	$module  ightarrow oldsymbol{ ext{module id }}; \ decls \ stat\_block \ oldsymbol{ ext{end id }}.$ $stat\_block  ightarrow oldsymbol{ ext{begin}} \ stat\_seq$ $\mid \epsilon$ $decls  ightarrow cnst\_decl \ type\_decl \ var\_decl \ proc\_decl\_seq$
	$cnst\_decl  ightarrow \mathbf{const} \ cnst\_ass$ $\mid \epsilon$ $cnst\_ass  ightarrow \mathbf{id} = expr \ ; \ cnst\_ass$ $\mid \epsilon$
	$type\_decl  ightarrow \mathbf{type}$ $type\_ass$ $\mid \epsilon$ $type\_ass  ightarrow \mathbf{id} = type$ ; $type\_ass$ $\mid \epsilon$ $type  ightarrow \mathbf{id}$
	$type  ightarrow \mathbf{id}$ $ type\_arr $ $ type\_rec $ $ \mathbf{integer} $ $ \mathbf{boolean} $
	$type\_arr  o \mathbf{array} \; expr  \mathbf{of} \; type \ type\_rec  o \mathbf{record} \; field\_list  \mathbf{end} \ field\_list  o \mathbf{id} : type \; ; field\_list \   \; \epsilon$
	$egin{aligned} var\_decl & ightarrow \mathbf{var\_list} \ &ert \epsilon \ var\_list & ightarrow id\_list  ext{: } type  ext{ ; } var\_list \ &ert \epsilon \end{aligned}$
	$proc\_decl\_seq  ightarrow proc\_decl$ ; $proc\_decl\_seq$ $\mid \epsilon$ $proc\_decl  ightarrow proc\_head$ ; $proc\_body$ $proc\_head  ightarrow procedure$ id $proc\_pars$
	$egin{aligned} proc\_pars & ightarrow \left( \begin{array}{c} par\_list   ight) \ &    \epsilon \ \ par\_list & ightarrow par  ext\_par \ & par  ightarrow \mathbf{var}  id\_list \colon type \ &    id\_list \colon type \end{aligned}$
	$egin{aligned} ext\_par & \Rightarrow  ext{; } par \ ext\_par \ &   \ \epsilon \ & id\_list  ightarrow  ext{id} \ ext\_id \ & ext\_id  ightarrow  ext{, }  ext{id} \ ext\_id \end{aligned}$
	$egin{aligned} & \epsilon \ & proc\_body  ightarrow decls \ stat\_block \ \mathbf{end} \ \mathbf{id} \ & stat\_seq  ightarrow stat \ ext\_stat \ & ext\_stat  ightarrow  ext{;} \ stat \ ext\_stat \end{aligned}$
	$egin{aligned} \epsilon \ stat  ightarrow \mathbf{id} \ id\_stat \ &   if\_stat \ &   while\_stat \end{aligned}$
	$id\_stat  ightarrow assignment \   proc\_call \   assignment  ightarrow sel := expr \   sel  ightarrow . id sel$
	$egin{array}{c} \mid expr  brack sel \ \mid \epsilon \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \$
	$expr\_list  ightarrow ext\_expr \   \ \epsilon \  $ $ext\_expr  ightarrow , \ expr \ ext\_expr \   \ \epsilon \  $
	$if\_stat  ightarrow \mathbf{if} \; expr  \mathbf{then} \; stat\_seq \; elif\_seq \; else\_stat  \mathbf{end}$ $elif\_seq  ightarrow  \mathbf{elsif} \; expr  \mathbf{then} \; stat\_seq \; elif\_seq$ $\mid \epsilon$ $else\_stat  ightarrow  \mathbf{else} \; stat\_seq$
	$egin{aligned} k & k \in \mathbb{R} & k$
	$egin{aligned} & \epsilon \ rel\_op  ightarrow =  \ \#\   < \   < =   >   > = \ simple\_expr  ightarrow + term\ post\_term \end{aligned}$
	$ $ - $term\ post\_term$ $ \ term\ post\_term$ $post\_term  o + term\ post\_term$ $ $ - $term\ post\_term$
	$egin{array}{c}    extbf{or}  ext{ } term  ext{ } post\_factor \ post\_factor  ightarrow * factor  ext{ } post\_factor \ & & factor  ext{ } post\_factor \ & & factor  ext{ } post\_factor \ & & & factor  ext{ } post\_factor \ & & & factor  ext{ } post\_factor \ & & & & & & & & & & & & & & & & & & $
	$egin{array}{c} \operatorname{\mathbf{div}}\ factor\ post\_factor \ & \operatorname{\mathbf{mod}}\ factor\ post\_factor \ & \epsilon \ & factor  ightarrow \operatorname{\mathbf{id}}\ sel \ & \end{array}$
	$  num\_token  $ $  (expr)$ $  \sim factor$
	irected Translation  → { env = null }
module $-$	<pre>module { for ( call in calls )</pre>
	{ $decl = \mathbf{new} \ Decl(\mathbf{id}, \ \mathbf{new} \ ModuleType()); \ env.put(decl) }$ { $g\_mod = \mathbf{new} \ Module(\mathbf{id}.lexeme); \ module.g\_mod = g\_mod }$ { $cur = env; \ env = \mathbf{new} \ Env(env); $ } { $decls.g\_mod = g\_mod; $ } $decls$
