Лабораторная работа № 2.2 «Абстрактные синтаксические деревья»

26 мая 2025 г.

Александр Старовойтов, ИУ9-61Б

Цель работы

Целью данной работы является получение навыков составления грамматик и проектирования синтаксических деревьев.

Индивидуальный вариант

Определения структур, объединений и перечислений языка Си. В инициализаторах перечислений допустимы знаки операций +, -, *, /, sizeof, операндами могут служить имена перечислимых значений и целые числа.

Числовые константы могут быть только целочисленными и десятичными.

Реализация

Исправление бага в parser_edsl

Исправлены баги с атрибутами и эпсилон-правилами в алгоритме Эрли.

https://github.com/bmstu-iu9/parser_edsl_python/pull/2

Абстрактный синтаксис

```
Program -> Definition Program | Definition
Definition -> Struct | Enum | Union
DefinitionOrVariable -> Definition | NumVar
NumVar -> NumType *? Variables ;
NumType -> INT | DOUBLE | FLOAT | CHAR | SHORT | LONG
Struct -> STRUCT NAME? StructFields? *? Variables? ;
StructFields -> { DefinitionOrVariable* }
Variables -> NAME (, NAME)*
Enum -> ENUM NAME? EnumFields? *? Variables? ;
```

```
EnumFields -> { EnumField (, EnumField)* ,? }
EnumField -> NAME EnumFieldRhs?
EnumFieldRhs -> = Expr
Expr -> NUMBER | NAME | Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr | SIZEOF(Expr) | ( Expr
Union -> UNION NAME? UnionFields? *? Variables? ;
UnionFields -> { DefinitionOrVariable* }
```

Лексическая структура и конкретный синтаксис

Лексическая структра:

```
NUMBER ::= [0-9]+
NAME ::= [a-zA-Z][a-zA-Z0-9_-]*
INT ::= int
```

Конкретный синтаксис:

```
Program -> Definition Program | Definition
Definition -> Struct | Enum | Union
DefinitionOrVariable -> Definition | NumVar
NumVar -> NumType PointerOpt Variables ;
PointerOpt -> * | ε
NumType -> INT | DOUBLE | FLOAT | CHAR | SHORT | LONG
Struct -> STRUCT NameOpt StructFieldsOpt PointerOpt VariablesOpt ;
NameOpt -> NAME | ε
VariablesOpt -> Variables | ε
StructFieldsOpt -> StructFields | ε
StructFields -> { DefinitionsOrVariables }
DefinitionsOrVariables -> DefinitionOrVariable DefinitionsOrVariables | ε
Variables -> NAME VariablesTail
VariablesTail -> , NAME VariablesTail | ε
Enum -> ENUM NameOpt EnumFieldsOpt PointerOpt VariablesOpt ;
EnumFieldsOpt -> EnumFields | ε
EnumFields -> { EnumField EnumFieldsTail CommaOpt }
CommaOpt -> , | ε
EnumFieldsTail -> , EnumField EnumFieldsTail | ε
EnumField -> NAME EnumFieldRhsOpt
EnumFieldRhsOpt -> EnumFieldRhs | ε
EnumFieldRhs -> = Expr
Expr -> NUMBER | NAME | Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr | SIZEOF ( Expr ) | -
Expr | ( Expr )
Union -> UNION NameOpt UnionFieldsOpt PointerOpt VariablesOpt ;
UnionFieldsOpt -> UnionFields | ε
UnionFields -> { DefinitionsOrVariables }
```

