Annex A
(informative)
Grammar Summary

A.1 Lexical Grammar

SourceCharacter ::
    any Unicode code point
InputElementDiv ::
    WhiteSpace
    LineTerminator
    Comment
    CommonToken
    DivPunctuator
    RightBracePunctuator
InputElementRegExp ::
    WhiteSpace
    LineTerminator
    Comment
    CommonToken
    RightBracePunctuator
    RegularExpressionLiteral
InputElementRegExpOrTemplateTail ::
    WhiteSpace
    LineTerminator
    Comment
    CommonToken
    RegularExpressionLiteral
    TemplateSubstitutionTail
InputElementTemplateTail ::
    WhiteSpace
    LineTerminator
    Comment
    CommonToken
    DivPunctuator
    TemplateSubstitutionTail
InputElementHashbangOrRegExp ::
    WhiteSpace
    LineTerminator
    Comment
    CommonToken
    HashbangComment
    RegularExpressionLiteral
WhiteSpace ::
    <TAB>
    <VT>
    <FF>
    <ZWNBSP>
    <USP>

LineTerminator ::
<LF>
<CR>
<LS>
<PS>
LineTerminatorSequence ::
<LF>
<CR> [lookahead ≠ <LF>]
<LS>
<PS>
<CR> <LF>
Comment ::
MultiLineComment
SingleLineComment
MultiLineComment ::
/* *MultiLineCommentChars*$_{opt}$ ***/**
MultiLineCommentChars ::
MultiLineNotAsteriskChar *MultiLineCommentChars*$_{opt}$
***** *PostAsteriskCommentChars*$_{opt}$
PostAsteriskCommentChars ::
MultiLineNotForwardSlashOrAsteriskChar *MultiLineCommentChars*$_{opt}$
***** *PostAsteriskCommentChars*$_{opt}$
MultiLineNotAsteriskChar ::
SourceCharacter but not *****
MultiLineNotForwardSlashOrAsteriskChar ::
SourceCharacter but not one of **/** or *****
SingleLineComment ::
// *SingleLineCommentChars*$_{opt}$
SingleLineCommentChars ::
SingleLineCommentChar *SingleLineCommentChars*$_{opt}$
SingleLineCommentChar ::
SourceCharacter but not *LineTerminator*
HashbangComment ::
#! *SingleLineCommentChars*$_{opt}$
CommonToken ::
IdentifierName
PrivateIdentifier
Punctuator
NumericLiteral
StringLiteral
Template
PrivateIdentifier ::
*IdentifierName*
IdentifierName ::
IdentifierStart
IdentifierName *IdentifierPart*
IdentifierStart ::
IdentifierStartChar
**** *UnicodeEscapeSequence*
IdentifierPart ::
IdentifierPartChar
**** *UnicodeEscapeSequence*

IdentifierStartChar ::
UnicodeIDStart
$
_

IdentifierPartChar ::
UnicodeIDContinue
$
<ZWNJ>
<ZWJ>

AsciiLetter :: **one of**
a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

UnicodeIDStart ::
any Unicode code point with the Unicode property “ID_Start”

UnicodeIDContinue ::
any Unicode code point with the Unicode property “ID_Continue”

ReservedWord :: **one of**
await break case catch class const continue debugger default delete do else enum export extends false finally for function if import in instanceof new null return super switch this throw true try typeof var void while with yield

Punctuator ::
OptionalChainingPunctuator
OtherPunctuator

OptionalChainingPunctuator ::
?. [lookahead ∉ *DecimalDigit*]

OtherPunctuator :: **one of**
{ ( ) [ ] . ... ; , < > <= >= == != === !== + - * % ** ++ -- << >> >>> & | ^ ! ~ && || ?? ? : = += -= *= %= **= <<= >>= >>>= &= |= ^= &&= ||= ??= =>

DivPunctuator ::
/
/=

RightBracePunctuator ::
}

NullLiteral ::
null

BooleanLiteral ::
true
false

NumericLiteralSeparator ::
_

NumericLiteral ::
DecimalLiteral
DecimalBigIntegerLiteral
NonDecimalIntegerLiteral[+Sep]
NonDecimalIntegerLiteral[+Sep] *BigIntLiteralSuffix*
LegacyOctalIntegerLiteral

DecimalBigIntegerLiteral ::
0 *BigIntLiteralSuffix*
NonZeroDigit *DecimalDigits*[+Sep] opt *BigIntLiteralSuffix*
NonZeroDigit *NumericLiteralSeparator* *DecimalDigits*[+Sep] *BigIntLiteralSuffix*

NonDecimalIntegerLiteral[Sep] ::
BinaryIntegerLiteral[?Sep]
OctalIntegerLiteral[?Sep]
HexIntegerLiteral[?Sep]
BigIntLiteralSuffix ::
n
DecimalLiteral ::
DecimalIntegerLiteral **.** *DecimalDigits*[+Sep] opt *ExponentPart*[+Sep] opt
. *DecimalDigits*[+Sep] *ExponentPart*[+Sep] opt
DecimalIntegerLiteral *ExponentPart*[+Sep] opt
DecimalIntegerLiteral ::
0
NonZeroDigit
NonZeroDigit *NumericLiteralSeparator*opt *DecimalDigits*[+Sep]
NonOctalDecimalIntegerLiteral
DecimalDigits[Sep] ::
DecimalDigit
DecimalDigits[?Sep] *DecimalDigit*
[+Sep] *DecimalDigits*[+Sep] *NumericLiteralSeparator* *DecimalDigit*
DecimalDigit :: **one of**
0 1 2 3 4 5 6 7 8 9
NonZeroDigit :: **one of**
1 2 3 4 5 6 7 8 9
ExponentPart[Sep] ::
ExponentIndicator *SignedInteger*[?Sep]
ExponentIndicator :: **one of**
e E
SignedInteger[Sep] ::
DecimalDigits[?Sep]
+ *DecimalDigits*[?Sep]
- *DecimalDigits*[?Sep]
BinaryIntegerLiteral[Sep] ::
0b *BinaryDigits*[?Sep]
0B *BinaryDigits*[?Sep]
BinaryDigits[Sep] ::
BinaryDigit
BinaryDigits[?Sep] *BinaryDigit*
[+Sep] *BinaryDigits*[+Sep] *NumericLiteralSeparator* *BinaryDigit*
BinaryDigit :: **one of**
0 1
OctalIntegerLiteral[Sep] ::
0o *OctalDigits*[?Sep]
0O *OctalDigits*[?Sep]
OctalDigits[Sep] ::
OctalDigit
OctalDigits[?Sep] *OctalDigit*
[+Sep] *OctalDigits*[+Sep] *NumericLiteralSeparator* *OctalDigit*
LegacyOctalIntegerLiteral ::
0 *OctalDigit*
LegacyOctalIntegerLiteral *OctalDigit*

NonOctalDecimalIntegerLiteral ::
0 *NonOctalDigit*
LegacyOctalLikeDecimalIntegerLiteral *NonOctalDigit*
NonOctalDecimalIntegerLiteral *DecimalDigit*
LegacyOctalLikeDecimalIntegerLiteral ::
0 *OctalDigit*
LegacyOctalLikeDecimalIntegerLiteral *OctalDigit*
OctalDigit :: **one of**
0 1 2 3 4 5 6 7
NonOctalDigit :: **one of**
8 9
HexIntegerLiteral[Sep] ::
0x *HexDigits*[?Sep]
0X *HexDigits*[?Sep]
HexDigits[Sep] ::
HexDigit
HexDigits[?Sep] *HexDigit*
[+Sep] *HexDigits*[+Sep] *NumericLiteralSeparator* *HexDigit*
HexDigit :: **one of**
0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F
StringLiteral ::
" *DoubleStringCharacters*opt **"**
' *SingleStringCharacters*opt **'**
DoubleStringCharacters ::
DoubleStringCharacter *DoubleStringCharacters*opt
SingleStringCharacters ::
SingleStringCharacter *SingleStringCharacters*opt
DoubleStringCharacter ::
SourceCharacter but not one of **"** or **** or *LineTerminator*
<LS>
<PS>
**** *EscapeSequence*
LineContinuation
SingleStringCharacter ::
SourceCharacter but not one of **'** or **** or *LineTerminator*
<LS>
<PS>
**** *EscapeSequence*
LineContinuation
LineContinuation ::
**** *LineTerminatorSequence*
EscapeSequence ::
CharacterEscapeSequence
0 [lookahead ∉ *DecimalDigit*]
LegacyOctalEscapeSequence
NonOctalDecimalEscapeSequence
HexEscapeSequence
UnicodeEscapeSequence
CharacterEscapeSequence ::
SingleEscapeCharacter
NonEscapeCharacter
SingleEscapeCharacter :: **one of**
' " \ b f n r t v

