Abstract Syntax of *LustreS* for the Open Source L2C Compiler

L2C team, System Software and Software Engineering Laboratory
Department of Computer Science and Technology, Tsinghua University

November 21, 2019

1 Program

⟨*program*⟩ ::= ⟨*general_program*⟩

⟨*general_program*⟩ ::= (⟨*type_block*⟩, ⟨*defs_block*⟩, ⟨*const_block*⟩, ⟨*node_block*⟩, ⟨*node_main*⟩)

⟨*type_block*⟩ ::= (*IDENT*, ⟨*type*⟩)*

⟨*defs_block*⟩ ::= (*IDENT*, ⟨*type*⟩)*

⟨*const_block*⟩ ::= (*IDENT*, ⟨*globvar*⟩)*

⟨*node_block*⟩ ::= (*IDENT*, ⟨*F*⟩)*

⟨*F*⟩ ::= ⟨*node*⟩

⟨*node_main*⟩ ::= *IDENT*

2 Type

⟨*type*⟩ ::= *Tvoid* (* the [void] type *)
| *Tint*(⟨*intsize*⟩, ⟨*signedness*⟩) (* integer types *)
| *Tfloat*(⟨*floatsize*⟩) (* floating-point types *)
| *Tpointer*(⟨*type*⟩) (* pointer types ([*ty]) *)
| *Tarray*(*IDENT*, ⟨*type*⟩, *Z*)
(* array types: array_type_id(ty ^ len), and *Z*: type of integer values *)
| *Tfunction*(⟨*typelist*⟩, ⟨*type*⟩) (* function types *)
| *Tstruct*(*IDENT*, ⟨*fieldlist*⟩) (* struct types *)

⟨*typelistt*⟩ ::= ⟨*type*⟩*

⟨*fieldlist*⟩ ::= (*IDENT*, ⟨*type*⟩)*

3 Const

⟨*globvar*⟩ ::= (⟨*gvar_info*⟩, ⟨*gvar_init*⟩, ⟨*gvar_readonly*⟩, ⟨*gvar_volatile*⟩)
(* ⟨*globvar*⟩: from AST.v in Compcert *)

⟨*gvar_info*⟩ ::= ⟨*V*⟩ (* language-dependent info, e.g. a type *)

⟨*gvar_init*⟩ ::= ⟨*init_data*⟩* (* initialization data *)

⟨*gvar_readonly*⟩ ::= ⟨*bool*⟩ (* read-only variable? (const) *)

⟨*gvar_volatile*⟩ ::= ⟨*bool*⟩ (* volatile variable? *)

⟨*V*⟩ ::= ⟨*type*⟩

⟨*init_data*⟩ ::= ..... (* ⟨*globvar*⟩: from AST.v in Compcert *)

4 Node

⟨*node*⟩ ::= ⟨*general_node*⟩

⟨*general_node*⟩ ::= (⟨*nd_kind*⟩, ⟨*nd_args*⟩, ⟨*nd_rets*⟩, ⟨*nd_flags*⟩, ⟨*nd_svars*⟩, ⟨*nd_vars*⟩,
⟨*nd_stmt*⟩, ⟨*nd_sid*⟩, ⟨*nd_fld*⟩, ⟨*nd_eqt*⟩)

⟨*nd_kind*⟩ ::= ⟨*bool*⟩ (* node kind. *)

⟨*nd_args*⟩ ::= ⟨*params*⟩ (* input parameters. *)

⟨*nd_rets*⟩ ::= ⟨*params*⟩ (* output parameters. *)

⟨*nd_flags*⟩ ::= ⟨*params*⟩

⟨*nd_svars*⟩ ::= ⟨*params*⟩ (* tempo variables. *)

⟨*nd_vars*⟩ ::= ⟨*params*⟩ (* local variables. *)

⟨*nd_stmt*⟩ ::= ⟨*S*⟩ (* statements. *)

⟨*nd_sid*⟩ ::= *IDENT* (* name of output struct. *)

⟨*nd_fld*⟩ ::= ⟨*fieldlist*⟩ (* fieldlist of output struct. *)