stat block –	$ \{ \ g\_proc = g\_mod.add(\ 'Main'\ ); \ stat\_block.g\_stats = g\_proc :: add\ \} $ $ stat\_block\ \mathbf{end\ id\ .} $ $ \{ \ env = cur\ \} $
$stat\_block$ $ decls$ $-$	$\{ stat\_seq.g\_stats = stat\_block.g\_stats \}$ $stat\_seq$ $\mid \epsilon$ $\rightarrow cnst\_decl \ type\_decl \ var\_decl$
$cnst\_decl$ –	$\{ proc\_decl\_seq.g\_mod = decls.g\_mod \}$ $proc\_decl\_seq$ $\Rightarrow \mathbf{const} \ cnst\_ass$ $\mid \epsilon$
$cnst\_ass$ –	$ \Rightarrow \mathbf{id} = expr $ $ \{ decl = \mathbf{new} \ Decl(\mathbf{id}, \ expr.type, \ \{ \ modifiers : [ \ Decl.Modifiers.Const \ ] \ \}) \ \} $ $ \{ env.put(\mathbf{id}.lexeme, \ decl) \ \} $ $ \vdots \ cnst\_ass_1 $
	$\mid \epsilon$ $ ightharpoonup \mathbf{type}\_ass$ $\mid \epsilon$ $ ightharpoonup \mathbf{id} = type$ $\{ decl = \mathbf{new} \ Decl(\mathbf{id}, \ \mathbf{new} \ Typedef(type.val)) \}$
type $-$	$ \{\;env.put(\mathbf{id}.lexeme,\;decl)\;\} $ $ ;\;type\_ass_1 $ $ \mid \epsilon $
	$ \{ \ decl = env.get(\mathbf{id}.lexeme); \ type.val = decl.type.val \} $ $   \ type\_arr$ $ \{ \ type.val = type\_arr.val \} $ $   \ type\_rec$
	$\{ type.val = type\_rec.val \}$ $  integer$ $\{ type.val = new IntegerType() \}$ $  boolean$
$type\_arr$ - $type\_rec$ -	
$field\_list$ $-$	{ field_list.l_fields = { } } field_list end { type_rec.val = new RecType(field_list.fields) }  id: type;  { deel = new Deel(id type val { modifiers : [ Deel Modifiers Van ] }) }
	$ \{ \ decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val, \ \{ \ modifiers : \ [ \ Decl.Modifiers.Var \ ] \ \} ) \ \} $ $ \{ \ field\_list.l\_fields.put(\mathbf{id}.lexeme, \ decl); \ field\_list_1.l\_fields = field\_list.l\_fields \ \} $ $ \{ \ field\_list.fields = field\_list_1.fields \ \} $ $   \ \epsilon \ \ \{ \ field\_list.fields = field\_list.l\_fields \ \} $
	$ ightarrow \mathbf{var}$ $var\_list$ $\mid \epsilon$ $ ightarrow id\_list: type;$ $\{$
	<pre>for(id in id_list.ids) {    env.put(id.lexeme, new Decl(id, type.val, { modifiers : [ Decl.Modifieres.Var ] })); }</pre>
$proc\_decl\_seq$ –	$egin{align*} & var\_list_1 \\ &   \; \epsilon \\ &  angle \; \{ \; proc\_decl.g\_mod = proc\_decl\_seq.g\_mod \; \} \\ & proc\_decl \; ; \\ & \end{align*}$
$proc\_decl$ –	$\{ proc\_decl\_seq_1.g\_mod = proc\_decl\_seq.g\_mod \}$ $proc\_decl\_seq_1$ $\mid \epsilon$ $\rightarrow \{ cur = env; \ env = \mathbf{new} \ Env(env) \}$
	$proc\_head; \\ \{proc\_body.g\_mod = proc\_decl.g\_mod; proc\_body.g\_proc = proc\_decl.g\_mod.add(proc\_head.id.lexeme)\} \\ \{proc\_body.decl\_id = proc\_head.id\} \\ proc\_body \\ \{env = cur; env.put(proc\_head.id, proc\_head.decl)\} \\ }$
	$ \begin{array}{l} \textbf{proc_pars} \\ \textbf{proc_pars} \\ \textbf{decl} = \textbf{new} \ Decl(\textbf{id}, \ \textbf{new} \ FunctionType(proc_pars.params)) \ \} \\ \textbf{proc_head.decl} = decl; \ proc_head.id = \textbf{id} \ \} \\ \textbf{proc_pars.params} = par\_list.params \ \} \end{array} $
$par\_list$ –	$\mid \epsilon \mid \{ proc\_pars.params = [ \ ] \}$ $\Rightarrow par \ ext\_par$ $\{ par\_list.params = [par.params,ext\_par.params ] \}$ $\Rightarrow \{ par.params = [ \ ] \}$
	$\mathbf{var}\ id\_list$ : $type$
	<pre>decl = new Decl(id, type.val, { modifiers: [ Decl.Modifiers.Var ] });     par.params.push(decl);     env.put(id.lexeme, decl); }    { par.params = [ ] }  id_list: type {     for(id in id_list.ids) {</pre>
	<pre>decl = new Decl(id, type.val, { modifiers: [ Decl.Modifiers.Var ] });     par.params.push(decl);     env.put(id.lexeme, decl); }  {    par.params = [ ] }  id_list: type {</pre>
