How ~~not~~ to become a dragon

C++ Senior Developer
Tech Lead
PVS-Studio

Lisiy Oleg

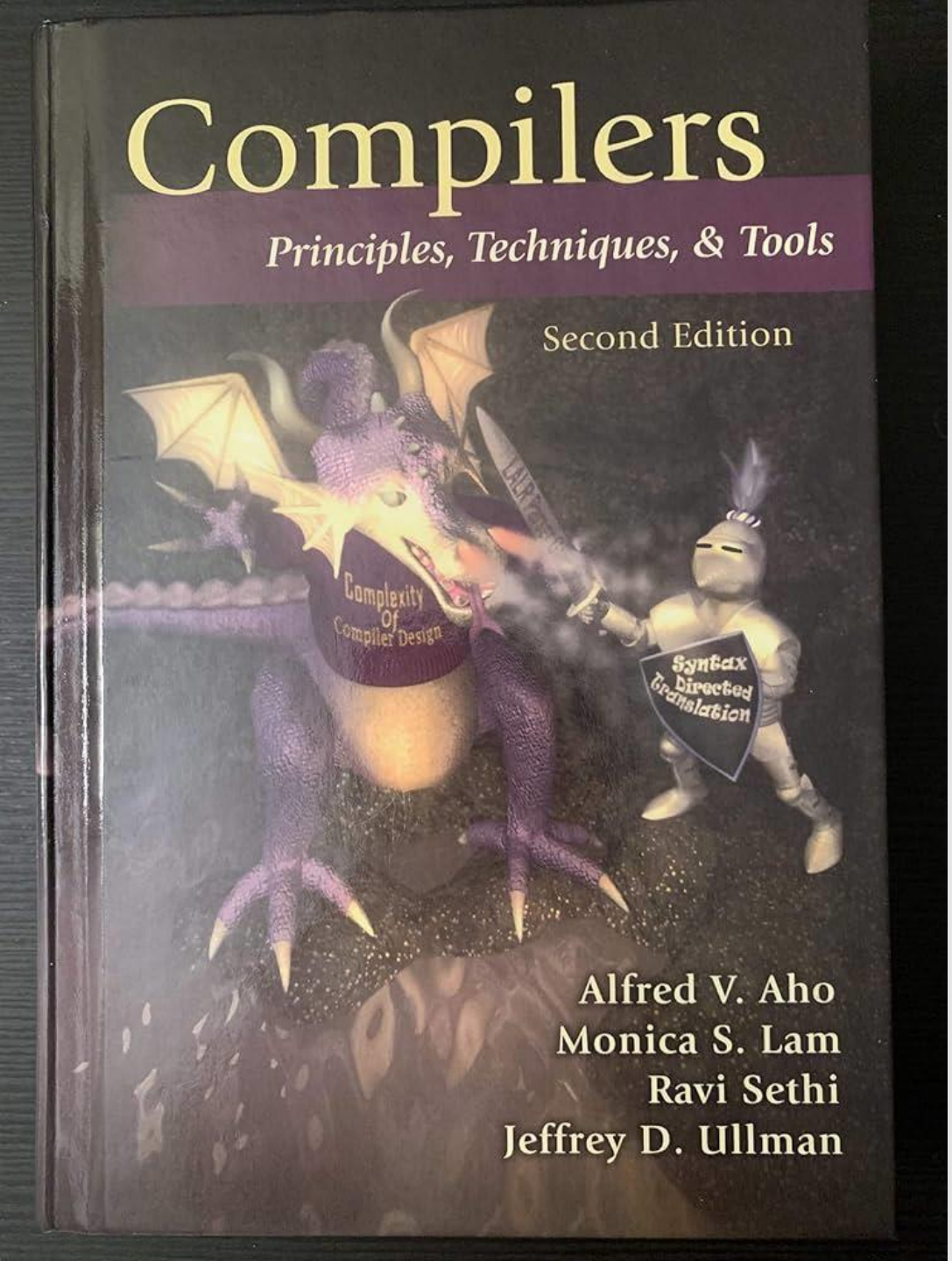
Yuri Minaev

C++ Senior Developer
PVS-Studio

He who defeateth the Dragon
Becometh the Dragon

The beginning

```
for(int j=0; j<n; j++)
{
    if(n!=2)
    {
        if(i<r)
        {
            if(j<t)
            {
                if(j>=k)
                {
                    if(j==k)
                        if(1==1)
                        {
                            {
                                a[i,j] = k + 1;
                            }
                        }
                        else{ a[i,j] = k; }
                    else{ a[i,j] = a[i,k]; }
                }
                else{ a[i,j] = a[i-1, j]; }
            }
        }
        else
        {
            if(a[i, j-1] - 1 > 0)
```