NonEscapeCharacter ::
    SourceCharacter but not one of *EscapeCharacter* or *LineTerminator*
EscapeCharacter ::
    SingleEscapeCharacter
    DecimalDigit
    x
    u
LegacyOctalEscapeSequence ::
    0 [lookahead ∈ { **8** , **9** }]
    NonZeroOctalDigit [lookahead ∉ *OctalDigit*]
    ZeroToThree *OctalDigit* [lookahead ∉ *OctalDigit*]
    FourToSeven *OctalDigit*
    ZeroToThree *OctalDigit* *OctalDigit*
NonZeroOctalDigit ::
    OctalDigit but not **0**
ZeroToThree :: **one of**
    0 **1** **2** **3**
FourToSeven :: **one of**
    4 **5** **6** **7**
NonOctalDecimalEscapeSequence :: **one of**
    8 **9**
HexEscapeSequence ::
    x *HexDigit* *HexDigit*
UnicodeEscapeSequence ::
    u *Hex4Digits*
    u{ *CodePoint* **}**
Hex4Digits ::
    HexDigit *HexDigit* *HexDigit* *HexDigit*
RegularExpressionLiteral ::
    / *RegularExpressionBody* **/** *RegularExpressionFlags*
RegularExpressionBody ::
    RegularExpressionFirstChar *RegularExpressionChars*
RegularExpressionChars ::
    [empty]
    RegularExpressionChars *RegularExpressionChar*
RegularExpressionFirstChar ::
    RegularExpressionNonTerminator but not one of ***** or **** or **/** or **[**
    RegularExpressionBackslashSequence
    RegularExpressionClass
RegularExpressionChar ::
    RegularExpressionNonTerminator but not one of **** or **/** or **[**
    RegularExpressionBackslashSequence
    RegularExpressionClass
RegularExpressionBackslashSequence ::
    **** *RegularExpressionNonTerminator*
RegularExpressionNonTerminator ::
    SourceCharacter but not *LineTerminator*
RegularExpressionClass ::
    [*RegularExpressionClassChars* **]**
RegularExpressionClassChars ::
    [empty]
    RegularExpressionClassChars *RegularExpressionClassChar*

RegularExpressionClassChar ::
RegularExpressionNonTerminator but not one of **]** or ****
RegularExpressionBackslashSequence
RegularExpressionFlags ::
[empty]
RegularExpressionFlags *IdentifierPartChar*
Template ::
NoSubstitutionTemplate
TemplateHead
NoSubstitutionTemplate ::
`** *TemplateCharacters*$_{opt}$ **`
TemplateHead ::
` *TemplateCharacters*$_{opt}$ **${**
TemplateSubstitutionTail ::
TemplateMiddle
TemplateTail
TemplateMiddle ::
} *TemplateCharacters*$_{opt}$ **${**
TemplateTail ::
} *TemplateCharacters*$_{opt}$ **`**
TemplateCharacters ::
TemplateCharacter *TemplateCharacters*$_{opt}$
TemplateCharacter ::
$ [lookahead ≠ **{**]
**** *TemplateEscapeSequence*
**** *NotEscapeSequence*
LineContinuation
LineTerminatorSequence
SourceCharacter but not one of **`** or **** or **$** or *LineTerminator*
TemplateEscapeSequence ::
CharacterEscapeSequence
0 [lookahead ∉ *DecimalDigit*]
HexEscapeSequence
UnicodeEscapeSequence
NotEscapeSequence ::
0 *DecimalDigit*
DecimalDigit but not **0**
x [lookahead ∉ *HexDigit*]
x *HexDigit* [lookahead ∉ *HexDigit*]
u [lookahead ∉ *HexDigit*] [lookahead ≠ **{**]
u *HexDigit* [lookahead ∉ *HexDigit*]
u *HexDigit* *HexDigit* [lookahead ∉ *HexDigit*]
u *HexDigit* *HexDigit* *HexDigit* [lookahead ∉ *HexDigit*]
u **{** [lookahead ∉ *HexDigit*]
u **{** *NotCodePoint* [lookahead ∉ *HexDigit*]
u **{** *CodePoint* [lookahead ∉ *HexDigit*] [lookahead ≠ **}**]
NotCodePoint ::
HexDigits$_{[\sim Sep]}$ but only if MV of *HexDigits* > 0x10FFFF
CodePoint ::
HexDigits$_{[\sim Sep]}$ but only if MV of *HexDigits* ≤ 0x10FFFF

A.2 Expressions

IdentifierReference[Yield, Await] :
Identifier
[~Yield] **yield**
[~Await] **await**

BindingIdentifier[Yield, Await] :
Identifier
yield
await

LabelIdentifier[Yield, Await] :
Identifier
[~Yield] **yield**
[~Await] **await**

Identifier :
IdentifierName but not *ReservedWord*

PrimaryExpression[Yield, Await] :
this
IdentifierReference[?Yield, ?Await]
Literal
ArrayLiteral[?Yield, ?Await]
ObjectLiteral[?Yield, ?Await]
FunctionExpression
ClassExpression[?Yield, ?Await]
GeneratorExpression
AsyncFunctionExpression
AsyncGeneratorExpression
RegularExpressionLiteral
TemplateLiteral[?Yield, ?Await, ~Tagged]
CoverParenthesizedExpressionAndArrowParameterList[?Yield, ?Await]

CoverParenthesizedExpressionAndArrowParameterList[Yield, Await] :
(*Expression*[+In, ?Yield, ?Await])
(*Expression*[+In, ?Yield, ?Await] ,)
( )
(... *BindingIdentifier*[?Yield, ?Await])
(... *BindingPattern*[?Yield, ?Await])
(*Expression*[+In, ?Yield, ?Await] , ... *BindingIdentifier*[?Yield, ?Await])
(*Expression*[+In, ?Yield, ?Await] , ... *BindingPattern*[?Yield, ?Await])

When processing an instance of the production
PrimaryExpression[Yield, Await] : *CoverParenthesizedExpressionAndArrowParameterList*[?Yield, ?Await]
the interpretation of *CoverParenthesizedExpressionAndArrowParameterList* is refined using the following grammar:

ParenthesizedExpression[Yield, Await] :
(*Expression*[+In, ?Yield, ?Await])

Literal :
NullLiteral
BooleanLiteral
NumericLiteral
StringLiteral

ArrayLiteral`[Yield, Await]` :
[*Elision*`opt` **]**
[*ElementList*`[?Yield, ?Await]` **]**
[*ElementList*`[?Yield, ?Await]` **,** *Elision*`opt` **]**

ElementList`[Yield, Await]` :
Elision`opt` *AssignmentExpression*`[+In, ?Yield, ?Await]`
Elision`opt` *SpreadElement*`[?Yield, ?Await]`
ElementList`[?Yield, ?Await]` **,** *Elision*`opt` *AssignmentExpression*`[+In, ?Yield, ?Await]`
ElementList`[?Yield, ?Await]` **,** *Elision*`opt` *SpreadElement*`[?Yield, ?Await]`

Elision :
,
Elision **,**

SpreadElement`[Yield, Await]` :
... *AssignmentExpression*`[+In, ?Yield, ?Await]`

ObjectLiteral`[Yield, Await]` :
{ }
{ *PropertyDefinitionList*`[?Yield, ?Await]` **}**
{ *PropertyDefinitionList*`[?Yield, ?Await]` **, }**

PropertyDefinitionList`[Yield, Await]` :
PropertyDefinition`[?Yield, ?Await]`
PropertyDefinitionList`[?Yield, ?Await]` **,** *PropertyDefinition*`[?Yield, ?Await]`

PropertyDefinition`[Yield, Await]` :
IdentifierReference`[?Yield, ?Await]`
CoverInitializedName`[?Yield, ?Await]`
PropertyName`[?Yield, ?Await]` **:** *AssignmentExpression*`[+In, ?Yield, ?Await]`
MethodDefinition`[?Yield, ?Await]`
... *AssignmentExpression*`[+In, ?Yield, ?Await]`

PropertyName`[Yield, Await]` :
LiteralPropertyName
ComputedPropertyName`[?Yield, ?Await]`

LiteralPropertyName :
IdentifierName
StringLiteral
NumericLiteral

ComputedPropertyName`[Yield, Await]` :
[*AssignmentExpression*`[+In, ?Yield, ?Await]` **]**

