Лабораторная работа № 2.2 «Абстрактные синтаксические деревья»

1 апреля 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

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

Индивидуальный вариант

Объявления типов и констант в Паскале:

В `record`'е точка с запятой разделяет поля и после `case` дополнительный `end` не ставится. См. https://bernd-oppolzer.de/PascalReport.pdf, третья с конца страница.

```
Type
  Coords = Record x, y: INTEGER end;
Const
  MaxPoints = 100;
type
  CoordsVector = array 1..MaxPoints of Coords;

(* графический и текстовый дисплеи *)
const
  Heigh = 480;
  Width = 640;
  Lines = 24
  Columns = 80;
type
  BaseColor = (red, green, blue, highlited);
  Color = set of BaseColor;
  GraphicScreen = array 1..Heigh of array 1..Width of Color;
  TextScreen = array 1..Lines of array 1..Columns of
    record
      Symbol : CHAR;
```

```
      SymColor : Color;
      BackColor : Color
    end;

{ определения токенов }
TYPE
  Domain = (Ident, IntNumber, RealNumber);
  Token = record
    fragment : record
      start, following : record
        row, col : INTEGER
      end
    end;
    case tokType : Domain of
      Ident : (
        name : array 1..32 of CHAR
      );
      IntNumber : (
        intval : INTEGER
      );
      RealNumber : (
        realval : REAL
      )
  end;

  Year = 1900..2050;

  List = record
    value : Token;
    next : ^List
  end;
```

Ключевые слова и идентификаторы не чувствительны к регистру.

Реализация

Абстрактный синтаксис

```
Program             → Block
Block               → CONST ConstDefs
                    | TYPE TypeDefs
ConstDefs           → ConstDef*
ConstDef            → Identifier = Constant ;
TypeDefs            → TypeDef*
TypeDef             → Identifier = Type ;
Constant            → [+-]? (ConstantIdentifier|UnsignedNumber)
```

```
                    | ' character+ '
Type                → SimpleType
                    | ^ TypeIdentifier
                    | PACKED? (
                       ARRAY [ SimpleType (, SimpleType)* OF Type ]
                        | FILE OF SimpleType
                        | RECORD FieldList END
                    )
FieldList         → Identifier (, Identifier)* : Type (; FieldList)?
                    | CASE Identifier : TypeIdentifier OF
                        Constant (, Constant)* : ( FieldList )
                       (; Constant (, Constant)* : ( FieldList ))*
SimpleType          → TypeIdentifier
                    | ( Identifier (, Identifier)* )
                    | Constant . . Constant
```

Лексическая структура и конкретный синтаксис

Лексическая структура в порядке убывания приоритетов доменов

```
TypeIdentifier      → INTEGER
                    | BOOLEAN
                    | REAL
                    | CHAR
                    | TEXT
ConstantIdentifier  → Identifier
Identifier          → Letter (Letter|Digit)*
Letter              → [a-zA-Z]
UnsignedInteger     → Digit+
UnsignedNumber            →  UnsignedInteger  (.  Digit+)?  (E  [+-
]? UnsignedInteger)?
Digit               → [0-9]
```