Arithmetic expressions

```
minutes = left + seconds * 60
```

Lexer

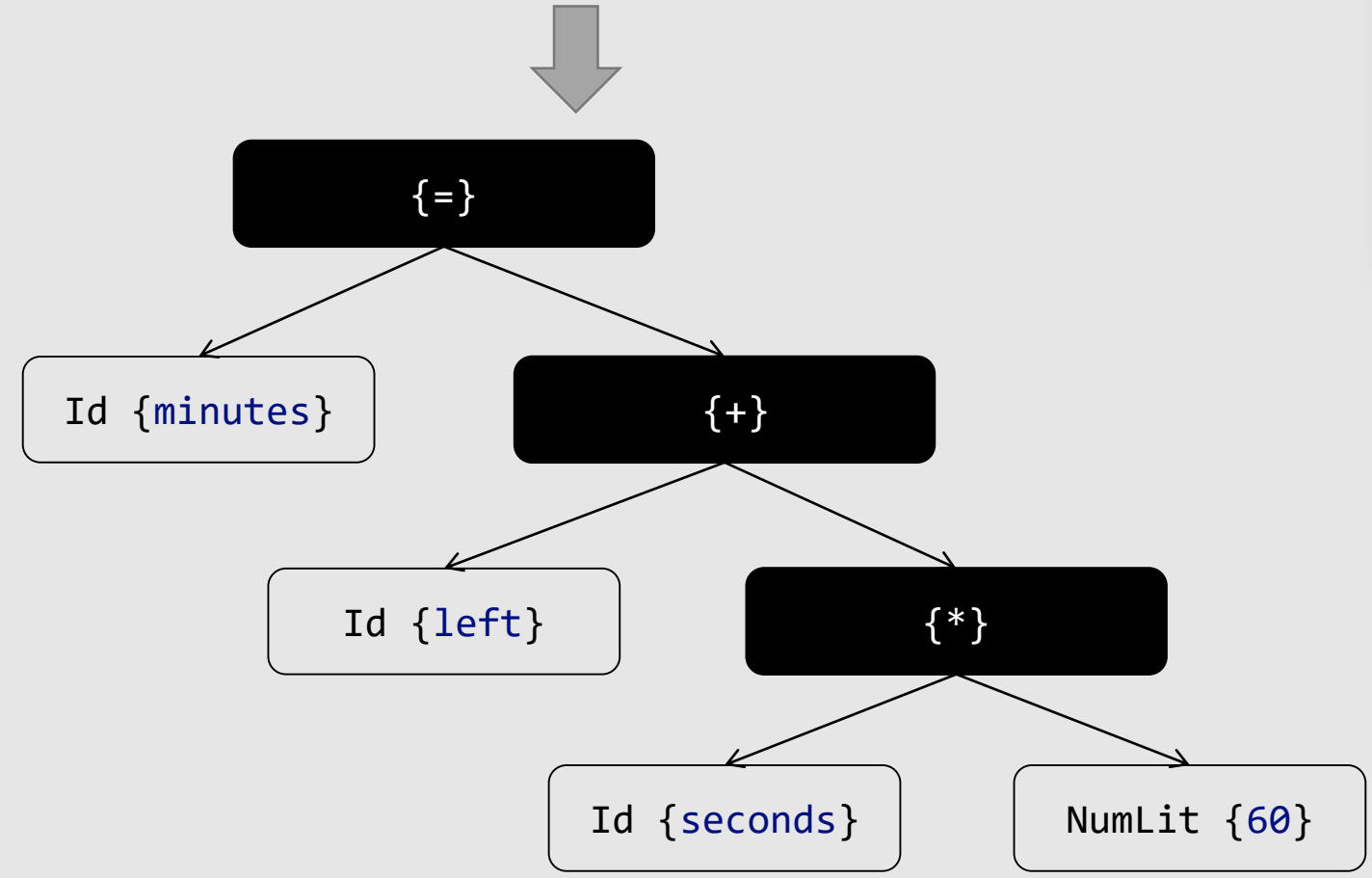
```
<id> <=> <id> <+> <id> <*> <60>
```

Parser

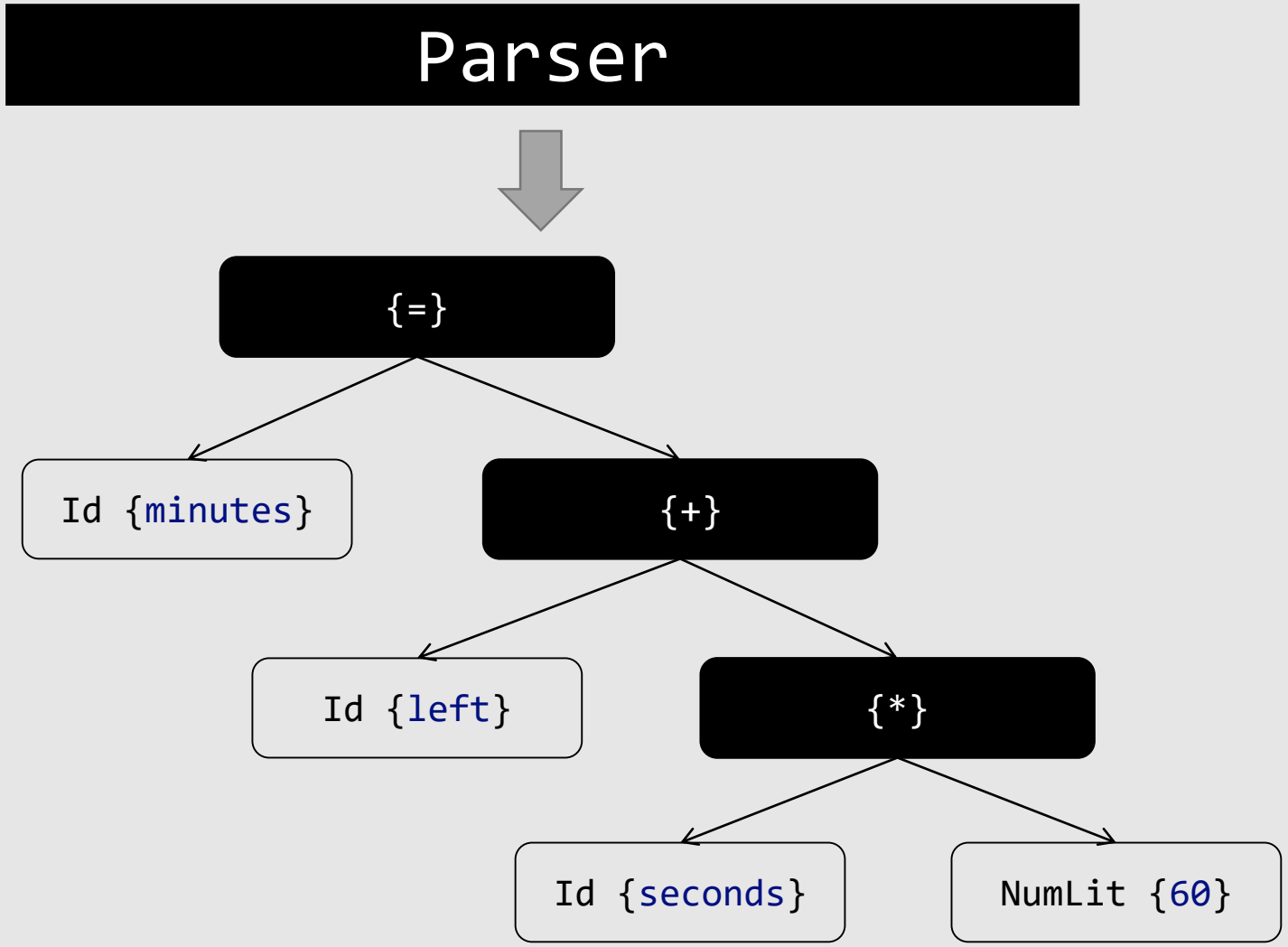
```
minutes = left + seconds * 60
```

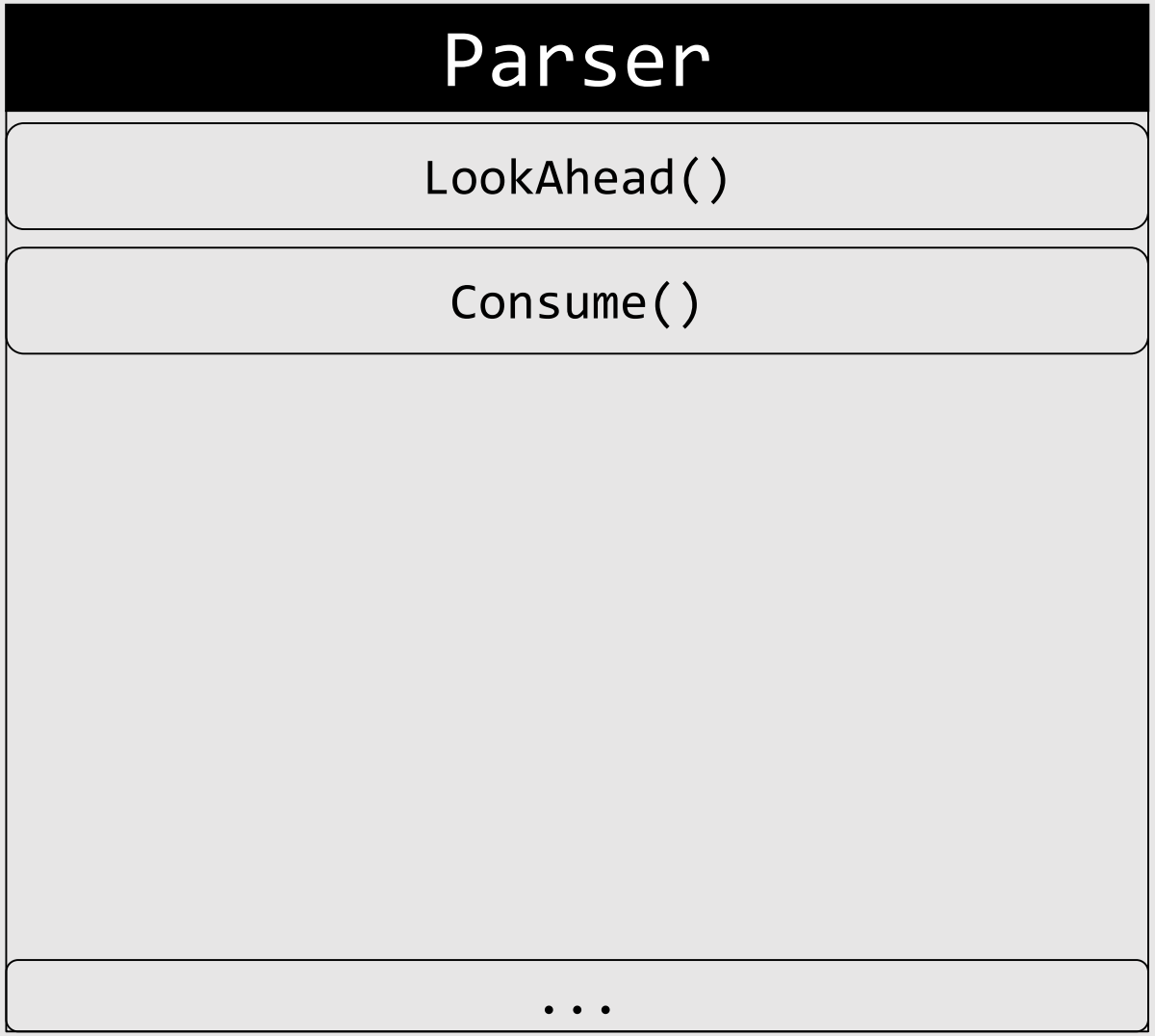
Lexer

Parser



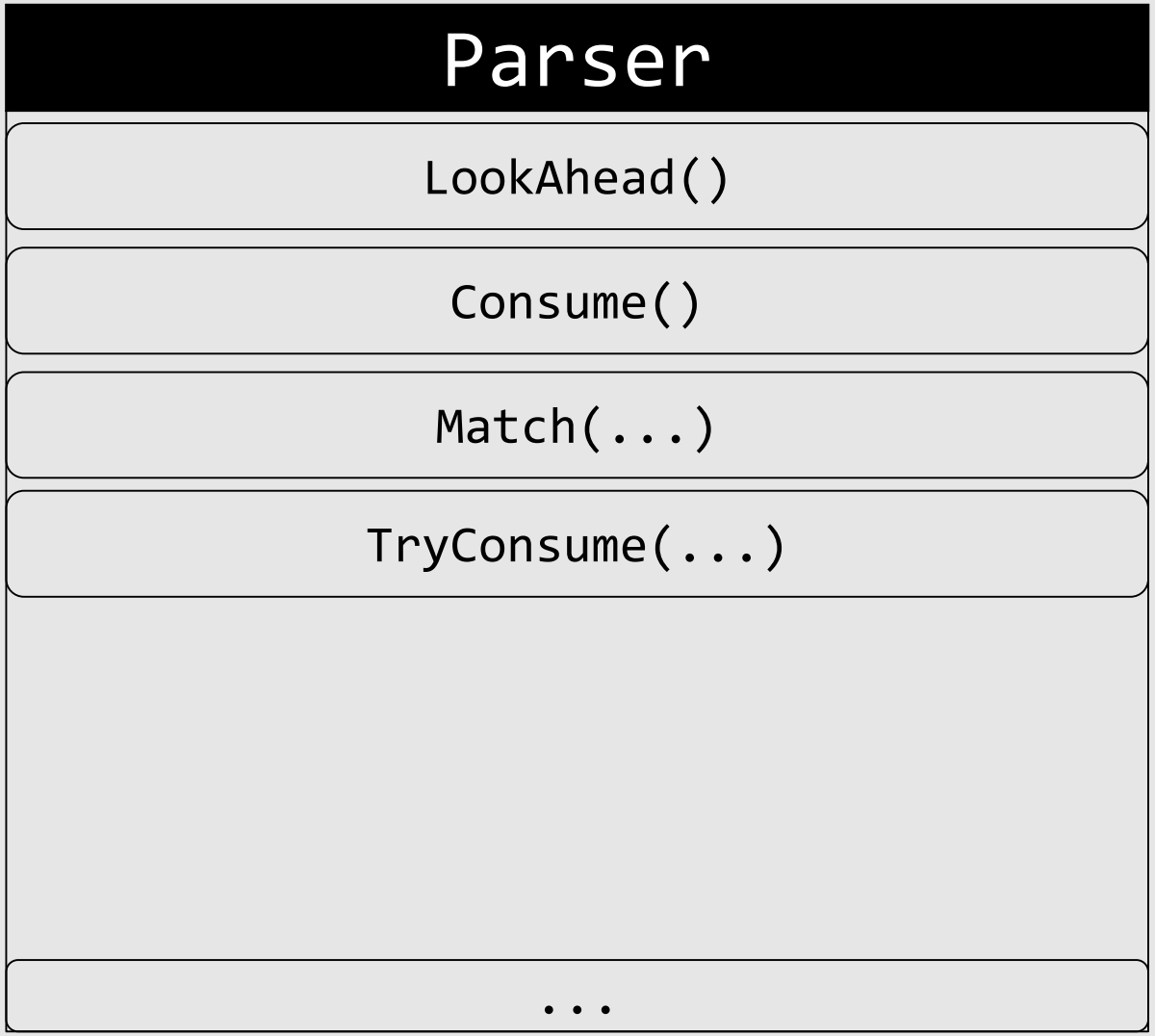
Sema


Parser
{=}
Id {minutes}
{+}
Id {left}
{*}
Id {seconds}
NumLit {60}



Reads the next token from input

Shifts lexer to the next token

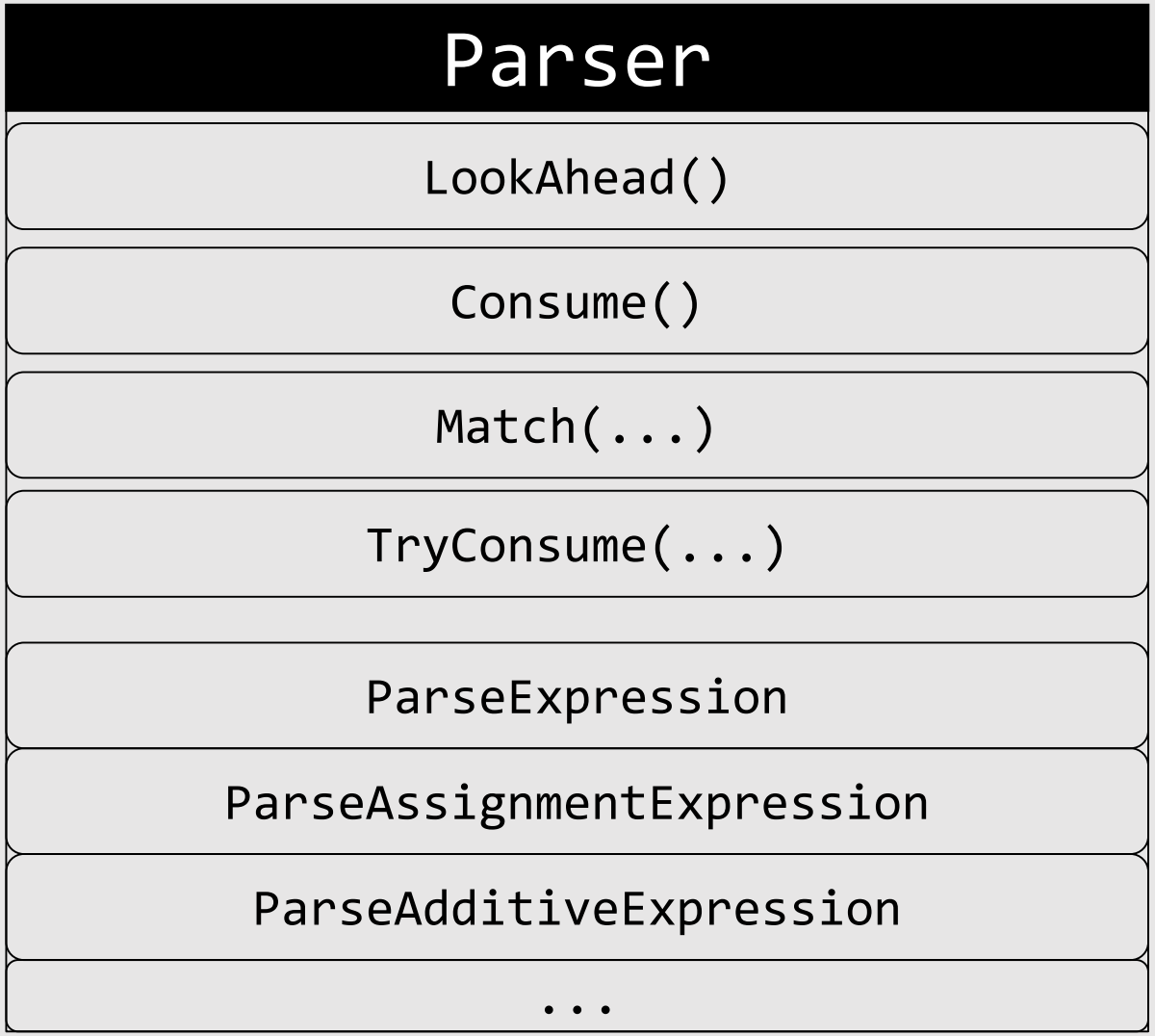


Reads the next token from input

Shifts lexer to the next token

Compares LookAhead with the arg

Match + Consume



Reads the next token from input

Shifts lexer to the next token

Compares LookAhead with the arg

Match + Consume

Parse input using Matchers and Consumers

C++ standard

Index	Summary	[gram]
A.1	General	[gram.general]
A.2	Keywords	[gram.key]
A.3	Lexical conventions	[gram.lex]
A.4	Basis	[gram.basis]
A.5	Expressions	[gram.expr]
A.6	Statements	[gram.stmt]
