\Chapter{Szintaktikai elemzés}

% TODO: Formális nyelvekkel, fordítóprogramokkal kapcsolatos könyvek hivatkozásai.

% http://www.informatik.uni-bremen.de/agbkb/lehre/ccfl/Material/ALSUdragonbook.pdf

Ennek a célja az, hogy megállapítsa, hogy érvényes programról van-e szó, illetve hogy majd egyszerűen fa struktúrába rendezhető legyen a program. (AST és CST problémaköre)

Ide kerül a szintaktikai elemzés elemzés elterjedt módjainak a bemutatása. Elvi szinten

\subsection{Példa az összehasonlításhoz}

Az alábbiakban a programozási nyelv nyelvtanának felírása történik meg Extended Backus-Naur From és szintaxis diagram segítségével.

Space ::= " ";

Tabulator ::= "\t";

LineBreak ::= "\n";

Operator ::= "+" | "-" | "\*" | "/";

Whitespace ::= " " | "\n" | "\t" | "\r" | "\r\n";

Digit ::= "0" | non\_zero\_digit;

NonZeroDigit ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

Number ::= [ "-" ], non\_zero\_digit, {digit};

Character ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z";

String ::= '"', ({Character – '"'} | Space), '"';

Identifier ::= Character, {Character | Digit};

Assignment ::= Identifier, "=", (Number | Identifier | String);

Function ::= 'funct', Space, Identifier, "(", {Identifier} ")", LineBreak,

{Tabulator, Assignment, LineBreak},

{Tabulator, Identifier, Operator, Identifier, LineBreak};

Class ::= 'Create', Space, Identifier, "(", {Space, Identifier}, ")", LineBreak,

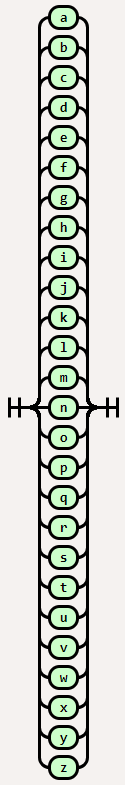
{Tabulator, Assignment, LineBreak},

{Tabulator, Function, LineBreak};

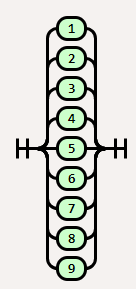
Program ::= {class};

A szintaxis diagrammal történő felírás

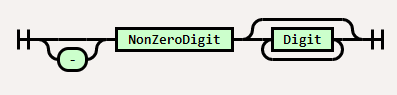
Egy Character leírása



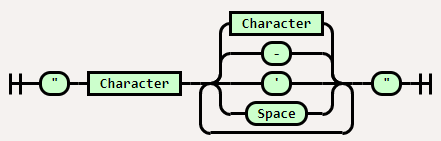
NonZeroDigit



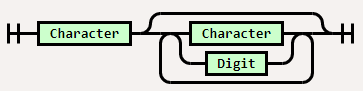
Number



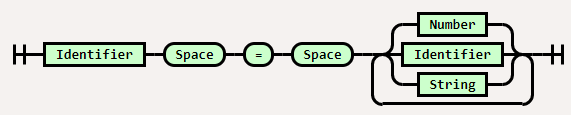
String



Identifier



Assignment



Egy példa forráskód részlet , melyben egy osztály és a benne lévő elemek láthatók

Create NewLangExample(elsoParam, masodikParam)

harmadikParam = 10

funct addAllParam()

elsoParam + masodikParam + harmadikParam

Az alábbiakban a fenti osztály példányosítása és használatának példája látható

pelda = NewLangExample(1, 3)

pelda.setElsoParam(5)

szam = pelda.addAllParam

\section{Java parser generátorok}

Az interneten sokféle parser generátor található, mely segítsétével a szintaktikai elemzés könnyebbé válik, megoldható.

A feladat megoldásához olyan parser generátorra van szükség, mely Java nyelvű, mivel a fordítóprogram ezen a nyelvek kerül megírása.

Emellett a parser generátorok feldolgozás szempontjából is sokfélék, és jelen feladathoz olyan generátort kellett keresni, mely reguláris nyelvvel képes működni.

Az alábbiakban a legelterjedtebb ilyen generátorokat vizsgáljuk meg.

\subsection{AnnoFlex}

Az Annoflex egy Java alapú parser generátor, mely szabadon letölthető és használható, sőt módosítható is. Az AnnoFlex implementálható mind az Eclipse, mind a InteliJ fejlesztőpi környezetekbe, külső eszközként, így bármikor használható lesz.

