A.2 Grammar

⟨*translation-unit*⟩ ::= ⟨*statement*⟩⋆ *(translation unit - Swift file)*

⟨*statement*⟩ ::= `';'` | ⟨*new-type-defn*⟩ | ⟨*global-const-defn*⟩ | ⟨*import-stmt*⟩ | ⟨*pragma-stmt*⟩ *(program statement)*
| ⟨*func-defn*⟩ | ⟨*block*⟩ | ⟨*if-stmt*⟩ | ⟨*switch-stmt*⟩ | ⟨*wait-stmt*⟩ | ⟨*foreach-loop*⟩ | ⟨*for-loop*⟩
| ⟨*iterate-loop*⟩ | ⟨*stmt-chain*⟩ | ⟨*update-stmt*⟩

⟨*new-type-defn*⟩ ::= `'type'` ⟨*type-name*⟩ `'{'` (⟨*var-decl*⟩ `';'`)⋆ `'}'` *(new struct or subtype)*
| `'type'` ⟨*type-name*⟩ ⟨*standalone-type*⟩ `';'` | `'typedef'` ⟨*type-name*⟩ ⟨*standalone-type*⟩ `';'`

⟨*global-const-defn*⟩ ::= `'global' 'const'` ⟨*var-decl*⟩ `';'` *(global constant)*

⟨*import-stmt*⟩ ::= `'import'` ⟨*module-path*⟩ `';'` | `'import'` ⟨*string-literal*⟩ `';'` *(module import)*

⟨*module-path*⟩ ::= ⟨*id*⟩ (`'.'` ⟨*id*⟩)⋆ *(module path)*

⟨*pragma-stmt*⟩ ::= `'pragma'` ⟨*id*⟩ ⟨*expr*⟩⋆ `';'` *(pragma statement)*

⟨*func-defn*⟩ ::= ⟨*swift-func-defn*⟩ | ⟨*app-func-defn*⟩ | ⟨*foreign-func-defn*⟩ *(function definitions)*

⟨*func-hdr*⟩ ::= ⟨*type-params*⟩? ⟨*formal-arg-list*⟩? ⟨*func-name*⟩ ⟨*formal-arg-list*⟩? *(function header)*

⟨*type-params*⟩ ::= `'<'` ⟨*var-name*⟩ (`','` ⟨*var-name*⟩)⋆ `'>'` *(type variable parameters)*

⟨*formal-arg-list*⟩ ::= `'('` (⟨*formal-arg*⟩ (`','` ⟨*formal-arg*⟩)⋆)? `')'` *(formal argument list)*

⟨*formal-arg*⟩ ::= ⟨*type-prefix*⟩ `'...'`? ⟨*var-name*⟩ ⟨*type-suffix*⟩ (`'='` ⟨*expr*⟩)? *(formal argument)*

⟨*swift-func-defn*⟩ ::= ⟨*annotation*⟩⋆ ⟨*func-hdr*⟩ ⟨*block*⟩ *(Swift function definition)*

⟨*app-func-defn*⟩ ::= ⟨*annotation*⟩⋆ `'app'` ⟨*func-hdr*⟩ `'{'` ⟨*app-body*⟩ `'}'` *(app function definition)*

⟨*app-body*⟩ ::= ⟨*app-arg-expr*⟩+ (`'@'`(`'stdin'` | `'stdout'` | `'stderr'`) `'='` ⟨*expr*⟩)⋆ `';'`? *(app function body)*

⟨*foreign-func-defn*⟩ ::= ⟨*annotation*⟩⋆ ⟨*func-hdr*⟩ ⟨*foreign-func-body*⟩ *(foreign function definition)*

⟨*foreign-func-body*⟩ ::= ⟨*string-literal*⟩ ⟨*string-literal*⟩ ⟨*string-literal*⟩? (`'['` ⟨*string-literal*⟩ *(foreign function body)*
| ⟨*multiline-string-literal*⟩`']'`)?