A.7	Declarations	[gam.dcl]
A.8	Modules	[gram.module]
A.9	Classes	[gram.class]
A.10	Overloading	[gram.over]
A.11	Templates	[gram.temp]
A.12	Exceptions handling	[gram.except]
A.13	Preprocessing directives	[gram.cpp]

C++ standard

Index	Summary	[gram]
A.1	General	[gram.general
A.2	Keywords	[gram.key]
A.3	Lexical conventions	[gram.lex]
A.4	Basis	[gram.basis]
A.5	Expressions	[gram.expr]
A.6	Statements	[gram.stmt]
A.7	Declarations	[gam.dcl]
A.8	Modules	[gram.module]
A.9	Classes	[gram.class]
A.10	Overloading	[gram.over]
A.11	Templates	[gram.temp]
A.12	Exceptions handling	[gram.except]
A.13	Preprocessing directives	[gram.cpp]

assignment-expression:
  conditional-expression
  yield-expression
  throw-expression

logical-or-expression
  assignment-operator initializer-clause
  assignment-operator:
    one of '= *= /= %= += - ...'

expression:
  assignment-expression
  expression , assignment-expression

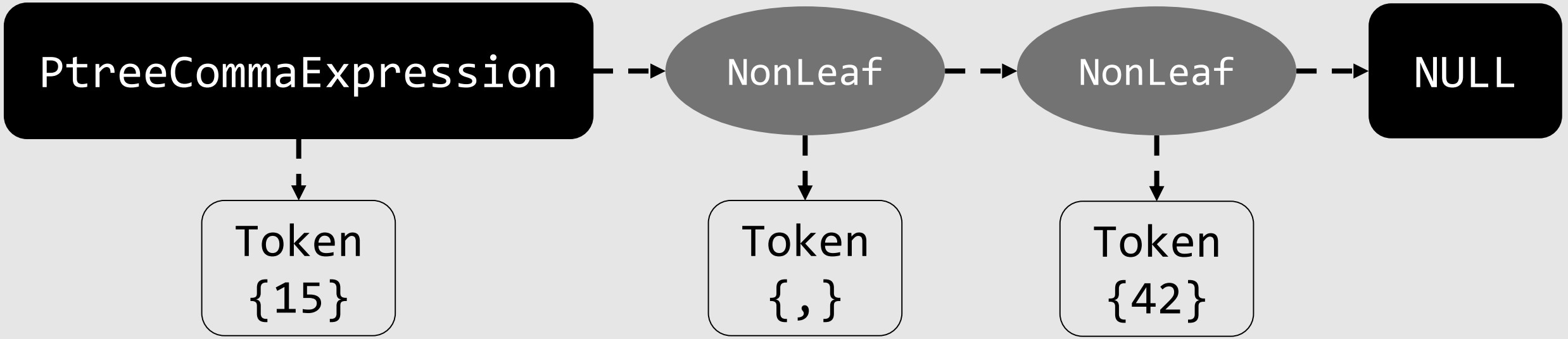
Expressions (Grammar)