AZ AnnoFlex megalkotáskor is a használata minél egyszerűbbé tétele volt a fő szempont, legegyszerűbb esetben az alábbi kódot kell megírnunk:

/\*\*

\* **@option** methodName = getNextToken

\* **@option** statistics = enabled

\*/

**public** **class** Example\_Annoflex {

/\*\* **@expr** [0-9]+ \*/ String createNumber() { **return** "number"; }

/\*\* **@expr** [a-zA-Z]+ \*/ String createIdentifier() { **return** "identifier"; }

/\*\* **@expr** [ \n\r\t\f]+ \*/ String createWhitespace() { **return** "whitespace"; }

//%%LEX-MAIN-START%%

//%%LEX-MAIN-END%%

}

Az AnnoFlexnek meg kell adni a beállításokat, méghozzá a megvalósítása szerint az osztálynév előtt és komment formájában kell megadni őket, valamint fontos, hogy a @option annotáció előzze meg őket. Példánkban a metódus nevét állítottuk be, illetve azt, hogy a generálás során statisztikát jelenítsen meg a program.

Ezután az osztályon belül meg kell adni a kifejezéseket és a metódusokat. Itt találkozhatunk több megkötéssel is az AnnoFlex tekintetében. A megadáskor mindenképpen @expr kifejezéssel kell kezdeni, melyet szintén kommentben kell elhelyezni.

A kifejezés után csak és kizárólag reguláris kifejezés állhat. A komment után magát a metódust kell megírni, melynél szintén van megkötés. A metódusok nem lehetnek static módosítóval ellátva, visszatérési értékük nem lehet csak primitív típus vagy String, de mindenképpen az összes így megadott metódusnak ugyanolyan visszatérési értékkel kell bírnia, egyetlen kivétellel, ami a void. Void visszatérési érték állhat más visszetérési érték mellett. További megkötés, hogy a metódusoknak nem lehet paraméterük, csak visszatérési értékük.

Minden egyes metódust csak egy darab reguláris kifejezés előzhet meg, ha több reguláris kifejezés is kellene, hogy ott álljon, akkor a reguláris kifejezések uniójával oldható ez meg, melyhez a | operátor használható.

Ez után következik egy tagek által határolt üres rész, ezt mindig a //%%LEX-MAIN-START%% és //%%LEX-MAIN-END%% határolja, és ide kerül legenerálásra a tulajdonképpeni kód. Ebben ad nagy segítséget az AnnoFlex.

Amennyiben elkészültünk a kifejezések megírásával, akkor a fejlesztőkörnyezetben lefuttathatjuk a AnnoFlex programot, mely eredménye a következő lesz:

/\*\*

\* **@option** methodName = getNextToken

\* **@option** statistics = enabled

\*/

**public** **class** Example\_Annoflex {

/\*\* **@expr** [0-9]+ \*/ String createNumber() { **return** "number"; }

/\*\* **@expr** [a-zA-Z]+ \*/ String createIdentifier() { **return** "identifier"; }

/\*\* **@expr** [ \n\r\t\f]+ \*/ String createWhitespace() { **return** "whitespace"; }

/\*\* **@expr** [^] \*/ String createMisc() { **return** "misc"; }

//%%LEX-MAIN-START%%

//================================================

// \_ \_\_\_\_\_ \_

// / \ \_ \_\_ \_ \_\_ \_\_\_ | \_\_\_| | \_\_\_ \_ \_\_

// / \_ \ | \_ \| \_ \ / \_ \| |\_ | |/ \_ \ \/ /

// / \_\_\_ \| | | | | | | (\_) | \_| | | \_\_/> <

// /\_/ \\_\\_| |\_|\_| |\_|\\_\_\_/|\_| |\_|\\_\_\_/\_/\\_\

//

//================================================

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Generation Statistics \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Rules: 4 \*

\* Lookaheads: 0 \*

\* Alphabet length: 4 \*

\* NFA states: 15 \*

\* DFA states: 5 \*

\* Static size: 308 Bytes \*

\* Instance size: 24 Bytes \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//=================

// Table Constants

//=================

/\*\*

\* Maps Unicode characters to DFA input symbols.

\*/

**private** **static** **final** **byte**[] ***CHARACTER\_MAP*** = *createCharacterMap*(

"\0\t\3\2\0\1\3\2\0\22\3\1\0\17\1\n\0\7\2\32\0\6\2\32");

/\*\*

\* The transition table of the DFA.

\*/

**private** **static** **final** **byte**[][] ***TRANSITION\_TABLE*** = *createTransitionTable*(

"\5\1\2\1\3\1\4\1\0\1\2\1\0\2\0\2\3\1\0\1\0\3\4\1\0\4");

/\*\*

\* Maps state numbers to action numbers.

\*/

**private** **static** **final** **byte**[] ***ACTION\_MAP*** = *createActionMap*(

"\0\1\1\1\2\1\3\1\4\1");

//===============

// String Fields

//===============

/\*\*

\* The current string to be scanned.

