Anhang A

Mini-Compiler in Java

```
/* ============================= Mini.java ================================

The Mini language is a modified small subset of Java/C.
A Mini program consists of a single possibly recursive function.
The language has no declarations (implicit type is integer).

Mini.java is a compiler-interpreter for Mini written in Java.
The compiler generates code for a virtual machine, which is a modified
small subset of the Java VM (integer code instead of byte code).

The one-pass compiler is implemented as a top-down recursive descent
parser calling the methods of lexical analysis and code generation.
The parser routines correspond to the the grammar rules in EBNF Notation.
The lexical analysis takes advantage of the Java class StreamTokenizer.

====================== source language syntax (EBNF) =====================

Program    = Function
Function   = identifier "(" identifier ")" Block
Block      = "{" [Statements] "}"
Statements = Statement Statements
Statement  = identifier "=" Expression ";" |
             "if" Condition Statement "else" Statement |
             "while" Condition Statement |
             "return" Expression ";" |
             Block |
             ";"
Condition  = "(" Expression ("=="|"!="|">"|"<") Expression ")"
Expression = Term {("+"|"-") Term}
Term       = Factor {("*"|"/") Factor}
Factor     = number |
             identifier |
             "(" Expression ")" |
             identifier "(" Expression ")"

================================ VM code =================================

 0    do nothing
 1 c  push constant c onto stack
```

```
 2 v  load variable v onto stack
 3 v  store stack value into variable v
 4    add two top elements of stack, replace by result
 5    subtract ...
 6    multiply ...
 7    divide ...
 8 a  jump to a if the two top elements of stack are equal
 9 a  jump if ... not equal
10 a  jump if ... less or equal
11 a  jump if ... greater or equal
12 a  unconditional jump to a
13 a  jump to subroutine start address a
14    return from function
15    stop ececution

================================ example =================================

source file "fac.mini":
-----------------------
fac(n) {
    if (n == 0)
        return 1;
    else
        return n * fac(n-1);
}

run:
----
<Java Compiler> Mini.java
<Java VM> Mini fac.mini 8

VMCode:13 3 15 2 1 1 0 9 14 1 1 14 12 25 2 1 2 1 1 1 5 13 3 6 14 0
Result:40320

======================================================================== */

//////////////////////////////////////////////////////////////////////////

import java.io.*;

public class Mini implements Globals {
    static String stab[] = new String[10]; // symbol table
    static int nv = 0;                     // number of variables
    static int code_size = 1000;
    static int stack_size = 100000;

    public static void main(String args[]) {
        Lex.init(args[0]);                   // init lexer
        Gen.init(code_size);                 // init generator
        Parser.program();                    // call parser
        Gen.show();                          // show VM code
        VM.init(Gen.code,nv,stack_size);     // initialize VM
        int arg = Integer.parseInt(args[1]); // input argument
        int res = VM.exec(arg);              // call VM
```

```
        System.out.println("Result:"+res);   // print result
    }

    protected static void error(String s,int n) {
        System.out.println("error "+s+" "+n);
        System.exit(0);
    }
}

//////////////////////////////////////////////////////////////////////////

interface Globals {

    // token

    final static int T_num = 1; // number
    final static int T_id  = 2; // identifier
    final static int T_eql = 3; // ==
    final static int T_neq = 4; // !=
    final static int T_grt = '>';
    final static int T_les = '<';
    final static int T_add = '+';
    final static int T_sub = '-';
    final static int T_mul = '*';
    final static int T_div = '/';
    final static int T_lbr = '(';
    final static int T_rbr = ')';
    final static int T_clb = '{';
    final static int T_crb = '}';
    final static int T_com = ',';
    final static int T_sem = ';';
    final static int T_ass = '=';
    final static int T_eof = '$';

