Creating Concise and Efficient Dynamic Analyses with ALDA

A. ALDA Full Syntax

We give the full syntax of ALDA in eBNF form. The whole ALDA program is roughly dividing into four sections as discussed in Section 3: <*type-decl*>, <*meta-decl*>, <*func-decl*> and <*insert-decl*>.

The major difference outlined between the syntax presented in Section 3 and the full syntax available here is the complete expansion of the *if-stmt*, *return-stmt*, and *expression-stmt*, as well as the operations they depend on. The *if-stmt* represents general conditional flows in the program, and supports if and else clauses. The *return-stmt* is used to represent return values from metadata propagation functions, and is generally used to pass function-local metadata out of a metadata propagation function.

Finally, the *expression-stmt* is expanded to include standard *C*-like operations. These operations include: logical operations or (||), and (&&); bitwise operations or (|), and (&), xor ($\wedge$); equality and relational operations ($==$, $<$, $>$, $<=$, $>=$, $!=$); shift expressions ($<<$, $>>$); additive expressions ($+$, $-$); multiplicative expressions ($*$, $/$, and $\%$), and cast expressions. The syntax also supports unary operations ($++$, $--$), and assignment operations ($=$), including assignment of common expressions (e.g. $*=$). The syntax also includes support for structure field accesses (.), and array index accesses ([]).

⟨*program*⟩ ::= ⟨*stmt*⟩*

⟨*stmt*⟩ ::= ⟨*type-decl*⟩ | ⟨*meta-decl*⟩ | ⟨*func-decl*⟩ | ⟨*insert-decl*⟩

⟨*type-decl*⟩ ::= ⟨*typename*⟩ ':=' ⟨*type*⟩ (':' **sync**)? (':' ⟨*number*⟩)?

⟨*type*⟩ ::= **int8** | **int16** | **int32** | **int64** | **pointer** | **lockid** | **threadid**

⟨*meta-decl*⟩ ::= ⟨*identifier*⟩ '=' ⟨*meta-type*⟩

⟨*meta-type*⟩ ::= ⟨*specifier*⟩ (⟨*map-type*⟩ | ⟨*set-type*⟩ | ⟨*typename*⟩)

⟨*set-type*⟩ ::= **set** '(' ⟨*typename*⟩ ')'

⟨*map-type*⟩ ::= **map** '(' ⟨*typename*⟩ ',' (⟨*typename*⟩ | ⟨*meta-type*⟩) ')'

⟨*specifier*⟩ ::= **universe::** | **bottom::** | ε

⟨*func-decl*⟩ ::= ⟨*typename*⟩? ⟨*funcname*⟩ '(' ⟨*func-arg-list*⟩? ')' '{' ⟨*func-body*⟩ '}'

⟨*func-body*⟩ ::= ⟨*subset-cpp-stmt*⟩*

⟨*func-arg*⟩ ::= ⟨*typename*⟩ ⟨*identifier*⟩

⟨*func-arg-list*⟩ ::= ⟨*func-arg*⟩ (',' ⟨*func-arg*⟩)* (**, ...**)? | ε

⟨*insert-decl*⟩ ::= **insert** (**before**|**after**) ⟨*insert-point*⟩ **call** ⟨*funcname*⟩ '(' ⟨*call-arg-list*⟩ ')'

⟨*insert-point*⟩ ::= **func** ⟨*identifier*⟩ | **LoadInst** | **StoreInst** | ...

⟨*call-arg-list*⟩ ::= ⟨*call-arg*⟩ | ⟨*call-arg*⟩ (',' ⟨*call-arg*⟩)* | ε

⟨*call-arg*⟩ ::= ⟨*call-arg-base*⟩ | ⟨*call-arg-base*⟩**.m** | **sizeof** '(' ⟨*call-arg-base*⟩ ')'

⟨*call-arg-base*⟩ ::= **$**⟨*number*⟩ | **$r** | **$p** | **$t**

⟨*subset-cpp-stmt*⟩ ::= (⟨*if-stmt*⟩ | ⟨*return-stmt*⟩ | ⟨*expression-stmt*⟩)*

⟨*if-stmt*⟩ ::= **if** '(' ⟨*expression*⟩ ')' ⟨*subset-cpp-stmt*⟩ (**else** ⟨*subset-cpp-stmt*⟩)?