Конкретный синтаксис представлен непосредственно в коде программы

Программная реализация

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

```python
# constant
class UnarSign(enum.Enum):
    Plus = 'PLUS'
    Minus = 'MINUS'

@dataclass
class Identifier:
    name : str

@dataclass
class ConstantIdentifier:
    identifier : Identifier

class Constant(abc.ABC): ...

@dataclass
class SignedIdentifierConstant(Constant):
    unar_sign : UnarSign
    constant_identifier : ConstantIdentifier

@dataclass
class UnsignedIdentifierConstant(Constant):
    constant_identifier : ConstantIdentifier

@dataclass
class SignedNumberConstant(Constant):
    unar_sign : UnarSign
    unsingned_number : float

@dataclass
class UnsignedNumberConstant(Constant):
    unsingned_number : float

@dataclass
class CharacterConstant(Constant):
    char_sequence : str

# simple type
class TypeIdentifier(abc.ABC): ...

class EnumTypeIdentifier(enum.Enum):
    Integer = 'INTEGER'
    Boolean = 'BOOLEAN'
    Real = 'REAL'
    Char = 'CHAR'
    Text = 'TEXT'
```

```python
@dataclass
class CommonTypeIdentifier(TypeIdentifier):
    common_type_identifier : EnumTypeIdentifier

@dataclass
class IdentifierTypeIdentificator:
    identifier : Identifier

class SimpleType(abc.ABC): ...

@dataclass
class DefaultSimpleType(SimpleType):
    type_identifier : TypeIdentifier

@dataclass
class ListSimpleType(SimpleType):
    identifier_list : tuple[Identifier]

@dataclass
class BoundedSimpleType(SimpleType):
    left_constant : Constant
    right_constant : Constant

# type
class Type(abc.ABC): ...

@dataclass
class DefaultType(Type):
    simple_type : SimpleType

@dataclass
class RefType(Type):
    type_identifier : TypeIdentifier

@dataclass
class PackedType(Type):
    simple_type : SimpleType

@dataclass
class ArrayType(Type):
    simple_types : tuple[SimpleType]
    type : Type

class FileType(DefaultType): ...
```

```
class SetType(DefaultType): ...

class RecordType(DefaultType): ...

# field list
@dataclass
class IdentifierWithType:
    identifier_list : tuple[Identifier]
    type : Type

@dataclass
class CaseVariant:
    class FieldList: ...

    constant_list : tuple[Constant]
    field_list : FieldList

@dataclass
class CaseBlock:
    identifier : Identifier
    type_identifier : TypeIdentifier
    case_variant_sequence : tuple[CaseVariant]

@dataclass
class FieldList:
    identifier_with_types_list : tuple[IdentifierWithType]
    case_block : typing.Optional[CaseBlock] = None

# block
class Block(abc.ABC): ...

@dataclass
class BlockConst(Block):
    identifier : Identifier
    constant : Constant

@dataclass
class BlockType(Block):
    identifier : Identifier
    type : Type

# program
@dataclass
class Program:
    block : Block
```

```
UNAR_SIGN = pe.Terminal(
    'UNAR_SIGN',
    r'[+-]?',
    str
)
IDENTIFIER = pe.Terminal(
    'IDENTIFIER',
    r'[a-zA-Z][a-zA-Z0-9]*',
    str.upper
)
UNSINGNED_NUMBER = pe.Terminal(
    'UNSINGNED_NUMBER',
    r'[0-9]+(\.[0-9]+)?(E[+-]?[0-9]+)?',
    float
)
CHAR_SEQUENCE = pe.Terminal(
    'CHAR_SEQUENCE',