⟨*nd_eqt*⟩ ::= (⟨*eqt*⟩, ⟨*sexp*⟩*)*

⟨*eqt*⟩ ::= *Eqt_assign*(⟨*eqf*⟩)
| *Eqt_counter*(⟨*eqf*⟩)

⟨*eqf*⟩ ::= (⟨*sexp*⟩, ⟨*sexp*⟩)

$\langle S \rangle$ ::= $(\langle stmt \rangle, \langle clock \rangle)^*$

$\langle params \rangle$ ::= $(IDENT, \langle type \rangle)^*$

$\langle stmt \rangle$ ::= $Sassign((IDENT, \langle type \rangle), \langle expr \rangle)$
| $Scall(\langle lhs \rangle, \langle calldef \rangle, \langle sexp \rangle^*)$
| $Sfor(\langle bool \rangle, \langle hstmt \rangle, \langle int \rangle)$
| $Sarydif((IDENT, \langle type \rangle), \langle int \rangle, \langle sexp \rangle^*)$
| $Smix((IDENT, \langle type \rangle), \langle sexp \rangle, \langle lindex \rangle^*, \langle sexp \rangle)$
| $Sstruct((IDENT, \langle type \rangle), \langle fieldlist \rangle, \langle sexp \rangle^*)$
| $Sfby((IDENT, \langle type \rangle), IDENT, \langle sexp \rangle, \langle sexp \rangle)$
| $Sfbyn((IDENT, \langle type \rangle), (IDENT, IDENT, IDENT), \langle int \rangle, \langle sexp \rangle, \langle sexp \rangle)$
| $Sarrow((IDENT, \langle type \rangle), \langle sexp \rangle, \langle sexp \rangle)$
| $Scurrent((IDENT, \langle type \rangle), \langle sexp \rangle, \langle sexp \rangle, \langle sexp \rangle)$
| $Sskip$

$\langle lhs \rangle$ ::= $(IDENT, \langle type \rangle)^*$

$\langle calldef \rangle$ ::= $(\langle cakind \rangle, \langle instid \rangle, \langle callid \rangle, \langle csid \rangle, \langle cfield \rangle, \langle argtys \rangle, \langle rettys \rangle)$

$\langle cakind \rangle$ ::= $\langle bool \rangle$ (* call kind. *)

$\langle instid \rangle$ ::= $IDENT$ (* name of call instance. *)

$\langle callid \rangle$ ::= $IDENT$ (* name of call node. *)

$\langle callnum \rangle$ ::= [$\langle int \rangle$] (* length of call instance array. *)

$\langle csid \rangle$ ::= $IDENT$ (* name of call struct. *)

$\langle cfield \rangle$ ::= $\langle fieldlist \rangle$ (* fieldlist of call struct. *)

$\langle argtys \rangle$ ::= $\langle type \rangle^*$ (* type list of input parameters in call node. *)

$\langle rettys \rangle$ ::= $(IDENT, \langle type \rangle)^*$ (* output parameters in call node. *)

$\langle lindex \rangle$ ::= $Label(IDENT)$
| $Index(\langle sexp \rangle)$

$\langle expr \rangle$ ::= $Expr(\langle sexp \rangle)$
| $Earyprj(\langle sexp \rangle, \langle sexp \rangle^*, \langle sexp \rangle)$
| $Ecase(\langle sexp \rangle, (\langle patn \rangle, \langle sexp \rangle)^*)$
| $Eif(\langle sexp \rangle, \langle sexp \rangle^*, \langle sexp \rangle)$
| $Eprefix(\langle binary\_operation \rangle, \langle sexp \rangle^*)$
| $Etypecmp(\langle bool \rangle, \langle sexp \rangle, \langle sexp \rangle)$

$\langle patn \rangle$ ::= $Pachar(\langle int \rangle)$
| $Paint(\langle int \rangle)$
| $Pabool(\langle bool \rangle)$
| $Pany$

