Transpiler van staal to js

Jens Vissenberg , Liam Leirs ,
Kobe De Broeck, Stein Vandenbroeke

Wat

Transpiler

```
int a = 1;
int b = 1;
int n = 1;
while(n < 20){
    print(a);
    int temp = a;
    a = a + b;
    b = temp;
    n = n + 1;
}
```

Outputs

```
0
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

Transpiler information

```
---------
Successfully transpiled!
---------
Successfully transpiled!
```

```
stein@LAPTOP-BAV8NCAR:~/test/TranspilerMB/build$ ./transpiler staalWebInt.c4 export.js
transpiling done
```

Overzicht

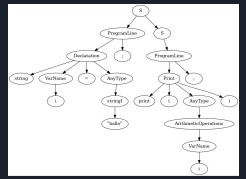
ε-nfa's

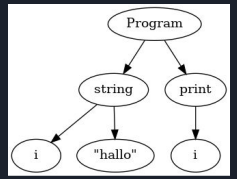
Parser mbv parse table

Interpretatie CST

Input file in staal -> Tokens -> CST -> AST -> JS code

```
string i = "hallo";
print(i);
```



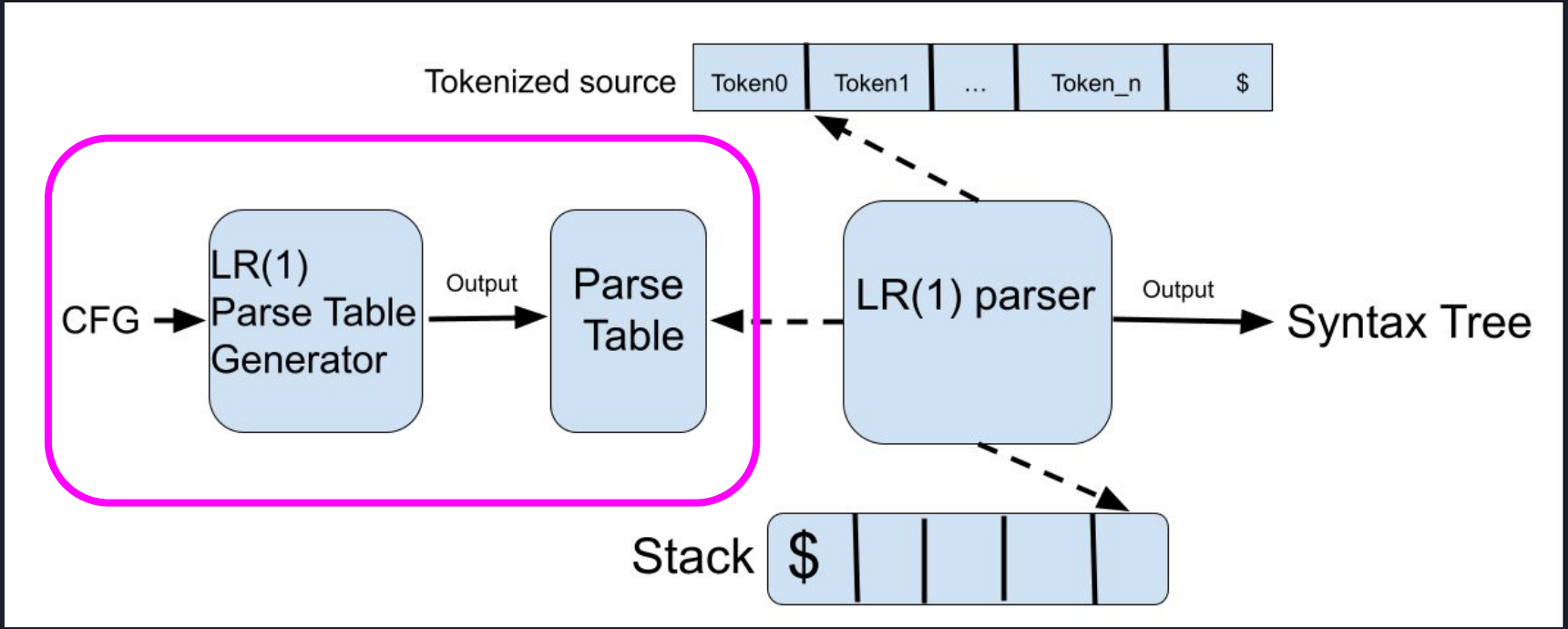