	$decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val, \ \{ \ modifiers: \ [ \ Decl.Modifiers.Var ] \});$ $par.params.push(decl);$ $env.put(\mathbf{id}.lexeme, \ decl);$ $\}$ $ \{ par.params = [ ] \} $ $id\_list: type$ $\{$ $for(id \ \mathbf{in} \ id\_list.ids) \{$ $decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val);$ $par.params.push(decl);$ $env.put(\mathbf{id}.lexeme, \ decl);$ $\}$ $\}$ $; par \ ext\_par_1$ $\{ ext\_par.params = [ \dots par.params, \dots ext\_par_1.params ] \}$ $ \epsilon \ \{ ext\_par.params = [ ] \}$
$id\_list$ $ ext\_id$ $-$	$decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val, \ \{ \ modifiers: \ [ \ Decl.Modifiers.Var \ ] \ \});$ $par.params.push(decl);$ $env.put(\mathbf{id}.lexeme, \ decl);$ $\}$ $ \{ \ par.params = [ \ ] \ \}$ $id\_list: type$ $\{$ $for(id\ \mathbf{in} \ id\_list.ids) \ \{$ $decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val);$ $par.params.push(decl);$ $env.put(\mathbf{id}.lexeme, \ decl);$ $\}$ $\}$ $; par \ ext\_par_1$ $\{ \ ext\_par.params = [ \par.params, \ext\_par_1.params \ ] \ \}$ $  \ \epsilon \ \{ \ ext\_par.params = [ \ ] \ \}$ $\Rightarrow \mathbf{id} \ ext\_id \ \{ \ id\_list.ids = [ \ \mathbf{id}, \ext\_id.ids \ ] \ \}$ $\Rightarrow \mathbf{id} \ ext\_id_1 \ \{ \ ext\_id.ids = [ \ \mathbf{id}, \ext\_id_1.ids \ ] \ \}$ $  \ \epsilon \ \{ \ ext\_id.ids = [ \ ] \ \}$ $\Rightarrow \{ \ decls.g\_mod = proc\_body.g\_mod \ \}$
$id\_list$ - $ext\_id$ - $proc\_body$ -	$decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val, \ \{ \ modifiers: \ [ \ Decl.Modifiers.Var \ ] \ \});$ $par.params.push(decl);$ $env.put(\mathbf{id}.lexeme, \ decl);$ $\}$ $ \{ \ par.params = [ \ ] \ \}$ $id\_list: type$ $\{$ $for(id \ \mathbf{in} \ id\_list.ids) \ \{$ $decl = \mathbf{new} \ Decl(\mathbf{id}, \ type.val);$ $par.params.push(decl);$ $env.put(\mathbf{id}.lexeme, \ decl);$ $\}$ $\}$ $; par \ ext\_par_1$ $\{ \ ext\_par.params = [ \par.params, \ext\_par_1.params \ ] \ \}$ $  \ \epsilon \ \{ \ ext\_par.params = [ \ \mathbf{id}, \ext\_id.ids \ ] \ \}$ $\Rightarrow \ \mathbf{id} \ ext\_id \ \{ \ id\_list.ids = [ \ \mathbf{id}, \ext\_id.ids \ ] \ \}$ $  \ \epsilon \ \{ \ ext\_id.ids = [ \ \mathbf{id}, \ext\_id.ids \ ] \ \}$ $  \ \epsilon \ \{ \ ext\_id.ids = [ \ \mathbf{id}, \ext\_id.ids \ ] \ \}$
$id\_list - \\ ext\_id - \\ proc\_body - \\ stat\_seq - \\$	<pre>decl = new Decl(id, type.val, { modifiers: [ Decl.Modifiers.Var ] });     par.params.push(decl);     env.put(id.lexeme, decl); }    { par.params = [ ] }  id_list: type  {     for(id in id_list.ids) {         decl = new Decl(id, type.val);         par.params.push(decl);         env.put(id.lexeme, decl);     }  }  }  ; par ext_par_{1     {ext_par.params = [par.params,ext_par_1.params ] }    { ext_par.params = [ id,ext_id.ids ] }  id ext_id { id_list.ids = [ id,ext_id.ids ] }  id ext_id_{1} { ext_id.ids = [ id,ext_id.ids ] }    { ext_id.ids = [ ] }  id decls.g_mod = proc_body.g_mod }  decls  { stat_block.g_stats = proc_body.g_proc :: add }  stat_block end id  } { stat.g_stats = stat_seq.g_stats } </pre>