    final static String kw[]  = {"if","else","while","return"};
    final static int kw0 = 10;
    final static int T_if     = kw0+0;
    final static int T_else   = kw0+1;
    final static int T_while  = kw0+2;
    final static int T_return = kw0+3;

    // target code (VM)

    final static int M_nop      =  0;
    final static int M_push     =  1;
    final static int M_load     =  2;
    final static int M_store    =  3;
    final static int M_add      =  4;
    final static int M_sub      =  5;
    final static int M_mul      =  6;
    final static int M_div      =  7;
    final static int M_if_cmpeq =  8;
    final static int M_if_cmpne =  9;
    final static int M_if_cmple = 10;
```

```
    final static int M_if_cmpge = 11;
    final static int M_goto     = 12;
    final static int M_jsr      = 13;
    final static int M_return   = 14;
    final static int M_stop     = 15;
}

///////////////////////////////////////////////////////////////////////////

// lexical analysis

class Lex extends Mini {
    static int num_val;   // attribute of number
    static int id_val;    // attribute of id (index of symbol table)

    private static FileInputStream source; // imported class
    private static StreamTokenizer st;     // imported class
    private static int tok;                // tokenizer token

    static void init(String file_name) {
        try {
            source = new FileInputStream(file_name);
        }
        catch (FileNotFoundException e) {
            error("filename"+" "+e.getMessage(),0);
        }
        st = new StreamTokenizer(source);
        st.ordinaryChar('/');
        st.ordinaryChar('-');
    }

    static int scan() {
        try {
            tok = st.nextToken();
            switch(tok) {
            case StreamTokenizer.TT_EOF:
                return T_eof;
            case StreamTokenizer.TT_NUMBER:
                num_val = (int)st.nval;
                return T_num;
            case StreamTokenizer.TT_WORD:
                int i = look_kw(st.sval);
                if (i >= 0)
                    return kw0+i;
                else {
                    id_val = enter_id(st.sval);
                    return T_id;
                }
            default:
                char c = (char)tok;
                switch(c) {
                case '=':
                    if ((char)st.nextToken() == '=')
                        return T_eql;
```

```
                    else
                        st.pushBack();
                    break;
                case '!':
                    if ((char)st.nextToken() == '=')
                        return T_neq;
                    else
                        st.pushBack();
                    break;
                }
            }
        }
        catch (IOException e) {
            error("i/o"+" "+e.getMessage(),0);
        }
        return tok;
    }

    private static int look_kw(String s) {
        int i;
        for (i = 0;i < kw.length && !s.equals(kw[i]);i++) {}
        if (i < kw.length)
            return i;
        else
            return -1;
    }

    private static int enter_id(String s) {
        int i;
        for (i = 0;i < nv && !s.equals(stab[i]);i++) {}
        if (i == nv) {
            stab[i] = s;
            nv++;
        }
        return i;
    }
}

/////////////////////////////////////////////////////////////////////////////

// parser (generates VM code)

class Parser extends Mini {
    private static int token;

    static void program() {
        next();
        Gen.start();
        function();
    }

    private static void function() {
        match(T_id);
        match(T_lbr);
```

```
    match(T_id);
    match(T_rbr);
    block();
    Gen.instr(M_nop);
}

private static void block() {
    match(T_clb);
    statements();
    match(T_crb);
}

private static void statements() {
    if (token != T_crb) {
        statement();
        statements();
    }
}

private static void statement() {
    int p1,p2;
    switch(token) {
    case T_id:
        int adr = Lex.id_val;
        next();
        match(T_ass);
        expression();
        Gen.instr(M_store,adr);
        match(T_sem);
        break;
    case T_if:
        next();
        condition();
        p1 = Gen.pc-1;
        statement();
        Gen.instr(M_goto,0);