⟨*var-decl*⟩ ::= ⟨*type-prefix*⟩ ⟨*var-decl-rest*⟩ (`','` ⟨*var-decl-rest*⟩)⋆ *(variable declaration)*

⟨*var-decl-rest*⟩ ::= ⟨*var-name*⟩ ⟨*type-suffix*⟩ ⟨*var-mapping*⟩? (`'='` ⟨*expr*⟩)? *(rest of variable declaration)*

⟨*type-prefix*⟩ ::= ⟨*type-name*⟩ | ⟨*param-type*⟩ *(type declaration prefix)*

⟨*param-type*⟩ ::= ⟨*type-name*⟩ `'<'` ⟨*standalone-type*⟩ `'>'` *(parameterized type)*

⟨*type-suffix*⟩ ::= (`'['` ⟨*standalone-type*⟩? `']'`)⋆ *(type declaration suffix)*

⟨*standalone-type*⟩ ::= ⟨*type-prefix*⟩ ⟨*type-suffix*⟩ *(standalone type)*

⟨*var-mapping*⟩ ::= `'<'` ⟨*expr*⟩ `'>'` *(variable mapping declaration)*

⟨*block*⟩ ::= `'{'` ⟨*statement*⟩⋆ `'}'` *(code block)*

⟨*stmt-chain*⟩ ::= ⟨*chainable-stmt*⟩ (`';'` | `'='` `'>'`⟨*stmt*⟩) *(statement chain)*

⟨*chainable-stmt*⟩ ::= ⟨*var-name*⟩ | ⟨*func-call*⟩ | ⟨*var-decl*⟩ | ⟨*assignment*⟩ *(statement that supports chaining)*

⟨*assignment*⟩ ::= (⟨*lval-list*⟩ | `'('` ⟨*lval-list*⟩ `')'`) (`'='` | `'+='`) ⟨*expr-list*⟩ *(assignment)*

⟨*update-stmt*⟩ ::= ⟨*var-name*⟩ `'<'` ⟨*id*⟩ `'>'` `':='` ⟨*expr*⟩ `';'` *(update statement)*

Figure A.1: Grammar for Swift/T variant of Swift programming language in EBNF syntax [33] extended with ? for optional elements and + for one or more repetitions. Figures A.2, A.4, and A.3 contain additional grammar rules.

⟨*if-stmt*⟩ ::= `'if'` `'('` ⟨*expr*⟩ `')'` ⟨*block*⟩ (`'else'` ⟨*block*⟩)? *(if statement)*

⟨*switch-stmt*⟩ ::= `'switch'` `'('` ⟨*expr*⟩ `')'` `'{'` ⟨*case*⟩⋆ ⟨*default*⟩? `'}'` *(switch statement)*

⟨*case*⟩ ::= `'case'` ⟨*int-literal*⟩ `':'` ⟨*stmt*⟩⋆ *(switch case)*

⟨*default*⟩ ::= `'default'` `':'` ⟨*stmt*⟩⋆ *(switch default case)*

⟨*wait-stmt*⟩ ::= `'wait'` `'deep'`? `'('` ⟨*expr-list*⟩ `')'` ⟨*block*⟩ *(wait statement)*

⟨*foreach-loop*⟩ ::= ⟨*annotation*⟩⋆ `'foreach'` ⟨*var-name*⟩ (`','` ⟨*var-name*⟩)? `'in'` ⟨*expr*⟩ ⟨*block*⟩ *(foreach loop)*

⟨*for-loop*⟩ ::= ⟨*annotation*⟩⋆ `'for'` `'('` ⟨*for-init-list*⟩ `';'` ⟨*expr*⟩ ⟨*;*⟩ ⟨*for-update-list*⟩ `')'` ⟨*block*⟩ *(for loop)*

⟨*for-init-list*⟩ ::= ⟨*for-init*⟩ (`','`⟨*for-init*⟩)⋆ *(for loop initializer list)*

⟨*for-init*⟩ ::= ⟨*for-assignment*⟩ | ⟨*type-prefix*⟩ ⟨*var-name*⟩ ⟨*type-suffix*⟩ `'='` ⟨*expr*⟩ *(for loop initializer)*

⟨*for-update-list*⟩ ::= ⟨*for-assignment*⟩ (`','`⟨*for-assignment*⟩)⋆ *(for loop update list)*

⟨*for-assignment*⟩ ::= ⟨*var-name*⟩ `'='` ⟨*expr*⟩ *(for loop assignment)*

⟨*iterate-loop*⟩ ::= `'iterate'` ⟨*var-name*⟩ ⟨*block*⟩ `'until'` `'('` ⟨*expr*⟩ `')'` *(iterate loop)*

Figure A.2: Extended Backus-Naur Form grammar for Swift/T control-flow statements.

