

MaxHs Grammar

<i>module</i>	→ module ModId <i>exports</i> where <i>body</i>	<i>bit_or_exp</i>	→ <i>bit_or_exp</i> <i>bit_xor_exp</i>
<i>exports</i>	→ ϵ	<i>bit_or_exp</i>	→ <i>bit_xor_exp</i>
<i>exports</i>	→ (<i>exports</i> 1)	<i>bit_xor_exp</i>	→ <i>bit_xor_exp</i> ^ <i>bit_and_exp</i>
<i>exports</i> 1	→ <i>var</i>	<i>bit_xor_exp</i>	→ <i>bit_and_exp</i>
<i>exports</i> 1	→ <i>var</i> , <i>exports</i> 1	<i>bit_and_exp</i>	→ <i>bit_and_exp</i> & <i>comp_exp</i>
<i>body</i>	→ { <i>decls</i> }	<i>bit_and_exp</i>	→ <i>comp_exp</i>
<i>decls</i>	→ <i>decl</i>	<i>comp_exp</i>	→ <i>add_exp</i> == <i>add_exp</i>
<i>decls</i>	→ <i>decl</i> ; <i>decls</i>	<i>comp_exp</i>	→ <i>add_exp</i> != <i>add_exp</i>
<i>decl</i>	→ <i>gendekl</i>	<i>comp_exp</i>	→ <i>add_exp</i> < <i>add_exp</i>
<i>decl</i>	→ <i>funlhs rhs</i>	<i>comp_exp</i>	→ <i>add_exp</i> > <i>add_exp</i>
<i>decl</i>	→ <i>var rhs</i>	<i>comp_exp</i>	→ <i>add_exp</i> <= <i>add_exp</i>
<i>gendekl</i>	→ <i>vars</i> :: <i>type</i>	<i>comp_exp</i>	→ <i>add_exp</i> >= <i>add_exp</i>
<i>vars</i>	→ <i>var</i>	<i>comp_exp</i>	→ <i>add_exp</i>
<i>vars</i>	→ <i>var</i> , <i>vars</i>	<i>add_exp</i>	→ <i>add_exp</i> + <i>mul_exp</i>
<i>type</i>	→ <i>atype</i>	<i>add_exp</i>	→ <i>add_exp</i> - <i>mul_exp</i>
<i>type</i>	→ <i>atype</i> -> <i>type</i>	<i>add_exp</i>	→ <i>mul_exp</i>
<i>atype</i>	→ ConId	<i>mul_exp</i>	→ <i>mul_exp</i> * <i>pre_exp</i>
<i>atype</i>	→ (<i>types</i>)	<i>mul_exp</i>	→ <i>mul_exp</i> / <i>pre_exp</i>
<i>atype</i>	→ [<i>type</i>]	<i>mul_exp</i>	→ <i>pre_exp</i>
<i>types</i>	→ <i>type</i>	<i>pre_exp</i>	→ - <i>exp</i> 10
<i>types</i>	→ <i>type</i> , <i>types</i>	<i>pre_exp</i>	→ <i>exp</i> 10
<i>funlhs</i>	→ <i>var args</i>	<i>exp</i> 10	→ let { <i>decls</i> } in <i>exp</i>
<i>args</i>	→ <i>var</i>	<i>exp</i> 10	→ if <i>exp</i> then <i>exp</i> else <i>exp</i>
<i>args</i>	→ <i>var args</i>	<i>exp</i> 10	→ <i>fexp</i>
<i>rhs</i>	→ = <i>exp</i>	<i>fexp</i>	→ <i>aexp</i>
<i>rhs</i>	→ = <i>exp</i> where { <i>decls</i> }	<i>fexp</i>	→ <i>fexp</i> <i>aexp</i>
<i>exp</i>	→ <i>bit_or_exp</i> :: <i>type</i>	<i>aexp</i>	→ <i>var</i>
<i>exp</i>	→ <i>bit_or_exp</i>	<i>aexp</i>	→ <i>literal</i>
		<i>aexp</i>	→ (<i>exps</i>)
		<i>aexp</i>	→ [<i>exps</i>]
		<i>exps</i>	→ <i>exp</i>
		<i>exps</i>	→ <i>exp</i> , <i>exps</i>