CoverInitializedName`[Yield, Await]` :
IdentifierReference`[?Yield, ?Await]` *Initializer*`[+In, ?Yield, ?Await]`

Initializer`[In, Yield, Await]` :
= *AssignmentExpression*`[?In, ?Yield, ?Await]`

TemplateLiteral`[Yield, Await, Tagged]` :
NoSubstitutionTemplate
SubstitutionTemplate`[?Yield, ?Await, ?Tagged]`

SubstitutionTemplate`[Yield, Await, Tagged]` :
TemplateHead *Expression*`[+In, ?Yield, ?Await]` *TemplateSpans*`[?Yield, ?Await, ?Tagged]`

TemplateSpans`[Yield, Await, Tagged]` :
TemplateTail
TemplateMiddleList`[?Yield, ?Await, ?Tagged]` *TemplateTail*
TemplateMiddleList`[Yield, Await, Tagged]` :
TemplateMiddle *Expression*`[+In, ?Yield, ?Await]`
TemplateMiddleList`[?Yield, ?Await, ?Tagged]` *TemplateMiddle* *Expression*`[+In, ?Yield, ?Await]`
MemberExpression`[Yield, Await]` :
PrimaryExpression`[?Yield, ?Await]`
MemberExpression`[?Yield, ?Await]` **[** *Expression*`[+In, ?Yield, ?Await]` **]**
MemberExpression`[?Yield, ?Await]` **.** *IdentifierName*
MemberExpression`[?Yield, ?Await]` *TemplateLiteral*`[?Yield, ?Await, +Tagged]`
SuperProperty`[?Yield, ?Await]`
MetaProperty
new *MemberExpression*`[?Yield, ?Await]` *Arguments*`[?Yield, ?Await]`
MemberExpression`[?Yield, ?Await]` **.** *PrivateIdentifier*
SuperProperty`[Yield, Await]` :
super **[** *Expression*`[+In, ?Yield, ?Await]` **]**
super **.** *IdentifierName*
MetaProperty :
NewTarget
ImportMeta
NewTarget :
new **.** **target**
ImportMeta :
import **.** **meta**
NewExpression`[Yield, Await]` :
MemberExpression`[?Yield, ?Await]`
new *NewExpression*`[?Yield, ?Await]`
CallExpression`[Yield, Await]` :
CoverCallExpressionAndAsyncArrowHead`[?Yield, ?Await]`
SuperCall`[?Yield, ?Await]`
ImportCall`[?Yield, ?Await]`
CallExpression`[?Yield, ?Await]` *Arguments*`[?Yield, ?Await]`
CallExpression`[?Yield, ?Await]` **[** *Expression*`[+In, ?Yield, ?Await]` **]**
CallExpression`[?Yield, ?Await]` **.** *IdentifierName*
CallExpression`[?Yield, ?Await]` *TemplateLiteral*`[?Yield, ?Await, +Tagged]`
CallExpression`[?Yield, ?Await]` **.** *PrivateIdentifier*

When processing an instance of the production
CallExpression`[Yield, Await]` : *CoverCallExpressionAndAsyncArrowHead*`[?Yield, ?Await]`
the interpretation of *CoverCallExpressionAndAsyncArrowHead* is refined using the following grammar:

CallMemberExpression`[Yield, Await]` :
MemberExpression`[?Yield, ?Await]` *Arguments*`[?Yield, ?Await]`

SuperCall`[Yield, Await]` :
super *Arguments*`[?Yield, ?Await]`
ImportCall`[Yield, Await]` :
import **(** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)**

Arguments[Yield, Await] :
( )
(*ArgumentList*[?Yield, ?Await])
(*ArgumentList*[?Yield, ?Await] ,)
ArgumentList[Yield, Await] :
AssignmentExpression[+In, ?Yield, ?Await]
... *AssignmentExpression*[+In, ?Yield, ?Await]
ArgumentList[?Yield, ?Await] , *AssignmentExpression*[+In, ?Yield, ?Await]
ArgumentList[?Yield, ?Await] , ... *AssignmentExpression*[+In, ?Yield, ?Await]
OptionalExpression[Yield, Await] :
MemberExpression[?Yield, ?Await] *OptionalChain*[?Yield, ?Await]
CallExpression[?Yield, ?Await] *OptionalChain*[?Yield, ?Await]
OptionalExpression[?Yield, ?Await] *OptionalChain*[?Yield, ?Await]
OptionalChain[Yield, Await] :
?. *Arguments*[?Yield, ?Await]
?. [*Expression*[+In, ?Yield, ?Await]]
?. *IdentifierName*
?. *TemplateLiteral*[?Yield, ?Await, +Tagged]
?. *PrivateIdentifier*
OptionalChain[?Yield, ?Await] *Arguments*[?Yield, ?Await]
OptionalChain[?Yield, ?Await] [*Expression*[+In, ?Yield, ?Await]]
OptionalChain[?Yield, ?Await] . *IdentifierName*
OptionalChain[?Yield, ?Await] *TemplateLiteral*[?Yield, ?Await, +Tagged]
OptionalChain[?Yield, ?Await] . *PrivateIdentifier*
LeftHandSideExpression[Yield, Await] :
NewExpression[?Yield, ?Await]
CallExpression[?Yield, ?Await]
OptionalExpression[?Yield, ?Await]
UpdateExpression[Yield, Await] :
LeftHandSideExpression[?Yield, ?Await]
LeftHandSideExpression[?Yield, ?Await] [no *LineTerminator* here] ++
LeftHandSideExpression[?Yield, ?Await] [no *LineTerminator* here] --
++ *UnaryExpression*[?Yield, ?Await]
-- *UnaryExpression*[?Yield, ?Await]
UnaryExpression[Yield, Await] :
UpdateExpression[?Yield, ?Await]
delete *UnaryExpression*[?Yield, ?Await]
void *UnaryExpression*[?Yield, ?Await]
typeof *UnaryExpression*[?Yield, ?Await]
+ *UnaryExpression*[?Yield, ?Await]
- *UnaryExpression*[?Yield, ?Await]
~ *UnaryExpression*[?Yield, ?Await]
! *UnaryExpression*[?Yield, ?Await]
[+Await] *AwaitExpression*[?Yield]
ExponentiationExpression[Yield, Await] :
UnaryExpression[?Yield, ?Await]
UpdateExpression[?Yield, ?Await] ** *ExponentiationExpression*[?Yield, ?Await]

MultiplicativeExpression`[Yield, Await]` :
ExponentiationExpression`[?Yield, ?Await]`
MultiplicativeExpression`[?Yield, ?Await]` *MultiplicativeOperator* *ExponentiationExpression*`[?Yield, ?Await]`
MultiplicativeOperator : **one of**
*** / %**
AdditiveExpression`[Yield, Await]` :
MultiplicativeExpression`[?Yield, ?Await]`
AdditiveExpression`[?Yield, ?Await]` **+** *MultiplicativeExpression*`[?Yield, ?Await]`
AdditiveExpression`[?Yield, ?Await]` **-** *MultiplicativeExpression*`[?Yield, ?Await]`
ShiftExpression`[Yield, Await]` :
AdditiveExpression`[?Yield, ?Await]`
ShiftExpression`[?Yield, ?Await]` **<<** *AdditiveExpression*`[?Yield, ?Await]`
ShiftExpression`[?Yield, ?Await]` **>>** *AdditiveExpression*`[?Yield, ?Await]`
ShiftExpression`[?Yield, ?Await]` **>>>** *AdditiveExpression*`[?Yield, ?Await]`
RelationalExpression`[In, Yield, Await]` :
ShiftExpression`[?Yield, ?Await]`
RelationalExpression`[?In, ?Yield, ?Await]` **<** *ShiftExpression*`[?Yield, ?Await]`
RelationalExpression`[?In, ?Yield, ?Await]` **>** *ShiftExpression*`[?Yield, ?Await]`
RelationalExpression`[?In, ?Yield, ?Await]` **<=** *ShiftExpression*`[?Yield, ?Await]`
RelationalExpression`[?In, ?Yield, ?Await]` **>=** *ShiftExpression*`[?Yield, ?Await]`
RelationalExpression`[?In, ?Yield, ?Await]` **instanceof** *ShiftExpression*`[?Yield, ?Await]`
`[+In]` *RelationalExpression*`[+In, ?Yield, ?Await]` **in** *ShiftExpression*`[?Yield, ?Await]`
`[+In]` *PrivateIdentifier* **in** *ShiftExpression*`[?Yield, ?Await]`
EqualityExpression`[In, Yield, Await]` :
RelationalExpression`[?In, ?Yield, ?Await]`
EqualityExpression`[?In, ?Yield, ?Await]` **==** *RelationalExpression*`[?In, ?Yield, ?Await]`
EqualityExpression`[?In, ?Yield, ?Await]` **!=** *RelationalExpression*`[?In, ?Yield, ?Await]`
EqualityExpression`[?In, ?Yield, ?Await]` **===** *RelationalExpression*`[?In, ?Yield, ?Await]`
EqualityExpression`[?In, ?Yield, ?Await]` **!==** *RelationalExpression*`[?In, ?Yield, ?Await]`
BitwiseANDExpression`[In, Yield, Await]` :
EqualityExpression`[?In, ?Yield, ?Await]`
BitwiseANDExpression`[?In, ?Yield, ?Await]` **&** *EqualityExpression*`[?In, ?Yield, ?Await]`
BitwiseXORExpression`[In, Yield, Await]` :
BitwiseANDExpression`[?In, ?Yield, ?Await]`
BitwiseXORExpression`[?In, ?Yield, ?Await]` **^** *BitwiseANDExpression*`[?In, ?Yield, ?Await]`
BitwiseORExpression`[In, Yield, Await]` :
BitwiseXORExpression`[?In, ?Yield, ?Await]`
BitwiseORExpression`[?In, ?Yield, ?Await]` **|** *BitwiseXORExpression*`[?In, ?Yield, ?Await]`
LogicalANDExpression`[In, Yield, Await]` :
BitwiseORExpression`[?In, ?Yield, ?Await]`
LogicalANDExpression`[?In, ?Yield, ?Await]` **&&** *BitwiseORExpression*`[?In, ?Yield, ?Await]`
LogicalORExpression`[In, Yield, Await]` :
LogicalANDExpression`[?In, ?Yield, ?Await]`
LogicalORExpression`[?In, ?Yield, ?Await]` **||** *LogicalANDExpression*`[?In, ?Yield, ?Await]`
CoalesceExpression`[In, Yield, Await]` :
CoalesceExpressionHead`[?In, ?Yield, ?Await]` **??** *BitwiseORExpression*`[?In, ?Yield, ?Await]`
CoalesceExpressionHead`[In, Yield, Await]` :
CoalesceExpression`[?In, ?Yield, ?Await]`
BitwiseORExpression`[?In, ?Yield, ?Await]`