Программная реализация

```
#!/usr/bin/env python3

import abc
import enum
import typing
from dataclasses import dataclass
import parser_edsl as pe
import re
from pprint import pprint
import sys


class DefinitionBase(abc.ABC):
    pass


# Definition -> Struct | Enum | Union
@dataclass
class Definition:
    data: DefinitionBase


# NumType -> INT | DOUBLE | FLOAT | CHAR | SHORT | LONG
class Type(enum.Enum):
    INT = 'int'
    DOUBLE = 'double'
    FLOAT = 'float'
    CHAR = 'char'
    SHORT = 'short'
    LONG = 'long'


@dataclass
class NumVariablesDefinition:
    typename: Type
    is_pointer: bool
    name_dimension: list[tuple[str, list[str|int]]]


# Struct -> STRUCT NameOpt StructFieldsOpt PointerOpt VariablesOpt ;
@dataclass
class Struct(DefinitionBase):
    name: str|None
    fields: list[Definition|NumVariablesDefinition]
```

```
    is_pointer: bool
    variables: list[str]


# Expr -> NUMBER | NAME | Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr | SIZEOF ( E
class Expr(abc.ABC):
    pass


@dataclass
class ExprNumber(Expr):
    number: int


@dataclass
class ExprName(Expr):
    name: str


@dataclass
class BinOpExpr(Expr):
    lhs: Expr
    op: str
    rhs: Expr


@dataclass
class UnOpExpr(Expr):
    op: str
    expr: Expr


# EnumField -> NAME EnumFieldRhsOpt
@dataclass
class EnumField:
    name: str
    rhs: Expr|None


# Enum -> ENUM NameOpt EnumFieldsOpt PointerOpt VariablesOpt ;
@dataclass
class Enum(DefinitionBase):
    name: str|None
    fields: list[EnumField]
    is_pointer: bool
    variables: list[str]
```

```python
# Union -> UNION NameOpt UnionFieldsOpt PointerOpt VariablesOpt ;
@dataclass
class Union(DefinitionBase):
    name: str|None
    fields: list[Definition|NumVariablesDefinition]
    is_pointer: bool
    variables: list[str]


# Program -> Definition Program | Definition
@dataclass
class Program:
    definitions: list[Definition]


INTEGER = pe.Terminal('INTEGER', '[0-9]+', int, priority=7)
VARNAME = pe.Terminal('VARNAME', '[A-Za-z][A-Za-z0-9_]*', str)

def make_keyword(image):
    return pe.Terminal(image, image, lambda name: None, priority=10)

KW_INT, KW_DOUBLE, KW_FLOAT, KW_CHAR, KW_SHORT, KW_LONG = \
    map(make_keyword, 'int double float char short long'.split())

KW_SIZEOF, KW_ENUM, KW_UNION, KW_STRUCT = \
    map(make_keyword, 'sizeof enum union struct'.split())

NProgram, NDefinition, NDefinitions, NStruct, NEnum, NUnion = \
    map(pe.NonTerminal, 'Program Definition Definitions Struct Enum Union'.split())

NDefinitionOrVariable, NNumVar, NNumType, NPointerOpt, NVariables = \
    map(pe.NonTerminal, 'DefinitionOrVariable NumVar NumType PointerOpt Variables'.split())

NNameOpt, NStructFieldsOpt, NVariablesOpt, NStructFields = \
    map(pe.NonTerminal, 'NameOpt StructFieldsOpt VariablesOpt StructFields'.split())

NDefinitionsOrVariables, NVariablesTail, NEnumFieldsOpt, NSizeOf  = \
    map(pe.NonTerminal, 'DefinitionsOrVariables VariablesTail, EnumFieldsOpt SizeOf'.split()

NEnumFields, NEnumField, NEnumFieldsTail, NCommaOpt, NDimensions  = \
    map(pe.NonTerminal, 'EnumFields EnumField EnumFieldsTail CommaOpt Dimensions'.split())

NEnumFieldRhsOpt, NExpr, NUnionFieldsOpt, NUnionFields = \
    map(pe.NonTerminal, 'EnumFieldRhsOpt Expr UnionFieldsOpt UnionFields'.split())
```

```
NEnumOther, NEnumOtherOther, NEnumFieldsRest, NEnumFieldsBody = \
    map(pe.NonTerminal, 'EnumOther EnumOtherOther EnumFieldsRest EnumFieldsBody'.split())

NFactor, NTerm, NAddOp, NMulOp = \
    map(pe.NonTerminal, 'Factor Term AddOp MulOp'.split())

# Program -> Definition Program | Definition
NProgram |= NDefinitions, Program
NDefinitions |= NDefinition, NDefinitions, lambda d, p: [d]+p
NDefinitions |= lambda: []

# Definition -> Struct | Enum | Union
NDefinition |= NStruct
NDefinition |= NEnum
NDefinition |= NUnion

# DefinitionOrVariable -> Definition | NumVar
NDefinitionOrVariable |= NDefinition
NDefinitionOrVariable |= NNumVar