expression:
  assignment-expression
  expression , assignment-expression

15	,	42

expression , assignment-expression

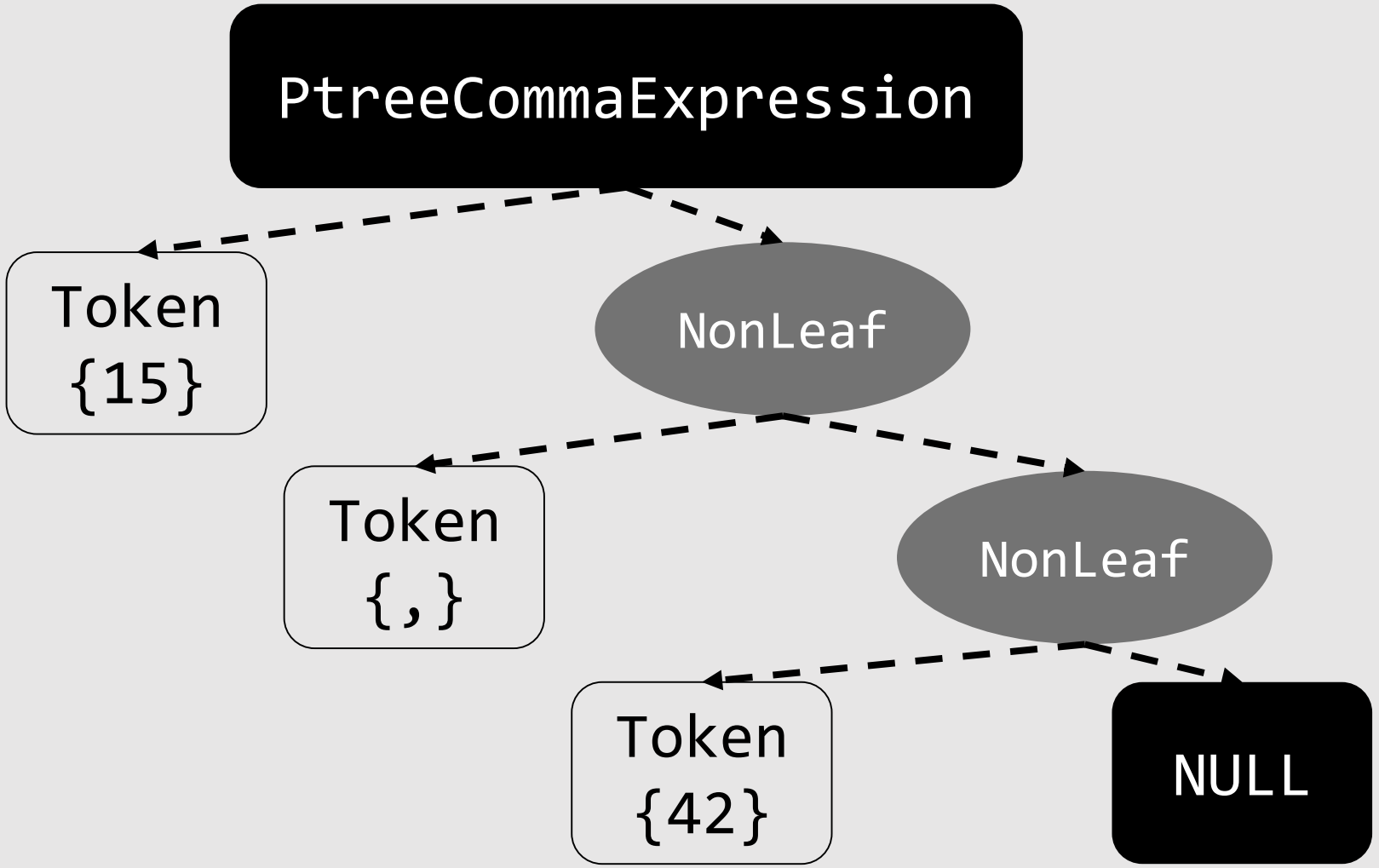
The tree



15	,	42

expression , assignment-expression

The tree


PtreeCommaExpression
Token
{15}
NonLeaf
Token
{,}
NonLeaf
Token
{42}
NULL

Recursive descent

```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```

Recursive descent

```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```

Recursive descent

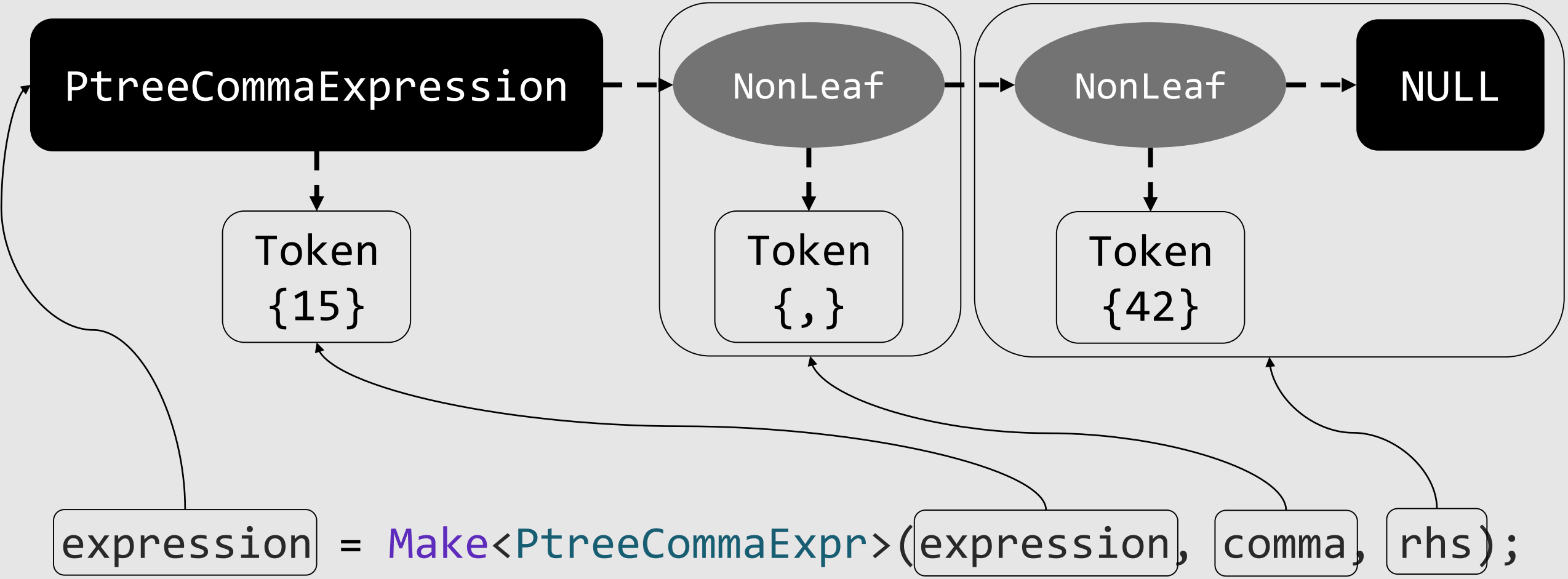
```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```

Recursive descent

```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```

Recursive descent

```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```


PtreeCommaExpression
NonLeaf
NonLeaf
NULL
Token
{15}
Token
{,}
Token
{42}
expression = Make<PtreeCommaExpr>(expression, comma, rhs);

Expressions (examples)

```
// assignment-expression:
//   conditional-expression
//   logical-or-expression assignment-operator initializer-clause
Tree *AssignmentExpression()
{
  auto lhs = LogicalOrExpression();

  // conditional-expression

  // assignment-operator
  if (Match('=', "*=", "/=", "%=", "+=", "-=", ">>=", "<<=", "&=", "^=", "|="))
  {
    auto assign = AssignmentOperator();
    auto inintClause = InitializerClause();
    return Make<PtreeAssignExpr>(lhs, assign, inintClause);
  }

  return lhs;
}
```

Expressions (examples)

```
// assignment-expression:
//   conditional-expression
//   logical-or-expression assignment-operator initializer-clause
Tree *AssignmentExpression()
{
  auto lhs = LogicalOrExpression();

  // conditional-expression

  // assignment-operator
  if (Match('=', "*=", "/=", "%=", "+=", "-=", ">>=", "<<=", "&=", "^=", "|="))
  {
    auto assign = AssignmentOperator();
    auto inintClause = InitializerClause();
    return Make<PtreeAssignExpr>(lhs, assign, inintClause);
  }

  return lhs;
}
```

Expressions (examples)

```
// assignment-expression:
//   conditional-expression
//   logical-or-expression assignment-operator initializer-clause
Tree *AssignmentExpression()
{
  auto lhs = LogicalOrExpression();

  // conditional-expression

  // assignment-operator
  if (Match('=', "*=", "/=", "%=", "+=", "-=", ">>=", "<<=", "&=", "^=", "|="))
  {
    auto assign = AssignmentOperator();
    auto inintClause = InitializerClause();
    return Make<PtreeAssignExpr>(lhs, assign, inintClause);
  }

  return lhs;
}
```

```
// conditional-expression:
//   logical-or-expression
//   ....
```

Expressions (examples)

```
// assignment-expression:
//   conditional-expression
//   logical-or-expression assignment-operator initializer-clause
Tree *AssignmentExpression()
{
  auto lhs = LogicalOrExpression();

  // conditional-expression
  if (Match("?"))
  {
    auto qMark = Consume();
    auto op1 = Expression();
    auto colon = TryConsume(':');
    auto op2 = AssignmentExpression();
    return Make<PtreeInfixExpr>(lhs, qMark, op1, colon, op2);
  }
  // assignment-operator
  return lhs;
}
```

```
// conditional-expression:
//   logical-or-expression
//   logical-or-expression ?
//     expression : assignment-expression
```

Expressions (examples)

```
// logical-or-expression
//   logical-and-expression
//   logical-or-expression || logical-and-expression
Tree *LogicalOrExpression()
{
  auto lhs = LogicalAndExpression();

  while (Match("||"))
  {
    auto lor = Consume();
    auto rhs = LogicalAndExpression();
    lhs = Make<PtreeInfixExpr>(lhs, lor, rhs);
  }

  return lhs;
}
```

Optional rules

```
// new-expression:
//   ::' new new-placement' ( type-id ) new-initializer'
//   ::' new new-placement' new-type-id new-initializer'