⟨*return-stmt*⟩ ::= **return** ⟨*expression*⟩ ';'

⟨*expression-stmt*⟩ ::= ⟨*expression*⟩ ';'

⟨*expression*⟩ ::= ⟨*assignment-expression*⟩ | ⟨*expression*⟩ ',' ⟨*assignment-expression*⟩

⟨*assignment-expression*⟩ ::= ⟨*logical-or-expression*⟩ | ⟨*unary-expression*⟩ ⟨*assignment-op*⟩ ⟨*assignment-expression*⟩

⟨*assignment-op*⟩ ::= '=' | '*=' | '/=' | '%=' | '&=' | '^=' | '|=' | '«=' | '»=' | '+=' | '/='

⟨*logical-or-expression*⟩ ::= ⟨*logical-and-expression*⟩ | ⟨*logical-or-expression*⟩ '||' ⟨*logical-and-expression*⟩

⟨*logical-and-expression*⟩ ::= ⟨*inclusive-or-expression*⟩ | ⟨*logical-and-expression*⟩ '&&' ⟨*inclusive-or-expression*⟩

⟨*inclusive-or-expression*⟩ ::= ⟨*exclusive-or-expression*⟩
| ⟨*inclusive-or-expression*⟩ '|' ⟨*exclusive-or-expression*⟩

⟨*exclusive-or-expression*⟩ ::= ⟨*and-expression*⟩
| ⟨*exclusive-or-expression*⟩ '^' ⟨*and-expression*⟩

⟨*and-expression*⟩ ::= ⟨*equality-expression*⟩
| ⟨*and-expression*⟩ '&' ⟨*equality-expression*⟩

⟨*equality-expression*⟩ ::= ⟨*relational-expression*⟩
| ⟨*equality-expression*⟩ ('=='|'!=')
⟨*relational-expression*⟩

⟨*relational-expression*⟩ ::= ⟨*shift-expression*⟩
| ⟨*relational-expression*⟩ ('>'|'<'|'<='|'>=')
⟨*shift-expression*⟩

⟨*shift-expression*⟩ ::= ⟨*additive-expression*⟩
| ⟨*shift-expression*⟩ ('»'|'«') ⟨*additive-expression*⟩

⟨*additive-expression*⟩ ::= ⟨*multiplicative-expression*⟩
| ⟨*additive-expression*⟩ ('+'|'-')
⟨*multiplicative-expression*⟩

⟨*multiplicative-expression*⟩ ::= ⟨*unary-expression*⟩
| ⟨*multiplicative-expression*⟩ ('*'|'/'|'%')
⟨*unary-expression*⟩

⟨*unary-expression*⟩ ::= ⟨*postfix-expression*⟩
| ('++'|'--') ⟨*unary-expression*⟩
| ('&'|'*'|'+'|'-'|'~'|'!') ⟨*unary-expression*⟩
| **sizeof** '(' ⟨*identifier*⟩ ')'

⟨*postfix-expression*⟩ ::= ⟨*primary-expression*⟩
| ⟨*postfix-expression*⟩ '[' ⟨*expression*⟩ ']'
| ⟨*postfix-expression*⟩ '(' ')'
| ⟨*postfix-expression*⟩ '(' ⟨*argument-expression-list*⟩ ')'
| ⟨*postfix-expression*⟩ ('.'|'->') ⟨*identifier*⟩
| ⟨*postfix-expression*⟩ ('++'|'--')

⟨*primary-expression*⟩ ::= ⟨*identifier*⟩
| ⟨*constant*⟩
| ⟨*str-constant*⟩
| '(' ⟨*expression*⟩ ')'

⟨*argument-expression-list*⟩ ::= ⟨*assignment-expression*⟩
| ⟨*argument-expression-list*⟩ ','
⟨*assignment-expression*⟩