⟨*sexp*⟩ ::= *Sconst*(⟨*const*⟩, ⟨*type*⟩) (* const expr. *)
| *Svar*(*IDENT*, ⟨*type*⟩) (* local variable. *)
| *Scvar*(*IDENT*, ⟨*type*⟩) (* global const variable. *)
| *Ssvar*(*IDENT*, ⟨*type*⟩) (* output struct variable. *)
| *Savar*(*IDENT*, ⟨*type*⟩) (* input variable. *)
| *Saryacc*(⟨*sexp*⟩, ⟨*sexp*⟩, ⟨*type*⟩) (* access to a index of a array. *)
| *Sfield*(⟨*sexp*⟩, *IDENT*, ⟨*type*⟩) (* access to a member of a struct. *)
| *Scast*(⟨*sexp*⟩, ⟨*type*⟩) (* type cast ([(ty) e]). *)
| *Sunop*(⟨*unary_operation*⟩, sexp¿, ⟨*type*⟩) (* unary operation. *)
| *Sbinop*(⟨*binary_operation*⟩, ⟨*sexp*⟩, ⟨*sexp*⟩, ⟨*type*⟩) (* binary operation. *)

⟨*const*⟩ ::= *Cint*(⟨*int*⟩)
| *Cfloat*(⟨*float*⟩)
| *Csingle*(⟨*float32*⟩)
| *Cbool*(⟨*bool*⟩)

⟨*unary_operation*⟩ ::= *Onotbool* (* boolean negation ([!] in C). *)
| *Onotint* (* integer complement ([ ] in C). *)
| *Oneg* (* opposite (unary [-]). *)
| *Oabsfloat* (* floating-point absolute value. *)

⟨*binary_operation*⟩ ::= *Oadd* (* addition (binary [+]). *)
| *Osub* (* subtraction (binary [-]). *)
| *Omul* (* multiplication (binary [*]). *)
| *Odivreal* (* division real ([/]). *)
| *Odivint* (* division integer ([div]). *)
| *Omod* (* remainder ([mod]). *)
| *Oand* (* and ([and]). *)
| *Oor* (* or ([or]). *)
| *Oxor* (* xor ([xor]). *)
| *Oeq* (* comparison ([=]). *)
| *One* (* comparison ([<>]). *)
| *Olt* (* comparison ([<]). *)
| *Ogt* (* comparison ([>]). *)
| *Ole* (* comparison ([<=]). *)
| *Oge* (* comparison ([>=]). *)

⟨*hstmt*⟩ ::= *Hmapary*((*IDENT*, ⟨*type*⟩), ⟨*binary_operation*⟩, ⟨*sexp*⟩, ⟨*sexp*⟩)
| *Hmaptycmp*((*IDENT*, ⟨*type*⟩), ⟨*bool*⟩, ⟨*sexp*⟩, ⟨*sexp*⟩)
| *Hmapunary*((*IDENT*, ⟨*type*⟩), ⟨*unary_operation*⟩, ⟨*sexp*⟩)
| *Hfoldary*((*IDENT*, ⟨*type*⟩), ⟨*binary_operation*⟩, ⟨*sexp*⟩, ⟨*sexp*⟩)
| *Hfoldunary*((*IDENT*, ⟨*type*⟩), ⟨*unary_operation*⟩, ⟨*sexp*⟩)
| *Hfoldcast*((*IDENT*, ⟨*type*⟩), ⟨*sexp*⟩, ⟨*type*⟩)
| *Harydef*((*IDENT*, ⟨*type*⟩), ⟨*sexp*⟩)
| *Haryslc*((*IDENT*, ⟨*type*⟩), ⟨*sexp*⟩, ⟨*int*⟩)
| *Hboolred*((*IDENT*, ⟨*type*⟩), ⟨*sexp*⟩)
| *Hmapcall*(⟨*lhs*⟩, ⟨*calldef*⟩, ⟨*sexp*⟩*)
| *Hmapfoldcall*((*IDENT*, ⟨*type*⟩), (*IDENT*, ⟨*type*⟩), ⟨*lhs*⟩, ⟨*calldef*⟩, ⟨*sexp*⟩, ⟨*sexp*⟩*)

5 Common

⟨*clock*⟩ ::= (*bool*, *IDENT*)*

⟨*int*⟩ ::= ..... (* from Integers.v *)

⟨*float*⟩ ::= binary64 (* from Integers.v*)

⟨*float32*⟩ ::= binary32 (* from Ibinary32.v *)