```
let i = "hallo";
console.log(i);
```

Overzicht LR(1) Parse-Table-Generatie

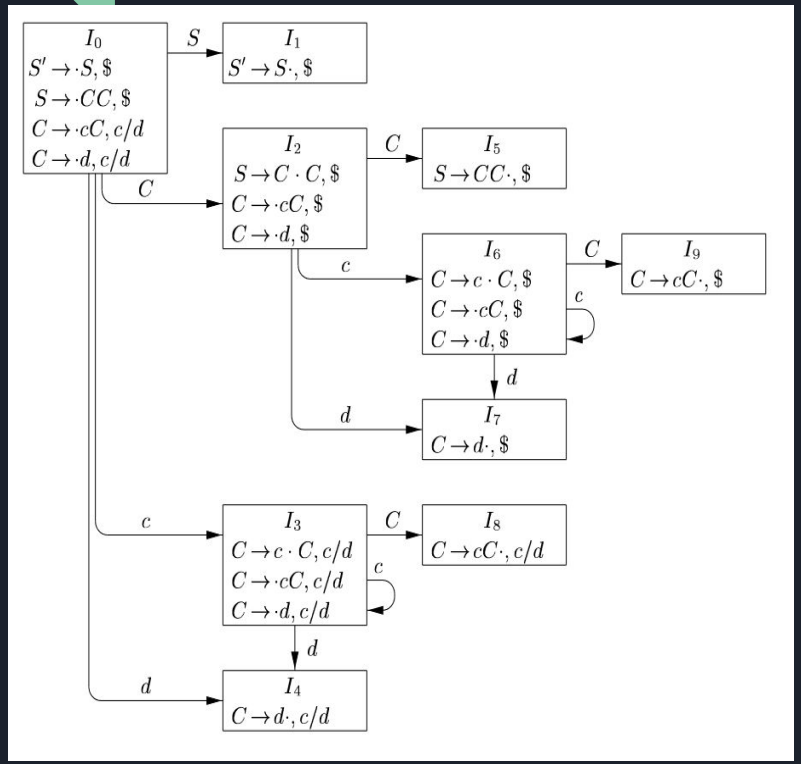
LR(1) parser generator

- Stap 1



$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \mid d
\end{array}
$$

Shift Reduce Parsing (Bottom up) == Uitvoeren van de LR(1) automaat mbv deze tabel



Parse Table

STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Hoe LR(1) automaat opstellen?

```
inline LR1Collection calcLR1Collection(const Grammar& g) {
    const auto& augmentedStartProduction = g.getProductionsOf(g.getStart()).at(0);
    Item startItem = Item{ augmentedStartProduction.get(), 0, g.getSymbol("$")};

    LR1Collection collection{};
    collection.states.push_back(lr1Closure({startItem}, g));

