Rekenmachine

Groep 8: Dara van Engelen, Roan Breëns, Jamie Lakchi

Parsing

Packrat Parser

- Parser combinator
  - oneindige lookahead
  - recursive descent
- Parsing Expression Grammar
  - LL(k) en LR(k) talen parsen
- Memoization → linear parse times, linear memory usage (when the grammar is constant)
- Zwakte: links-recursie, oplossing seed-parsing
- Why? It insists upon itself

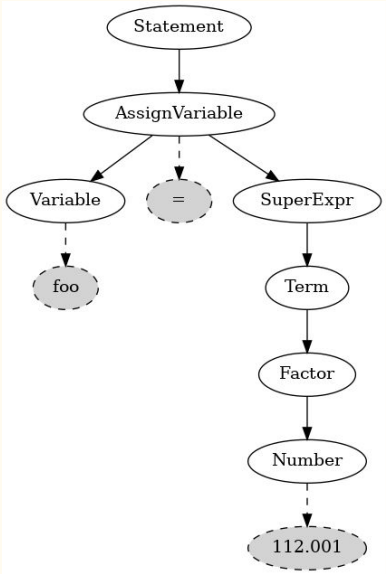
Parsing Expression Grammars

CFG met syntactic sugar (regex achtig):

Expr ← Additive
Additive ← Multiplicative (('+' / '-') Multiplicative)*
Multiplicative ← Primary (('*' / '/') Primary)*
Primary ← Number / '(' Expr ')'
Number ← [0-9]+

Parser Combinator

- Veel kleine/modulaire parsers samenvoegen
- Recursive descent parsing




```
auto originalPosition: position = _streamPosition;
auto nrDigits: (lambda) = [&]()-> size_t {
  size_t i = 0;
  for (; (i + _streamPosition) < _stream.size(); ++i) {
    char c = _stream[i + _streamPosition];
    if (!('0' <= c && c <= '9')) {
      break;
    }
  }
  return i;
};

_streamPosition += nrDigits();

if (_streamPosition < _stream.size()) {
  if (_stream[_streamPosition] == '.') {
    ++_streamPosition;
    _streamPosition += nrDigits();
  }
}

if (_streamPosition == originalPosition) {
  return nullptr;
}

return create_shared<ParseTreeNode>(
    _stream.substr(pos: originalPosition, n: _streamPosition - originalPosition));
```

Parser Combinator

- Veel kleine/modulaire parsers samenvoegen
- Recursive descent parsing
- Memoization!
  - Linear parse time
  - Linear memory usage

	Index						
	1	2	3	4	5	6	7
S							
A				3			
M				1		1	
P	1		5	1		1	
D	1			1		1	
	2	*	(	3	+	4	)

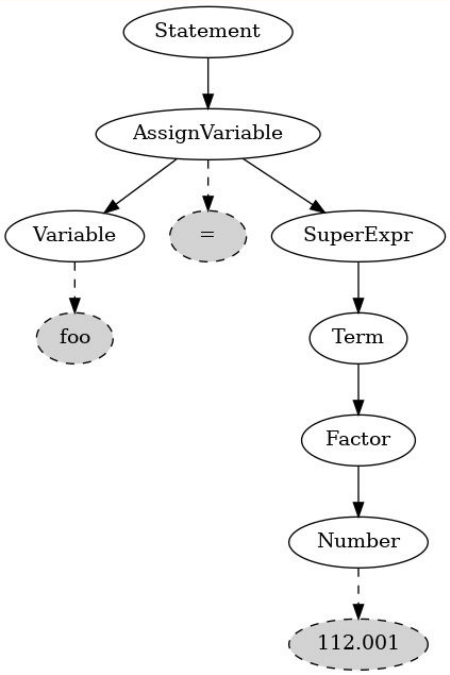
Parser Combinator

- Veel kleine/modulaire parsers samenvoegen
- Recursive descent parsing
- Memoization!
  - Linear parse time
  - Linear memory usage
  - Link-recursie ook hier opgelost
    - Vb: S → S ‘a’ | ‘a’

	Index						
	1	2	3	4	5	6	7
S							
A				3			
M				1		1	
P	1		5	1		1	
D	1	LR		1		1	
	2	*	(	3	+	4	)

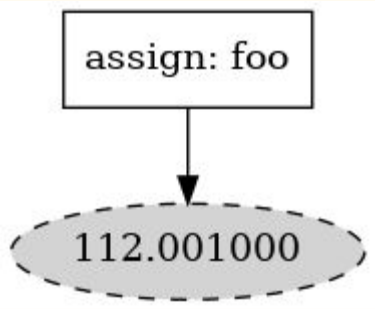
Parse Tree Evaluation

Abstract Syntax Tree conversion



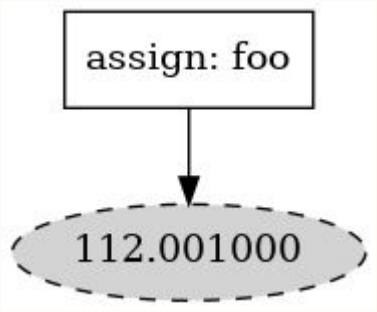

```
shared_ptr<ASTNode>
ASTConverter::_convert(shared_ptr<rats::ParseTreeNode> &root) {
  static const hashmap<string, c_functype> converters{
      [0]={sym::Evaluate, &ASTConverter::c_evaluate},
      [1]={sym::AssignFunction, &ASTConverter::c_assignfunc},
      [2]={sym::AssignVariable, &ASTConverter::c_assignvar},
      [3]={sym::SuperExpr, &ASTConverter::c_superexpr},
      [4]={sym::Expr, &ASTConverter::c_expr},
      [5]={sym::Term, &ASTConverter::c_term},
      [6]={sym::Factor, &ASTConverter::c_factor},
      [7]={sym::Number, &ASTConverter::c_number},
      [8]={sym::Function, &ASTConverter::c_function},
      [9]={sym::Variable, &ASTConverter::c_variable}};

  auto rsym: string = root->_value;
  auto it: const_iterator = converters.find(x: rsym);

  if (it != converters.end()) {
    return (this->*(it->second))(root);
  }

  return nullptr;
}
```

conversion



AST evaluation

- Eerst valideren (naam conflicten)
- Komt overeen met berekening
- Sommige AST's passen geheugen aan
- Functie opslaan als AST
  - Parameter naam conflicten bestaan niet



Demo