$id\_list - \\ ext\_id - \\ proc\_body - \\ stat\_seq - \\$	$decl = \textbf{new} \ Decl(\textbf{id}, \ type.val, \ \{ \ modifiers: \ [ \ Decl.Modifiers.Var \ ] \ \}); \\ par.params.push(decl); \\ env.put(\textbf{id}.lexeme, \ decl); \\ \} \\   \{ par.params = [ ] \} \\ id\_list: type \\ \{ \\ for(id \ \textbf{in} \ id\_list.ids) \ \{ \\ decl = \textbf{new} \ Decl(\textbf{id}, \ type.val); \\ par.params.push(decl); \\ env.put(\textbf{id}.lexeme, \ decl); \\ \} \\ \} \\ ; par \ ext\_par. \\ \{ ext\_par.params = [par.params,ext\_par_1.params \ ] \ \} \\   \epsilon \ \{ ext\_par.params = [par.params,ext\_d.ids \ ] \\   \epsilon \ \{ ext\_t.id.ids = [ \textbf{id},ext\_id.ids \ ] \\ \} \\ ; \textbf{id} \ ext\_id \ \{ id\_list.ids = [ \textbf{id},ext\_id_1.ids \ ] \\   \epsilon \ \{ ext\_id.ids = [ ] \ \} \\ \} \\ \{ decls.g\_mod = proc\_body.g\_mod \ \} \\ decls \\ \{ stat\_block.g\_stats = proc\_body.g\_proc :: add \ \} \\ stat\_block \ end \ \textbf{id} \\ \Leftrightarrow \{ stat\_stat_s\_stats = stat\_seq.g\_stats \ \} \\ stat \\ \{ ext\_stat,g\_stats = stat\_seq.g\_stats \ \} \\ ext\_stat \\ = ext\_stat \\ = ext\_stat.g\_stats = ext\_stat.g\_stats \ \} \\ stat \\ \{ ext\_stat_1.g\_stats = ext\_stat.g\_stats \ \} \\ ext\_stat \\ = ext\_stat_1.g\_stats = ext\_stat.g\_stats \ \} \\ ext\_stat_1.g\_stat_2.g\_stat_2.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g\_stat_3.g$
$id\_list - ext\_id - stat\_seq - stat\_seq - stat - s$	decl = new Decl(id, type.val, { modifiers : [Decl.Modifiers.Var] });   par.params.push(decl);   env.put(id.lexeme, decl);   }   { par.params = [] }   id. list : type   {   for(id in id.list.ids) {   decl = new Decl(id, type.val);   par.params.push(decl);   env.put(id.lexeme, decl);   }   }   ; par ext_par_{1} {   ext_par.params = [par.params,ext_par_params] }   e { ext_par.params = [] }   id ext.id { id.list.ids = [id,ext_id.ids] }   + id ext.id { id.list.ids = [id,ext_id.ids] }   e { ext_id.ids = [] }   decls.g.mod = proc_body.g.mod }   decls   stat_block.g.stats = proc_body.g.proc :: add }   stat_block.g.stats = stat_seq.g.stats }   stat   ext_stat_g.stats = stat_seq.g.stats }   stat   ext.stat_g.stats = ext_stat.g.stats }   ext_stat_l.g.stats = ext_stat.g.stats }   ext_stat_l.g.stats = ext_stat.g.stats }   ext_stat_l.g.stats = stat_seq.stats; id.stat.id = id }   id.stat   stat.g.stats.add(if.stat.statement) }   while_stat { stat.g.stats.add(while_stat.statement) }   while_stat { stat.g.stats.add(while_stat.statement) }
$id\_list - ext\_id - stat\_seq - stat\_seq - stat - s$	<pre>decl = new Decl(id, type.val, { modifiers: [ Decl.Modifiers.Var ] });     par.params.push(decl);     env.put(id.lexeme, decl); } } { par.params = [ ] } id.list: type {     for(id in id_list.ids) {         decl = new Decl(id, type.val);         par.params.push(decl);         env.put(id.lexeme, decl);     } } }; par.params.push(decl); env.put(id.lexeme, decl); } } } ; par ext.par, {     ext.par.params = [par.params,ext.par.params ] }   e {    ext.par.params = [ ] } e {    ext.par.params = [ ] } e {    ext.par.params = [ ] } e {    ext.di.dis = [ id,ext.id.ids ] } e {    ext.di.dis = [ ] } e {    ext.par.params = [ ] e {    ext.par.params = [ ] } e {    ext.siat.par.params = [ ] } e {    ext.siat.par.par.params = [ ] } e {    ext.siat.par.par.par.par.par.par.par.par.par.par</pre>
$id\_list - ext\_id - stat\_seq - stat\_seq - stat - s$	decl = new Decl(id, type.val, { modifiers:   Decl.Modifiers.Var   });     par.params.push(decl);     env.put(id.lexeme, decl); } }
$id\_list - ext\_id - stat\_seq - stat\_seq - stat - s$	decl = new Decl(id, type.val, { modifiers:   Decl.Modifiers.Var   });     par.params.push(decl);     env.put(id.lexeme, decl); } }
