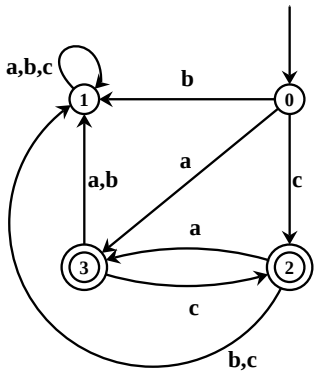
UNIVERSITÄT KONSTANZ
Fachbereich Informatik & Informationswissenschaft
Prof. Dr. D. Wagner / C. Fieß / M. Gaertler / F. Schulz

Klausur:	**Haupttermin**	**Mo, 15.07.2002, 12:00 - 14:00 Uhr, R611**
	Nachtermin	**Mi, 09.10.2002, 14:00 - 16:00 Uhr, R711**

13. Übungsblatt

Aufgabe 1: Geben Sie eine rechtslineare Grammatik an, welche die Sprache erzeugt, die folgender deterministische endliche Automat aus Aufgabe 3 des 1. Übungsblatts akzeptiert:



Aufgabe 2: Betrachten Sie die regulären Ausdrücke z, n, g und r aus Aufgabe 1 des 3. Übungsblatts (das Alphabet war $\Sigma = \{0,1,2,3,4,5,6,7,8,9,.,e,E,+,-\}$):

$$\begin{aligned} z &= 0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9 \\ n &= zz^* \\ g &= (+ \cup - \cup \varepsilon)n \\ r &= g(.n \cup \varepsilon)((e \cup E)g \cup \varepsilon) \end{aligned}$$

a) Geben Sie eine rechtslineare Grammatik über dem Alphabet Σ an, deren erzeugte Sprache die Vereinigung der regulären Ausdrücke z, n und g ist.

b) Kann die Sprache jedes regulären Ausdrucks durch eine rechtslineare Grammatik erzeugt werden?

c) Geben Sie eine Grammatik ihrer Wahl an, deren erzeugte Sprache gleich der durch r beschriebenen Sprache ist.

Aufgabe 3: Die Sprache der korrekten Klammerausdrücke kann definiert werden durch:

- ε ist ein korrekter Klammerausdruck
- Ist w ein korrekter Klammerausdruck, so auch (w)
- Sind v und w korrekte Klammerausdrücke, so auch vw
- Sonst ist nichts korrekter Klammerausdruck

Geben Sie eine kontextfreie Grammatik an, die die Sprache der korrekten Klammerausdrücke erzeugt.

Aufgabe 4: Zeigen Sie, dass die Sprache $\{ww | w \in \Sigma^*\}$ über dem Alphabet $\Sigma = \{0,1\}$ nicht durch eine kontextfreie Grammatik erzeugt werden kann.

The Grammar of the Java Programming Language

Taken from: Java Language Specification
Second Edition
http://java.sun.com/docs/books/jls/

The grammar below uses the following BNF-style conventions:

[x] denotes zero or one occurrences of x.
{x} denotes zero or more occurrences of x.
x | y means one of either x or y.