ShortCircuitExpression[In, Yield, Await] :
LogicalORExpression[?In, ?Yield, ?Await]
CoalesceExpression[?In, ?Yield, ?Await]
ConditionalExpression[In, Yield, Await] :
ShortCircuitExpression[?In, ?Yield, ?Await]
ShortCircuitExpression[?In, ?Yield, ?Await] **?** *AssignmentExpression*[+In, ?Yield, ?Await] **:** *AssignmentExpression*[?In, ?Yield, ?Await]
AssignmentExpression[In, Yield, Await] :
ConditionalExpression[?In, ?Yield, ?Await]
[+Yield] *YieldExpression*[?In, ?Await]
ArrowFunction[?In, ?Yield, ?Await]
AsyncArrowFunction[?In, ?Yield, ?Await]
LeftHandSideExpression[?Yield, ?Await] **=** *AssignmentExpression*[?In, ?Yield, ?Await]
LeftHandSideExpression[?Yield, ?Await] *AssignmentOperator* *AssignmentExpression*[?In, ?Yield, ?Await]
LeftHandSideExpression[?Yield, ?Await] **&&=** *AssignmentExpression*[?In, ?Yield, ?Await]
LeftHandSideExpression[?Yield, ?Await] **||=** *AssignmentExpression*[?In, ?Yield, ?Await]
LeftHandSideExpression[?Yield, ?Await] **??=** *AssignmentExpression*[?In, ?Yield, ?Await]
AssignmentOperator : **one of**
***= /= %= += -= <<= >>= >>>= &= ^= |= **=**

In certain circumstances when processing an instance of the production *AssignmentExpression*[In, Yield, Await] : *LeftHandSideExpression*[?Yield, ?Await] **=** *AssignmentExpression*[?In, ?Yield, ?Await]
the interpretation of *LeftHandSideExpression* is refined using the following grammar:

AssignmentPattern[Yield, Await] :
ObjectAssignmentPattern[?Yield, ?Await]
ArrayAssignmentPattern[?Yield, ?Await]
ObjectAssignmentPattern[Yield, Await] :
{ }
{ *AssignmentRestProperty*[?Yield, ?Await] **}**
{ *AssignmentPropertyList*[?Yield, ?Await] **}**
{ *AssignmentPropertyList*[?Yield, ?Await] **,** *AssignmentRestProperty*[?Yield, ?Await] opt **}**
ArrayAssignmentPattern[Yield, Await] :
[*Elision*opt *AssignmentRestElement*[?Yield, ?Await] opt **]**
[*AssignmentElementList*[?Yield, ?Await] **]**
[*AssignmentElementList*[?Yield, ?Await] **,** *Elision*opt *AssignmentRestElement*[?Yield, ?Await] opt **]**
AssignmentRestProperty[Yield, Await] :
... *DestructuringAssignmentTarget*[?Yield, ?Await]
AssignmentPropertyList[Yield, Await] :
AssignmentProperty[?Yield, ?Await]
AssignmentPropertyList[?Yield, ?Await] **,** *AssignmentProperty*[?Yield, ?Await]
AssignmentElementList[Yield, Await] :
AssignmentElisionElement[?Yield, ?Await]
AssignmentElementList[?Yield, ?Await] **,** *AssignmentElisionElement*[?Yield, ?Await]
AssignmentElisionElement[Yield, Await] :
*Elision*opt *AssignmentElement*[?Yield, ?Await]
AssignmentProperty[Yield, Await] :
IdentifierReference[?Yield, ?Await] *Initializer*[+In, ?Yield, ?Await] opt
PropertyName[?Yield, ?Await] **:** *AssignmentElement*[?Yield, ?Await]

AssignmentElement[Yield, Await] :
DestructuringAssignmentTarget[?Yield, ?Await] *Initializer*[+In, ?Yield, ?Await] opt
AssignmentRestElement[Yield, Await] :
... *DestructuringAssignmentTarget*[?Yield, ?Await]
DestructuringAssignmentTarget[Yield, Await] :
LeftHandSideExpression[?Yield, ?Await]

Expression[In, Yield, Await] :
AssignmentExpression[?In, ?Yield, ?Await]
Expression[?In, ?Yield, ?Await] , *AssignmentExpression*[?In, ?Yield, ?Await]

A.3 Statements

Statement[Yield, Await, Return] :
BlockStatement[?Yield, ?Await, ?Return]
VariableStatement[?Yield, ?Await]
EmptyStatement
ExpressionStatement[?Yield, ?Await]
IfStatement[?Yield, ?Await, ?Return]
BreakableStatement[?Yield, ?Await, ?Return]
ContinueStatement[?Yield, ?Await]
BreakStatement[?Yield, ?Await]
[+Return] *ReturnStatement*[?Yield, ?Await]
WithStatement[?Yield, ?Await, ?Return]
LabelledStatement[?Yield, ?Await, ?Return]
ThrowStatement[?Yield, ?Await]
TryStatement[?Yield, ?Await, ?Return]
DebuggerStatement
Declaration[Yield, Await] :
HoistableDeclaration[?Yield, ?Await, ~Default]
ClassDeclaration[?Yield, ?Await, ~Default]
LexicalDeclaration[+In, ?Yield, ?Await]
HoistableDeclaration[Yield, Await, Default] :
FunctionDeclaration[?Yield, ?Await, ?Default]
GeneratorDeclaration[?Yield, ?Await, ?Default]
AsyncFunctionDeclaration[?Yield, ?Await, ?Default]
AsyncGeneratorDeclaration[?Yield, ?Await, ?Default]
BreakableStatement[Yield, Await, Return] :
IterationStatement[?Yield, ?Await, ?Return]
SwitchStatement[?Yield, ?Await, ?Return]
BlockStatement[Yield, Await, Return] :
Block[?Yield, ?Await, ?Return]
Block[Yield, Await, Return] :
{ *StatementList*[?Yield, ?Await, ?Return] opt }
StatementList[Yield, Await, Return] :
StatementListItem[?Yield, ?Await, ?Return]
StatementList[?Yield, ?Await, ?Return] *StatementListItem*[?Yield, ?Await, ?Return]
StatementListItem[Yield, Await, Return] :
Statement[?Yield, ?Await, ?Return]
Declaration[?Yield, ?Await]