$id\_list - ext\_id - stat\_seq - stat\_seq - stat - s$	decl = new Decl(id, type.val, { modifiers: [Decl.Modifiers.Var] });     per.params.push(decl);     env.put(id.lexeme, decl); } }{ {     for(id in id.list.ids) {         decl = new Decl(id, type.val);         per.params.push(decl);     env.put(id.lexeme, decl); } } } } par.params.push(decl); env.put(id.lexeme, decl); } } par.params.push(decl); env.put(id.lexeme, decl); } } par.params = [par.params,ext_par.params] } {     ext_par.params = [par.params,ext_par.params] } {     ext_det_par.params = [] } } dect.id (id.list.ids = [id,ext_id.ids] } } dect.id (id.list.ids = [id,ext_id.ids] } {     ext_tid.ids = [1] } } dect.genod = proc_body.g_mod } decls {     stat.block_g_stats = proc_body.g_mod } decls {     stat.block_g_stats = proc_body.g_moe :: add} } stat.block_g_stats = stat_seq.g_stats } stat {     ext_stat_g_stats = stat_seq.g_stats } ext_stat. } * * * * { stat_s_stat_s = stat_seq.g_stats } ext_stat. * * * * { stat_s_stat_s = stat_seq.g_stats } ext_stat. * * * * * * * * * * * * * * * * * * *
$id\_list - ext\_id - stat\_seq - stat\_seq - stat - s$	deel = new Deellid   type-val, { modifiers : [Deel.Modifiers.Var]};   par params push(deel);   par params push(deel);   }   { par params = [] }   id hist   type   { for (id in id. list.ids) {
$id\_list = ext\_id = stat\_seq = stat\_seq = stat = s$	deci = new Decitid, typeval, { modifiers:   Deci.Modifiers.Var   };   par.params.push(deci);   par.params.push(deci);   }   }   }   { par.params =     }   id   dist: type   {   for(id in id.list.ida) { deci = new Decitid, type.val); par.params.push(deci); env.puvt(id.exeme, deci); }   }   ** par.ext_par_  {ext_par.params = [par.params,ext_par.params] }   {ext_par.params = [par.paramsext_par.params] }   {ext_par.params = [par.paramsext_id.ids] }   ** id ext id; {ext.id.ids =   id,ext.id.ids] }   ** id ext id; {ext.id.ids =   id,ext.id.ids] }   ** {ext_id.ids =     }   ** {ext_id.ids =     }   ** {ext_id.ids =     }   ** {ext.di.dis = stat.seq.g.stats}   decis   {ext.id.ids = stat.seq.g.stats}   ** decis   {ext.id.ids = stat.seq.g.stats}   ** id.t.ids   {ext.id.ids = stat.seq.g.stats}   ** id.t.ids   {ext.stat.g.stats = stat.seq.g.stats}   ** id.t.ids   {ext.stat.g.stats = stat.seq.stats}   ** id.t.ids   {ext.stat.g.stats = stat.seq.stats}   ** id.t.ids   {ext.stat.g.stats = stat.seq.stats}   ** id.t.ids   {ext.stat.g.stats = ext.stat.g.stats}   ** id.t.ids   {ext.stat.g.stats = ext.stat.g.stats}   ** id.t.ids   {ext.stat.g.stats.add(if.stat.statement) }   ** id.stat.g.stats.add(stat.id.getLexeme()); proc.call.id = id.stat.id }   ** id.stat.g.stats.add(proc.call.statement) }   ** if id.stat.g.stats.add(proc.call.id.act.pars.type = sel_ttype }   ** if id.stat.et = "" if id.stat.et
$id\_list = ext\_id = ext\_id = stat\_seq = stat\_seq = stat =$	deci = new Deci(ld, type.val, { modifiers: [Decl.Modifiers.Var]});   par.params.push(deci);   empsyl(ld.lezeme, deci);   }   { par.params.e
$id\_list = ext\_id = ext\_id = stat\_seq = stat\_seq = stat =$	decl = new Decl(d, type.val, { modifiers: [Decl.Modifiers.Var]});   par.params.puh(decl);   cros.puk(ld.lezeme, decl);   }   { par.params.e. [] id. list.ids) {   decl = new Decl(id, type.val);   par.params.puh(ldecl);   ent.puk(ld.lezeme, decl);   }   } }   **paret.par*  { ext.par.params.eext.par.params.] }   { ext.par.params.e. [par.paramsext.par.params.] }   { ext.par.params.e. [] ipar.paramsext.par.params.] }   { ext.par.par.params.e. [] ipar.params.] }   { ext.par.params.e. [] ipar.paramsext.par.params.] }   { ext.par.par.params.e. [] ipar.paramspar.params.] }   { ext.par.par.par.par.par.paramspar.paramspar.params.] }   { ext.par.par.par.par.par.par.paramspar.paramspar.params.] }   { ext.par.par.par.par.par.par.par.par.par.par
id_list ext_id proc_body  stat_seq  ext_stat  stat  stat  sel  expr_list	decl = new Decl[id.tpseval. { modifiers:   Decl.Modifiers.Var   }};   par parmays.mi(decl);   emv.gui(id.lexeme, decl);   }   }   {   forcid in id.list.ids}   {   decl = new Decl[id.tpseval];