        p2 = Gen.pc-1;
        match(T_else);
        Gen.setjump(p1);
        statement();
        Gen.setjump(p2);
        break;
    case T_while:
        next();
        p1 = Gen.pc;
        condition();
        p2 = Gen.pc-1;
        statement();
        Gen.instr(M_goto,p1);
        Gen.setjump(p2);
        break;
    case T_return:
        next();
        expression();
```

```
        Gen.instr(M_return);
        match(T_sem);
        break;
    case T_clb:
        block();
        break;
    case T_sem:
        next();
        break;
    default:
        error("syntax",token);
    }
}

private static void condition() {
    match(T_lbr);
    expression();
    int rop = token;
    next();
    expression();
    match(T_rbr);
    switch(rop) {
    case T_eql:
        Gen.instr(M_if_cmpne,0);
        break;
    case T_neq:
        Gen.instr(M_if_cmpeq,0);
        break;
    case T_grt:
        Gen.instr(M_if_cmple,0);
        break;
    case T_les:
        Gen.instr(M_if_cmpge,0);
        break;
    default:
        error("syntax",token);
    }
}

private static void expression() {
    term();
    while (token == T_add || token == T_sub) {
        switch (token) {
        case T_add:
            next();
            term();
            Gen.instr(M_add);
            break;
        case T_sub:
            next();
            term();
            Gen.instr(M_sub);
            break;
        }
```

```
    }
}

private static void term() {
    factor();
    while (token == T_mul || token == T_div) {
        switch (token) {
        case T_mul:
            next();
            term();
            Gen.instr(M_mul);
            break;
        case T_div:
            next();
            term();
            Gen.instr(M_div);
            break;
        }
    }
}

private static void factor() {
    switch(token) {
    case T_num:
        Gen.instr(M_push,Lex.num_val);
        next();
        break;
    case T_id:
        if (Lex.id_val != 0) {
            Gen.instr(M_load,Lex.id_val);
            next();
        }
        else {
            next();
            match(T_lbr);
            expression();
            match(T_rbr);
            Gen.instr(M_jsr,Gen.start_adr);
        }
        break;
    case T_lbr:
        next();
        expression();
        match(T_rbr);
        break;
    default:
        error("syntax",token);
    }
}

private static void next() {
    token = Lex.scan();
}
```

```
    private static void match(int x) {
        if (token == x)
            next();
        else
            error("syntax",token);
    }
}

//////////////////////////////////////////////////////////////////////////

// code generator

class Gen extends Mini {
    static int code[];    // target code
    static int pc;        // program counter
    static int start_adr; // start address

    static void init(int code_size) {
        code = new int[code_size];
        pc = 0;
    }

    static void start() {
        instr(M_jsr,3);
        instr(M_stop);
        start_adr = pc;
    }

    static void instr(int operator) {
        code[pc] = operator;
        pc = pc+1;
    }

    static void instr(int operator,int operand) {
        code[pc] = operator;
        code[pc+1] = operand;
        pc = pc+2;
    }

    static void setjump(int adr) {
        code[adr] = pc;
    }

    static void show() {
        System.out.print("VMCode:");
        for (int i = 0;i < pc;i++)
            System.out.print(code[i]+" ");
        System.out.println();
    }
}

//////////////////////////////////////////////////////////////////////////

// virtual machine
```

```
class VM extends Mini {
    private static int p[]; // program code
    private static int ip;  // instruction pointer
    private static int s[]; // stack
    private static int sp;  // stack pointer
    private static int fp;  // frame pointer
    private static int fs;  // frame size

    static void init(int code[],int nv,int stack_size) {
        p = code;
        ip = 0;
        s = new int[stack_size];
        sp = 0;
        fp = 0;
        fs = nv;
    }

    static int exec(int arg) {
        s[0] = arg;
        sp++;
        while (p[ip] != M_stop) {
            switch(p[ip]) {
            case M_push:
                s[sp] = p[ip+1];
                sp++;