LexicalDeclaration`[In, Yield, Await]` :
LetOrConst *BindingList*`[?In, ?Yield, ?Await]` **;**
LetOrConst :
let
const
BindingList`[In, Yield, Await]` :
LexicalBinding`[?In, ?Yield, ?Await]`
BindingList`[?In, ?Yield, ?Await]` **,** *LexicalBinding*`[?In, ?Yield, ?Await]`
LexicalBinding`[In, Yield, Await]` :
BindingIdentifier`[?Yield, ?Await]` *Initializer*`[?In, ?Yield, ?Await] opt`
BindingPattern`[?Yield, ?Await]` *Initializer*`[?In, ?Yield, ?Await]`
VariableStatement`[Yield, Await]` :
var *VariableDeclarationList*`[+In, ?Yield, ?Await]` **;**
VariableDeclarationList`[In, Yield, Await]` :
VariableDeclaration`[?In, ?Yield, ?Await]`
VariableDeclarationList`[?In, ?Yield, ?Await]` **,** *VariableDeclaration*`[?In, ?Yield, ?Await]`
VariableDeclaration`[In, Yield, Await]` :
BindingIdentifier`[?Yield, ?Await]` *Initializer*`[?In, ?Yield, ?Await] opt`
BindingPattern`[?Yield, ?Await]` *Initializer*`[?In, ?Yield, ?Await]`
BindingPattern`[Yield, Await]` :
ObjectBindingPattern`[?Yield, ?Await]`
ArrayBindingPattern`[?Yield, ?Await]`
ObjectBindingPattern`[Yield, Await]` :
{ }
{ *BindingRestProperty*`[?Yield, ?Await]` **}**
{ *BindingPropertyList*`[?Yield, ?Await]` **}**
{ *BindingPropertyList*`[?Yield, ?Await]` **,** *BindingRestProperty*`[?Yield, ?Await] opt` **}**
ArrayBindingPattern`[Yield, Await]` :
[*Elision*`opt` *BindingRestElement*`[?Yield, ?Await] opt` **]**
[*BindingElementList*`[?Yield, ?Await]` **]**
[*BindingElementList*`[?Yield, ?Await]` **,** *Elision*`opt` *BindingRestElement*`[?Yield, ?Await] opt`
]
BindingRestProperty`[Yield, Await]` :
... *BindingIdentifier*`[?Yield, ?Await]`
BindingPropertyList`[Yield, Await]` :
BindingProperty`[?Yield, ?Await]`
BindingPropertyList`[?Yield, ?Await]` **,** *BindingProperty*`[?Yield, ?Await]`
BindingElementList`[Yield, Await]` :
BindingElisionElement`[?Yield, ?Await]`
BindingElementList`[?Yield, ?Await]` **,** *BindingElisionElement*`[?Yield, ?Await]`
BindingElisionElement`[Yield, Await]` :
Elision`opt` *BindingElement*`[?Yield, ?Await]`
BindingProperty`[Yield, Await]` :
SingleNameBinding`[?Yield, ?Await]`
PropertyName`[?Yield, ?Await]` **:** *BindingElement*`[?Yield, ?Await]`
BindingElement`[Yield, Await]` :
SingleNameBinding`[?Yield, ?Await]`
BindingPattern`[?Yield, ?Await]` *Initializer*`[+In, ?Yield, ?Await] opt`
SingleNameBinding`[Yield, Await]` :
BindingIdentifier`[?Yield, ?Await]` *Initializer*`[+In, ?Yield, ?Await] opt`

BindingRestElement`[Yield, Await]` :
... *BindingIdentifier*`[?Yield, ?Await]`
... *BindingPattern*`[?Yield, ?Await]`

EmptyStatement :
;

ExpressionStatement`[Yield, Await]` :
[lookahead ∉ { **{** , **function** , **async** [no *LineTerminator* here] **function** , **class** , **let** **[** }] *Expression*`[+In, ?Yield, ?Await]` **;**

IfStatement`[Yield, Await, Return]` :
if **(** *Expression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]` **else** *Statement*`[?Yield, ?Await, ?Return]`
if **(** *Expression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]` [lookahead ≠ **else**]

IterationStatement`[Yield, Await, Return]` :
DoWhileStatement`[?Yield, ?Await, ?Return]`
WhileStatement`[?Yield, ?Await, ?Return]`
ForStatement`[?Yield, ?Await, ?Return]`
ForInOfStatement`[?Yield, ?Await, ?Return]`

DoWhileStatement`[Yield, Await, Return]` :
do *Statement*`[?Yield, ?Await, ?Return]` **while** **(** *Expression*`[+In, ?Yield, ?Await]` **)** **;**

WhileStatement`[Yield, Await, Return]` :
while **(** *Expression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`

ForStatement`[Yield, Await, Return]` :
for **(** [lookahead ≠ **let** **[**] *Expression*`[~In, ?Yield, ?Await] opt` **;** *Expression*`[+In, ?Yield, ?Await] opt` **;** *Expression*`[+In, ?Yield, ?Await] opt` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** **var** *VariableDeclarationList*`[~In, ?Yield, ?Await]` **;** *Expression*`[+In, ?Yield, ?Await] opt` **;** *Expression*`[+In, ?Yield, ?Await] opt` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** *LexicalDeclaration*`[~In, ?Yield, ?Await]` *Expression*`[+In, ?Yield, ?Await] opt` **;** *Expression*`[+In, ?Yield, ?Await] opt` **)** *Statement*`[?Yield, ?Await, ?Return]`

ForInOfStatement`[Yield, Await, Return]` :
for **(** [lookahead ≠ **let** **[**] *LeftHandSideExpression*`[?Yield, ?Await]` **in** *Expression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** **var** *ForBinding*`[?Yield, ?Await]` **in** *Expression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** *ForDeclaration*`[?Yield, ?Await]` **in** *Expression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** [lookahead ∉ { **let** , **async** **of** }] *LeftHandSideExpression*`[?Yield, ?Await]` **of** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** **var** *ForBinding*`[?Yield, ?Await]` **of** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
for **(** *ForDeclaration*`[?Yield, ?Await]` **of** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
`[+Await]` **for await** **(** [lookahead ≠ **let**] *LeftHandSideExpression*`[?Yield, ?Await]` **of** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
`[+Await]` **for await** **(** **var** *ForBinding*`[?Yield, ?Await]` **of** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`
`[+Await]` **for await** **(** *ForDeclaration*`[?Yield, ?Await]` **of** *AssignmentExpression*`[+In, ?Yield, ?Await]` **)** *Statement*`[?Yield, ?Await, ?Return]`

ForDeclaration`[Yield, Await]` :
LetOrConst *ForBinding*`[?Yield, ?Await]`

ForBinding[Yield, Await] :
BindingIdentifier[?Yield, ?Await]
BindingPattern[?Yield, ?Await]

ContinueStatement[Yield, Await] :
continue ;
continue [no *LineTerminator* here] *LabelIdentifier*[?Yield, ?Await] ;

BreakStatement[Yield, Await] :
break ;
break [no *LineTerminator* here] *LabelIdentifier*[?Yield, ?Await] ;

ReturnStatement[Yield, Await] :
return ;
return [no *LineTerminator* here] *Expression*[+In, ?Yield, ?Await] ;

WithStatement[Yield, Await, Return] :
with (*Expression*[+In, ?Yield, ?Await]) *Statement*[?Yield, ?Await, ?Return]

SwitchStatement[Yield, Await, Return] :
switch (*Expression*[+In, ?Yield, ?Await]) *CaseBlock*[?Yield, ?Await, ?Return]

CaseBlock[Yield, Await, Return] :
{ *CaseClauses*[?Yield, ?Await, ?Return] opt **}**
{ *CaseClauses*[?Yield, ?Await, ?Return] opt *DefaultClause*[?Yield, ?Await, ?Return] *CaseClauses*[?Yield, ?Await, ?Return] opt **}**

CaseClauses[Yield, Await, Return] :
CaseClause[?Yield, ?Await, ?Return]
CaseClauses[?Yield, ?Await, ?Return] *CaseClause*[?Yield, ?Await, ?Return]

CaseClause[Yield, Await, Return] :
case *Expression*[+In, ?Yield, ?Await] **:** *StatementList*[?Yield, ?Await, ?Return] opt

DefaultClause[Yield, Await, Return] :
default : *StatementList*[?Yield, ?Await, ?Return] opt

LabelledStatement[Yield, Await, Return] :
LabelIdentifier[?Yield, ?Await] **:** *LabelledItem*[?Yield, ?Await, ?Return]

LabelledItem[Yield, Await, Return] :
Statement[?Yield, ?Await, ?Return]
FunctionDeclaration[?Yield, ?Await, ~Default]

