Formal grammar

Ask AI

Feedback

Compact Grammar

Compact language version 0.17.0.

Notational note: In the grammar productions below, ellipses are used to specify repetition. The notation *X* ... *X*, where *X* is a grammar symbol, represents zero or more occurrences of *X*. The notation *X s* ... *s X*, where *X* is a grammar symbol and *s* is a separator such as a comma or or semicolon, represents zero or more occurrences of *X* separated by *s*. In either case, when the ellipsis is marked with the superscript 1, the notation represents a sequence containing at least one *X*. When such a sequence is followed by s^{opt}, an optional trailing separator is allowed, but only if there is at least one *X*. For example, id … id represents zero or more ids, and expr , …^{1} , expr ,$^{\text{opt}}$ represents one or more comma-separated exprs possibly followed by an extra comma.

end-of-file (*eof*)

end of file

identifier (*id*, *module-name*, *function-name*, *struct-name*, *enum-name*, *contract-name*, *tvar-name*)

identifiers have the same syntax as Typescript identifiers

field-literal (*nat*)

a field literal is 0 or a natural number formed from a sequence of digits starting with 1-9, e.g. 723, whose value does not exceed the maximum field value

string-literal (*str*, *file*)

a string literal has the same syntax as a Typescript string

version-literal (*version*)

a version literal takes the form nat or nat.nat or nat.nat.nat, e.g., 1.2 or 1.2.3, representing major, minor, and bugfix versions

Compact (*program*)

program → *pelt* … *pelt* *eof*

Feedback

Program-element (*pelt*)

pelt → *pragma*
→ *incld*
→ *mdefn*
→ *idecl*
→ *xdecl*
→ *ldecl*
→ *lconstructor*
→ *cdefn*
→ *edecl*
→ *wdecl*
→ *ecdecl*
→ *struct*
→ *enumdef*

Pragma (*pragma*)

pragma → `pragma` *id* *version-expr* `;`

Version-expression (*version-expr*)

version-expr → *version-expr* `||` $version\text{-}expr_0$
→ $version\text{-}expr_0$

Version-expression$_0$ ($version\text{-}expr_0$)

$version\text{-}expr_0$ → $version\text{-}expr_0$ `&&` *version-term*
→ *version-term*

Version-Term (*version-term*)

version-term → *version-atom*
→ `!` *version-atom*

Feedback

→ `<` *version-atom*
→ `<=` *version-atom*
→ `>=` *version-atom*
→ `>` *version-atom*
→ `(` *version-expr* `)`

Version-atom (*version-atom*)

version-atom → *nat*
→ *version*

Include (*incld*)

incld → `include` *file* `;`

Module-definition (*mdefn*)

mdefn → `export`^{opt} `module` *module-name* *gparams*^{opt} `{` *pelt* … *pelt* `}`

Generic-parameter-list (*gparams*)

gparams → `<` *generic-param* `,` … `,` *generic-param* `,`^{opt} `>`

Generic-parameter (*generic-param*)

generic-param → `#` *tvar-name*
→ *tvar-name*

Import-declaration (*idecl*)

idecl → `import` *import-name* *gargs*^{opt} *import-prefix*^{opt} `;`

Import-name (*import-name*)

Feedback

import-name → *id*
→ *file*

Import-prefix (*import-prefix*)

import-prefix → `prefix` *id*

Generic-argument-list (*gargs*)

gargs → `<` *garg* `,` … `,` *garg* `,`$^{\text{opt}}$ `>`

Generic-argument (*garg*)

garg → *nat*
→ *type*

Export-declaration (*xdecl*)

xdecl → `export` `{` *id* `,` … `,` *id* `,`$^{\text{opt}}$ `}` `;`$^{\text{opt}}$

Ledger-declaration (*ldecl*)

ldecl → `export`$^{\text{opt}}$ `sealed`$^{\text{opt}}$ `ledger` *id* `:` *type* `;`

Constructor (*lconstructor*)

lconstructor → `constructor` *pattern-parameter-list* *block*

Circuit-definition (*cdefn*)

cdefn → `export`$^{\text{opt}}$ `pure`$^{\text{opt}}$ `circuit` *function-name* *gparams*$^{\text{opt}}$ *pattern-parameter-list* `:` *type* *block*

External-declaration (*edecl*)

Feedback

edecl → `export`$^{\text{opt}}$ `circuit` *id* *gparams*$^{\text{opt}}$ *simple-parameter-list* `:` *type* `;`

Witness-declaration (*wdecl*)

wdecl → `export`$^{\text{opt}}$ `witness` *id* *gparams*$^{\text{opt}}$ *simple-parameter-list* `:` *type* `;`