                ip = ip+2;
                break;
            case M_load:
                s[sp] = s[fp+p[ip+1]];
                sp++;
                ip = ip+2;
                break;
            case M_store:
                s[fp+p[ip+1]] = s[sp-1];
                sp--;
                ip = ip+2;
                break;
            case M_add:
                sp--;
                s[sp-1] = s[sp-1] + s[sp];
                ip++;
                break;
            case M_sub:
                sp--;
                s[sp-1] = s[sp-1] - s[sp];
                ip++;
                break;
            case M_mul:
                sp--;
                s[sp-1] = s[sp-1] * s[sp];
                ip++;
                break;
            case M_div:
```

```
            sp--;
            s[sp-1] = s[sp-1] / s[sp];
            ip++;
            break;
        case M_if_cmpeq:
            sp = sp-2;
            ip = s[sp] == s[sp+1] ? p[ip+1]:ip+2;
            break;
        case M_if_cmpne:
            sp = sp-2;
            ip = s[sp] != s[sp+1] ? p[ip+1]:ip+2;
            break;
        case M_if_cmple:
            sp = sp-2;
            ip = s[sp] <= s[sp+1] ? p[ip+1]:ip+2;
            break;
        case M_if_cmpge:
            sp = sp-2;
            ip = s[sp] >= s[sp+1] ? p[ip+1]:ip+2;
            break;
        case M_goto:
            ip = p[ip+1];
            break;
        case M_jsr:
            s[sp] = ip+2;      // save return address
            s[sp+1] = fp;      // save fp
            fp = sp+2;         // set fp
            sp = fp+fs;        // set sp
            s[fp+1] = s[fp-3]; // copy argument
            ip = p[ip+1];      // goto start address
            break;
        case M_return:
            s[fp-3] = s[sp-1]; // copy return value
            sp = fp-2;         // reset sp
            fp = s[sp+1];      // reset fp
            ip = s[sp];        // goto return address
            break;
        default:
            error("runtime",p[ip]);
        }
    }
    return s[0];
  }
}

///////////////////////////////////////////////////////////////////////////
```

Anhang B

Mini-Compiler in Prolog

```
/* ============================== mini.pl ================================

The Mini language is a modified small subset of Java/C.
A Mini program consists of a single possibly recursive function.
The language has no declarations (implicit type integer).

mini.pl is a compiler-interpreter for Mini written in Prolog.
The compiler generates a syntax tree which is executed by a recursive
interpreter.

The compiler has two passes (lexical analysis and syntax analysis).
Both are implemented with definite clause grammars (DCG).
The target language (syntax tree) is specified by Prolog predicates.

======================= source language syntax (DCG) =====================

program    --> function
function   --> identifier,"(",identifier,")",block.
block      --> "{",statements,"}".
statements --> statement,statements.
statements --> "".

statement  --> identifier,"=",expression,";".
statement  --> "if",condition,statement,"else",statement.
statement  --> "while",condition,statement.
statement  --> "return",expression,";".
statement  --> block.
statement  --> ";".

condition  --> "(",expression,("==";"!=";">";"<"),expression,")".

expression --> term,exp_rest.
exp_rest   --> ("+";"-"),term,exp_rest.
exp_rest   --> "".

term       --> factor,term_rest.
term_rest  --> ("*";"/"),factor,term_rest.
term_rest  --> "".
```

```
factor     --> number.
factor     --> identifier.
factor     --> identifier,"(",expression,")".
factor     --> "(",expression,")".

============================ target language ========================== */

% syntax tree

is_int_prog(p(F)) :-
    is_fun(F).

is_fun(fun(Id,Pm,S)):-
    is_id(Id),
    is_id(Pm),
    is_stm(S).

is_stm(seq(S1,S2)) :-
    is_stm(S1),
    is_stm(S2).
is_stm(asm(Id,E)) :-
    is_id(Id),
    is_exp(E).
is_stm(ifs(E,S1,S2)) :-
    is_exp(E),
    is_stm(S1),
    is_stm(S2).
is_stm(whs(E,S)) :-
    is_exp(E),
    is_stm(S).
is_stm(ret(E)) :-
    is_exp(E).
is_stm(emp).

is_exp(con(N)) :-
    is_con(N).
is_exp(var(Id)) :-
    is_id(Id).
is_exp(fct(Id,E)) :-
    is_id(Id),
    is_exp(E).
is_exp(bin(Op,E1,E2)) :-
    is_opr(Op),
    is_exp(E1),
    is_exp(E2).