ThrowStatement[Yield, Await] :
throw [no *LineTerminator* here] *Expression*[+In, ?Yield, ?Await] ;

TryStatement[Yield, Await, Return] :
try *Block*[?Yield, ?Await, ?Return] *Catch*[?Yield, ?Await, ?Return]
try *Block*[?Yield, ?Await, ?Return] *Finally*[?Yield, ?Await, ?Return]
try *Block*[?Yield, ?Await, ?Return] *Catch*[?Yield, ?Await, ?Return] *Finally*[?Yield, ?Await, ?Return]

Catch[Yield, Await, Return] :
catch (*CatchParameter*[?Yield, ?Await]) *Block*[?Yield, ?Await, ?Return]
catch *Block*[?Yield, ?Await, ?Return]

Finally[Yield, Await, Return] :
finally *Block*[?Yield, ?Await, ?Return]

CatchParameter[Yield, Await] :
BindingIdentifier[?Yield, ?Await]
BindingPattern[?Yield, ?Await]

DebuggerStatement :
debugger ;

A.4 Functions and Classes

UniqueFormalParameters`[Yield, Await]` :
FormalParameters`[?Yield, ?Await]`

FormalParameters`[Yield, Await]` :
[empty]
FunctionRestParameter`[?Yield, ?Await]`
FormalParameterList`[?Yield, ?Await]`
FormalParameterList`[?Yield, ?Await]` **,**
FormalParameterList`[?Yield, ?Await]` **,** *FunctionRestParameter*`[?Yield, ?Await]`

FormalParameterList`[Yield, Await]` :
FormalParameter`[?Yield, ?Await]`
FormalParameterList`[?Yield, ?Await]` **,** *FormalParameter*`[?Yield, ?Await]`

FunctionRestParameter`[Yield, Await]` :
BindingRestElement`[?Yield, ?Await]`

FormalParameter`[Yield, Await]` :
BindingElement`[?Yield, ?Await]`

FunctionDeclaration`[Yield, Await, Default]` :
function *BindingIdentifier*`[?Yield, ?Await]` **(** *FormalParameters*`[~Yield, ~Await]` **) {**
FunctionBody`[~Yield, ~Await]` **}**
`[+Default]` **function (** *FormalParameters*`[~Yield, ~Await]` **) {** *FunctionBody*`[~Yield, ~Await]`
}

FunctionExpression :
function *BindingIdentifier*`[~Yield, ~Await] opt` **(** *FormalParameters*`[~Yield, ~Await]` **) {**
FunctionBody`[~Yield, ~Await]` **}**

FunctionBody`[Yield, Await]` :
FunctionStatementList`[?Yield, ?Await]`

FunctionStatementList`[Yield, Await]` :
StatementList`[?Yield, ?Await, +Return] opt`

ArrowFunction`[In, Yield, Await]` :
ArrowParameters`[?Yield, ?Await]` [no *LineTerminator* here] **=>** *ConciseBody*`[?In]`

ArrowParameters`[Yield, Await]` :
BindingIdentifier`[?Yield, ?Await]`
CoverParenthesizedExpressionAndArrowParameterList`[?Yield, ?Await]`

ConciseBody`[In]` :
[lookahead ≠ **{**] *ExpressionBody*`[?In, ~Await]`
{ *FunctionBody*`[~Yield, ~Await]` **}**

ExpressionBody`[In, Await]` :
AssignmentExpression`[?In, ~Yield, ?Await]`

When processing an instance of the production
ArrowParameters`[Yield, Await]` : *CoverParenthesizedExpressionAndArrowParameterList*`[?Yield, ?Await]`
the interpretation of *CoverParenthesizedExpressionAndArrowParameterList* is refined using the following grammar:

ArrowFormalParameters`[Yield, Await]` :
(*UniqueFormalParameters*`[?Yield, ?Await]` **)**

AsyncArrowFunction`[In, Yield, Await]` :
async [no *LineTerminator* here] *AsyncArrowBindingIdentifier*`[?Yield]` [no *LineTerminator* here] **=>** *AsyncConciseBody*`[?In]`
CoverCallExpressionAndAsyncArrowHead`[?Yield, ?Await]` [no *LineTerminator* here] **=>** *AsyncConciseBody*`[?In]`
AsyncConciseBody`[In]` :
[lookahead ≠ **{**] *ExpressionBody*`[?In, +Await]`
{ *AsyncFunctionBody* **}**
AsyncArrowBindingIdentifier`[Yield]` :
BindingIdentifier`[?Yield, +Await]`
CoverCallExpressionAndAsyncArrowHead`[Yield, Await]` :
MemberExpression`[?Yield, ?Await]` *Arguments*`[?Yield, ?Await]`

When processing an instance of the production *AsyncArrowFunction*`[In, Yield, Await]` : *CoverCallExpressionAndAsyncArrowHead*`[?Yield, ?Await]` [no *LineTerminator* here] **=>** *AsyncConciseBody*`[?In]` the interpretation of *CoverCallExpressionAndAsyncArrowHead* is refined using the following grammar:

AsyncArrowHead :
async [no *LineTerminator* here] *ArrowFormalParameters*`[~Yield, +Await]`

MethodDefinition`[Yield, Await]` :
ClassElementName`[?Yield, ?Await]` **(** *UniqueFormalParameters*`[~Yield, ~Await]` **) {** *FunctionBody*`[~Yield, ~Await]` **}**
GeneratorMethod`[?Yield, ?Await]`
AsyncMethod`[?Yield, ?Await]`
AsyncGeneratorMethod`[?Yield, ?Await]`
get *ClassElementName*`[?Yield, ?Await]` **( ) {** *FunctionBody*`[~Yield, ~Await]` **}**
set *ClassElementName*`[?Yield, ?Await]` **(** *PropertySetParameterList* **) {** *FunctionBody*`[~Yield, ~Await]` **}**
PropertySetParameterList :
FormalParameter`[~Yield, ~Await]`
GeneratorDeclaration`[Yield, Await, Default]` :
function ***** *BindingIdentifier*`[?Yield, ?Await]` **(** *FormalParameters*`[+Yield, ~Await]` **) {** *GeneratorBody* **}**
[+Default] **function** ***** **(** *FormalParameters*`[+Yield, ~Await]` **) {** *GeneratorBody* **}**
GeneratorExpression :
function ***** *BindingIdentifier*`[+Yield, ~Await]` `opt` **(** *FormalParameters*`[+Yield, ~Await]` **) {** *GeneratorBody* **}**
GeneratorMethod`[Yield, Await]` :
***** *ClassElementName*`[?Yield, ?Await]` **(** *UniqueFormalParameters*`[+Yield, ~Await]` **) {** *GeneratorBody* **}**
GeneratorBody :
FunctionBody`[+Yield, ~Await]`
YieldExpression`[In, Await]` :
yield
yield [no *LineTerminator* here] *AssignmentExpression*`[?In, +Yield, ?Await]`
yield [no *LineTerminator* here] ***** *AssignmentExpression*`[?In, +Yield, ?Await]`

AsyncGeneratorDeclaration[Yield, Await, Default] :
async [no *LineTerminator* here] **function** * *BindingIdentifier*[?Yield, ?Await] (*FormalParameters*[+Yield, +Await]) { *AsyncGeneratorBody* }
[+Default] **async** [no *LineTerminator* here] **function** * (*FormalParameters*[+Yield, +Await]) { *AsyncGeneratorBody* }

AsyncGeneratorExpression :
async [no *LineTerminator* here] **function** * *BindingIdentifier*[+Yield, +Await] opt (*FormalParameters*[+Yield, +Await]) { *AsyncGeneratorBody* }

AsyncGeneratorMethod[Yield, Await] :
async [no *LineTerminator* here] * *ClassElementName*[?Yield, ?Await] (*UniqueFormalParameters*[+Yield, +Await]) { *AsyncGeneratorBody* }

AsyncGeneratorBody :
FunctionBody[+Yield, +Await]

AsyncFunctionDeclaration[Yield, Await, Default] :
async [no *LineTerminator* here] **function** *BindingIdentifier*[?Yield, ?Await] (*FormalParameters*[~Yield, +Await]) { *AsyncFunctionBody* }
[+Default] **async** [no *LineTerminator* here] **function** (*FormalParameters*[~Yield, +Await]) { *AsyncFunctionBody* }

AsyncFunctionExpression :
async [no *LineTerminator* here] **function** *BindingIdentifier*[~Yield, +Await] opt (*FormalParameters*[~Yield, +Await]) { *AsyncFunctionBody* }