⟨*id*⟩ ::= (⟨*alpha*⟩ | `'_'`) (⟨*alpha*⟩ | `'_'`⟨*digit*⟩)⋆ *(identifier)*

⟨*string-literal*⟩ ::= `'"'` (`'\'` . | ¬`'"'`)⋆ `'"'` *(string literal)*

⟨*multiline-string-literal*⟩ ::= `'----\n'` .?⋆ `'----'` | `'"""\n'` .?⋆ `'"""'` *(multi-line string literal)*

⟨*int-literal*⟩ ::= [⟨*digit*⟩]⋆ | `'0x'`([⟨*digit*⟩] | [`'a'`..`'f'`] | [`'A'`..`'F'`])⋆ *(integer literal)*

⟨*decimal*⟩ ::= ⟨*digit*⟩+ `'.'` ⟨*digit*⟩+ *(decimal floating point literal)*

⟨*sci-decimal*⟩ ::= ⟨*digit*⟩+ (`'.'` ⟨*digit*⟩+)? (`'e'` | `'E'`) `'-'`? ⟨*digit*⟩+ *(decimal scientific notation literal)*

⟨*digit*⟩ ::= `'0'`..`'9'` *(decimal digit)*

⟨*alpha*⟩ ::= [`'a'`..`'z'`] | [`'A'`..`'Z'`] *(alphabet character)*

⟨*line-comment*⟩ ::= (`'//'` | `'#'`) (¬`'\n'`)∗ `'\n'` *(single-line comment)*

⟨*multi-line-comment*⟩ ::= `'/*'` .?⋆ `'*/'` *(multi-line comment)*

Figure A.3: Extended Backus-Naur Form grammar for Swift/T lexer tokens. ?⋆ means non-greedy matching.

⟨*expr*⟩ ::= ⟨*or-expr*⟩ *(RValue expression)*

⟨*or-expr*⟩ ::= ⟨*and-expr*⟩ | ⟨*or-expr*⟩ `'||'` ⟨*and-expr*⟩ *(or expression)*

⟨*and-expr*⟩ ::= ⟨*eq-expr*⟩ | ⟨*and-expr*⟩ `'&&'` ⟨*eq-expr*⟩ *(and expression)*

⟨*eq-expr*⟩ ::= ⟨*cmp-expr*⟩ | ⟨*eq-expr*⟩ (`'=='` | `'!='`) ⟨*eq-expr*⟩ *(equality expression)*

⟨*cmp-expr*⟩ ::= ⟨*add-expr*⟩ | ⟨*cmp-expr*⟩ (`'<'` | `'<='` | `'>'` | `'>='`) ⟨*add-expr*⟩ *(comparison expression)*

⟨*add-expr*⟩ ::= ⟨*mult-expr*⟩ | ⟨*add-expr*⟩ (`'+'` | `'-'`) ⟨*mult-expr*⟩ *(addition precedence expression)*

⟨*mult-expr*⟩ ::= ⟨*pow-expr*⟩ | ⟨*mult-expr*⟩ (`'*'` | `'/'` | `'%/'` | `'%%'` | `'%'`) ⟨*pow-expr*⟩ *(multiplication precedence expression)*

⟨*unary-expr*⟩ ::= ⟨*unary-expr*⟩ | ⟨*pow-expr*⟩ `'**'` ⟨*unary-expr*⟩ *(power expression)*

⟨*unary-expr*⟩ ::= ⟨*postfix-expr*⟩ | (`'-'` | `'!'`) ⟨*postfix-expr*⟩ *(unary expression)*