<pre>id_list = ext_id =  proc_body =  stat_seq =  ext_stat =  stat =  id_stat =  sel =  proc_call =  ext_pars =  ext_expr =  e</pre>	decl = new Declifd, type.cul, {  modifiers  :   Decl.Modifiers  Var    };   purs paramas
<pre>id_list = ext_id =  proc_body =  stat_seq =  ext_stat =  stat =  id_stat =  sel =  proc_call =  ext_pars =  ext_expr =  e</pre>	decl = new Declifd, type-oil, { modifiers:   Decl.Modifiers.Var   };
id_list - ext_id -  proc_body -  stat_seq -  ext_stat -  stat -  id_stat -  sel -  act_pars -  if_stat -	deci = new Decilid. type.val. (modifiers : [Deci.Modifiers Var   ]); por parama publical; env.put[id.leaeme, deci]; } } [for.parama = [] id.list type { for (im id.list.ide) {     deci = new Decilid. (gue.val);     pur parama.publical; env.put[id.leaeme, deci]; } } * **: pur cat par. {     cet par. parama.epar.paramacat.par.parama.]} {         cet par. parama.epar.paramacat.par.parama.]} {         cet par. parama.epar.paramacat.par.parama.]} {         cet par.parama.epar.paramacat.di.ide]} {         cet par.parama.epar.paramadi.} {         cet par.parama.epar.paramadi.} {         cet par.parama.epar.paramadi.} {         cet stat.d. (cet.stid.ide][] {         cet.di.ide.epar.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.parama.e.]} {         cet.di.ide.epar.par.par.parama.e.]} {         cet.di.ide.epar.par.par.par.par.par.par.par.par.par.
id_list - ext_id -  proc_body -  stat_seq -  ext_stat -  stat -  id_stat -  sel -  act_pars -  if_stat -	doc! = new Doc[ld. type.od., { modifiers : [ Decl.Modifiers Vor ] }); per paramy bilded!; empatible.ceme, dec!); } } { [(pro params = []); id. hit type {     for (id in id. list.ids) {         dec! = new Doc[ld. type.od];         per paramy bilded!;         empatible.ceme, dec!); } }; } }; pro response = [], id. hit type {     for (id in id. list.ids) {         dec! = new Doc[ld. type.od];         per paramy bilded!;         empatible.ceme, dec!); } }; }; }; pro response = [], id. dec! = new Doc[ld. type.od]; } {     cet. paramose = [], per params,ext port, params ] } {     {        cet. params = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] } }; {         cet. paramose = [], per params,ext port, params ] }  }; {         cet. paramose = [], per params,ext port, params ] }  }; {         cet. paramose = [], per params,ext paramose ] }  {         cet. paramose = [], per params,ext paramose ] }  {         cet. paramose = [], per paramoseext paramose ] }  {         cet. paramose = [], per paramoseext paramose ] }  {         cet. paramose = [], per paramoseext paramose ] }  {         cet. paramose = [], per paramoseext paramose ] }  {         cet. paramose = [], per paramoseext paramose ] }  {         cet. paramose = [], per paramoseext paramose ] }  {         cet. paramose = [], per paramoseext param
id_list - ext_id -  proc_body -  stat_seq -  ext_stat -  stat -  id_stat -  sel -  act_pars -  if_stat -	deed = new Decilial, type.col.,   modifiers:   DeciModifiers Vor   \};   per persons multiplied);   every upital learner, deci);   }   }   { Jornan
id_list - ext_id - proc_body -  stat_seq -  ext_stat -  stat -  id_stat -  sel -  proc_call -  act_pars -  ext_expr -  if_stat -  elif_seq -  elif_seq -	deci
id_list - ext_id - proc_body -  stat_seq -  ext_stat -  stat -  id_stat -  sel -  proc_call -  act_pars -  ext_expr -  if_stat -  elif_seq -  elif_seq -	deci = new Deci(id, type=u), { nodifices : [DeciModifices Vive] };   propromos = []
id_list = ext_id =  proc_body =  stat_seq =  ext_stat =  stat =  id_stat =  set =  set =  proc_call =  act_pars =  ext_expr =  ext_expr =  ext_expr =  while_stat =  while_stat =  while_stat =	decis = new Deci(ht, types, https://decis.org/dicters/ver]);