AsyncMethod[Yield, Await] :
async [no *LineTerminator* here] *ClassElementName*[?Yield, ?Await] (*UniqueFormalParameters*[~Yield, +Await]) { *AsyncFunctionBody* }

AsyncFunctionBody :
FunctionBody[~Yield, +Await]

AwaitExpression[Yield] :
await *UnaryExpression*[?Yield, +Await]

ClassDeclaration[Yield, Await, Default] :
class *BindingIdentifier*[?Yield, ?Await] *ClassTail*[?Yield, ?Await]
[+Default] **class** *ClassTail*[?Yield, ?Await]

ClassExpression[Yield, Await] :
class *BindingIdentifier*[?Yield, ?Await] opt *ClassTail*[?Yield, ?Await]

ClassTail[Yield, Await] :
ClassHeritage[?Yield, ?Await] opt { *ClassBody*[?Yield, ?Await] opt }

ClassHeritage[Yield, Await] :
extends *LeftHandSideExpression*[?Yield, ?Await]

ClassBody[Yield, Await] :
ClassElementList[?Yield, ?Await]

ClassElementList[Yield, Await] :
ClassElement[?Yield, ?Await]
ClassElementList[?Yield, ?Await] *ClassElement*[?Yield, ?Await]

ClassElement[Yield, Await] :
MethodDefinition[?Yield, ?Await]
static *MethodDefinition*[?Yield, ?Await]
FieldDefinition[?Yield, ?Await] ;
static *FieldDefinition*[?Yield, ?Await] ;
ClassStaticBlock
;

FieldDefinition[Yield, Await] :
ClassElementName[?Yield, ?Await] *Initializer*[+In, ?Yield, ?Await] opt

ClassElementName[Yield, Await] :
PropertyName[?Yield, ?Await]
PrivateIdentifier
ClassStaticBlock :
static { *ClassStaticBlockBody* }
ClassStaticBlockBody :
ClassStaticBlockStatementList
ClassStaticBlockStatementList :
StatementList[~Yield, +Await, ~Return] opt

A.5 Scripts and Modules

Script :
*ScriptBody*opt
ScriptBody :
StatementList[~Yield, ~Await, ~Return]
Module :
*ModuleBody*opt
ModuleBody :
ModuleItemList
ModuleItemList :
ModuleItem
ModuleItemList *ModuleItem*
ModuleItem :
ImportDeclaration
ExportDeclaration
StatementListItem[~Yield, +Await, ~Return]
ModuleExportName :
IdentifierName
StringLiteral
ImportDeclaration :
import *ImportClause* *FromClause* **;**
import *ModuleSpecifier* **;**
ImportClause :
ImportedDefaultBinding
NameSpaceImport
NamedImports
ImportedDefaultBinding **,** *NameSpaceImport*
ImportedDefaultBinding **,** *NamedImports*
ImportedDefaultBinding :
ImportedBinding
NameSpaceImport :
*** as** *ImportedBinding*
NamedImports :
{ }
{ *ImportsList* **}**
{ *ImportsList* **,** **}**
FromClause :
from *ModuleSpecifier*
ImportsList :
ImportSpecifier
ImportsList **,** *ImportSpecifier*
ImportSpecifier :
ImportedBinding
ModuleExportName **as** *ImportedBinding*

ModuleSpecifier :
StringLiteral
ImportedBinding :
BindingIdentifier[~Yield, +Await]
ExportDeclaration :
export *ExportFromClause* *FromClause* **;**
export *NamedExports* **;**
export *VariableStatement*[~Yield, +Await]
export *Declaration*[~Yield, +Await]
export default *HoistableDeclaration*[~Yield, +Await, +Default]
export default *ClassDeclaration*[~Yield, +Await, +Default]
export default [lookahead ∉ { **function** , **async** [no *LineTerminator* here] **function** , **class** }] *AssignmentExpression*[+In, ~Yield, +Await] **;**
ExportFromClause :
*****
*** as** *ModuleExportName*
NamedExports
NamedExports :
{ }
{ *ExportsList* **}**
{ *ExportsList* **,** **}**
ExportsList :
ExportSpecifier
ExportsList **,** *ExportSpecifier*
ExportSpecifier :
ModuleExportName
ModuleExportName **as** *ModuleExportName*

A.6 Number Conversions

StringNumericLiteral ::::
*StrWhiteSpace*opt
*StrWhiteSpace*opt *StrNumericLiteral* *StrWhiteSpace*opt
StrWhiteSpace ::::
StrWhiteSpaceChar *StrWhiteSpace*opt
StrWhiteSpaceChar ::::
WhiteSpace
LineTerminator
StrNumericLiteral ::::
StrDecimalLiteral
NonDecimalIntegerLiteral[~Sep]
StrDecimalLiteral ::::
StrUnsignedDecimalLiteral
+ *StrUnsignedDecimalLiteral*
- *StrUnsignedDecimalLiteral*
StrUnsignedDecimalLiteral ::::
Infinity
DecimalDigits[~Sep] **.** *DecimalDigits*[~Sep] opt *ExponentPart*[~Sep] opt
. *DecimalDigits*[~Sep] *ExponentPart*[~Sep] opt
DecimalDigits[~Sep] *ExponentPart*[~Sep] opt

All grammar symbols not explicitly defined by the *StringNumericLiteral* grammar have the definitions used in the Lexical Grammar for numeric literals.

StringIntegerLiteral ::: *StrWhiteSpace*[opt]
StrWhiteSpace[opt] *StrIntegerLiteral* *StrWhiteSpace*[opt]
StrIntegerLiteral :::
SignedInteger[~Sep]
NonDecimalIntegerLiteral[~Sep]

A.7 Time Zone Offset String Format

UTCOffset :::
TemporalSign *Hour*
TemporalSign *Hour* *HourSubcomponents*[+Extended]
TemporalSign *Hour* *HourSubcomponents*[~Extended]
TemporalSign :::
ASCIISign
<MINUS>
ASCIISign ::: **one of**
+ **-**
Hour :::
0 *DecimalDigit*
1 *DecimalDigit*
20
21
22
23
HourSubcomponents[Extended] :::
TimeSeparator[?Extended] *MinuteSecond*
TimeSeparator[?Extended] *MinuteSecond* *TimeSeparator*[?Extended] *MinuteSecond* *TemporalDecimalFraction*[opt]
TimeSeparator[Extended] :::
[+Extended] **:**
[~Extended] [empty]
MinuteSecond :::
0 *DecimalDigit*
1 *DecimalDigit*
2 *DecimalDigit*
3 *DecimalDigit*
4 *DecimalDigit*
5 *DecimalDigit*
TemporalDecimalFraction :::
TemporalDecimalSeparator *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit*
TemporalDecimalSeparator *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit* *DecimalDigit*

TemporalDecimalSeparator ::: **one of**
`.` `,`

A.8 Regular Expressions

Pattern`[UnicodeMode, UnicodeSetsMode, NamedCaptureGroups]` ::
Disjunction`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`

Disjunction`[UnicodeMode, UnicodeSetsMode, NamedCaptureGroups]` ::
Alternative`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`
Alternative`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `|`
Disjunction`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`

Alternative`[UnicodeMode, UnicodeSetsMode, NamedCaptureGroups]` ::
[empty]
Alternative`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`
Term`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`

Term`[UnicodeMode, UnicodeSetsMode, NamedCaptureGroups]` ::
Assertion`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`
Atom`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]`
Atom`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` *Quantifier*

Assertion`[UnicodeMode, UnicodeSetsMode, NamedCaptureGroups]` ::
`^`
`$`
`\b`
`\B`
`(?=` *Disjunction*`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `)`
`(?!` *Disjunction*`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `)`
`(?<=` *Disjunction*`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `)`
`(?<!` *Disjunction*`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `)`

Quantifier ::
QuantifierPrefix
QuantifierPrefix `?`

QuantifierPrefix ::
`*`
`+`
`?`
`{` *DecimalDigits*`[~Sep]` `}`
`{` *DecimalDigits*`[~Sep]` `,}`
`{` *DecimalDigits*`[~Sep]` `,` *DecimalDigits*`[~Sep]` `}`

Atom`[UnicodeMode, UnicodeSetsMode, NamedCaptureGroups]` ::
PatternCharacter
`.`
`\` *AtomEscape*`[?UnicodeMode, ?NamedCaptureGroups]`
CharacterClass`[?UnicodeMode, ?UnicodeSetsMode]`
`(` *GroupSpecifier*`[?UnicodeMode]` `opt`
Disjunction`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `)`
`(?:` *Disjunction*`[?UnicodeMode, ?UnicodeSetsMode, ?NamedCaptureGroups]` `)`