\*/

**private** String string = "";

//===============

// Region Fields

//===============

/\*\*

\* The start of the scan region.

\*/

**private** **int** regionStart;

/\*\*

\* The end of the scan region.

\*/

**private** **int** regionEnd;

//============

// Dot Fields

//============

/\*\*

\* The start position of the next scan.

\*/

**private** **int** dot;

//==============

// Match Fields

//==============

/\*\*

\* The start of the last match.

\*/

**private** **int** matchStart;

/\*\*

\* The end of the last match.

\*/

**private** **int** matchEnd;

//===============

// Table Methods

//===============

/\*\*

\* Creates the character map of the scanner.

\*

\* **@param** characterMapData The compressed data of the character map.

\* **@return** The character map of the scanner.

\*/

**private** **static** **byte**[] createCharacterMap(String characterMapData) {

**byte**[] characterMap = **new** **byte**[123];

**int** length = characterMapData.length();

**int** i = 0;

**int** j = 0;

**while** (i < length) {

**byte** curValue = (**byte**)characterMapData.charAt(i++);

**for** (**int** x=characterMapData.charAt(i++);x>0;x--) {

characterMap[j++] = curValue;

}

}

**return** characterMap;

}

/\*\*

\* Creates the transition table of the scanner.

\*

\* **@param** transitionTableData The compressed data of the transition table.

\* **@return** The transition table of the scanner.

\*/

**private** **static** **byte**[][] createTransitionTable(String transitionTableData) {

**byte**[][] transitionTable = **new** **byte**[5][4];

**int** length = transitionTableData.length();

**int** i = 0;

**int** j = 0;

**int** k = 0;

**while** (i < length) {

**byte** curValue = (**byte**)((**short**)transitionTableData.charAt(i++) - 1);

**for** (**int** x=transitionTableData.charAt(i++);x>0;x--) {

transitionTable[j][k++] = curValue;

}

**if** (k == 4) {

k = 0;

j++;

}

}

**return** transitionTable;

}

/\*\*

\* Creates the action map of the scanner.

\*

\* **@param** actionMapData The compressed data of the action map.

\* **@return** The action map of the scanner.

\*/

**private** **static** **byte**[] createActionMap(String actionMapData) {

**byte**[] actionMap = **new** **byte**[5];

**int** length = actionMapData.length();

**int** i = 0;

**int** j = 0;

**while** (i < length) {

**byte** curValue = (**byte**)((**short**)actionMapData.charAt(i++) - 1);

**for** (**int** x=actionMapData.charAt(i++);x>0;x--) {

actionMap[j++] = curValue;

}

}

**return** actionMap;

}

//================

// String Methods

//================

/\*\*

\* Sets the string to be scanned. The scan region is set to the entire

\* string.

\*

\* **@param** string The new string to be scanned.

\*/

**public** **void** setString(String string) {

**this**.string = string != **null** ? string : "";

regionStart = 0;

regionEnd = **this**.string.length();

dot = 0;

matchStart = 0;

matchEnd = 0;

}

/\*\*

\* Returns the current string to be scanned.

\*

\* **@return** The current string to be scanned.

\*/

**public** String getString() {

**return** string;

}

//================

// Region Methods

//================

/\*\*

\* Sets the scan region. The dot is clipped to the region if necessary.

\*

\* **@param** start The start of the scan region (inclusive).

\* **@param** end The end of the scan region (exclusive).

\* **@throws** IllegalArgumentException If the region is out of range

\*/

**public** **void** setRegion(**int** start, **int** end) {

**if** ((start < 0) || (end > string.length()) || (start > end)) {

**throw** **new** IllegalArgumentException("region offsets out of range");

}

regionStart = start;

regionEnd = end;

**if** (dot < start) {

dot = start;

}

**if** (dot > end) {

dot = end;

}

}

/\*\*

\* Returns the start of the scan region.

\*

\* **@return** The start of the scan region.

\*/

**public** **int** getRegionStart() {

**return** regionStart;

}

/\*\*

\* Returns the end of the scan region.

\*

\* **@return** The end of the scan region.

\*/

**public** **int** getRegionEnd() {

**return** regionEnd;

}

//=============

// Dot Methods

//=============

/\*\*

\* Sets the position at which the next scan starts.

\*

\* **@param** dot The new position at which the next scan starts.

\* **@throws** IllegalArgumentException If the position is out of range

\*/

**public** **void** setDot(**int** dot) {

**if** ((dot < regionStart) || (dot > regionEnd)) {

**throw** **new** IllegalArgumentException("dot out of range");

}

**this**.dot = dot;

}

/\*\*

\* Returns the position at which the next scan starts.