Ptree *NewExpression()
{
  Ptree *colonColon = nullptr; // opt
  if (Match("::")) {
    colonColon = MakeTree(Consume());
  }

  Ptree *new_ = MakeTree(TryConsume(VivaCore::tkNew));
  if (!new_) {
    return colonColon ? SyntaxError(....) : nullptr;
  }
  Ptree *newPlacement = NewPlacement(); // opt
  // ....
}
```

```
// new-expression:
//   ::' new new-placement' ( type-id ) new-initializer'
//   ::' new new-placement' new-type-id new-initializer'

Ptree *NewExpression()
{
  Ptree *colonColon = nullptr; // opt
  if (Match("::")) {
    colonColon = MakeTree(Consume());
  }

  Ptree *new_ = MakeTree(TryConsume(VivaCore::tkNew));
  if (!new_) {
    return colonColon ? SyntaxError(....) : nullptr;
  }
  Ptree *newPlacement = NewPlacement(); // opt
  // ....
}
```

```
// new-expression:
//   ::' new new-placement' ( type-id ) new-initializer'
//   ::' new new-placement' new-type-id new-initializer'

Ptree *NewExpression()
{
  // ....
  Ptree *typeIdClause = nullptr;
  if (Match('(')) {
    auto lParen = MakeTree(Consume());
    if (!lParen) return SyntaxError(....);
    auto typeId = MakeTree(TypeId());
    if (!typeId) return SyntaxError(....);
    auto rParen = MakeTree(TryConsume(')'));
    if (!rParen) return SyntaxError(....);
    typeIdClause = MakeTree(lParen, typeId, rParen);
  }
  else
    typeIdClause = NewTypeId();
  if (!typeIdClause) return SyntaxError(....);
  auto newInitializer = NewInitializer(); // opt
  // ....
}
```

```
// new-expression:
//   ::' new new-placement' ( type-id ) new-initializer'
//   ::' new new-placement' new-type-id new-initializer'

Ptree *NewExpression()
{
  // ....
  Ptree *typeIdClause = nullptr;
  if (Match('(')) {
    auto lParen = MakeTree(Consume());
    if (!lParen) return SyntaxError(....);
    auto typeId = MakeTree(TypeId());
    if (!typeId) return SyntaxError(....);
    auto rParen = MakeTree(TryConsume(')'));
    if (!rParen) return SyntaxError(....);
    typeIdClause = MakeTree(lParen, typeId, rParen);
  }
  else
    typeIdClause = NewTypeId();
  if (!typeIdClause) return SyntaxError(....);
  auto newInitializer = NewInitializer(); // opt
  // ....
}
```

```
Ptree *NewExpression()
{
  // ....
  if (colonColon){
    if (newPlacement){
      if (newInitializer) return Make<PtreeNewExpression>(colonColon, new_,
                                    newPlacement, typeIdClause, newInitializer);
      return Make<PtreeNewExpression>(colonColon, new_, newPlacement, typeIdClause);
    }
    else if (newInitializer) return Make<PtreeNewExpression>(colonColon, new_,
                                                typeIdClause, newInitializer);
    return Make<PtreeNewExpression>(colonColon, new_, typeIdClause);
  }
  else if (newPlacement){
    if (newInitializer) return Make<PtreeNewExpression>(new_, newPlacement,
                                                 typeIdClause, newInitializer);
    return Make<PtreeNewExpression>(newPlacement, new_, typeIdClause);
  }

  return Make<PtreeNewExpression>(new_, typeIdClause);
}
```

```
Ptree *NewExpression()
{
  // ....
  if (colonColon){
    if (newPlacement){
      if (newInitializer) return Make<PtreeNewExpression>(colonColon, new_,
                                    newPlacement, typeIdClause, newInitializer);
      return Make<PtreeNewExpression>(colonColon, new_, newPlacement, typeIdClause);
    }
    else if (newInitializer) return Make<PtreeNewExpression>(colonColon, new_,
                                                typeIdClause, newInitializer);
    return Make<PtreeNewExpression>(colonColon, new_, typeIdClause);
  }
  else if (newPlacement){
    if (newInitializer) return Make<PtreeNewExpression>(new_, newPlacement,
                                                typeIdClause, newInitializer);
    return Make<PtreeNewExpression>(newPlacement, new_, typeIdClause);
  }