LL grammar

<i>module</i>	→ module ModId exports where body	<i>bit_xor_exp</i>	→ bit_and_exp bit_xor_exp'
<i>exports</i>	→ ε	<i>bit_xor_exp'</i>	→ ε
<i>exports</i>	→ (var exports')	<i>bit_xor_exp'</i>	→ ^ bit_and_exp bit_xor_exp'
<i>exports'</i>	→ ε	<i>bit_and_exp</i>	→ comp_exp bit_and_exp'
<i>exports'</i>	→ , var exports'	<i>bit_and_exp'</i>	→ ε
<i>body</i>	→ {decls}	<i>bit_and_exp'</i>	→ & comp_exp bit_and_exp'
<i>decls</i>	→ ε	<i>comp_exp</i>	→ add_exp comp_exp'
<i>decls</i>	→ decl decls'	<i>comp_exp'</i>	→ ε
<i>decls'</i>	→ ε	<i>comp_exp'</i>	→ === add_exp
<i>decls'</i>	→ ; decl decls'	<i>comp_exp'</i>	→ !== add_exp
<i>decl</i>	→ var decl'	<i>comp_exp'</i>	→ < add_exp
<i>decl'</i>	→ gendekl'	<i>comp_exp'</i>	→ > add_exp
<i>decl'</i>	→ funlhs' rhs	<i>comp_exp'</i>	→ <= add_exp
<i>decl'</i>	→ rhs	<i>comp_exp'</i>	→ >= add_exp
<i>gendekl'</i>	→ vars' :: type	<i>add_exp</i>	→ mul_exp add_exp'
<i>vars'</i>	→ ε	<i>add_exp'</i>	→ ε
<i>vars'</i>	→ , vars'	<i>add_exp'</i>	→ + mul_exp add_exp'
<i>type</i>	→ atype type'	<i>add_exp'</i>	→ - mul_exp add_exp'
<i>type'</i>	→ ε	<i>mul_exp</i>	→ pre_exp mul_exp'
<i>type'</i>	→ -> atype type'	<i>mul_exp'</i>	→ ε
<i>atype</i>	→ ConId	<i>mul_exp'</i>	→ * pre_exp mul_exp'
<i>atype</i>	→ (types)	<i>mul_exp'</i>	→ / pre_exp mul_exp'
<i>atype</i>	→ [type]	<i>pre_exp</i>	→ - exp10
<i>types</i>	→ type types'	<i>pre_exp</i>	→ exp10
<i>types'</i>	→ ε	<i>exp10</i>	→ let {decls} in exp
<i>types'</i>	→ , type types'	<i>exp10</i>	→ if exp then exp else exp
<i>funlhs'</i>	→ var args'	<i>exp10</i>	→ fexp
<i>args'</i>	→ ε	<i>fexp</i>	→ aexp fexp'
<i>args'</i>	→ var args'	<i>fexp'</i>	→ ε
<i>rhs</i>	→ = exp	<i>fexp'</i>	→ aexp fexp'
<i>rhs</i>	→ = exp where {decls}	<i>aexp</i>	→ var
<i>exp</i>	→ bit_or_exp :: type	<i>aexp</i>	→ literal
<i>exp</i>	→ bit_or_exp	<i>aexp</i>	→ (exps)
<i>bit_or_exp</i>	→ bit_xor_exp bit_or_exp'	<i>aexp</i>	→ [exps]
<i>bit_or_exp'</i>	→ ε	<i>exps</i>	→ exp exps'
<i>bit_or_exp'</i>	→ bit_xor_exp bit_or_exp'	<i>exps'</i>	→ ε
		<i>exps'</i>	→ , exp exps'