SyntaxCharacter :: **one of**
`^ $ \ . * + ? ( ) [ ] { } |`

PatternCharacter ::
SourceCharacter but not *SyntaxCharacter*

AtomEscape[UnicodeMode, NamedCaptureGroups] ::
DecimalEscape
CharacterClassEscape[?UnicodeMode]
CharacterEscape[?UnicodeMode]
[+NamedCaptureGroups] **k** *GroupName*[?UnicodeMode]

CharacterEscape[UnicodeMode] ::
ControlEscape
c *AsciiLetter*
0 [lookahead ∉ *DecimalDigit*]
HexEscapeSequence
RegExpUnicodeEscapeSequence[?UnicodeMode]
IdentityEscape[?UnicodeMode]

ControlEscape :: **one of**
f n r t v

GroupSpecifier[UnicodeMode] ::
? *GroupName*[?UnicodeMode]

GroupName[UnicodeMode] ::
< *RegExpIdentifierName*[?UnicodeMode] **>**

RegExpIdentifierName[UnicodeMode] ::
RegExpIdentifierStart[?UnicodeMode]
RegExpIdentifierName[?UnicodeMode] *RegExpIdentifierPart*[?UnicodeMode]

RegExpIdentifierStart[UnicodeMode] ::
IdentifierStartChar
**** *RegExpUnicodeEscapeSequence*[+UnicodeMode]
[~UnicodeMode] *UnicodeLeadSurrogate* *UnicodeTrailSurrogate*

RegExpIdentifierPart[UnicodeMode] ::
IdentifierPartChar
**** *RegExpUnicodeEscapeSequence*[+UnicodeMode]
[~UnicodeMode] *UnicodeLeadSurrogate* *UnicodeTrailSurrogate*

RegExpUnicodeEscapeSequence[UnicodeMode] ::
[+UnicodeMode] **u** *HexLeadSurrogate* **\u** *HexTrailSurrogate*
[+UnicodeMode] **u** *HexLeadSurrogate*
[+UnicodeMode] **u** *HexTrailSurrogate*
[+UnicodeMode] **u** *HexNonSurrogate*
[~UnicodeMode] **u** *Hex4Digits*
[+UnicodeMode] **u{** *CodePoint* **}**

UnicodeLeadSurrogate ::
any Unicode code point in the inclusive interval from U+D800 to U+DBFF

UnicodeTrailSurrogate ::
any Unicode code point in the inclusive interval from U+DC00 to U+DFFF

Each **\u** *HexTrailSurrogate* for which the choice of associated **u** *HexLeadSurrogate* is ambiguous shall be associated with the nearest possible **u** *HexLeadSurrogate* that would otherwise have no corresponding **\u** *HexTrailSurrogate*.

HexLeadSurrogate ::
Hex4Digits but only if the MV of *Hex4Digits* is in the inclusive interval from 0xD800 to 0xDBFF

HexTrailSurrogate ::
Hex4Digits but only if the MV of *Hex4Digits* is in the inclusive interval from 0xDC00 to 0xDFFF

HexNonSurrogate ::
Hex4Digits but only if the MV of *Hex4Digits* is not in the inclusive interval from 0xD800 to 0xDFFF

IdentityEscape[UnicodeMode] ::
[+UnicodeMode] *SyntaxCharacter*
[+UnicodeMode] **/**
[~UnicodeMode] *SourceCharacter* but not *UnicodeIDContinue*

DecimalEscape ::
NonZeroDigit *DecimalDigits*[~Sep] opt [lookahead ∉ *DecimalDigit*]

CharacterClassEscape[UnicodeMode] ::
d
D
s
S
w
W
[+UnicodeMode] **p{** *UnicodePropertyValueExpression* **}**
[+UnicodeMode] **P{** *UnicodePropertyValueExpression* **}**

UnicodePropertyValueExpression ::
UnicodePropertyName **=** *UnicodePropertyValue*
LoneUnicodePropertyNameOrValue

UnicodePropertyName ::
UnicodePropertyNameCharacters

UnicodePropertyNameCharacters ::
UnicodePropertyNameCharacter *UnicodePropertyNameCharacters*opt

UnicodePropertyValue ::
UnicodePropertyValueCharacters

LoneUnicodePropertyNameOrValue ::
UnicodePropertyValueCharacters

UnicodePropertyValueCharacters ::
UnicodePropertyValueCharacter *UnicodePropertyValueCharacters*opt

UnicodePropertyValueCharacter ::
UnicodePropertyNameCharacter
DecimalDigit

UnicodePropertyNameCharacter ::
AsciiLetter
_

CharacterClass[UnicodeMode, UnicodeSetsMode] ::
[[lookahead ≠ **^**] *ClassContents*[?UnicodeMode, ?UnicodeSetsMode] **]**
[^ *ClassContents*[?UnicodeMode, ?UnicodeSetsMode] **]**

ClassContents[UnicodeMode, UnicodeSetsMode] ::
[empty]
[~UnicodeSetsMode] *NonemptyClassRanges*[?UnicodeMode]
[+UnicodeSetsMode] *ClassSetExpression*

NonemptyClassRanges[UnicodeMode] ::
ClassAtom[?UnicodeMode]
ClassAtom[?UnicodeMode] *NonemptyClassRangesNoDash*[?UnicodeMode]
ClassAtom[?UnicodeMode] **-** *ClassAtom*[?UnicodeMode] *ClassContents*[?UnicodeMode, ~UnicodeSetsMode]

NonemptyClassRangesNoDash[UnicodeMode] ::
ClassAtom[?UnicodeMode]
ClassAtomNoDash[?UnicodeMode] *NonemptyClassRangesNoDash*[?UnicodeMode]
ClassAtomNoDash[?UnicodeMode] **-** *ClassAtom*[?UnicodeMode] *ClassContents*[?UnicodeMode, ~UnicodeSetsMode]

ClassAtom[UnicodeMode] ::
- **-**
- *ClassAtomNoDash*[?UnicodeMode]

ClassAtomNoDash[UnicodeMode] ::
- *SourceCharacter* but not one of **** or **]** or **-**
- **** *ClassEscape*[?UnicodeMode]

ClassEscape[UnicodeMode] ::
- **b**
- [+UnicodeMode] **-**
- *CharacterClassEscape*[?UnicodeMode]
- *CharacterEscape*[?UnicodeMode]

ClassSetExpression ::
- *ClassUnion*
- *ClassIntersection*
- *ClassSubtraction*

ClassUnion ::
- *ClassSetRange* *ClassUnion*$_{opt}$
- *ClassSetOperand* *ClassUnion*$_{opt}$

ClassIntersection ::
- *ClassSetOperand* **&&** [lookahead ≠ **&**] *ClassSetOperand*
- *ClassIntersection* **&&** [lookahead ≠ **&**] *ClassSetOperand*

ClassSubtraction ::
- *ClassSetOperand* **--** *ClassSetOperand*
- *ClassSubtraction* **--** *ClassSetOperand*

ClassSetRange ::
- *ClassSetCharacter* **-** *ClassSetCharacter*

ClassSetOperand ::
- *NestedClass*
- *ClassStringDisjunction*
- *ClassSetCharacter*

NestedClass ::
- **[** [lookahead ≠ **^**] *ClassContents*[+UnicodeMode, +UnicodeSetsMode] **]**
- **[^** *ClassContents*[+UnicodeMode, +UnicodeSetsMode] **]**
- **** *CharacterClassEscape*[+UnicodeMode]

ClassStringDisjunction ::
- **\q{** *ClassStringDisjunctionContents* **}**

ClassStringDisjunctionContents ::
- *ClassString*
- *ClassString* **|** *ClassStringDisjunctionContents*

ClassString ::
- [empty]
- *NonEmptyClassString*

NonEmptyClassString ::
- *ClassSetCharacter* *NonEmptyClassString*$_{opt}$

ClassSetCharacter ::
- [lookahead ∉ *ClassSetReservedDoublePunctuator*] *SourceCharacter* but not *ClassSetSyntaxCharacter*
- **** *CharacterEscape*[+UnicodeMode]
- **** *ClassSetReservedPunctuator*
- **\b**

ClassSetReservedDoublePunctuator :: **one of**
- **&& !! ## $$ %% ** ++ ,, .. :: ;; << == >> ?? @@ ^^ `` ~~**

ClassSetSyntaxCharacter :: **one of**
`( ) [ ] { } / - \ |`
ClassSetReservedPunctuator :: **one of**
`& - ! # % , : ; < = > @ ` ~`