Compiler Construction/Fall 2015/Project Milestone 2
Template for Syntax-Directed Definition

Eva Rose
evarose@cs.nyu.edu

Kristoffer Rose
krisrose@cs.nyu.edu

Assigned Thursday 10/15/2015, due Monday 10/26/2015 at 8am

This document is a template for syntax-directed definitions (SDD) over the abstract syntax of the *SubScript* language used in the Fall 2015 project of the class. In the abstract syntax production rules we use the following abbreviations for the HACS sorts in the *SubScript* syntax.

NAME	ABBREVIATION	NAME	ABBREVIATION
CallSignature	CS	NameType	NT
Declaration	D	NameTypeList	NTL
Expression	E	NameTypeListTail	NTLT
ExpressionList	EL	Program	P
ExpressionListTail	ELT	STRING	**str**
IDENTIFIER	**id**	Statement	S
INTEGER	**int**	Statements	Ss
IfTail	IT	Type	T
KeyValue	KV	TypeList	TL
KeyValueList	KVL	TypeListTail	TLT
KeyValueListTail	KVLT	Unit	U
Member	M	Units	Us
Members	Ms		

Below a template for writing an SDD over *SubScript*, with the full abstract syntax.

Attributes. We shall use the following attributes:

- ok is a synthesized boolean on P, ..., which is *true* if and only if P is well-typed.
- $synd$ is a synthesized declaration records (containing mappings of names and types) on P, ...
- n is a synthesized name on P, ...
- $lexval$ is the synthesized attribute for taking the literal string on P, ...
- t is a synthesized types on P, ...
- pt is a synthesized parameter types on P, ...
- $lexval$ is a literal values on $terminals$.
- env is a inherited declaration records (containing mappings of names and types) on P, ...
- $||$ measn concatenation

SDD Helpers. Below is the description of helpers used in the SDD, where X is the qualified abbreviations that satisfy the given productions.

- $\mathit{Define}(X.\mathit{env}, \textbf{id}.\mathit{lexval})$ is to check whether there exist such **id** in the current environment of X.
- $\mathit{Extend}(X, \textbf{id}.\mathit{lexval}, T_1)$ is to add the identifier **id** with type T_1 in the current content of X.
- $\mathit{ExtendT}(X, \textbf{id}.\mathit{lexval}, T_1)$ is to add the type which comes from **id** in the current content of X.
- $\mathit{Lookup}(X.\mathit{env}, \textbf{id}.\mathit{lexval})$ is to lookup the type of the identifier **id** in the current environment of X.
- $\mathit{AddTypeCheck}(E_1, E_2)$ is to check the type of the Expression E_1 and E_2. The rule as follows:
  if $(E_1.t \in \mathit{TSet}$ && $E_2.t \in \mathit{TSet})$ {
    if $(E_1.t = \textbf{number}$ && $E_2.t = \textbf{number})$
      return **number**
    return **string**
  }
  $\mathit{returnTypeError}$
- $(\mathit{Boolean\ Expression})?\ \mathit{Value0} :\ \mathit{Value1}$ is the ternary operation. If the boolean expression is true, it calculates Value0. Otherwise it calculates Value1.
- A && B is the boolean operation to represent both condition A and condition B have to be true.
- $\mathit{LValue}(E) : \mathit{LValue}$ evaluates the expression to true if E is an $\mathit{Identifier}$ or it is a member access $E.m$ that is not a function member.
- $\mathit{Uniq}(P_{outer}.\mathit{env}, D_{inner}.\mathit{synd})$: evaluates to true if in the scope of D there exists an distinct $\mathit{Identifier}$. Otherwise return false.
- $\mathit{KvlTypeChecker}(\mathit{KVL})$: evaluates to true if all pair elements satisfy $\mathit{identifier}_i.t = E_i.t$ (permutation allowed). Otherwise return TypeError.
- $\mathit{TSet}_1 = \{\textbf{number}, \textbf{string}\}$.

Asscumptions Once the type error occurs, the whole program stops.

Explanations The procedure to apply the semantic rule is that if there is synthesized declaration records, then the rule to extend them will be apply first. Because those declarations need to go up to the root to tell the program they are defined, and then those $\mathit{non-terminals}$ can the inherited attributes env to pass them down to the children in the tree.

