Gramática Inicial do Pyscal

A ideia inicial da gramática foi juntar algumas palavras reservadas do Python, com a forma rígida de programacão do Pascal, por isso o nome Pyscal. Primeiramente montamos uma ideia de gramática, para ter nocão de como a sintaxe ficaria.

Pyscal Versão 0.0.0 theta

<S> → program;<INST> begin <PROGRAM> end;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | epsilon
<ATTR> → identifier = <EXP>; | identifier = <CONTENT>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if <LOGEXP> <PROGRAM> | if <LOGEXP> <PROGRAM> else <PROGRAM> | epsilon
<LOGEXP> → identifier comp_op identifier | identifier | identifier logic_op identifier
<IO>→ read identifier | write <CONTENT>

aritmética
<EXP> → <T><E'> | <T>
<E'> → +<T><E'> | -<T><E'> | <T>
<T> → <F><T'> | <F>
<T'> → *<F><T'> | /<F><T'> | <F>
<F> → identifier | digit | (<EXP>)

aritmetica certa
E -> T E'
E' -> + T E' | -TE' |epsilon
T -> F T'
T' -> * F T' | /FT' |epsilon
F -> (E) | int

ref: http://stackoverflow.com/a/22919146
ref:
http://stackoverflow.com/questions/23845198/correct-ll1-grammar-for-arithmetic-expressions

Pyscal Versão 0.0.0.5 phi

Organizando algumas produções e melhorando a gramática

<S> → program;<INST> begin <PROGRAM> end;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | epsilon
<ATTR> → identifier = <EXP>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if <LOGEXP> <PROGRAM> | if <LOGEXP> <PROGRAM> else <PROGRAM> | epsilon
<LOGEXP> → identifier comp_op identifier | identifier | identifier logic_op identifier
<IO>→ read identifier | write <CONTENT>

aritmética
<EXP> → <T><E'>
<E'> → +<T><E'> | -<T><E'> | epsilon
<T> → <F><T'>
<T'> → *<F><T'> | /<F><T'> | epsilon
<F> → identifier | digit | boolean | string | char | (<EXP>)

Pyscal Versão 0.0.1 zeta

Fatorando, tirando não determinismos diretos

<S> → program;<INST> begin <PROGRAM> end;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | epsilon
<ATTR> → identifier = <EXP>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if<COND'>
<COND'> →<LOGEXP><COND''>
<COND''> → <PROGRAM><COND'''>
<COND'''> → else <PROGRAM> | epsilon
<LOGEXP> → identifier<LOGEXP'>
<LOGEXP'> → comp_op identifier | logic_op identifier | epsilon
<IO>→ read identifier | write <CONTENT>

aritmética
<EXP> → <T><E'>
<E'> → +<T><E'> | -<T><E'> | epsilon
<T> → <F><T'>
<T'> → *<F><T'> | /<F><T'> | epsilon
<F> → identifier | digit | boolean | string | char | (<EXP>)

Pyscal Versão 0.0.1.5 eta

Fatorando, tirando não determinismos indiretos

<S> → program;<INST> begin <PROGRAM> end;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | epsilon
<ATTR> → identifier = <EXP>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if<COND'>
<COND'> →<LOGEXP><COND''>
<COND''> → <PROGRAM><COND'''>
<COND'''> → else <PROGRAM> | epsilon
<LOGEXP> → identifier<LOGEXP'>
<LOGEXP'> → comp_op identifier | logic_op identifier | epsilon
<IO>→ read identifier | write <CONTENT>
<EXP> → <T><E'>
<E'> → +<T><E'> | -<T><E'> | epsilon
<T> → <F><T'>
<T'> → *<F><T'> | /<F><T'> | epsilon
<F> → identifier | digit | boolean | string | char | (<EXP>)

Não tinham não determinismos indiretos YEAH!

Pyscal Versão 0.0.2 iota

Calculando First e Follow

<S> → program;<INST> begin <PROGRAM> end;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | epsilon
<ATTR> → identifier = <EXP>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if<COND'>
<COND'> →<LOGEXP><COND''>
<COND''> → <PROGRAM><COND'''>
<COND'''> → else <PROGRAM> | epsilon
<LOGEXP> → identifier<LOGEXP'>
<LOGEXP'> → comp_op identifier | logic_op identifier | epsilon
<IO>→ read identifier | write <CONTENT>
<EXP> → <T><E'>
<E'> → +<T><E'> | -<T><E'> | epsilon
<T> → <F><T'>
<T'> → *<F><T'> | /<F><T'> | epsilon
<F> → identifier | digit | boolean | string | char | (<EXP>)

LEGENDA: Follow calculado na primeira parte do algoritmo, Produção percorrida pela segunda parte do algoritmo, Follow atualizado pela segunda parte do algoritmo.