External-contract-declaration (*ecdecl*)

ecdecl → `export`$^{\text{opt}}$ `contract` *contract-name* `{` *ecdecl-circuit* `;` … `;` *ecdecl-circuit* `;`$^{\text{opt}}$ `}` `;`$^{\text{opt}}$
→ `export`$^{\text{opt}}$ `contract` *contract-name* `{` *ecdecl-circuit* `,` … `,` *ecdecl-circuit* `,`$^{\text{opt}}$ `}` `;`$^{\text{opt}}$

External-contract-circuit (*ecdecl-circuit*)

ecdecl-circuit → `pure`$^{\text{opt}}$ `circuit` *id* *simple-parameter-list* `:` *type*

Structure-definition (*struct*)

struct → `export`$^{\text{opt}}$ `struct` *struct-name* *gparams*$^{\text{opt}}$ `{` *typed-identifier* `;` … `;` *typed-identifier* `;`$^{\text{opt}}$ `}` `;`$^{\text{opt}}$
→ `export`$^{\text{opt}}$ `struct` *struct-name* *gparams*$^{\text{opt}}$ `{` *typed-identifier* `,` … `,` *typed-identifier* `,`$^{\text{opt}}$ `}` `;`$^{\text{opt}}$

Enum-definition (*enumdef*)

enumdef → `export`$^{\text{opt}}$ `enum` *enum-name* `{` *id* `,` …^{1} `,` *id* `,`$^{\text{opt}}$ `}` `;`$^{\text{opt}}$

Typed-identifier (*typed-identifier*)

typed-identifier → *id* `:` *type*

Simple-parameter-list (*simple-parameter-list*)

simple-parameter-list → `(` *typed-identifier* `,` … `,` *typed-identifier* `,`$^{\text{opt}}$ `)`

Feedback

Typed-pattern (*typed-pattern*)

typed-pattern → *pattern* **:** *type*

Pattern-parameter-list (*pattern-parameter-list*)

pattern-parameter-list → **(** *typed-pattern* **,** … **,** *typed-pattern* **,**$^{\text{opt}}$ **)**

Type (*type*)

type → *tref*
→ `Boolean`
→ `Field`
→ `Uint < ` *tsize* ` >`
→ `Uint < ` *tsize* ` .. ` *tsize* ` >`
→ `Bytes < ` *tsize* ` >`
→ `Opaque < ` *str* ` >`
→ `Vector < ` *tsize* ` , ` *type* ` >`
→ `[` *type* **,** … **,** *type* **,**$^{\text{opt}}$ `]`

Type-reference (*tref*)

tref → *id* *gargs*$^{\text{opt}}$

Type-size (*tsize*)

tsize → *nat*
→ *id*

Block (*block*)

block → **{** *stmt* … *stmt* **}**

Statement (*stmt*)

stmt → *expr-seq* `;`
→ `return` *expr-seq* `;`
→ `return ;`
→ `if (` *expr-seq* `)` *stmt* `else` *stmt*
→ `if (` *expr-seq* `)` *stmt*
→ `for ( const` *id* `of` *nat* `..` *nat* `)` *stmt*
→ `for ( const` *id* `of` *expr-seq* `)` *stmt*
→ `const` *cbinding* `, …`[1] `,` *cbinding* `;`
→ *block*

Pattern (*pattern*)

pattern → *id*
→ `[` $pattern^{opt}$ `, … ,` $pattern^{opt}$ $,^{opt}$ `]`
→ `{` *pattern-struct-elt* `, … ,` *pattern-struct-elt* $,^{opt}$ `}`

Pattern-tuple-element (*pattern-tuple-elt*)

pattern-tuple-elt → (*empty*)
→ *pattern*

Pattern-struct-element (*pattern-struct-elt*)

pattern-struct-elt → *id*
→ *id* `:` *pattern*

Expression-sequence (*expr-seq*)

expr-seq → *expr*
→ *expr* `, …`[1] `,` *expr* `,` *expr*

Expression (*expr*)

expr → $expr_0$ `?` *expr* `:` *expr*

$\rightarrow$ *$expr_0$* `=` *expr*

$\rightarrow$ *$expr_0$* `+=` *expr*

$\rightarrow$ *$expr_0$* `-=` *expr*

$\rightarrow$ *$expr_0$*

Expression$_0$ (*$expr_0$*)

$expr_0 \rightarrow$ *$expr_0$* `||` *$expr_1$*

$\rightarrow$ *$expr_1$*

Expression$_1$ (*$expr_1$*)

$expr_1 \rightarrow$ *$expr_1$* `&&` *$expr_2$*

$\rightarrow$ *$expr_2$*

Expression$_2$ (*$expr_2$*)

$expr_2 \rightarrow$ *$expr_2$* `==` *$expr_3$*

$\rightarrow$ *$expr_2$* `!=` *$expr_3$*

$\rightarrow$ *$expr_3$*

Expression$_3$ (*$expr_3$*)