# NumVar -> NumType PointerOpt Variables ;
NNumVar |= NNumType, NPointerOpt, NVariables, ';', NumVariablesDefinition

# PointerOpt -> * | ε
NPointerOpt |= '*', lambda: True
NPointerOpt |= lambda: False

# NumType -> INT | DOUBLE | FLOAT | CHAR | SHORT | LONG
NNumType |= KW_INT, lambda: Type.INT
NNumType |= KW_DOUBLE, lambda: Type.DOUBLE
NNumType |= KW_CHAR, lambda: Type.CHAR
NNumType |= KW_SHORT, lambda: Type.SHORT
NNumType |= KW_LONG, lambda: Type.LONG

# Struct -> STRUCT NameOpt StructFieldsOpt PointerOpt VariablesOpt ;
NStruct |= KW_STRUCT, NNameOpt, NStructFieldsOpt, NPointerOpt, NVariablesOpt, ';', Struct

# NameOpt -> NAME | ε
NNameOpt |= VARNAME
NNameOpt |= lambda: ''

# VariablesOpt -> Variables | ε
NVariablesOpt |= NVariables
NVariablesOpt |= lambda: []
```

```
# StructFieldsOpt -> StructFields | ε
NStructFieldsOpt |= NStructFields
NStructFieldsOpt |= lambda: []

# StructFields -> { DefinitionsOrVariables }
NStructFields |= '{', NDefinitionsOrVariables, '}'

# DefinitionsOrVariables -> DefinitionOrVariable DefinitionsOrVariables | ε
NDefinitionsOrVariables |= NDefinitionOrVariable, NDefinitionsOrVariables, lambda d, arr: [d
NDefinitionsOrVariables |= lambda: []

# Variables -> NAME , Variables | Variables
NVariables |= VARNAME, NDimensions, ',', NVariables, lambda name, dimensions, arr: [(name, d
NVariables |= VARNAME, NDimensions, lambda name, dimensions: [(name, dimensions)]

NDimensions |= '[', NExpr, ']', NDimensions, lambda expr, other: [expr]+other
NDimensions |= lambda: []


NEnum |= KW_ENUM, VARNAME, NEnumOther, lambda name, other: Enum(name, *other)
NEnum |= KW_ENUM, NEnumOther, lambda other: Enum('', *other)

NEnumOther |= NEnumFields, NEnumOtherOther, lambda fields, other: [fields]+other
NEnumOther |= NEnumOtherOther, lambda other: [[]]+other

NEnumOtherOther |= '*', NVariables, ';', lambda v: [True, v]
NEnumOtherOther |= NVariables, ';', lambda v: [False, v]
NEnumOtherOther |= ';', lambda: [False, []]

NEnumFields |= '{', NEnumFieldsBody

NEnumFieldsBody |= '}', lambda: []
NEnumFieldsBody |= NEnumField, NEnumFieldsRest, lambda l, r: [l]+r
NEnumFieldsRest |= ',', NEnumField, NEnumFieldsRest, lambda l, r: [l]+r
NEnumFieldsRest |= ',', '}', lambda: []
NEnumFieldsRest |= '}', lambda: []

# EnumFieldsTail -> , EnumField EnumFieldsTail | ε
NEnumFieldsTail |= ',', NEnumField, NEnumFieldsTail, lambda f, t: [f]+t
NEnumFieldsTail |= lambda: []

# EnumField -> NAME EnumFieldRhsOpt
NEnumField |= VARNAME, '=', NExpr, EnumField
NEnumField |= VARNAME, lambda name: EnumField(name, None)

# Expr -> NUMBER | NAME | Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr | SIZEOF ( V
```

```
NExpr |= NTerm
NExpr |= '+', NTerm, lambda t: UnOpExpr('+', t)
NExpr |= '-', NTerm, lambda t: UnOpExpr('-', t)
NExpr |= NExpr, NAddOp, NTerm, BinOpExpr
NTerm |= NFactor
NTerm |= NTerm, NMulOp, NFactor, BinOpExpr
NMulOp |= '*', lambda: '*'
NMulOp |= '/', lambda: '/'
NAddOp |= '+', lambda: '+'
NAddOp |= '-', lambda: '-'
NFactor |= INTEGER, lambda v: ExprNumber(int(v))
NFactor |= '(', NExpr, ')'
NFactor |= KW_SIZEOF, '(', NSizeOf, ')', lambda inner: UnOpExpr('sizeof', ExprName(inner))
NFactor |= VARNAME, ExprName
NSizeOf |= KW_ENUM, VARNAME, lambda y: 'enum '+y
NSizeOf |= KW_UNION, VARNAME, lambda y: 'union '+y
NSizeOf |= KW_STRUCT, VARNAME, lambda y: 'struct '+y
NSizeOf |= VARNAME
NSizeOf |= NNumType

# Union -> UNION NameOpt UnionFieldsOpt PointerOpt VariablesOpt ;
NUnion |= KW_UNION, NNameOpt, NUnionFieldsOpt, NPointerOpt, NVariablesOpt, ';', Union

# UnionFieldsOpt -> UnionFields | ε
NUnionFieldsOpt |= NUnionFields
NUnionFieldsOpt |= lambda: []

# UnionFields -> { DefinitionsOrVariables }
NUnionFields |= '{', NDefinitionsOrVariables, '}'

p = pe.Parser(NProgram)

p.add_skipped_domain('\\s')
p.add_skipped_domain(r"(?:\/\/.*)|(?:\/\*(?:.|\n)*?\*\/)")

for filename in sys.argv[1:]:
    with open(filename) as f:
        tree = p.parse_earley(f.read())
        pprint(tree)
```