FI(<S>) = {program}	FO(<S>) = {$, }
FI(<INST>) = {type, epsilon}	FO(<INST>) = {begin}
FI(<CONTENT>) = {digit, boolean, string, char}	FO(<CONTENT>) = {;, identifier, while, if, read, write, end, else}
FI(<PROGRAM>) = {identifier, while, if, read, write, epsilon}	**FO(<PROGRAM>) = {end, else, identifier, while, if, read, write}**
FI(<ATTR>) = {identifier}	FO(<ATTR>) = {identifier, while, if, read, write, end, else}
FI(<LOOP>) = {while}	FO(<LOOP>) = {identifier, while, if, read, write, end, else}
FI(<COND>) = {if}	FO(<COND>) = {identifier, while, if, read, write, end, else}
FI(<COND'>) = {identifier}	FO(<COND'>) = {identifier, while, if, read, write, end, else}
FI(<COND''>) = {identifier, while, if, read, write, epsilon}	**FO(<COND''>) = {identifier, while, if, read, write, end, else}**
FI(<COND'''>) = {else, epsilon}	**FO(<COND'''>) = {identifier, while, if, read, write, end, else}**
FI(<LOGEXP>) = {identifier}	FO(<LOGEXP>) = {begin, identifier, while, if, read, write, end, else}
FI(<LOGEXP'>) = {comp_op, logic_op, epsilon}	FO(<LOGEXP'>) = {begin, identifier, while, if, read, write, end, else}

FI(<IO>) = {read, write}	FO(<IO>) = {identifier, while, if, read, write, end, else}
FI(<EXP>) = {identifier, digit, boolean, string, char, (}	FO(<EXP>) = {;,)}
FI(<E'>) = {+, -, epsilon}	FO(<E'>) = {;,)}
FI(<T>) = {identifier, digit, boolean, string, char, (}	FO(<T>) = {+, -, ;,)}
FI(<T'>) = {*, /, epsilon}	FO(<T'>) = { +, -, ;,)}
FI(<F>) = {identifier, digit, boolean, string, char, (}	FO(<F>) = {*, /, +, -, ;,)}

A gramática montada não é LL(1), porque o aluno que fatorou não sabe fatorar :B.

Pyscal Versão 0.1.0 phi

Fatorando a gramática, agora do jeito certo!

<S> → program;<INST> begin <PROGRAM> end;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | epsilon
<ATTR> → identifier = <EXP>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if <LOGEXP> <PROGRAM><COND'>
<COND'> →else<PROGRAM> | epsilon
<LOGEXP> → identifier comp_op identifier | identifier | identifier logic_op identifier
<IO>→ read identifier | write <CONTENT>
<EXP> → <T><E'>
<E'> → +<T><E'> | -<T><E'> | epsilon
<T> → <F><T'>
<T'> → *<F><T'> | /<F><T'> | epsilon
<F> → identifier | digit | boolean | string | char | (<EXP>)

Pyscal Versão 0.1.1 iota

Calculando First e Follow

<S> → program;<INST> begin <PROGRAM>;
<INST> → type identifier = <CONTENT>; <INST> | epsilon
<CONTENT> → digit | boolean | string | char
<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | end
<ATTR> → identifier = <EXP>;
<LOOP> → while <LOGEXP> begin <PROGRAM> end;
<COND> → if <LOGEXP> <PROGRAM><COND'>
<COND'> →else<PROGRAM>end | end
<LOGEXP> → identifier<LOGEXP'>
<LOGEXP'> → comp_op identifier | epsilon | logic_op identifier
<IO>→ read identifier | write <CONTENT>
<EXP> → <T><E'>
<E'> → +<T><E'> | -<T><E'> | epsilon
<T> → <F><T'>
<T'> → *<F><T'> | /<F><T'> | epsilon
<F> → identifier | digit | boolean | string | char | (<EXP>)

LEGENDA: Follow calculado na primeira parte do algoritmo, Produção percorrida pela segunda parte do algoritmo, Follow atualizado pela segunda parte do algoritmo.

