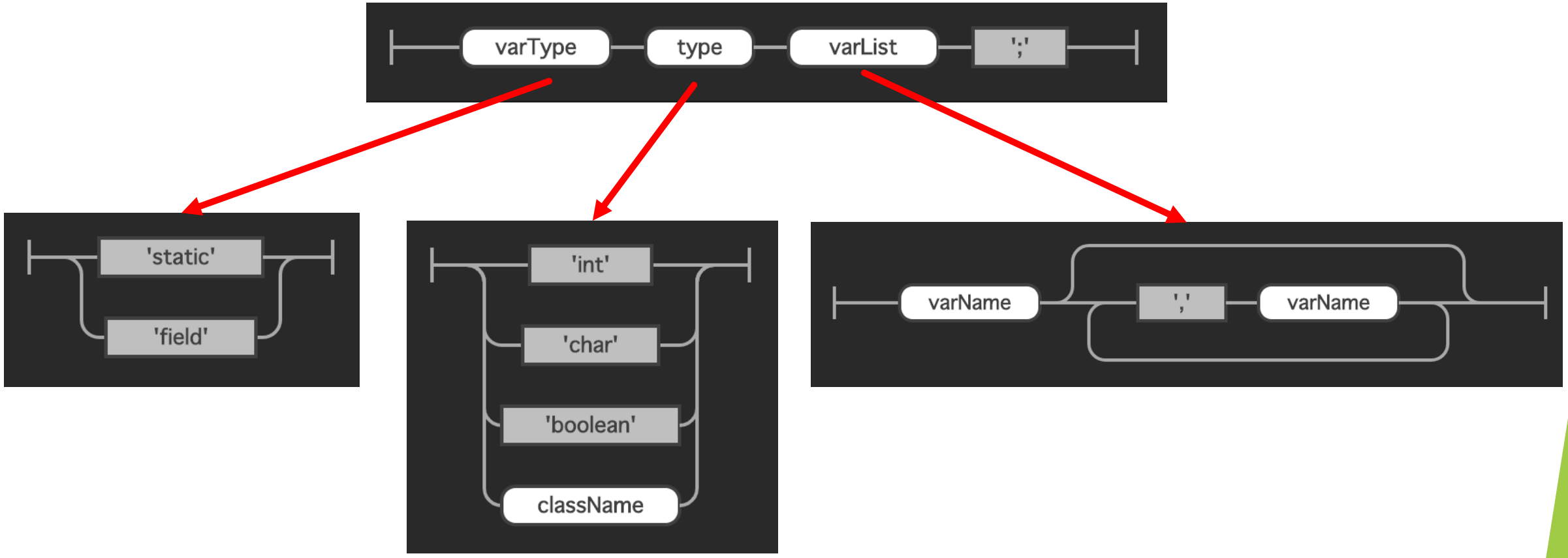
文法

- 例: class


'class'
className
'{'
classVarDec
subroutineDec
'}'
identifier
varType
type
varList
';'
subroutineKind
returnType
subroutineName
'('
parameterList
')'
subroutineBody

文法

- 例: class


varType
type
varList
';'
'static'
'field'
'int'
'char'
'boolean'
className
varName
','
varName