PRODUCTION	SEMANTIC RULES
$P \to Us_1$	$P.ok = Us_1.ok, Us_1.\mathit{env} = Us_1.\mathit{synd}$
$Us \to U_1\ Us_2$	$Us.ok = U_1.ok$ && $U_2.ok, U_1.\mathit{env} = Us.\mathit{env}$ $U_2.\mathit{env} = U_1.\mathit{env}, Us.\mathit{synd} = \mathit{Extend}(Us_2.\mathit{synd}, U_1.n, U_1.pt)$

Production	Semantic Rules
$Us \to U_1$	$Us.ok = U_1.ok, U_1.env = Us.env, Us.env = Extend(Us.synd, U_1.synd, U_1.pt)$
$U \to D_1$	$U.ok = D_1.ok, D_1.env = U.env, U.env = Extend(U.synd, D_1.synd, D.pt)$
$U \to S_1$	$U.ok = S_1.ok, S_1.env = U.env, U.env = Extend(U.synd, D_1.synd, D_1.pt)$
$D \to$ **interface** $\mathbf{id}_1$ **{** Ms_2 **}**	$D.ok = Ms_2.ok \;\&\&\; Uniq(Ms_2),\; D.pt = Ms_2.pt$ $Ms_2.env = Extend(D.env,\; \mathbf{id}_1.n,\; \mathbf{id}_1),\; D.n = \mathbf{id}_1.lexval$
$D \to$ **function** $\mathbf{id}_1$ CS_2 **{** Ss_3 **}**	$D.ok = Ss_3.ok \;\&\&\; CS_2.ok,\; D.n = \mathbf{id}_1.lexval,\; \mathbf{id}_1.t = CS_2.t$ $D.pt = CS_2.pt,\; Ss_3.env = Extend(D.env,\; \mathbf{id}_1.lexval,\; \mathbf{id}_1.t)$
$Ms \to M_1\; Ms_2$	$Ms.ok = M_1.ok \;\&\&\; Ms_2.ok,\; Ms.env = Ms.synd$ $Ms.synd = Extend(Ms_2.synd, M_1.n, M_1.pt)$ $M_1.env = Ms.env,\; Ms_2.env = M_1.env$
$Ms \to \epsilon$	$Ms.ok = true,\; Ms.synd = \epsilon$
$M \to \mathbf{id}_1$ **:** T_2 **;**	$M.n = \mathbf{id}_1.lexval,\; M.t = T_2.t,\; M.ok = true$
$M \to \mathbf{id}_1$ CS_2 **{** Ss_3 **}**	$M.n = \mathbf{id}_1.lexval, M.pt = CS_2.pt, M.ok = CS_2.ok \;\&\&\; Ss_3.ok \quad Ss_3.env = Extend(M.env, CS_2.n, CS_2.pt),\; M.t = CS_2.t \quad M.env = M.synd, M.synd = Extend(Ss_3.synd, Cs_2.n, Cs_2.pt)$
$CS \to$ **(** NTL_1 **)** **:** T_2	$CS.ok = uniq(NTL_1), NTL_1.env = CS.env$ $CS.t = T_2.t,\; CS.pt = NTL.t,\; CS.n = \mathbf{id}_2.lexval \| T2.t$
$NTL \to NT_1\; NTLT_2$	$NTL.ok = NT_1.ok \;\&\&\; NTL_2.ok,\; NTL.t = NT_1.t \| NTLT_2.pt$ $NT_1.env = NTL.env, NTL_2.env = NTL.env$
$NTL \to \epsilon$	$NTL.ok = true$
$NTLT \to$ **,** $NT_1\; NTLT_2$	$NTLT.ok = NT_1.ok \;\&\&\; NTLT_2.ok,\; NT_1.env = NTLT.env$ $NTLT_2.env = Extend(NTLT.env, NT_1.syn, NT_1.t)$ $NTLT.t = NT_1.t \| NTLT_2.pt \quad NTLT.t = Extend(NTLT_2.synd, NT_1.syn, NT_1.t)$
$NTLT \to \epsilon$	$NTLT.ok = true,\; NTLT.t = \epsilon$
$NT \to \mathbf{id}_1$ **:** T_2	$NT.syn = \mathbf{id}_1.lexval, NT.t = T_2.t$
$Ss \to S_1\; Ss_2$	$Ss.ok = S_1.ok \;\&\&\; Ss_2.ok,\; S_1.env = Ss.env$ $Ss_2.env = Extend(Ss.env, S_1.n, S_1.t)$