id_list - ext_id - ext_id -  proc_body -  stat_seq -  ext_stat -  id_stat -  set -  sel -  act_pars -  ext_expr -	ded = new Declife agreed, {mod for x : [Decl Mod for x lor ]}; per garanta production; ent guild lacente, decl; }    Total in it lacender
id_list - ext_id - ext_id -  proc_body -  stat_seq -  ext_stat -  id_stat -  set -  sel -  act_pars -  ext_expr -	ded = new Declin, typecold, (modifiers : [Decl.Modifiers Not  ]); pro-gramma = []  **Aport Note
id_list - ext_id - ext_id - proc_body - stat_seq - ext_stat -  id_stat -  id_stat -  sel -  sel -  act_pars -  ext_expr -  if_stat -  while_stat -  while_stat -	
id_list - ext_id - ext_id - proc_body - stat_seq - ext_stat -  id_stat -  id_stat -  sel -  sel -  act_pars -  ext_expr -  if_stat -  while_stat -  while_stat -	
id_list - ext_id - ext_id -  proc_body - stat_seq -  ext_stat -  id_stat -  id_stat -  sel -  sel -  act_pars -  ext_expr -  ext_expr -  ext_expr -  vhile_stat -  while_stat -	deed = new Packful, age-and, { modifiers   [Pack Modifiers   bar   ]};   per systems as middled;   modified in the Intel (
id_list - ext_id - ext_id -  proc_body - stat_seq -  ext_stat -  id_stat -  id_stat -  sel -  sel -  act_pars -  ext_expr -  ext_expr -  ext_expr -  vhile_stat -  while_stat -	dest = men Brotijk spysosk (molifores   Dest Bandy Fore New [1]);  per per men publisherem, decit;  }
id_list - ext_id - ext_id -  proc_body - stat_seq -  ext_stat -  id_stat -  id_stat -  sel -  sel -  act_pars -  ext_expr -  ext_expr -  ext_expr -  vhile_stat -  while_stat -	
id_list - ext_id - ext_id -  proc_body - stat_seq -  ext_stat -  id_stat -  id_stat -  sel -  sel -  act_pars -  ext_expr -  ext_expr -  ext_expr -  vhile_stat -  while_stat -	decided   special,   month(see,   2004.3 of, fee, see,   1)
id_list - ext_id - proc_body - stat_seq - ext_stat -  stat -  id_stat -  id_stat -  sel -  proc_call -  act_pars -  expr_list -  ext_expr -  if_stat -  while_stat -  while_stat -	def = mem bereith spaces, (most press;   20ct. Most plans long   1);  gen promoses and publishment death;
id_list - ext_id - proc_body - stat_seq - ext_stat -  stat -  id_stat -  id_stat -  sel -  proc_call -  act_pars -  expr_list -  ext_expr -  if_stat -  while_stat -  while_stat -	
id_list - ext_id - proc_body - stat_seq - ext_stat -  stat -  id_stat -  id_stat -  sel -  proc_call -  act_pars -  expr_list -  ext_expr -  if_stat -  while_stat -  while_stat -	
id_list - ext_id - proc_body - stat_seq - ext_stat - stat - stat - id_stat - sel - acsignment - sel - act_pars - expr_list - ext_expr - ext_expr - rel_op - simple_expr - rel_op - rel_op - simple_expr -	deed = meet incriptifi, type-med. [mobilities of 17] per parameter incriptifi, type-med. [mobilities of 18]
id_list - ext_id - proc_body - stat_seq - ext_stat - stat - stat - id_stat - sel - acsignment - sel - act_pars - expr_list - ext_expr - ext_expr - rel_op - simple_expr - rel_op - rel_op - simple_expr -	and men Analish capends [ model from [ Deckholos [ model from [ model from [ model from
id_list - ext_id - ext_id - proc_body -  stat_seq - ext_stat -  stat -  id_stat -  id_stat -  sel -  sel -  proc_call -  act_pars -  exp_list -  ext_expr -  if_stat -  while_stat -  vhile_stat -  rel_op -  rel_op -  rel_op -  rel_op -	
id_list - ext_id - ext_id - proc_body -  stat_seq - ext_stat -  stat -  id_stat -  id_stat -  sel -  sel -  proc_call -  act_pars -  exp_list -  ext_expr -  if_stat -  while_stat -  vhile_stat -  rel_op -  rel_op -  rel_op -  rel_op -	solid and seed to the control of the