    r'(?<=\')[^\']+(?=\')',
    str
)

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

KW_INTEGER  = make_keyword('INTEGER')
KW_BOOLEAN  = make_keyword('BOOLEAN')
KW_REAL     = make_keyword('REAL')
KW_CHAR     = make_keyword('CHAR')
KW_TEXT     = make_keyword('TEXT')
KW_PACKED   = make_keyword('PACKED')
KW_ARRAY    = make_keyword('ARRAY')
KW_OF       = make_keyword('OF')
KW_FILE     = make_keyword('FILE')
KW_SET      = make_keyword('SET')
KW_RECORD   = make_keyword('RECORD')
KW_END      = make_keyword('END')
KW_CASE     = make_keyword('CASE')
KW_CONST    = make_keyword('CONST')
KW_TYPE     = make_keyword('TYPE')

# constant
NConstant               = pe.NonTerminal('constant')
NUnarSign               = pe.NonTerminal('unar sign')
```

```
NConstantIdentifier     = pe.NonTerminal('constant identifier')
# simple type
NSimpleType             = pe.NonTerminal('simple type')
NIdentifierList         = pe.NonTerminal('identifier list')
NTypeIdentifier         = pe.NonTerminal('type identifier')
NCommonTypeIdentifier   = pe.NonTerminal('common type identifier')
# type
NType                   = pe.NonTerminal('type')
NTypeAfterPacked        = pe.NonTerminal('type after packed')
NSimpleTypeList         = pe.NonTerminal('simple type list')
# field list
NFieldList              = pe.NonTerminal('field list')
NIdentifierWithTypeList = pe.NonTerminal('identifier with type list')
NIdentifierWithTypeSeq  = pe.NonTerminal('identifier with type seq')
NIdentifierWithType     = pe.NonTerminal('identifier with type')
NCaseBlock              = pe.NonTerminal('case block')
NCaseVariantSequence    = pe.NonTerminal('case block sequence')
NCaseVariant            = pe.NonTerminal('case block')
NConstantList           = pe.NonTerminal('constant list')
# block
NBlock                  = pe.NonTerminal('block')
NBlockConstSequence     = pe.NonTerminal('block const sequence')
NBlockConst             = pe.NonTerminal('block const')
NBlockTypeSequence      = pe.NonTerminal('block type sequence')
NBlockType              = pe.NonTerminal('block type')
# program
NProgram                = pe.NonTerminal('program')

# constant
NConstant |= NUnarSign, NConstantIdentifier, SignedIdentifierConstant
NConstant |= NConstantIdentifier, UnsignedIdentifierConstant
NConstant |= NUnarSign, UNSINGNED_NUMBER, SignedNumberConstant
NConstant |= UNSINGNED_NUMBER, UnsignedNumberConstant
NConstant |= '\'', CHAR_SEQUENCE, '\'', CharacterConstant

NUnarSign |= '+', lambda: UnarSign.Plus
NUnarSign |= '-', lambda: UnarSign.Minus

NConstantIdentifier |= IDENTIFIER

# simple type
NSimpleType |= NTypeIdentifier, DefaultSimpleType
NSimpleType |= '(', NIdentifierList, ')', ListSimpleType
NSimpleType |= NConstant, '..', NConstant, BoundedSimpleType

NTypeIdentifier |= NCommonTypeIdentifier, CommonTypeIdentifier
```

```
NTypeIdentifier |= IDENTIFIER, IdentifierTypeIdentificator

NCommonTypeIdentifier |= KW_INTEGER, lambda: EnumTypeIdentifier.Integer
NCommonTypeIdentifier |= KW_BOOLEAN, lambda: EnumTypeIdentifier.Boolean
NCommonTypeIdentifier |= KW_REAL, lambda: EnumTypeIdentifier.Real