\*

\* **@return** The position at which the next scan starts.

\*/

**public** **int** getDot() {

**return** dot;

}

//===============

// Match Methods

//===============

/\*\*

\* Returns the start (inclusive) of the last match.

\*

\* **@return** The start (inclusive) of the last match.

\*/

**public** **int** getMatchStart() {

**return** matchStart;

}

/\*\*

\* Returns the end (exclusive) of the last match.

\*

\* **@return** The end (exclusive) of the last match.

\*/

**public** **int** getMatchEnd() {

**return** matchEnd;

}

/\*\*

\* Returns the length of the last match.

\*

\* **@return** The length of the last match.

\*/

**public** **int** getMatchLength() {

**return** matchEnd - matchStart;

}

/\*\*

\* Returns the text of the last match.

\*

\* **@return** The text of the last match.

\*/

**public** String getMatchText() {

**return** string.substring(matchStart,matchEnd);

}

/\*\*

\* Returns a character relative to the start of the last match.

\*

\* **@param** index The index of the character relative to the last match.

\* **@return** The character at the specified position.

\* **@throws** IndexOutOfBoundsException If the specified index is invalid

\*/

**public** **char** getMatchChar(**int** index) {

**int** stringIndex = matchStart + index;

**if** ((stringIndex < regionStart) || (stringIndex >= regionEnd)) {

**throw** **new** IndexOutOfBoundsException("match character not available");

}

**return** string.charAt(stringIndex);

}

//==============

// Scan Methods

//==============

/\*\*

\* Performs at the current position the next step of the lexical analysis

\* and returns the result.

\*

\* **@return** The result of the next step of the lexical analysis.

\* **@throws** IllegalStateException If a lexical error occurs

\*/

**public** String getNextToken() {

**if** (dot < regionEnd) {

// find longest match

**int** curState = 0;

**int** iterator = dot;

**int** matchState = -1;

**int** matchPosition = 0;

**do** {

**char** curChar = string.charAt(iterator);

curState = ***TRANSITION\_TABLE***[curState][curChar >= 123 ?

0 : ***CHARACTER\_MAP***[curChar]];

**if** (curState == -1) {

**break**;

}

**if** (***ACTION\_MAP***[curState] != -1) {

matchState = curState;

matchPosition = iterator;

}

} **while** (++iterator < regionEnd);

// match found, perform action

**if** (matchState != -1) {

**int** endPosition = matchPosition + 1;

matchStart = dot;

matchEnd = endPosition;

dot = endPosition;

**switch**(***ACTION\_MAP***[matchState]) {

**case** 0: **return** createNumber();

**case** 1: **return** createIdentifier();

**case** 2: **return** createWhitespace();

**case** 3: **return** createMisc();

}

}

// no match found, set match values and report as error

matchStart = dot;

matchEnd = dot;

**throw** **new** IllegalStateException("invalid input");

}

// no match found, set match values and return to caller

matchStart = dot;

matchEnd = dot;

**return** **null**;

}

//%%LEX-MAIN-END%%

}

A fenti kódban látható, hogy a program legenerálta a szükséges metódusokat és funkciókat. Ezután már nem kell mást tenni, mint egy futtatható osztályt készíteni a példához:

**public** **class** Example\_AnnoRun {

**public** **static** **void** main(String[] args) {

Example\_Annoflex anno = **new** Example\_Annoflex();

anno.setString("Ez 1 teszt string");

System.***out***.println("Scan:" + anno.getString());

String token = anno.getNextToken();

**while** (token != **null**) {

System.***out***.println(token + ":" + anno.getMatchText());

token = anno.getNextToken();

}

}

}

Ebben csak létrehozunk egy példányt az osztályból, hozzáadunk egy szöveget és lefuttatjuk a programot. A képernyőre a következő eredményt fogja a program kiírni:

Scan:Ez 1 teszt string

identifier:Ez

whitespace:

number:1

whitespace:

identifier:teszt

whitespace:

identifier:string

Látható, hogy identifier-nek jelezte a szövegeket, a számot numberként jelenítette meg és megtalálta a fehér karaktereket is, azaz a szóközöket.

\subsection{JFlex}

A JFlex egy Java nyelvhez Java nyelven írt szkenner generátor, azaz ez is egy olyan program melynek különféle reguláris nyelven írt kifejezéseket kell megadni mely alapján a beolvasott szöveget, programkódot elemzi. A JFlex a JLex átírt változata, melynek átírásakor a cél a teljes unicode támogatás és platformfüggetlenség volt, illetve a gyors szkenner generálás, kényelmes szintaktika és az is, hogy kompatibilis legyen a JLex-el.

\subsection{...}

% TODO: Összeszedni 4-5 használhatót, és csinálni hozzá példákat!