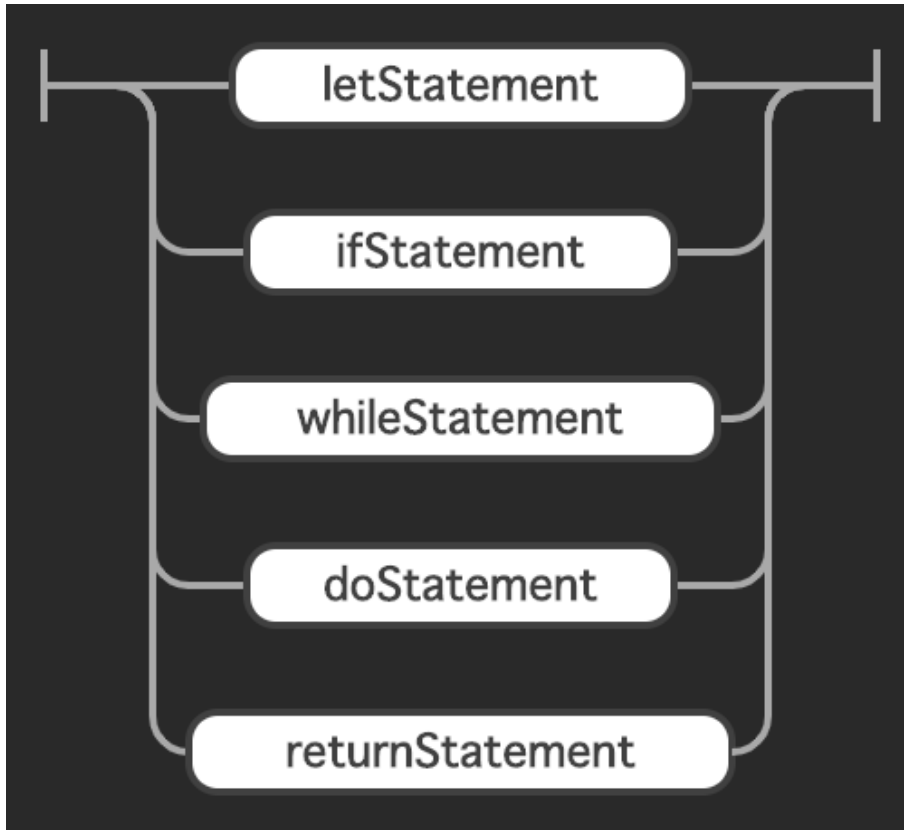
文法

- 文法に従い、終端要素になるまで (これ以上潜れなくなるまで) 辿って行き、終端要素を出力する。
- 全ての要素が終端要素に到達するまで再帰的に解析すれば良い。

- ただし、入力によってパターンが変わるので、先読みして決定しないといけない。
  - statement や、引数を含む含まない…など

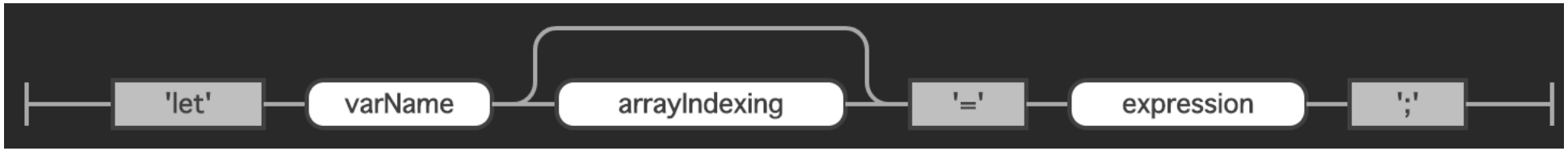
文法

- 例: statement
  - let statement
  - if statement
  - while statement
  - do statement
  - return statement

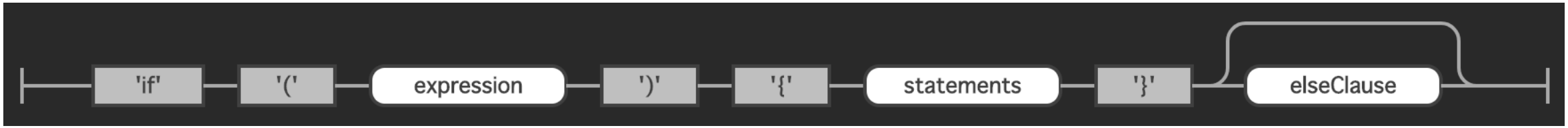


文法

- 例: let statement
  - キーワードとして let があったら…



- 例: if statement
  - キーワードとして if があったら…



課題

- Jack プログラムをトークナイズし、xxxT.xml を作成する (前回)
- xxxT.xml をパースし、xxx.xml を作成する

- ArrayTest, ExpressionLessSquare, Square の順がよいです。

Antlr (展望のやつ

- ANTLR は、構造化テキストまたはバイナリファイルの読み取り、処理、実行、または翻訳のための強力なパーサージェネレーターです。
  言語、ツール、およびフレームワークの構築に広く使用されています。ANTLRは文法から、構文解析ツリーを構築およびウォークできるパーサーを生成します。

- EBNFに似た記法で文法を記述します
  https://github.com/lulichn/nand2tetris/blob/master/src/compiler-java/src/main/antlr/net/lulichn/n2t/compiler/Jack.g4

- これだけでASTが取得できる。あとはすきなように。

```
var lexer = new JackLexer(CharStreams.fromStream(is));
var stream = new CommonTokenStream(lexer);
var parser = new JackParser(stream);
```

参考実装

- Java Antlr版
  https://github.com/lulichn/nand2tetris/tree/master/src/compiler-java

- Rust 汚いやつ
  https://github.com/lulichn/nand2tetris/tree/master/src/Compiler