NCommonTypeIdentifier |= KW_CHAR, lambda: EnumTypeIdentifier.Char
NCommonTypeIdentifier |= KW_TEXT, lambda: EnumTypeIdentifier.Text

NIdentifierList |= IDENTIFIER, lambda id: (id,)
NIdentifierList |= (
    IDENTIFIER, ',', NIdentifierList,
    lambda id, idlist: (id, *idlist)
)

# type
NType |= NSimpleType, DefaultType
NType |= '^', NTypeIdentifier, RefType
NType |= KW_PACKED, NTypeAfterPacked, PackedType
NType |= NTypeAfterPacked

NTypeAfterPacked |= (
    KW_ARRAY, NSimpleTypeList, KW_OF, NType,
    ArrayType
)
NTypeAfterPacked |= KW_FILE, KW_OF, NType, FileType
NTypeAfterPacked |= KW_SET, KW_OF, NSimpleType, SetType
NTypeAfterPacked |= KW_RECORD, NFieldList, KW_END, RecordType

NSimpleTypeList |= NSimpleType, lambda st: (st,)
NSimpleTypeList |= (
    NSimpleType, ',', NSimpleTypeList,
    lambda st, stlist: (st, *stlist)
)

# field list
NFieldList |= NIdentifierWithTypeList, FieldList
NFieldList |= NIdentifierWithTypeSeq, NCaseBlock, FieldList

NIdentifierWithTypeList |= NIdentifierWithType, lambda iwt: (iwt,)
NIdentifierWithTypeList |= (
    NIdentifierWithType, ';', NIdentifierWithTypeList,
    lambda iwt, iwtlist: (iwt, *iwtlist)
)

NIdentifierWithTypeSeq |= NIdentifierWithType, ';', lambda iwt: (iwt,)
NIdentifierWithTypeSeq |= (
```

```
    NIdentifierWithType, ';', NIdentifierWithTypeSeq,
    lambda iwt, iwtseq: (iwt, *iwtseq)
)

NIdentifierWithType |= NIdentifierList, ':', NType, IdentifierWithType

NCaseBlock |= (
    KW_CASE, IDENTIFIER, ':', NTypeIdentifier, KW_OF,
    NCaseVariantSequence,
    CaseBlock
)

NCaseVariantSequence |= NCaseVariant, lambda cblock: (cblock,)
NCaseVariantSequence |= (
    NCaseVariant, ';', NCaseVariantSequence,
    lambda cb, cbseq: (cb, *cbseq)
)

NCaseVariant |= (
    NConstantList, ':', '(', NFieldList, ')', CaseVariant
)

NConstantList |= NConstant, lambda c: (c,)
NConstantList |= (
    NConstant, ',', NConstantList,
    lambda c, clist: (c, *clist)
)

# block
NBlock |= (
    KW_CONST, NBlockConstSequence, NBlock,
    lambda bcseq, block: (bcseq, *block)
)
NBlock |= (
    KW_TYPE, NBlockTypeSequence, NBlock,
    lambda btseq, block: (btseq, *block)
)
NBlock |= lambda: ()

NBlockConstSequence |= NBlockConst, lambda bc: (bc,)
NBlockConstSequence |= (
    NBlockConst, NBlockConstSequence,
    lambda bc, bcseq: (bc, *bcseq)
)

NBlockConst |= IDENTIFIER, '=', NConstant, ';', BlockConst
```

```
NBlockTypeSequence |= NBlockType, lambda bt: (bt,)
NBlockTypeSequence |= (
    NBlockType, NBlockTypeSequence,
    lambda bt, btseq: (bt, *btseq)
)

NBlockType |= IDENTIFIER, '=', NType, ';', BlockType

# program
NProgram |= NBlock, Program


p = pe.Parser(NProgram)
assert p.is_lalr_one()

p.add_skipped_domain(r'\s')
p.add_skipped_domain(r'{[^}]*}')
p.add_skipped_domain(r'\(\*([^*]|\*[^\)])*\*\)')

for filename in sys.argv[1:]:
    try:
        with open(filename) as f:
            tree = p.parse(f.read())
            pprint(tree)
    except pe.Error as e:
        print(f'Ошибка {e.pos}: {e.message}')
    except Exception as e:
        print(e)
```