    bool progressMade {true};
    while(progressMade) {
        progressMade = false;
        for(size_t i = 0; i < collection.states.size(); ++i) {
            for(const auto& X : g.getSymbols()) {
                auto stateToGoTo = lr1Goto(collection.states[i], X.get(), g);
                if(!stateToGoTo.empty()) {
                    auto stateToGoToIndex = KDBFacilities::find(collection.states, stateToGoTo);
                    if(stateToGoToIndex >= 0) {
                        if(collection.transitions.find({i, X.get()}) == collection.transitions.end()) {
                            progressMade = true;
                            collection.transitions[{i, X.get()}] = stateToGoToIndex;
                        }
                    }
                    else {
                        progressMade = true;
                        collection.states.push_back(stateToGoTo);
                        collection.transitions[{i, X.get()}] = collection.states.size()-1;
                        //std::cout << stateToGoTo << "\n\n";
                    }
                }
            }
        }
    }
    return collection;
```

void $items(G')$ {
    initialize C to $\{\text{CLOSURE}(\{[S' \to \cdot S, \$]\})\}$;
    repeat
        for (each set of items I in C)
            for (each grammar symbol X)
                if ($\text{GOTO}(I, X)$ is not empty and not in C)
                    add $\text{GOTO}(I, X)$ to C;
    until no new sets of items are added to C;
}

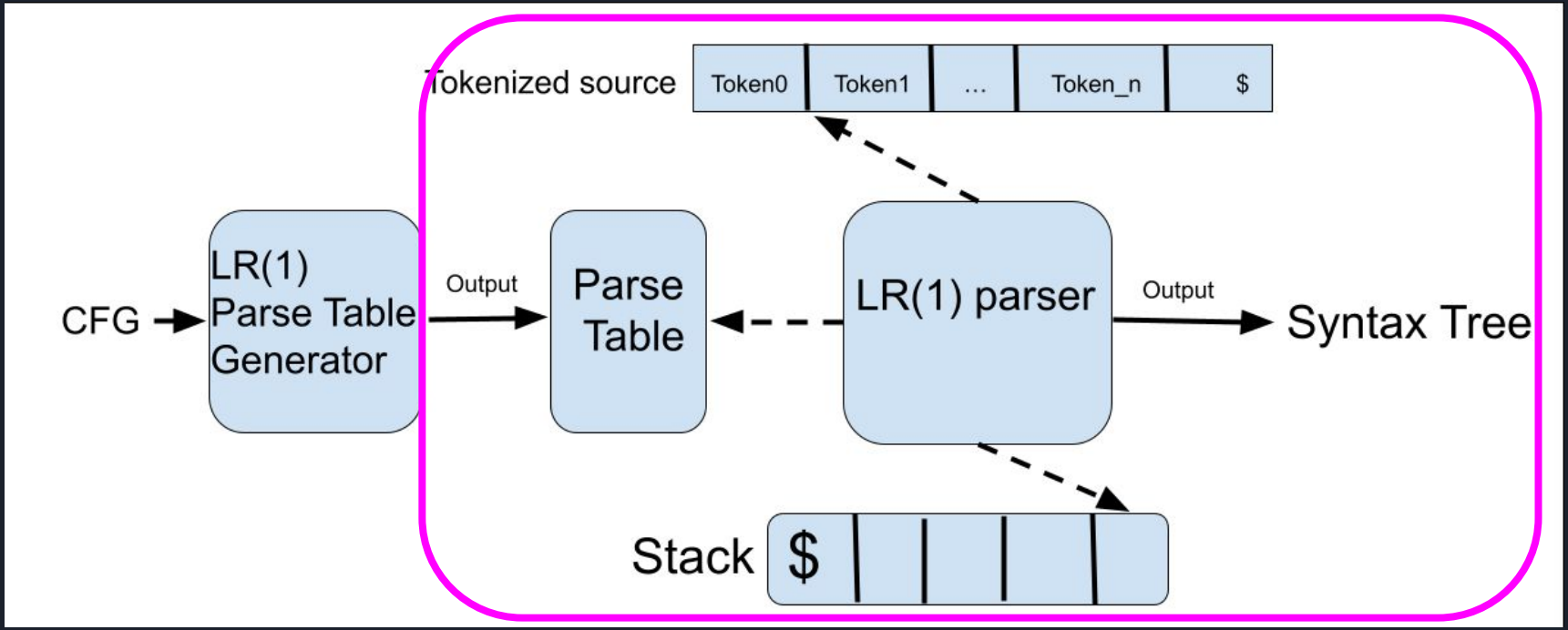
STaal Parse Table

- 151 states

Serialiseren naar json

LR(1) parser

- Stap 2



Tokenizer:

Keywords: "print", "int", "double", "char", "string"
Seperators: " ", ";", "(", ")", "=", "!=", "<=", ">=", "<", ">", "==", "{", "}", "+", "-", "*", "/"
Types:

- string: [STRINGTOKEN]
- char: [CHARTOKEN]
- getallen: [NUMBERTOKEN]
- variabele namen: [VARNAMETOKEN]

Tokenizer:

Lees file

Splits file o.b.v. seperators

```
int a = 1;
int b = 1;
int n = 1;
while(n < 20){
    print(a);
    int temp = a;
    a = a + b;
    b = temp;
    n = n + 1;
}
```

```
1: int | a | = | 1 | ; |
2: int | b | = | 1 | ; |
3: int | n | = | 1 | ; |
4: while | ( | n | < | 20 | ) | { |
5: print | ( | a | ) | ; |
6: int | temp | = | a | ; |
7: a | = | a | + | b | ; |
8: b | = | temp | ; |
9: n | = | n | + | 1 | ; |
10: } |
```

Lezen van strings en operators

Tokenizer:

ENFA 1:

- gegenereerd met lijst van keywords
- check split file op keywords

ENFA 2:

- herkennen van:
  - strings
  - chars
  - getallen
  - variabele namen

Woord herkend => creëer token

Tokenizer:

```
1: int | a | = | 1 | ; |
2: int | b | = | 1 | ; |
3: int | n | = | 1 | ; |
4: while | ( | n | < | 20 | ) | { |
5: print | ( | a | ) | ; |
6: int | temp | = | a | ; |
7: a | = | a | + | b | ; |
8: b | = | temp | ; |
9: n | = | n | + | 1 | ; |
10: } |
```

Tokens:

- text
- type
- lijnnummer
- positie in de lijn

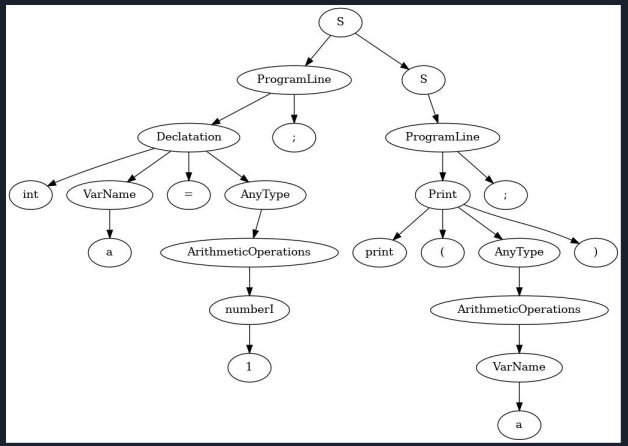
ENFA 1: type = woord, text = / `{int:  at (1, 1)}`

ENFA 2: type = afhankelijk eindstaat, text = woord `{[VARNAMETOKEN]: a at (1, 2)}`

End token: `{$: $ at (11, 1)}`

CST Constructie

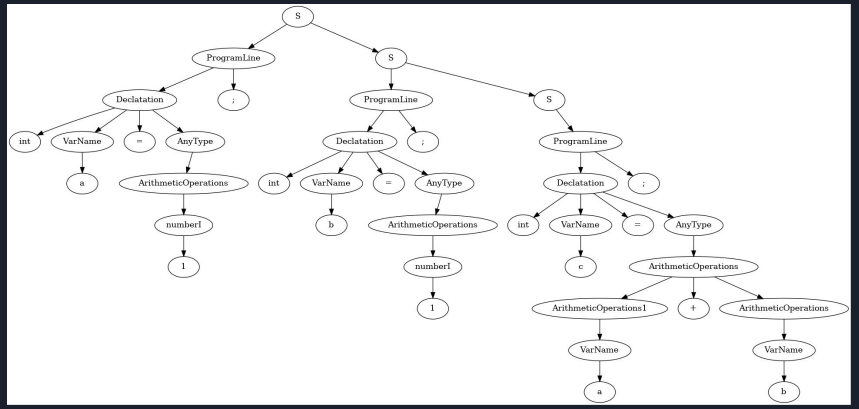
- Tree weergave van LR parsing
- LR -> CST algoritme:
  - Stack van states (state + symbol = index)
  - Shift = Nieuwe node
  - Reduce = Nieuwe parent + children




```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

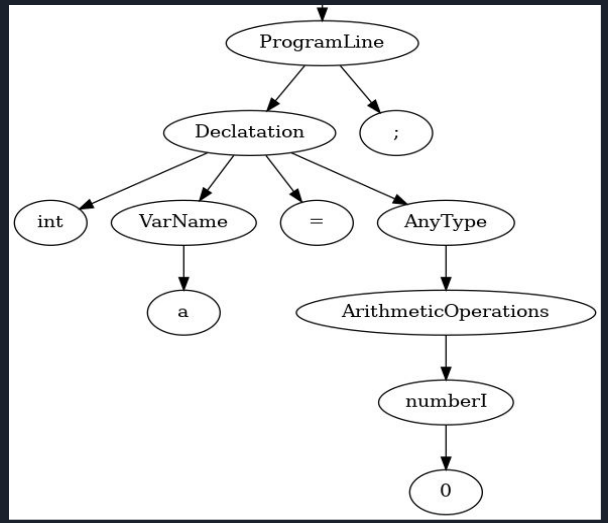
CST Weergave

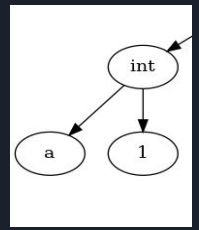
- DOT language
- Te genereren via CST.generateDOT()