  return Make<PtreeNewExpression>(new_, typeIdClause);
}
```

1

2

3

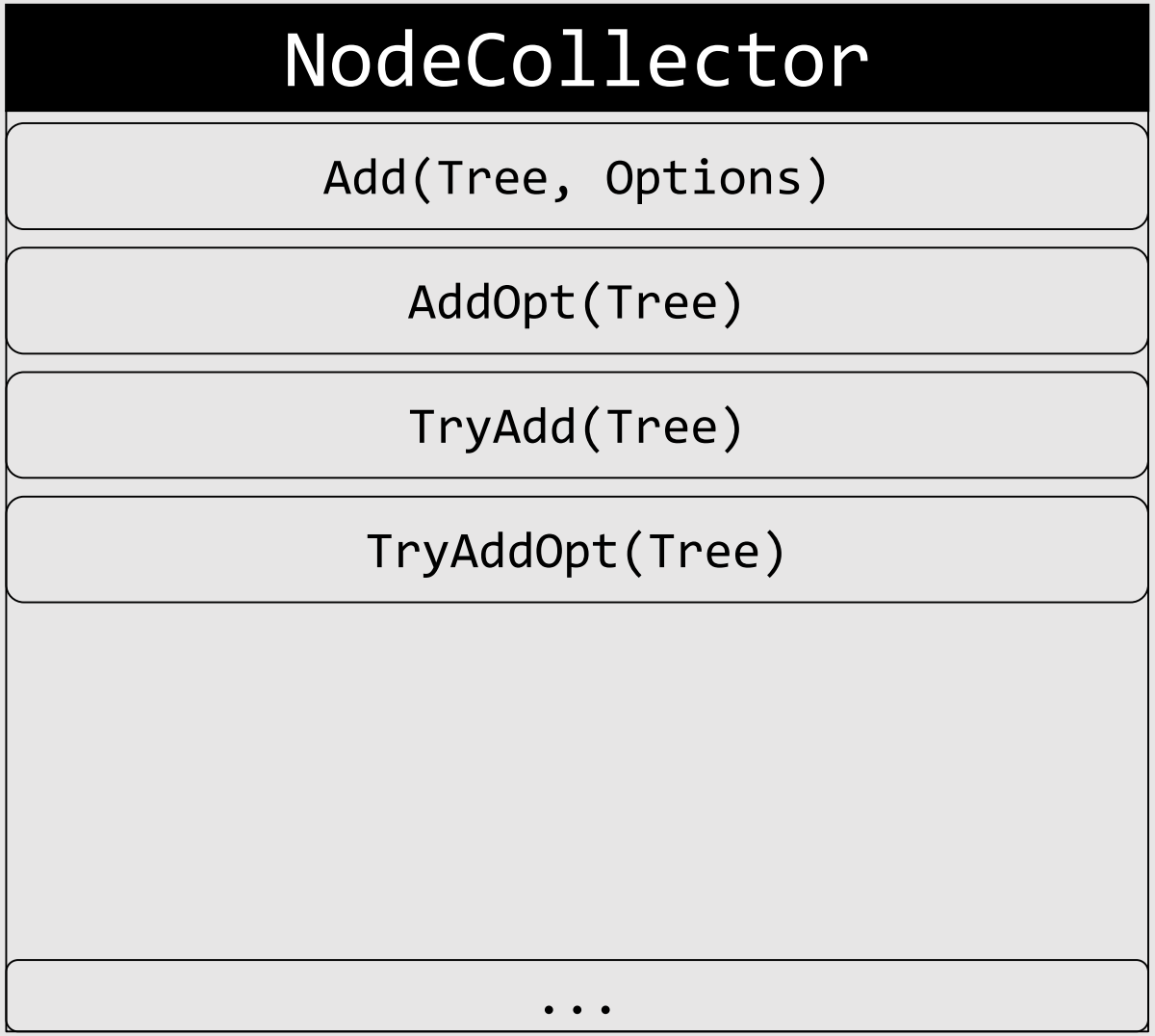
4

5

6

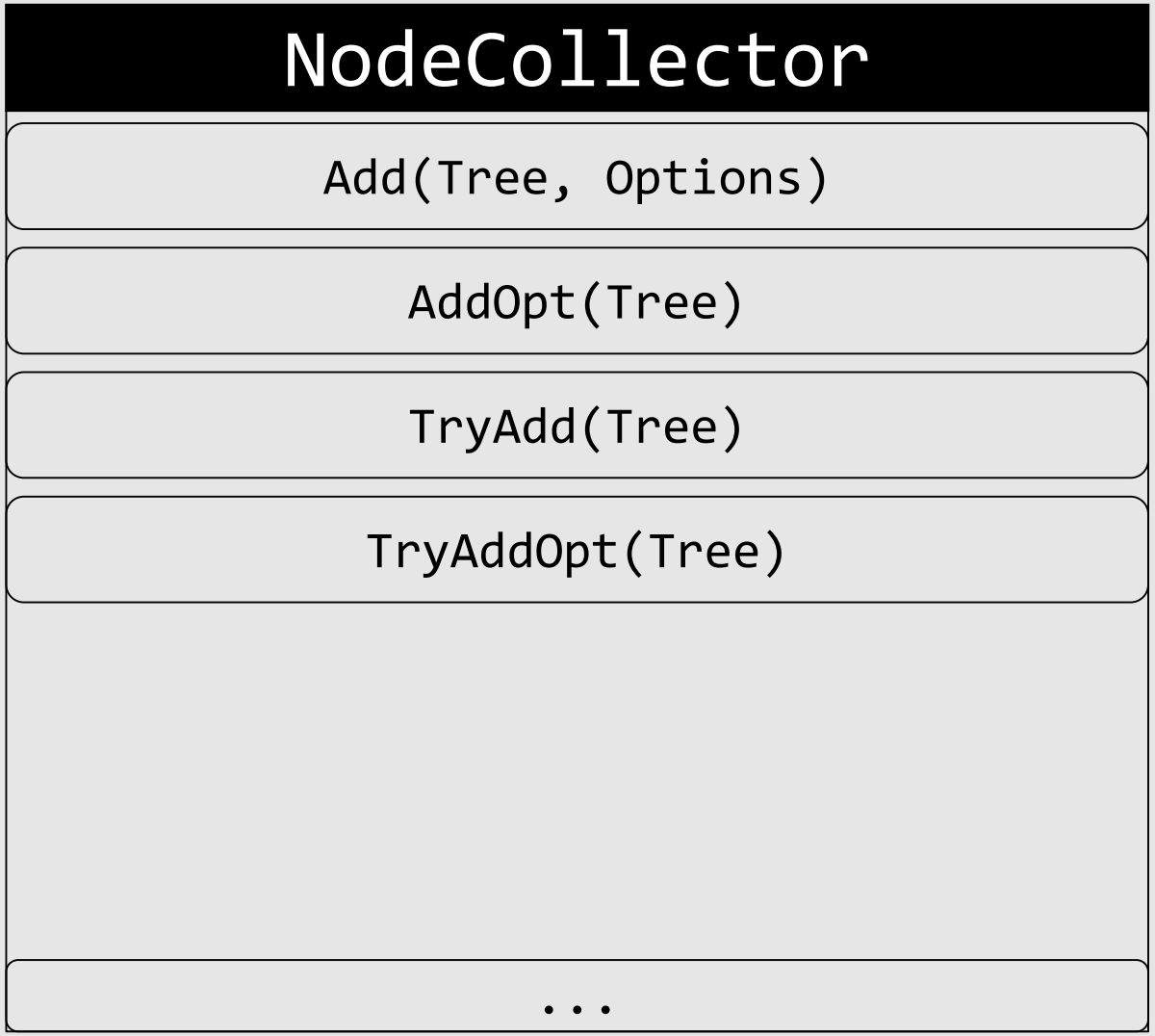
7

Syntax Tree Builder


NodeCollector
Add(Tree, Options)
AddOpt(Tree)
TryAdd(Tree)
TryAddOpt(Tree)
...

NodeCollector
Add(Tree, Options)
AddOpt(Tree)
TryAdd(Tree)
TryAddOpt(Tree)
...

Options
None
Option
Trivia
Ignore

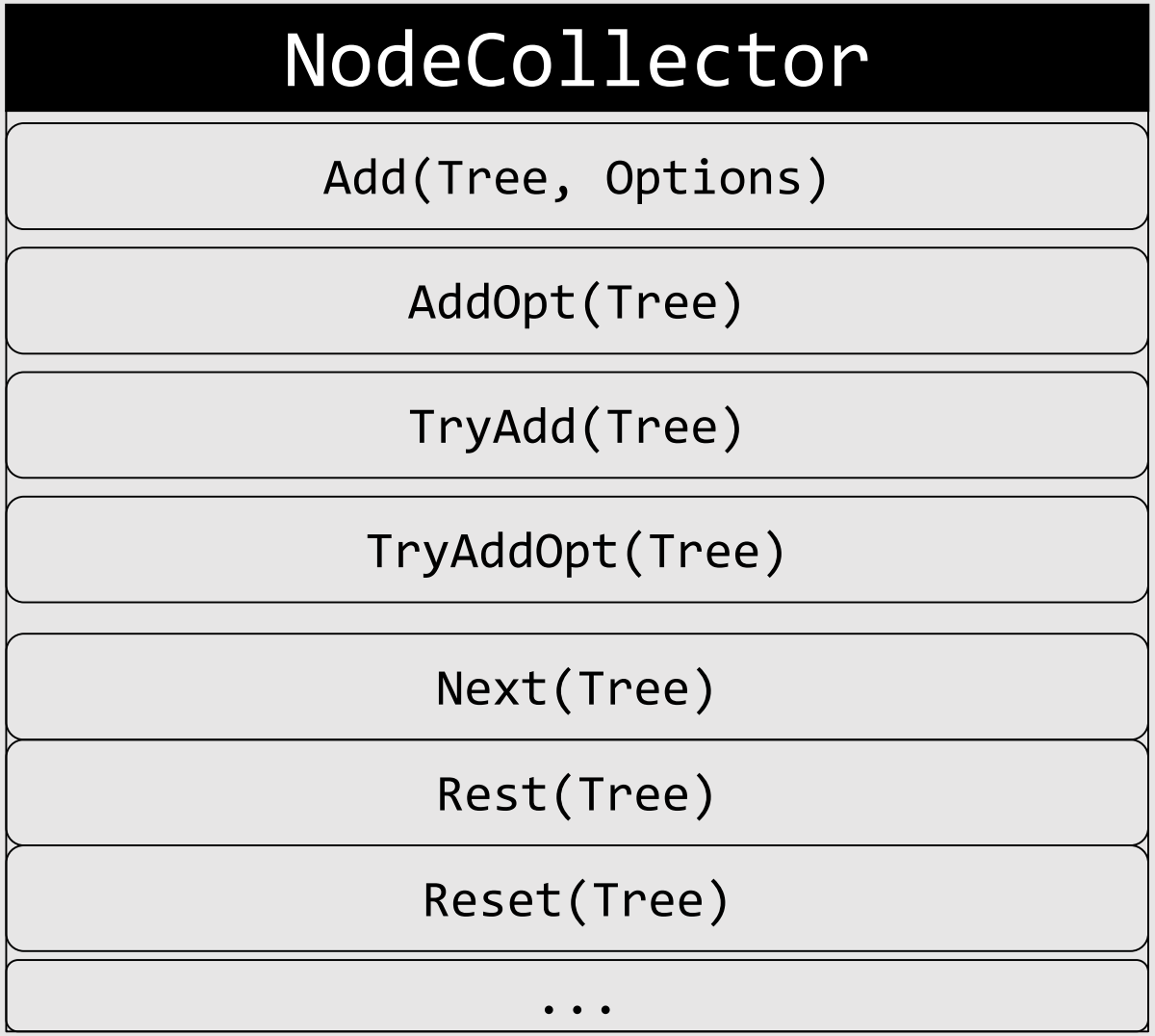


Adds the tree node to the collection

Adds and marks as optional

Adds if not null

Adds if not null and marks as optional



Adds the tree node to the collection

Adds and marks as optional

Adds if not null

Adds if not null and marks as optional

Before

```
// new-expression:
//   ::' new new-placement' ( type-id ) new-initializer'
//   ::' new new-placement' new-type-id new-initializer'
Ptree *NewExpression()
{
  Ptree *colonColon = nullptr; // opt
  if (Match("::")) {
    colonColon = MakeTree(Consume());
  }

  Ptree *new_ = MakeTree(TryConsume(VovaCpre::tkNew));
  if (!new_) {
    return colonColon ? SyntaxError(....) : nullptr;
  }
  Ptree *newPlacement = NewPlacement(); // opt
  // ....
}
```

After

```
// new-expression:
//   ::' new new-placement' ( type-id ) new-initializer'
//   ::' new new-placement' new-type-id new-initializer'
Tree *Parser::NewExpression()
{
  auto collector = TreeBuilder().MakeCollector();

  collector.AddOpt(TryConsume("::")); // opt
  collector.Add   (TryConsume(tk::NEW));
  collector.AddOpt(NewPlacement());   // opt