⟨*postfix-expr*⟩ ::= ⟨*base-expr*⟩ | ⟨*postfix-expr*⟩(⟨*array-subscript*⟩ | ⟨*struct-subscript*⟩) *(postfix expression)*

⟨*array-subscript*⟩ ::= `'['` ⟨*expr*⟩ `']'` *(array key subscript)*

⟨*struct-subscript*⟩ ::= `'.'` ⟨*id*⟩ *(struct member subscript)*

⟨*base-expr*⟩ ::= ⟨*literal*⟩ | ⟨*func-call*⟩ | ⟨*var-name*⟩ | `'('` ⟨*expr*⟩ `')'` | ⟨*tuple-constructor*⟩ | ⟨*array-constructor*⟩ *(base expressions)*

⟨*func-call*⟩ ::= ⟨*annotation*⟩ ⋆ ⟨*func-name*⟩ `'('` ⟨*func-call-arg-list*⟩ `')'` *(function call)*

⟨*func-call-arg-list*⟩ ::= (⟨*expr*⟩ | ⟨*kw-expr*⟩) (`','` (⟨*expr*⟩ | ⟨*kw-expr*⟩))⋆ *(function call argument list)*

⟨*tuple-constructor*⟩ ::= `'('` ⟨*expr*⟩ (`','` ⟨*expr*⟩) + `')'` *(tuple constructor)*

⟨*array-constructor*⟩ ::= ⟨*array-list-constructor*⟩ | ⟨*array-range-constructor*⟩ | ⟨*array-kv-constructor*⟩ *(array constructor)*

⟨*array-list-constructor*⟩ ::= `'['` ⟨*expr-list*⟩? `']'` *(array list constructor)*

⟨*array-range-constructor*⟩ ::= `'['` ⟨*expr*⟩ `':'` ⟨*expr*⟩ (`':'` ⟨*expr*⟩)? `']'` *(array range constructor)*

⟨*array-kv-constructor*⟩ ::= `'{'` (⟨*array-kv-elem*⟩ (`','` ⟨*array-kv-elem*⟩)⋆)? `'}'` *(array key-value constructor)*

⟨*array-kv-elem*⟩ ::= ⟨*expr*⟩ `':'` ⟨*expr*⟩ *(array key-value)*

⟨*annotation*⟩ ::= `'@'` ⟨*id*⟩ | `'@'` ⟨*kw-expr*⟩ *(annotation)*

⟨*kw-expr*⟩ ::= ⟨*id*⟩ `'='` ⟨*expr*⟩ *(keyword argument)*

⟨*literal*⟩ ::= ⟨*string-literal*⟩ | ⟨*multiline-string-literal*⟩ | ⟨*int-literal*⟩ | ⟨*float-literal*⟩ | ⟨*bool-literal*⟩ *(literal value)*

⟨*float-literal*⟩ ::= ⟨*decimal*⟩ | ⟨*sci-decimal*⟩ | `'inf'` | `'NaN'` *(floating point literal)*

⟨*bool-literal*⟩ ::= `'true'` | `'false'` *(boolean literal)*

⟨*expr-list*⟩ ::= ⟨*expr*⟩ (`','` ⟨*expr*⟩)⋆ *(comma-separated expression list)*

⟨*type-name*⟩ ::= ⟨*id*⟩ *(type name)*

⟨*var-name*⟩ ::= ⟨*id*⟩ *(variable name)*

⟨*func-name*⟩ ::= ⟨*id*⟩ *(function name)*

⟨*lval-list*⟩ ::= ⟨*lval-expr*⟩ (`','`⟨*lval-expr*⟩)⋆ *(assignment LValue expression list)*

⟨*lval-expr*⟩ ::= ⟨*var-name*⟩ (⟨*array-subscript*⟩ | ⟨*struct-subscript*⟩)⋆ *(assignment LValue expression)*

⟨*app-arg-expr*⟩ ::= `'@'`?⟨*var-name*⟩ | ⟨*literal*⟩ | ⟨*array-constructor*⟩ | `'('` ⟨*expr*⟩ `')'` *(app function argument expression)*

Figure A.4: Extended Backus-Naur Form grammar for Swift/T expressions.