The Grammar:

```
Identifier:
        IDENTIFIER

QualifiedIdentifier:
        Identifier { . Identifier }

Literal:
        IntegerLiteral
        FloatingPointLiteral
        CharacterLiteral
        StringLiteral
        BooleanLiteral
        NullLiteral

Expression:
        Expression1 [AssignmentOperator Expression1]]

AssignmentOperator:
        =
        +=
        -=
        *=
        /=
        &=
        |=
        ^=
        %=
        <<=
        >>=
        >>>=

Type:
        Identifier {   .   Identifier } BracketsOpt
        BasicType

StatementExpression:
        Expression

ConstantExpression:
        Expression

Expression1:
        Expression2 [Expression1Rest]

Expression1Rest:
        [  ?   Expression   :   Expression1]

Expression2 :
        Expression3 [Expression2Rest]

Expression2Rest:
        {Infixop Expression3}
        Expression3 instanceof Type

Infixop:
        ||
        &&
        |
        ^
        &
        ==
        !=
        <
        >
        <=
        >=
        <<
        >>
        >>>
        +
        -
        *
        /
        %

Expression3:
        PrefixOp Expression3
        (   Expr | Type   )   Expression3
        Primary {Selector} {PostfixOp}

Primary:
        ( Expression )
        this [Arguments]
        super SuperSuffix
        Literal
        new Creator
        Identifier { . Identifier }[ IdentifierSuffix]
        BasicType BracketsOpt .class
        void.class

IdentifierSuffix:
        [ ( ] BracketsOpt   .   class | Expression ])
        Arguments
        .   ( class | this | super Arguments | new InnerCreator )

PrefixOp:
        ++
        --
        !
        ~
        +
        -

PostfixOp:
        ++
        --

Selector:
        . Identifier [Arguments]
        . this
        . super SuperSuffix
        . new InnerCreator
        [ Expression ]

SuperSuffix:
        Arguments
        . Identifier [Arguments]

BasicType:
        byte
        short
        char
        int
        long
        float
        double
        boolean

ArgumentsOpt:
        [ Arguments ]

Arguments:
        ( [Expression { , Expression }] )

BracketsOpt:
        {[]}

Creator:
        QualifiedIdentifier ( ArrayCreatorRest  | ClassCreatorRest )

InnerCreator:
        Identifier ClassCreatorRest

ArrayCreatorRest:
        [ ( ] BracketsOpt ArrayInitializer | Expression ] {[ Expression ]} BracketsOpt )

ClassCreatorRest:
        Arguments [ClassBody]

ArrayInitializer:
        { [VariableInitializer {, VariableInitializer} [,] ] }

VariableInitializer:
        ArrayInitializer
        Expression

ParExpression:
        ( Expression )

Block:
        { BlockStatements }

BlockStatements:
        { BlockStatement }

BlockStatement :
        LocalVariableDeclarationStatement
        ClassOrInterfaceDeclaration
        [Identifier :] Statement

LocalVariableDeclarationStatement:
        [final] Type VariableDeclarators   ;

Statement:
        Block
        if ParExpression Statement [else Statement]
        for ( ForInitOpt   ;   [Expression]   ;   ForUpdateOpt ) Statement
        while ParExpression Statement
        do Statement while ParExpression   ;
        try Block ( Catches | [Catches] finally Block )
        switch ParExpression { SwitchBlockStatementGroups }
        synchronized ParExpression Block
        return [Expression] ;
        throw Expression   ;
        break [Identifier]
        continue [Identifier]
        ;
        ExpressionStatement
        Identifier   :   Statement

Catches:
        CatchClause {CatchClause}

CatchClause:
        catch ( FormalParameter ) Block

SwitchBlockStatementGroups:
        { SwitchBlockStatementGroup }

SwitchBlockStatementGroup:
        SwitchLabel BlockStatements

SwitchLabel:
        case ConstantExpression   :
        default:

MoreStatementExpressions:
        { , StatementExpression }

ForInit:
        StatementExpression MoreStatementExpressions
        [final] Type VariableDeclarators

ForUpdate:
        StatementExpression MoreStatementExpressions

ModifiersOpt:
        { Modifier }

Modifier:
        public
        protected
        private
        static
        abstract
        final
        native
        synchronized
        transient
        volatile
        strictfp

VariableDeclarators:
        VariableDeclarator { ,   VariableDeclarator }

VariableDeclaratorsRest:
        VariableDeclaratorRest { ,   VariableDeclarator }

ConstantDeclaratorsRest:
        ConstantDeclaratorRest { ,   ConstantDeclarator }

VariableDeclarator:
        Identifier VariableDeclaratorRest

ConstantDeclarator:
        Identifier ConstantDeclaratorRest

VariableDeclaratorRest:
        BracketsOpt [  =   VariableInitializer]

ConstantDeclaratorRest:
        BracketsOpt   =   VariableInitializer

VariableDeclaratorId:
        Identifier BracketsOpt

CompilationUnit:
        [package QualifiedIdentifier   ;  ] {ImportDeclaration}
{TypeDeclaration}

ImportDeclaration:
        import Identifier {   .   Identifier } [   .     *   ] ;

TypeDeclaration:
        ClassOrInterfaceDeclaration
        ;

ClassOrInterfaceDeclaration:
        ModifiersOpt (ClassDeclaration | InterfaceDeclaration)

ClassDeclaration:
        class Identifier [extends Type] [implements TypeList] ClassBody

InterfaceDeclaration:
        interface Identifier [extends TypeList] InterfaceBody

TypeList:
        Type {  ,   Type}

ClassBody:
        { {ClassBodyDeclaration} }

InterfaceBody:
        { {InterfaceBodyDeclaration} }

ClassBodyDeclaration:
        ;
        [static] Block
        ModifiersOpt MemberDecl

MemberDecl:
        MethodOrFieldDecl
        void Identifier MethodDeclaratorRest
        Identifier ConstructorDeclaratorRest
        ClassOrInterfaceDeclaration

MethodOrFieldDecl:
        Type Identifier MethodOrFieldRest

MethodOrFieldRest:
        VariableDeclaratorRest
        MethodDeclaratorRest

InterfaceBodyDeclaration:
        ;
        ModifiersOpt InterfaceMemberDecl

InterfaceMemberDecl:
        InterfaceMethodOrFieldDecl
        void Identifier VoidInterfaceMethodDeclaratorRest
        ClassOrInterfaceDeclaration

InterfaceMethodOrFieldDecl:
        Type Identifier InterfaceMethodOrFieldRest

InterfaceMethodOrFieldRest:
        ConstantDeclaratorsRest ;
        InterfaceMethodDeclaratorRest

MethodDeclaratorRest:
                FormalParameters BracketsOpt [throws QualifiedIdentifierList] (
MethodBody |   ;  )

VoidMethodDeclaratorRest:
                FormalParameters [throws QualifiedIdentifierList] ( MethodBody |   ;  )

InterfaceMethodDeclaratorRest:
        FormalParameters BracketsOpt [throws QualifiedIdentifierList]   ;

VoidInterfaceMethodDeclaratorRest:
        FormalParameters [throws QualifiedIdentifierList]
  ;

ConstructorDeclaratorRest:
        FormalParameters [throws QualifiedIdentifierList]
MethodBody

QualifiedIdentifierList:
        QualifiedIdentifier {  ,   QualifiedIdentifier}

FormalParameters:
        ( [FormalParameter { , FormalParameter}] )

FormalParameter:
        [final] Type VariableDeclaratorId

MethodBody:
        Block
```