Тестирование

Входные данные

```
struct Coords {
```

```
  int x, y;
};
```

Вывод на `stdout`

```
Program(definitions=[Struct(name='Coords',
                  fields=[NumVariablesDefinition(typename=<Type.INT: 'int'>,
                                                 is_pointer=False,
                                                name_dimension=[('x',
                                                                 []),
                                                                ('y',
                                                                 [])])],
                          is_pointer=False,
                          variables=[])])
```

Входные данные

```
enum Color {
  COLOR_RED = 1,
  COLOR_GREEN = 2,
  COLOR_BLUE = 4,
  COLOR_HIGHLIGHT = 8, // запятая после последнего необязательна
};
```

Вывод на `stdout`

```
Program(definitions=[Enum(name='Color',
                          fields=[EnumField(name='COLOR_RED',
                                            rhs=ExprNumber(number=1)),
                                  EnumField(name='COLOR_GREEN',
                                            rhs=ExprNumber(number=2)),
                                  EnumField(name='COLOR_BLUE',
                                            rhs=ExprNumber(number=4)),
                                 EnumField(name='COLOR_HIGHLIGHT',
                                           rhs=ExprNumber(number=8))],
                          is_pointer=False,
                          variables=[])])
```

Вывод

Получил навыки составления грамматик и проектирования синтаксических деревьев.