  // add ( type-id ) or new-type-id in collector

  collector.AddOpt(NewInitializer());
  // ....
}
```

Before

```
Ptree *NewExpression()
{
  // ....
  Ptree *typeIdClause = nullptr;
  if (Match('(')) {
    auto lParen = MakeTree(Consume());
    if (!lParen) return SyntaxError(....);
    auto typeId = MakeTree(TypeId());
    if (!typeId) return SyntaxError(....);
    auto rParen = MakeTree(TryConsume(')'));
    if (!rParen) return SyntaxError(....);
    typeIdClause = MakeTree(lParen, typeId, rParen);
  }
  else
    typeIdClause = NewTypeId();
  if (!typeIdClause) return SyntaxError(....);
  auto newInitializer = NewInitializer(); // opt
  // ....
}
```

After

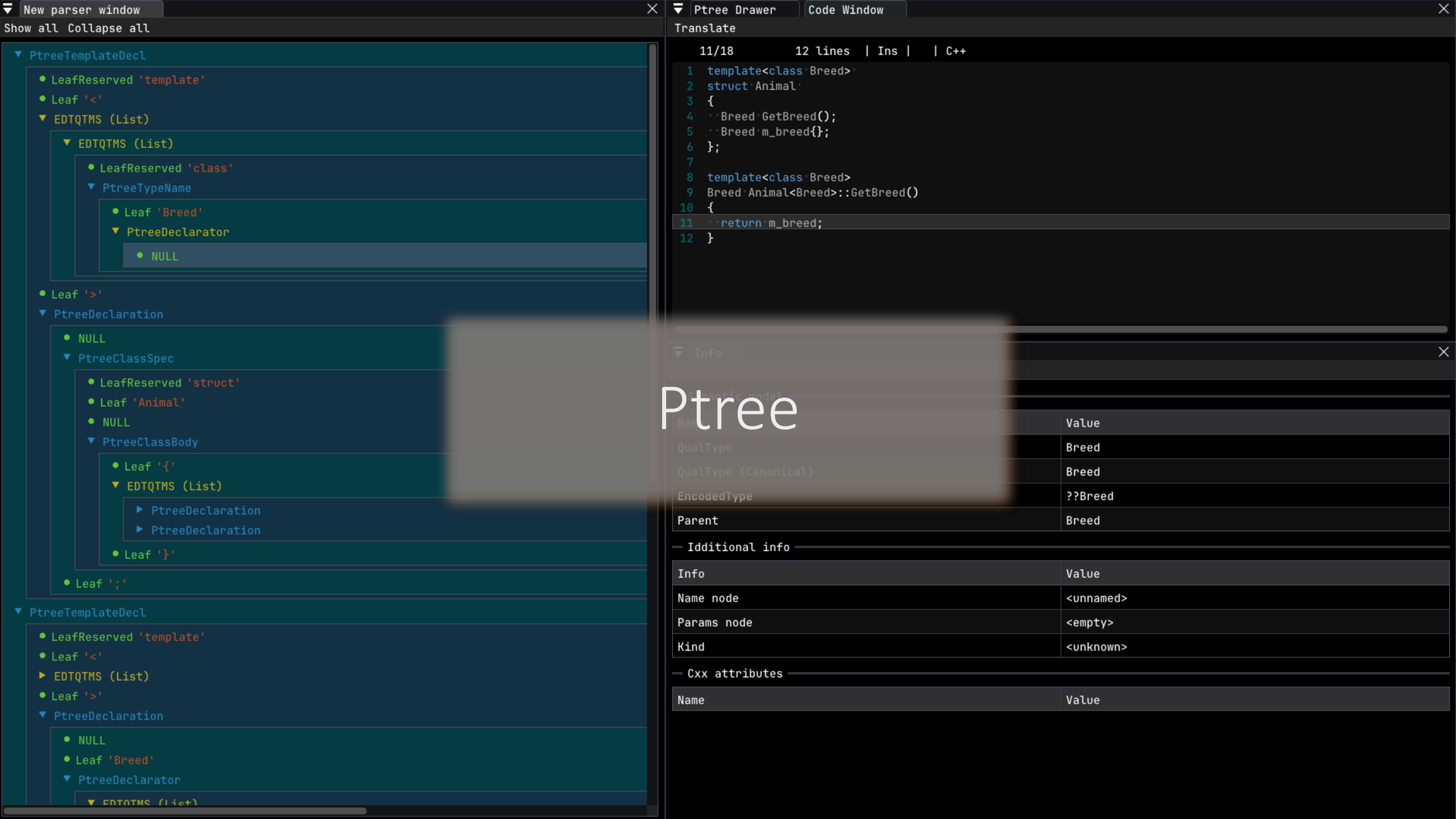
```
Tree *Parser::NewExpression()
{
  // ....

  if (Match('('))
  {
    auto parensTypeId = m_treeBuilder.MakeCollector();
    parensTypeId.Add(Consume());
    parensTypeId.Add(TypeId());
    parensTypeId.Add(Consume(')'));
    collector.Add(MakeTree(parensTypeId.Next(), parensTypeId.Rest()));
  }
  else
  {
    collector.Add(NewTypeId());
  }

  // ....
}
```

Before

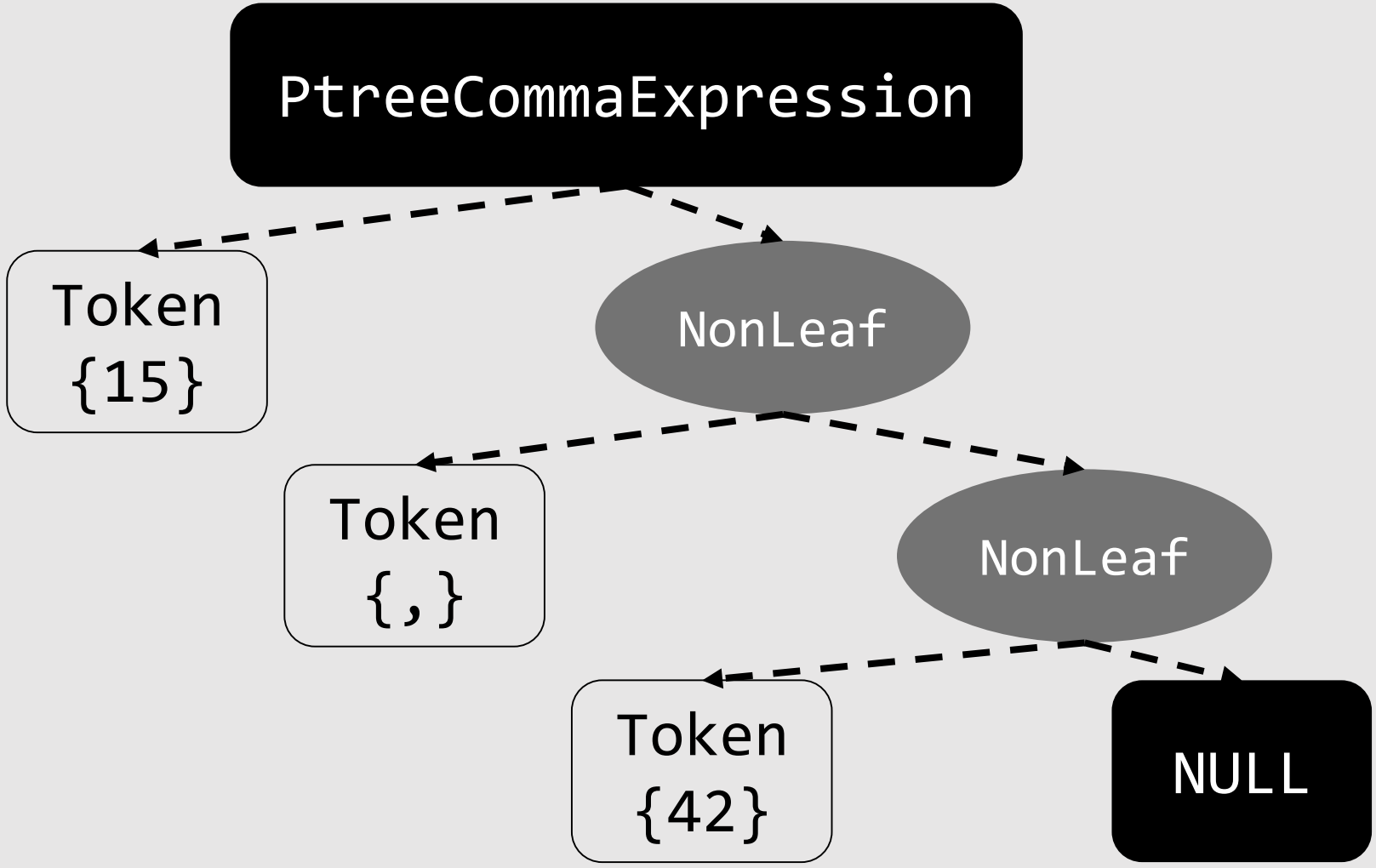
```
Ptree *NewExpression()
{
  // ....
  if (colonColon){
    if (newPlacement){
      if (newInitializer) return Make<PtreeNewExpression>(colonColon, new_,
                                    newPlacement, typeIdClause, newInitializer);
      return Make<PtreeNewExpression>(colonColon, new_, newPlacement, typeIdClause);
    }
    else if (newInitializer) return Make<PtreeNewExpression>(colonColon, new_,
                                                  typeIdClause, newInitializer);
    return Make<PtreeNewExpression>(colonColon, new_, typeIdClause);
  }
  else if (newPlacement){
    if (newInitializer) return Make<PtreeNewExpression>(new_, newPlacement,
                                                   typeIdClause, newInitializer);
    return Make<PtreeNewExpression>(newPlacement, new_, typeIdClause);
  }