PRODUCTION	SEMANTIC RULES
$Ss \to \epsilon$	$Ss.ok = true$
$S \to \textbf{\{}\ Ss_1\ \textbf{\}}$	$S.ok = Ss_1.ok,\ Ss_1.env = S.env,\ S.n = \epsilon,\ S.t = \epsilon$
$S \to \textbf{var}\ NT_1\ \textbf{;}$	$S.ok = true,\ NT_1.env = S.env, S.n = NT_1.n, S.t = NT_1.t$
$S \to E_1\ \textbf{;}$	$S.ok = E_1.ok,\ E_1.env = S.env,\ S.n = \epsilon,\ S.t = \epsilon$
$S \to \textbf{;}$	$S.ok = true, S.n = \epsilon, S.t = \epsilon$
$S \to \textbf{if}\ (\ E_1\ )\ IT_2$	$S.ok = E_1.ok\ \&\&\ IT_2.ok, E_1.env = S.env,\ IT_2.env = S.env$ $S.n = \epsilon,\ S.t = \epsilon$
$S \to \textbf{while}\ (\ E_1\ )\ S_2$	$S.ok = E_1.ok\ \&\&\ IT_2.ok, E_1.env = S.env,\ S_2.env = S.env$ $S.n = \epsilon,\ S.t = \epsilon$
$S \to \textbf{return}\ E_1\ ;$	$S.ok = Lookup(S.env, \textbf{return})\ \&\&\ E1.t, S.n = \epsilon,\ S.t = \epsilon$
$S \to \textbf{return}\ ;$	$S.ok = Lookup(S.env, \textbf{return})\ \&\&\ \textbf{void}, S.n = \epsilon,\ S.t = \epsilon$
$IT \to S_1\ \textbf{else}\ S_2$	$IT.ok = S_1.ok\ \&\&\ S_2.ok,\ S_1.env = IT.env,\ S_2.env = IT.env$
$IT \to S_1$	$IT.ok = S_1.ok, S_1.env = IT.env$
$T \to \textbf{boolean}$	$T.t = \textbf{boolean}$
$T \to \textbf{number}$	$T.t = \textbf{number}$
$T \to \textbf{string}$	$T.t = \textbf{string}$
$T \to \textbf{void}$	$T.t = \textbf{void}$
$T \to \textbf{id}_1$	$T.t = Define(Us_1.env, \textbf{id}_1.lexval)?\ \textbf{id}_1.lexval : UndefinedError$
$T \to \textbf{(}\ TL_1\ \textbf{)}\ \ {=}{>}\ \ T_2$	$T.t = T_2$
$T \to \textbf{\{}\ NTL_1\ \textbf{\}}$	$T.t = NTL_1.t$
$TL \to T_1\ TLT_2$	$T_1.env = ExtendT(TL.env, T_1.syn), TLT_2.env = T_1.env,$
$TL \to \epsilon$	$TL.t = \epsilon$
$TLT \to \textbf{,}\ T_1\ TLT_2$	$TLT_2.env = T_1.env,\ TLT.ok = true$ $T_1.env = ExtendT(TLT.syn,\ , T_1.syn)$
$TLT \to \epsilon$	$TLT.t = \epsilon$
$E \to \textbf{id}_1$	$E.t = Define(E.env,\ \textbf{id}_1)?\ Lookup(E.env,\ \textbf{id}_1) :\ TypeError$
$E \to \textbf{str}_1$	$E.t = \textbf{string}$