 $\{\;post\_factor_1.l\_text = post\_factor.l\_text\;||\;'mod'\;||\;factor.text\;\}$ 

 $\{\ decl = env.get(\mathbf{id}.lexeme);\ sel.parent\_type = decl.type\ \}$ 

 $\{\ factor.type = sel.type;\ factor.text = \mathbf{id}.lexeme \ ||\ sel.text\ \}$ 

 $\{\ factor.type = expr.type;\ factor.text = '('\mid\mid expr.text\mid\mid\ ')'\ \}$ 

 $\{\ factor.type = \mathbf{new}\ IntegerType();\ factor.text = \mathbf{num\_token}.lexeme\ \}$ 

 $\{ \textit{ factor.type} = \mathbf{new} \; \textit{BooleanType}(); \; \textit{factor.text} = ' \sim' \; || \; \textit{factor}_1.text \; \}$ 

 $\{\ post\_factor.type = post\_factor.l\_type\ \}$ 

 $\{\ post\_factor.text = post\_factor.l\_text\ \}$ 

 $\{\ post\_factor.type = post\_factor_1.type;\ post\_factor.text = post\_factor_1.text\ \}$ 

 $post\_factor_1$ 

| num\_token

|(expr)

 $| \sim \mathit{factor}_1$ 

 $factor o \mathbf{id}$ 

Oberon-0 Analysis

**BNF Grammar**