$expr_3 \rightarrow$ *$expr_4$* `<` *$expr_4$*

$\rightarrow$ *$expr_4$* `<=` *$expr_4$*

$\rightarrow$ *$expr_4$* `>=` *$expr_4$*

$\rightarrow$ *$expr_4$* `>` *$expr_4$*

$\rightarrow$ *$expr_4$*

Expression$_4$ (*$expr_4$*)

Feedback

$expr_4 \rightarrow$ *$expr_4$* `as` *type*

$\rightarrow \mathit{expr}_5$

Expression$_5$ (expr_5)

$\mathit{expr}_5 \rightarrow \mathit{expr}_5 \; \texttt{+} \; \mathit{expr}_6$
$\quad\quad\;\; \rightarrow \mathit{expr}_5 \; \texttt{-} \; \mathit{expr}_6$
$\quad\quad\;\; \rightarrow \mathit{expr}_6$

Expression$_6$ (expr_6)

$\mathit{expr}_6 \rightarrow \mathit{expr}_6 \; \texttt{*} \; \mathit{expr}_7$
$\quad\quad\;\; \rightarrow \mathit{expr}_7$

Expression$_7$ (expr_7)

$\mathit{expr}_7 \rightarrow \texttt{!} \; \mathit{expr}_7$
$\quad\quad\;\; \rightarrow \mathit{expr}_8$

Expression$_8$ (expr_8)

$\mathit{expr}_8 \rightarrow \mathit{expr}_8 \; \texttt{[} \; \mathit{nat} \; \texttt{]}$
$\quad\quad\;\; \rightarrow \mathit{expr}_8 \; \texttt{.} \; \mathit{id}$
$\quad\quad\;\; \rightarrow \mathit{expr}_8 \; \texttt{.} \; \mathit{id} \; \texttt{(} \; \mathit{expr} \; \texttt{,} \ldots \texttt{,} \; \mathit{expr} \; \texttt{,}^{\text{opt}} \; \texttt{)}$
$\quad\quad\;\; \rightarrow \mathit{expr}_9$

Expression$_9$ (expr_9)

$\mathit{expr}_9 \rightarrow \mathit{fun} \; \texttt{(} \; \mathit{expr} \; \texttt{,} \ldots \texttt{,} \; \mathit{expr} \; \texttt{,}^{\text{opt}} \; \texttt{)}$
$\quad\quad\;\; \rightarrow \texttt{map} \; \texttt{(} \; \mathit{fun} \; \texttt{,} \; \mathit{expr} \; \texttt{,} \ldots^{1} \texttt{,} \; \mathit{expr} \; \texttt{,}^{\text{opt}} \; \texttt{)}$
$\quad\quad\;\; \rightarrow \texttt{fold} \; \texttt{(} \; \mathit{fun} \; \texttt{,} \; \mathit{expr} \; \texttt{,} \; \mathit{expr} \; \texttt{,} \ldots^{1} \texttt{,} \; \mathit{expr} \; \texttt{,}^{\text{opt}} \; \texttt{)}$
$\quad\quad\;\; \rightarrow \texttt{[} \; \mathit{expr} \; \texttt{,} \ldots \texttt{,} \; \mathit{expr} \; \texttt{,}^{\text{opt}} \; \texttt{]}$

Feedback

→ *tref* `{` *struct-arg* `,` … `,` *struct-arg* `,`$^{\text{opt}}$ `}`
→ `assert` `(` *expr* `,` *str* `)`
→ `disclose` `(` *expr* `)`
→ *term*

Term (*term*)

term → *id*
→ `true`
→ `false`
→ *nat*
→ *str*
→ `pad` `(` *nat* `,` *str* `)`
→ `default` `<` *type* `>`
→ `(` *expr-seq* `)`

Structure-argument (*struct-arg*)

struct-arg → *expr*
→ *id* `:` *expr*
→ `...` *expr*

Function (*fun*)

fun → *id* *gargs*$^{\text{opt}}$
→ *arrow-parameter-list* *return-type*$^{\text{opt}}$ `=>` *block*
→ *arrow-parameter-list* *return-type*$^{\text{opt}}$ `=>` *expr*
→ `(` *fun* `)`

Return-type (*return-type*)

return-type → `:` *type*

Optionally-typed pattern (*optionally-typed-pattern*)

Optionally-typed-pattern (*optionally-typed-pattern*)

optionally-typed-pattern → *pattern*
→ *typed-pattern*

Const-Binding (*cbinding*)

cbinding → *optionally-typed-pattern* **=** *expr*

Arrow-parameter-list (*arrow-parameter-list*)

arrow-parameter-list → **(** *optionally-typed-pattern* **,** … **,** *optionally-typed-pattern* **,**$^{\text{opt}}$ **)**

Feedback

Feedback