Production	Semantic Rules
$E \rightarrow \textbf{int}_1$	$E.t = \textbf{number}$
$E \rightarrow E_1 \; \textbf{.} \; \textbf{id}_2$	$E_1.env = E.env, E.t = (\text{Define}(E.env, \textbf{id}_2)\;\&\&\;\text{Lookup}(E.env, \textbf{id}_2) = \textbf{id}_2.t)?\; \textbf{id}_2.t : \; \text{TypeError}$
$E \rightarrow E_1 \; \textbf{(} \; EL_2 \; \textbf{)}$	$E_1.env = E.env,\; EL_2.env = E.env,\; E.t = E_1.t \parallel EL_2.t$
$E \rightarrow \textbf{!}\; E_1$	$E_1.env = E.env, E.t = (E_1.t = \textbf{boolean})?\; E_1.t : \; \text{TypeError}$
$E \rightarrow \sim E_1$	$E_1.env = E.env, E.t = (E_1.t = \textbf{number})?\; E_1.t : \; \text{TypeError}$
$E \rightarrow \; -E_1$	$E_1.env = E.env, E.t = (E_1.t = \textbf{number})?\; E_1.t : \; \text{TypeError}$
$E \rightarrow \; +E_1$	$E_1.env = E.env, E.t = (E_1.t = \textbf{number})?\; E_1.t : \; \text{TypeError}$
$E \rightarrow E_1 * E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = \textbf{number}\;\&\&\;E_2.t = \textbf{number})?\; E_1.t : \text{TypeError}$
$E \rightarrow E_1 \,/\, E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = \textbf{number}\;\&\&\;E_2.t = \textbf{number})?\; E_1.t : \text{TypeError}$
$E \rightarrow E_1 \,\%\, E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = \textbf{number}\;\&\&\;E_2.t = \textbf{number})?\; E_1.t : \text{TypeError}$
$E \rightarrow E_1 + E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (\text{AddTypeCheck}(E_1.t, E_2.t))?\; E_1.t : \text{TypeError}$
$E \rightarrow E_1 - E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = \textbf{number}\;\&\&\;E_2.t = \textbf{number})?\; E_1.t : \text{TypeError}$
$E \rightarrow E_1 <= E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t\;\&\&\;E_1.t \in \text{TSet}_1)?\; \textbf{boolean} : \; \text{TypeError}$
$E \rightarrow E_1 >= E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t\;\&\&\;E_1.t \in \text{TSet}_1)?\; \textbf{boolean} : \; \text{TypeError}$
$E \rightarrow E_1 < E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t\;\&\&\;E_1.t \in \text{TSet}_1)?\; \textbf{boolean} : \; \text{TypeError}$
$E \rightarrow E_1 > E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t\;\&\&\;E_1.t \in \text{TSet}_1)?\; \textbf{boolean} : \; \text{TypeError}$
$E \rightarrow E_1 == E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t)?\; \textbf{boolean} : \; \text{TypeError}$
$E \rightarrow E_1 \,!\!= E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t)?\; \textbf{boolean} : \; \text{TypeError}$
$E \rightarrow E_1 \,\&\, E_2$	$E_1.env = E.env,\; E_2.env = E.env$ $E.t = (E_1.t = E_2.t)?\; \textbf{number} : \; \text{TypeError}$

PRODUCTION	SEMANTIC RULES
$E \to E_1 \mid E_2$	$E_1.env = E.env,\ E_2.env = E.env$ $E.t = (E_1.t = E_2.t = \textbf{number})?\ \textbf{number} :\ TypeError$
$E \to E_1\ \&\&\ E_2$	$E_1.env = E.env,\ E_2.env = E.env$ $E.t = (E_1.t = E_2.t = \textbf{boolean})?\ \textbf{boolean} :\ TypeError$
$E \to E_1 \parallel E_2$	$E_1.env = E.env,\ E_2.env = E.env$ $E.t = (E_1.t = E_2.t = \textbf{boolean})?\ \textbf{boolean} :\ TypeError$
$E \to E_1 = E_2$	$E_1.env = E.env,\ E_2.env = E.env$ $E.t = (E_1.t = E_2.t\ \&\&\ LValue(E_1))?\ E_1.t :\ TypeError$
$E \to E_1 \mathrel{+}= E_2$	$E_1.env = E.env, E_2.env = E.env$ $E.t = (E_1.t = E_2.t\ \&\&\ LValue(E_1))?\ AddTypeCheck(E_1, E_2) :\ TypeError$
$E \to E_1 = \{\ KVL_2\ \}$	$E_1.env = E.env, KVL_2.env = E.env,\ E.t = E_1.t$ $E1.t = KVL_2.ok?\ KVL_2.t :\ TypeError$
$KVL \to KV_1\ KVLT_2$	$KV_1.env = KVL.env,\ KVLT_2.env = KVL.env$ $KVL.ok = KvlTypeCheck(KVL)$ $KVL.t = KvlTypeCheck(KVL)?\ (KVL.t = ExtendT(KVT2.synd, (KV1.n),\ KV1.t))) : TypeError$
$KVL \to \epsilon$	$KVL.t = \epsilon$
$KVLT \to \textbf{,}\ KV_1\ KVLT_2$	$KV_1.env = KVLT.env,\ KVLT_2.env = KV_1.env$ $KVLT.t = ExtendT(KVT2.synd, (KV1.n), KV1.t))$
$KVLT \to \epsilon$	$KVLT.t = \epsilon$
$EL \to E\ ELT$	$E.env = EL.env, ELT.env = EL.env$ $E.t = ExtendT(EL.env, E.n, E.t)$
$EL \to \epsilon$	$EL.t = \epsilon$
$ELT \to ,\ E\ ELT_1$	$E.env = ELT.env, ELT_1.env = E.env$ $ELT.t = ExtendT(ELT_1.snyd, E.n, E.t)$
$ELT \to \epsilon$	$ELT = \epsilon$
$KV \to \textbf{id}_1 : E_2$	$E_2.env = KV.env,\ KV.n = \textbf{id}_1.lexval, KV.t = E_2.t$