FI(<S>) = {program}	FO(<S>) = {$}
FI(<INST>) = {type, epsilon}	FO(<INST>) = {begin}
FI(<CONTENT>) = {digit, boolean, string, char}	FO(<CONTENT>) = {;, identifier, while, if, read, write}
FI(<PROGRAM>) = {identifier, while, if, read, write, end}	FO(<PROGRAM>) = {else, identifier, while, if, read, write}
FI(<ATTR>) = {identifier}	FO(<ATTR>) = {identifier, while, if, read, write}
FI(<LOOP>) = {while}	FO(<LOOP>) = {identifier, while, if, read, write}
FI(<COND>) = {if}	FO(<COND>) = {identifier, while, if, read, write}
FI(<COND'>) = {else}	FO(<COND'>) = {identifier, while, if, read, write}
FI(<LOGEXP>) = {identifier}	FO(<LOGEXP>) = {begin, identifier, while, if, read, write, else}
FI(<LOGEXP'>) = {comp_op, logic_op, epsilon}	FO(<LOGEXP'>) = {begin, identifier, while, if, read, write, else}
FI(<IO>) = {read, write}	FO(<IO>) = {identifier, while, if, read, write}
FI(<EXP>) = {identifier, digit, boolean, string, char, (}	FO(<EXP>) = {;,)}
FI(<E'>) = {+, -, epsilon}	FO(<E'>) = {;,)}
FI(<T>) = {identifier, digit, boolean, string, char, (}	FO(<T>) = {+, -, ;,)}
FI(<T'>) = {*, /, epsilon}	FO(<T'>) = {+, -, ;,)}

FI(<F>) = {identifier, digit, boolean, string, char, (}	FO(<F>) = {*, /, +, -, ;,)}

Ela é LL(1)!!! Agora é só montar a tabela e ser feliz!

Pyscal Versão 0.1.1 iota

Numerando as produções

1: **<S>** → program;<INST> begin <PROGRAM>;
2, 3: **<INST>** → type identifier = <CONTENT>; <INST> | epsilon
4, 5, 6, 7: **<CONTENT>** → digit | boolean | string | char
8, 9, 10, 11, 12: **<PROGRAM>** → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | end
13: **<ATTR>** → identifier = <EXP>;
14: **<LOOP>** → while <LOGEXP> begin <PROGRAM> end;
15: **<COND>** → if <LOGEXP> <PROGRAM><COND'>
16, 17: **<COND'>** →else<PROGRAM>end | end
18: **<LOGEXP>** → identifier<LOGEXP'>
19, 20, 21: **<LOGEXP'>** → comp_op identifier | epsilon | logic_op identifier
22, 23: **<IO>**→ read identifier | write <CONTENT>
24: **<EXP>** → <T><E'>
25, 26, 27: **<E'>** → +<T><E'> | -<T><E'> | epsilon
28: **<T>** → <F><T'>
29, 30, 31: **<T'>** → *<F><T'> | /<F><T'> | epsilon
32, 33, 34, 35, 36, 37: **<F>** → identifier | digit | boolean | string | char | (<EXP>)

Pyscal Versão 0.1.2 theta (char-less)

1: **<S>** → program;<INST> begin <PROGRAM>;
2, 3: **<INST>** → type identifier = <CONTENT>; <INST> | epsilon
4, 5, 6, 7: **<CONTENT>** → digit | boolean | string | list
8, 9, 10, 11, 12: **<PROGRAM>** → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | end
13: **<ATTR>** → identifier = <EXP>;
14: **<LOOP>** → while <LOGEXP> begin <PROGRAM> end;
15: **<COND>** → if <LOGEXP> <PROGRAM><COND'>
16, 17: **<COND'>** →else<PROGRAM>end | end
18: **<LOGEXP>** → identifier<LOGEXP'>
19, 20, 21: **<LOGEXP'>** → comp_op identifier | epsilon | logic_op identifier
22, 23: **<IO>**→ read identifier | write <CONTENT>
24: **<EXP>** → <T><E'>
25, 26, 27: **<E'>** → +<T><E'> | -<T><E'> | epsilon
28: **<T>** → <F><T'>
29, 30, 31: **<T'>** → *<F><T'> | /<F><T'> | epsilon
32, 33, 34, 35, 36: **<F>** → identifier | digit | boolean | string | (<EXP>)