AST

- Alle nuttige informatie uit de CST halen





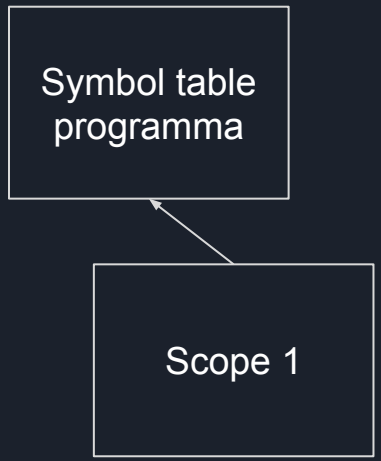
AST Constructie

class: Cst node
functie: toAst();

```
else if(this->getValue() == "Initalization"){
    AstVar* astVar = dynamic_cast<AstVar *>(this->children[0]->toAst(program));
    AstNode* astValue = this->children[2]->toAst(program);
    return new AstIntalisation( token: this->getChildren()[1]->getToken(), astVar, astValue);
}
```

```
class AstDeclartion;
class AstIntalisation;
class AstWhile;
class AstIf;
class AstBody;
class AstCondition;
class AstArithmeticOperations;
class AstVar;
class AstProgram;
class AstValue;
class AstVarOrValue;
class AstParentheses;
class AstPrint;
```

Symbol table



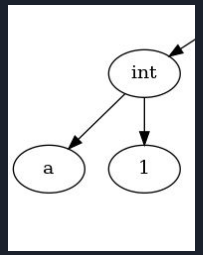

```
int i = 0;
if(i < 20){
    string i = "hallo";
    print(i);
}
```

```
----------
Successfully transpiled!
```

```
int i = 0;
string i = "hallo";
print(i);
```

```
---------
error on line: 2 [Variable i is declared multiple times in the same scope]
```

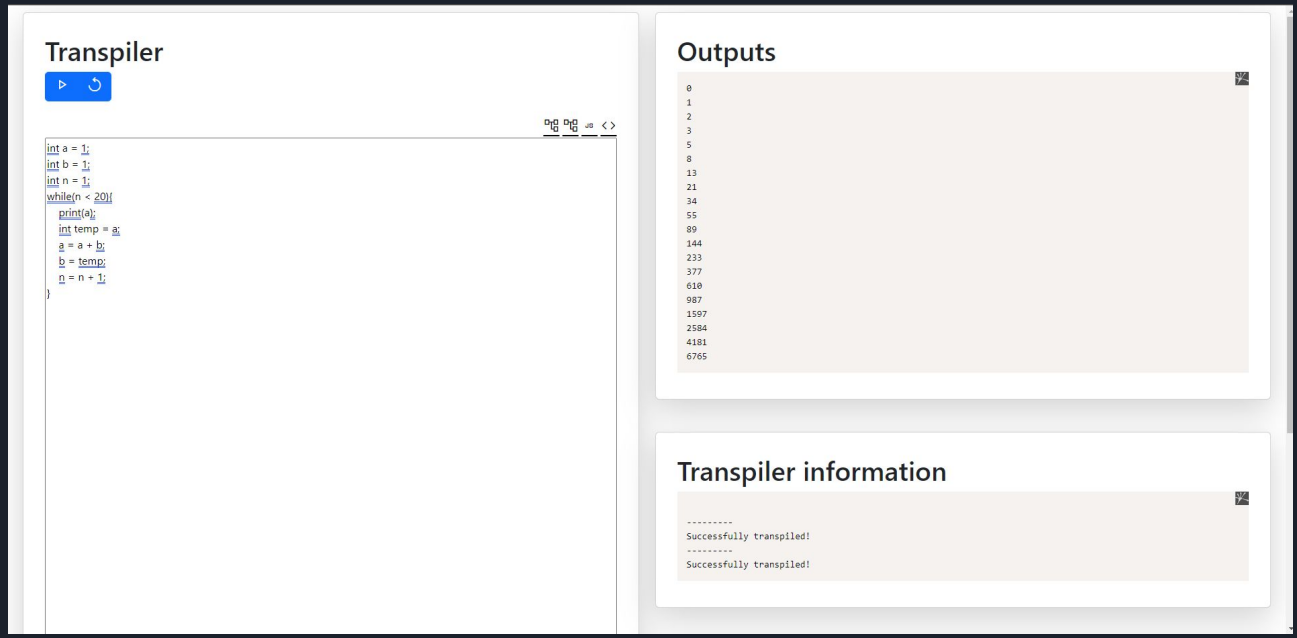
AST to code



let a = 1

```
std::string AstDeclartion::getJsCode(int scopeCount) const {
    return "let " + var->getTokenText() + " = " + value->getJsCode(scopeCount) + ";";
}
```

Demo


Transpiler
int a = 1;
int b = 1;
int n = 1;
while(n < 20){
print(a);
int temp = a;
a = a + b;
b = temp;
n = n + 1;
}
Outputs
0
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
Transpiler information

Successfully transpiled!

Successfully transpiled!

Einde