is_id(Id):- atom(Id).
is_con(N):- number(N).
is_opr(Op):- member(Op,[=,#,>,<,+,-,*,/]).

/* ============================== example ================================

source file "fac.mini":
------------------------
```

```
fac(n) {
    if (n == 0)
        return 1;
    else
        return n * fac(n-1);
}

run:
----
mini('fac.mini',8,Result,SynTree,_).

Result = 40320
SynTree =
    p(fun(fac,n,
        ifs(bin(=,var(n),con(0)),
            ret(con(1)),
            ret(bin(*,var(n),fct(fac,bin(-,var(n),con(1))))))))

========================================================================= */
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% main

mini(SourceFile,In,Out,SynTree,Msg) :-
    compiler(SourceFile,SynTree,CompMsg),
    (CompMsg='compilation ok' ->
        interpreter(SynTree,In,Out,Msg)
        ;
        Msg=CompMsg).

compiler(P0,P,M) :-
    run(lex_analysis(P0,P1),M='lexical error',
    run(syn_analysis(P1,P), M='syntax error',
    run(fail,               M='compilation ok',
    true))).

interpreter(P,In,Out,M) :-
    run(is_int_prog(P),     M='target program error',
    run(number(In),         M='input data error',
    run(execution(P,In,Out),M='execution error',
    run(fail,               M='execution ok',
    true)))).

run(Pass,_Error,Next) :-
    Pass,Next.
run(Pass,Error,_Next) :-
    \+Pass,Error.

:- use_module(library(lists)).
:- use_module(library(assoc)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% lexical analysis
```

```
lex_analysis(SourceFile,Tokens) :-
    reader(SourceFile,Chars),
    phrase(tokens(Tokens),Chars).

tokens(Ts)     --> whitespace,
                   tokens(Ts).
tokens([T|Ts]) --> tok(T),
                   !, % single solution:longest input match
                   tokens(Ts).
tokens([])     --> "".

tok('if')      --> "if".
tok('else')    --> "else".
tok('while')   --> "while".
tok('return')  --> "return".
tok('{')       --> "{".
tok('}')       --> "}".
tok(';')       --> ";".
tok(',')       --> ",".
tok('(')       --> "(".
tok(')')       --> ")".
tok(rop(=))    --> "==".
tok(rop(#))    --> "!=".
tok(rop(>))    --> ">".
tok(rop(<))    --> "<".
tok(aop(+))    --> "+".
tok(aop(-))    --> "-".
tok(mop(*))    --> "*".
tok(mop(/))    --> "/".
tok('=')       --> "=".
tok(id(I))     --> ident(Cs),{name(I,Cs)}.
tok(num(N))    --> number(Cs),{name(N,Cs)}.

ident([C|Cs])  --> letter(C),identr(Cs).
identr([C|Cs]) --> letter(C),identr(Cs).
identr([C|Cs]) --> digit(C),identr(Cs).
identr([])     --> "".

number([C|Cs]) --> digit(C),number(Cs).
number([C])    --> digit(C).

letter(C)      --> [C],{0'A=<C,C=<0'Z;0'a=<C,C=<0'z}.
digit(C)       --> [C],{0'0=<C,C=<0'9}.
whitespace     --> [C],{C=<0' }.

% ------------------------------------------------------------------------

reader(F,P) :-
    open(F,read,S),
    read_chars(S,P),
    close(S),
    !.
```

```
read_chars(S,[]) :-
    at_end_of_stream(S).
read_chars(S,[C|Cs]):-
    \+at_end_of_stream(S),
    get0(S,C),
    read_chars(S,Cs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% syntax analysis

syn_analysis(Tokens,SynTree) :-
    phrase(prog(SynTree),Tokens).

prog(p(F))          --> func(F).

func(fun(Id,Pm,S))  --> [id(Id)],['('],[id(Pm)],[')'],block(S).

block(S)            --> ['{'],stms(S),['}'].