Тестирование

Входные данные

```
Type
  Coords = Record x, y: INTEGER end;
Const
  MaxPoints = 100;
type
  CoordsVector = array 1..MaxPoints of Coords;

const
  Heigh = 480;
  Width = 640;
  Lines = 24;
```

```
  Columns = 80;
type
  BaseColor = (red, green, blue, highlited);
  Color = set of BaseColor;
  GraphicScreen = array 1..Heigh of array 1..Width of Color;
  TextScreen = array 1..Lines of array 1..Columns of
    record
      Symbol : CHAR;
      SymColor : Color;
      BackColor : Color
    end;

(* определения токенов }
{ определения токенов *)
(* определения токенов *)
{ определения токенов }
{ определения токенов *)
(* определения токенов }
TYPE
  Domain = (Ident, IntNumber, RealNumber);
  Token = record
    fragment : record
      start, following : record
        row, col : INTEGER
      end
    end;
    case tokType : Domain of
      Ident : (
        name : array 1..32 of CHAR
      );
      IntNumber : (
        intval : INTEGER
      );
      RealNumber : (
        realval : REAL
      )
  end;

  Year = 1900..2050;

  List = record
    value : Token;
    next : ^List
  end;
```

Вывод на `stdout`

```
Program(block=((BlockType(identifier='COORDS',
                  type=RecordType(simple_type=FieldList(identifier_with_types_list=(Identifie
                                                                                    'Y'),
                                                                          type=DefaultType(s
                                        case_block=None))),),
               (BlockConst(identifier='MAXPOINTS',
                  constant=UnsignedNumberConstant(unsingned_number=100.0)),),
               (BlockType(identifier='COORDSVECTOR',
                  type=ArrayType(simple_types=(BoundedSimpleType(left_constant=UnsignedNumber
                                               right_constant=UnsignedIdentifierConstant(co
                                 type=DefaultType(simple_type=DefaultSimpleType(type_identifier=Id
               (BlockConst(identifier='HEIGH',
                  constant=UnsignedNumberConstant(unsingned_number=480.0)),
                BlockConst(identifier='WIDTH',
                  constant=UnsignedNumberConstant(unsingned_number=640.0)),
                BlockConst(identifier='LINES',
                  constant=UnsignedNumberConstant(unsingned_number=24.0)),
                BlockConst(identifier='COLUMNS',
                  constant=UnsignedNumberConstant(unsingned_number=80.0))),
               (BlockType(identifier='BASECOLOR',
                  type=DefaultType(simple_type=ListSimpleType(identifier_list=('RED',
                                                               'GREEN',
                                                                'BLUE',
                                                               'HIGHLITED')))),
                BlockType(identifier='COLOR',
                  type=SetType(simple_type=DefaultSimpleType(type_identifier=IdentifierTypeId
                BlockType(identifier='GRAPHICSCREEN',
                  type=ArrayType(simple_types=(BoundedSimpleType(left_constant=UnsignedNumber
                                               right_constant=UnsignedIdentifierConstant(co
                                 type=ArrayType(simple_types=(BoundedSimpleType(left_constant=Unsi
                                                               right_constant=UnsignedIdentifierC
                                                type=DefaultType(simple_type=DefaultSimpleType(type_ide
                BlockType(identifier='TEXTSCREEN',
                  type=ArrayType(simple_types=(BoundedSimpleType(left_constant=UnsignedNumber
                                               right_constant=UnsignedIdentifierConstant(co
                                 type=ArrayType(simple_types=(BoundedSimpleType(left_constant=Unsi
                                                               right_constant=UnsignedIdentifierC
                                                type=RecordType(simple_type=FieldList(identifier_with_t

                                                                                    Identifierw

                                                                                    Identifierw

                                                                  case_block=None))))))),
```

```
(BlockType(identifier='DOMAIN',
  type=DefaultType(simple_type=ListSimpleType(identifier_list=('IDENT',
                                                'INTNUMBER',
                                                'REALNUMBER')))),
 BlockType(identifier='TOKEN',
  type=RecordType(simple_type=FieldList(identifier_with_types_list=(Identifie
                                                                    type=RecordType(si
                                  case_block=CaseBlock(identifier='TOKTYPE',
                                                       type_identifier=IdentifierTypeIdenti
                                                       case_variant_sequence=(CaseVariant(c
                                                                              field_list=F
                                                                    CaseVariant(constant
                                                                              field_list=F
                                                                    CaseVariant(constant
                                                                              field_list=F
 BlockType(identifier='YEAR',
  type=DefaultType(simple_type=BoundedSimpleType(left_constant=UnsignedNumber
                                       right_constant=UnsignedNumberConstant(unsing
 BlockType(identifier='LIST',
  type=RecordType(simple_type=FieldList(identifier_with_types_list=(Identifie
                                                                    type=DefaultType(s
                                                  IdentifierWithType(identifier_l
                                                                    type=RefType(type_
                                     case_block=None))))))
```

Вывод

В результате выполнения данной работы были получены навыки составления грамматик и проектирования синтаксических деревьев.