mod ::=

x	
{}	
[val exp]	
[val :typ]	
[type typ]	
[type :$kind$]	
[data typ]	
[data :typ]	
[unit mod]	
[unit :sig]	
new mod	
{module l = mod}	
mod.l	
link x = mod_1 with mod_2	
link x = mod_1 seals mod_2	
mod :> sig	
(mod)	
[val exp:typ]	⇝ [val exp] with [val :typ]
[type]	⇝ [type : #]
[type typ:$kind$]	⇝ [type typ] with [type :$kind$]
[data typ:$kind$]	⇝ [data typ] with [type :$kind$]
[data :typ:$kind$]	⇝ [data :typ] with [type :$kind$]
[unit mod:sig]	⇝ [unit mod] with [unit :sig]
[module mod]	⇝ mod
[module :mod]	⇝ !mod
[module mod_1:mod_2]	⇝ mod_1 with !mod_2
{dec_1⟨,⟩ ⋯ ⟨,⟩dec_n}	⇝ link x = {dec_1} with ⋯ link x = {$dec_n[x.x_i/x_i]$} with {}
!mod	⇝ new mod
mod_1 with mod_2	⇝ link x = mod_1 with mod
mod_1 seals mod_2	⇝ link x = mod_1 seals mod
link x : mod_1 with mod_2	⇝ link x = !mod_1 with mod
link x : mod_1 seals mod_2	⇝ link x = !mod_1 seals mod
let dec_1⟨,⟩ ⋯ ⟨,⟩dec_n in mod	⇝ {dec_1⟨,⟩ ⋯ ⟨,⟩dec_n, module l = mod}.l
let x = mod_1 in mod_2	⇝ let module x = mod_1 in mod_2
fn x = mod_1 in mod_2	⇝ [unit link x = {module Arg = mod_1} with {module Res = $mod_2[x.\text{Arg}/x]$
fn x : mod_1 in mod_2	⇝ fn x = !mod_1 in mod_2
mod_1 mod_2	⇝ ({module Arg = mod_2} with !mod_1).Res

dec ::=

val p ⟨[α_1, ⋯ ,α_n]⟩ (x_1:typ_1) ⋯ (x_m:typ_m) : typ	⇝ p = [val : ⟨forall [α_1, ⋯ ,α_n] ->⟩ typ_1 -> ⋯ -> typ_m -> typ]
val p ⟨[α_1, ⋯ ,α_n]⟩ (x_1:typ_1) ⋯ (x_m:typ_m) ⟨: typ⟩ = exp	⇝ p = [val ⟨fn [α_1, ⋯ ,α_n] ->⟩ fn x_1:typ_1 -> ⋯ fn x_m:typ_m -> exp ⟨: ⟨forall [α_1, ⋯ ,α_n] ->⟩ typ_1 -> ⋯ -> typ_m -> typ⟩]
type p ⟨[α_1, ⋯ ,α_n]⟩	⇝ p = [type ⟨: #n -> #⟩]
type p ⟨[α_1, ⋯ ,α_n]⟩ : $kind$	⇝ p = [type : ⟨#n ->⟩ $kind$]
type p ⟨[α_1, ⋯ ,α_n]⟩ ⟨: $kind$⟩ = typ	⇝ p = [type ⟨fn [α_1, ⋯ ,α_n] ->⟩ typ ⟨: ⟨#n ->⟩ $kind$⟩]
data p ⟨[α_1, ⋯ ,α_n]⟩ ⟨: $kind$⟩ = typ	⇝ p = [data ⟨fn [α_1, ⋯ ,α_n] ->⟩ typ ⟨: ⟨#n ->⟩ $kind$⟩]
data p ⟨[α_1, ⋯ ,α_n]⟩ ⟨: $kind$⟩ : typ	⇝ p = [data : ⟨fn [α_1, ⋯ ,α_n] ->⟩ typ ⟨: ⟨#n ->⟩ $kind$⟩]
unit p : sig	⇝ p = [unit :sig]
unit p ⟨: sig⟩ = mod	⇝ p = [unit mod⟨:sig⟩]
module p : mod	⇝ p = [module :mod]
module p ⟨: mod_1⟩ = mod_2	⇝ p = [module mod_2⟨:mod_1⟩]
module x.ls = mod	⇝ x = {module l_1 = ⋯ {module l_n = mod} ⋯ }
do exp	⇝ val x = exp

$prog$ ::=

$dec_1\langle,\rangle \ldots \langle,\rangle dec_n$

mod $\rightsquigarrow$ `unit it =` mod

sig ::=

mod `import (`ls_1`,` $\ldots$`,`ls_n`)`

mod `export (`ls_1`,` $\ldots$`,`ls_n`)`

mod $\rightsquigarrow$ mod `import ()`

ls ::= $\epsilon \mid ls.l$

p ::= $x \mid x\,.ls$

$kind$ ::=

`#`

`#`n `->` `#`

`# ->` `#` $\rightsquigarrow$ `#1 ->` `#`

typ ::=

`!`mod

`int`

`string`

`(`typ_1`,` $\ldots$`,`typ_n`)`

`(`typ_1`|` $\ldots$ `|`typ_n`)`

typ_1 `->` typ_2

`forall [`α_1`,` $\ldots$`,`α_n`]` `->` typ

`fn [`α_1`,` $\ldots$`,`α_n`]` `->` typ

typ `[`typ_1`,` $\ldots$`,`typ_n`]`

`(`typ`)`

p $\rightsquigarrow$ `!`p

`bool` $\rightsquigarrow$ `((`$\mid$`())`

exp ::=

`!`mod

n

s

exp_1 `+` exp_2

exp_1 `-` exp_2

exp_1 `==` exp_2

exp_1 `<` exp_2

exp_1 `++` exp_2

`(`exp_1`,` $\ldots$`,`exp_n`)`

exp`#`n

exp`@`n`[`typ`]`

`case` exp `of` x_1`->`exp_1 `|` $\ldots$ `|` x_n`->`exp_n

`fn` x`:`typ `->` exp

exp_1 exp_2

`fn [`α_1`,` $\ldots$`,`α_n`]` `->` exp

exp `[`typ_1`,` $\ldots$`,`typ_n`]`

`in` mod `[`typ_1`,` $\ldots$`,`typ_n`]` exp

`out` mod `[`typ_1`,` $\ldots$`,`typ_n`]` exp

`let module` x `=` mod `in` exp

`print` exp

`(`exp`)`

p $\rightsquigarrow$ `!`p

`@`n`[`typ`]` $\rightsquigarrow$ `()@`n`[`typ`]`

`false` $\rightsquigarrow$ `@1[bool]`

`true` $\rightsquigarrow$ `@2[bool]`

`if` exp_1 `then` exp_2 `else` exp_3 $\rightsquigarrow$ `case` exp_1 `of` x`->`exp_3 `|` x`->`exp_2

`let` $dec_1\langle,\rangle \ldots \langle,\rangle dec_n$ `in` exp $\rightsquigarrow$ `let` dec_1 `in` $\ldots$ `let` dec_n `in` exp

`let` x `=` exp_1 `in` exp_2 $\rightsquigarrow$ `let` x `=` `[val` exp_1`]` `in` exp_2

`(`exp `:` typ`)` $\rightsquigarrow$ `let` x `=` `[val` exp`:`typ`]` `in` x

exp_1 `;` exp_2 $\rightsquigarrow$ `let` x `=` exp_1 `in` exp_2