stms(S)             --> stm(S1),stmr(S1,S).
stms(emp)           --> [].
stmr(S1,seq(S1,S))  --> stm(S2),stmr(S2,S).
stmr(S,S)           --> [].

stm(asm(Id,E))      --> [id(Id)],['='],exp(E),[';'].
stm(ifs(E,S1,S2))   --> ['if'],cond(E),stm(S1),['else'],stm(S2).
stm(whs(E,S))       --> ['while'],cond(E),stm(S).
stm(ret(E))         --> ['return'],exp(E),[';'].
stm(S)              --> block(S).
stm(emp)            --> [';'].

cond(bin(Op,E1,E2)) --> ['('],exp(E1),[rop(Op)],exp(E2),[')'].

exp(E)              --> term(E1),expr(E1,E).
expr(E1,E)          --> [aop(Op)],term(E2),expr(bin(Op,E1,E2),E).
expr(E,E)           --> [].

term(E)             --> factor(E1),termr(E1,E).
termr(E1,E)         --> [mop(Op)],factor(E2),termr(bin(Op,E1,E2),E).
termr(E,E)          --> [].

factor(con(N))      --> [num(N)].
factor(var(Id))     --> [id(Id)].
factor(fct(Id,E))   --> [id(Id)],['('],exp(E),[')'].
factor(E)           --> ['('],exp(E),[')'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% execute target program (syntax tree)
% Env..Environment, V..Value
% return statements produce a return value replacing the environment

execution(p(F),Vin,Vout) :-
```

```
    exec(F,Vin,Vout).

exec(fun(Id,Pm,S),Vin,Vout) :-
    new_env(fun(Id,Pm,S),Env0),
    put_env(Pm,Vin,Env0,Env1),
    exec(S,Env1,Vout),
    number(Vout). % fail if no return value

exec(seq(S1,S2),Env0,Env) :-
    exec(S1,Env0,Env1),
    (number(Env1) -> % return value ?
        Env = Env1   % don't execute S2
        ;
        exec(S2,Env1,Env)).
exec(asm(Id,E),Env0,Env) :-
    exec(E,Env0,V),
    put_env(Id,V,Env0,Env).
exec(ifs(E,S1,_),Env0,Env) :-
    exec(E,Env0,1),
    exec(S1,Env0,Env).
exec(ifs(E,_,S2),Env0,Env) :-
    exec(E,Env0,0),
    exec(S2,Env0,Env).
exec(whs(E,S),Env0,Env) :-
    exec(E,Env0,1),
    exec(S,Env0,Env1),
    exec(whs(E,S),Env1,Env).
exec(whs(E,_),Env,Env) :-
    exec(E,Env,0).
exec(ret(E),Env0,V) :-
    exec(E,Env0,V).
exec(emp,Env,Env).

exec(con(V),_,V).
exec(var(Id),Env,V) :-
    get_env(Id,Env,V).
exec(fct(_,E),Env,Vout) :-
    exec(E,Env,Vin),
    Env = env(F,_),
    exec(F,Vin,Vout).
exec(bin(Op,E1,E2),Env,V) :-
    exec(E1,Env,V1),
exec(E2,Env,V2),
exec(Op,V1,V2,V).

exec(+,V1,V2,V) :- V is V1+V2.
exec(-,V1,V2,V) :- V is V1-V2.
exec(*,V1,V2,V) :- V is V1*V2.
exec(/,V1,V2,V) :- V is V1/V2.
exec(=,V1,V2,V) :- V1=V2 -> V=1;V=0.
exec(#,V1,V2,V) :- V1=V2 -> V=0;V=1.
exec(>,V1,V2,V) :- V1>V2 -> V=1;V=0.
exec(<,V1,V2,V) :- V1<V2 -> V=1;V=0.
```

```
new_env(F,env(F,B)) :- % B..Bindings of Variables
empty_assoc(B).

put_env(Id,V,env(F,B0),env(F,B)) :-
put_assoc(Id,B0,V,B).

get_env(Id,env(_,B),V) :-
get_assoc(Id,B,V). % fail if not assigned

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```