  return Make<PtreeNewExpression>(new_, typeIdClause);
}
```

After

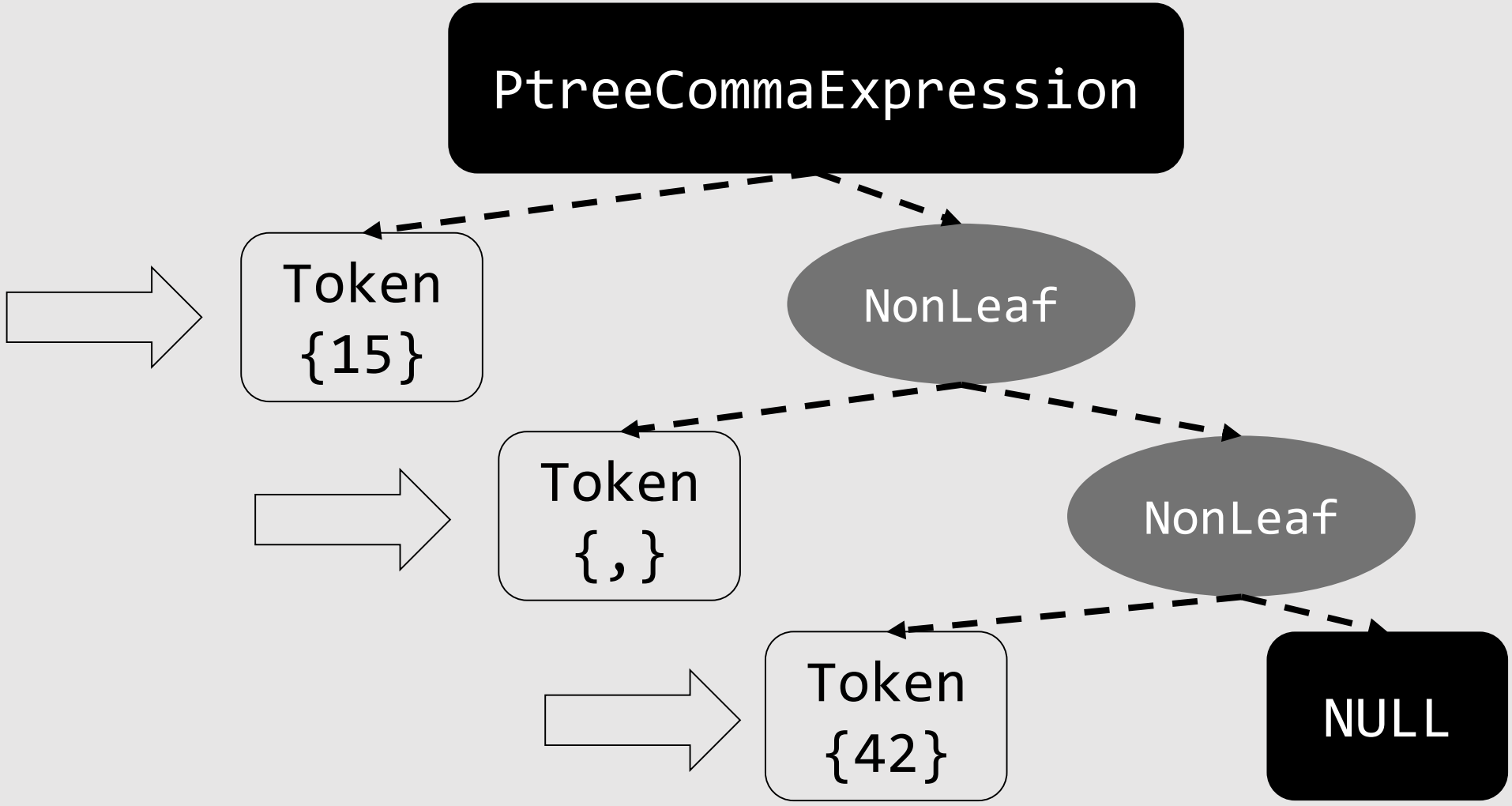
```
Ptree *NewExpression()
{
  // ....
  return Make<PtreeNewExpression>(builder.Next(), builder.Rest());
}
```


New parser window
Show all Collapse all
PtreeTemplateDecl
LeafReserved 'template'
Leaf '<'
EDTQTMS (List)
EDTQTMS (List)
LeafReserved 'class'
PtreeTypeName
Leaf 'Breed'
PtreeDeclarator
NULL
Leaf '>'
PtreeDeclaration
NULL
PtreeClassSpec
LeafReserved 'struct'
Leaf 'Animal'
NULL
PtreeClassBody
Leaf '{'
EDTQTMS (List)
PtreeDeclaration
PtreeDeclaration
Leaf '}'
Leaf ';'
PtreeTemplateDecl
LeafReserved 'template'
Leaf '<'
EDTQTMS (List)
Leaf '>'
PtreeDeclaration
NULL
Leaf 'Breed'
PtreeDeclarator
Ptree Drawer
Code Window
Translate
11/18 12 lines | Ins | | C++
1 template<class Breed>
2 struct Animal
3 {
4 Breed GetBreed();
5 Breed m_breed{};
6 };
7
8 template<class Breed>
9 Breed Animal<Breed>::GetBreed()
10 {
11 return m_breed;
12 }
Info
Value
QualType
Breed
QualType (Canonical)
Breed
EncodedType
??Breed
Parent
Breed
Iddditional info
Info
Value
Name node
<unnamed>
Params node
<empty>
Kind
<unknown>
Cxx attributes
Name
Value
Ptree

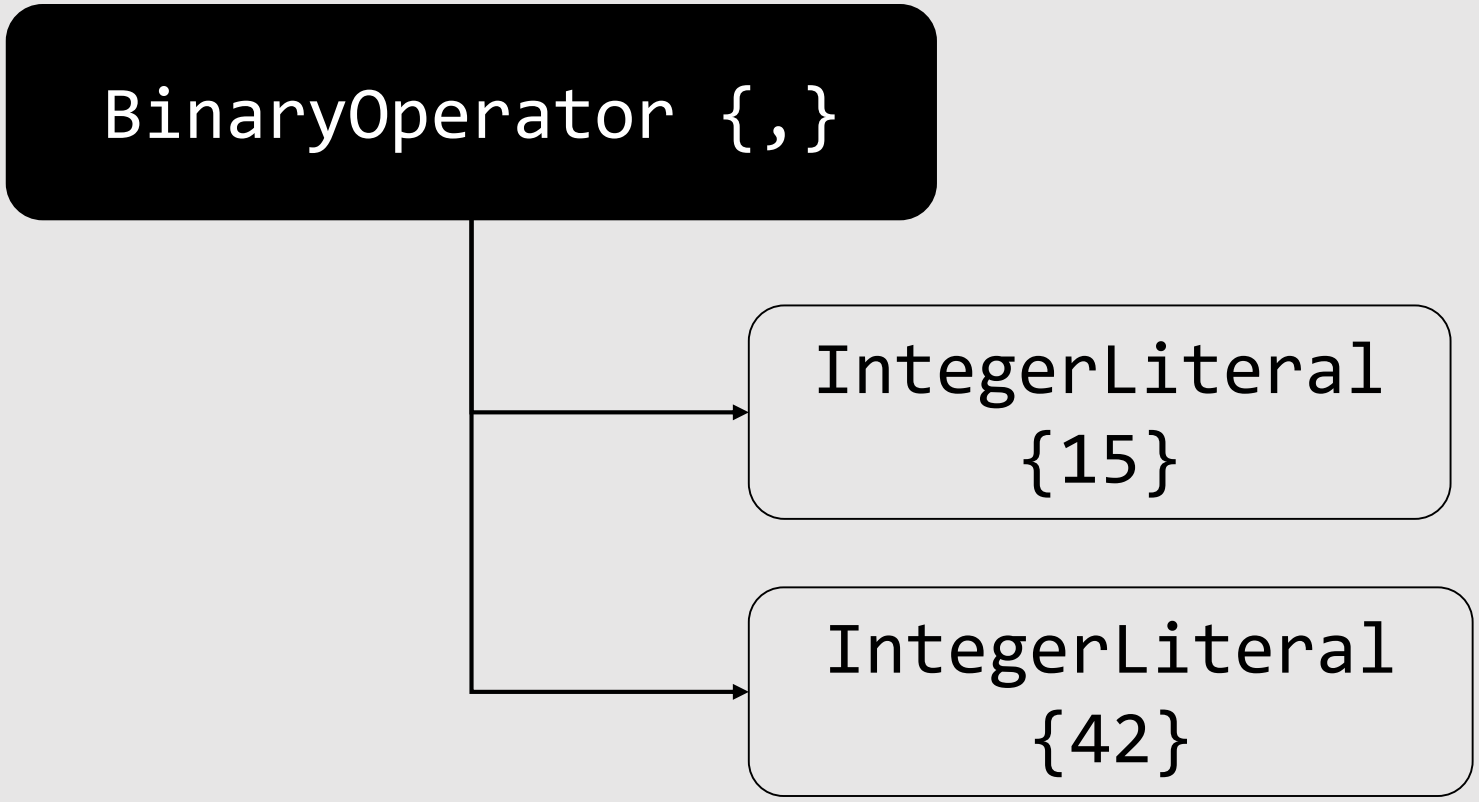
Syntax Tree


PtreeCommaExpression
Token
{15}
NonLeaf
Token
{,}
NonLeaf
Token
{42}
NULL

Syntax Tree


PtreeCommaExpression
Token
{15}
NonLeaf
Token
{,}
NonLeaf
Token
{42}
NULL

Abstract Syntax Tree



Syntax Tree

```
template<class Breed>
struct Animal
{
  Breed GetBreed();
  Breed m_breed{};
};

template<class Breed>
Breed Animal<Breed>::GetBreed()
{
  return m_breed;
}
```

Syntax Tree

```
TranslationUnitDecl
|-ClassTemplateDecl Animal
| |-TemplateTypeParmDecl Breed
| `-CXXRecordDecl struct Animal definition
|   ....
`-CXXMethodDecl GetBreed 'Breed ()'
  `-CompoundStmt
    `-ReturnStmt
      `-MemberExpr 'Breed’ -> m_breed
        `-CXXThisExpr 'Animal<Breed> *'
```

```
template<class Breed>
struct Animal
{
  Breed GetBreed();
  Breed m_breed{};
};

template<class Breed>
Breed Animal<Breed>::GetBreed()
{
  return m_breed;
}
```

AST Interface

```
enum class NodeKind : std::uint8_t
{
  Error,
  // ....
  // Expressions
  ImplicitVoidExpr,
  LiteralExpr,
  DeclRefExpr,
  CxxThisExpr,
  LambdaExpr,
  CxxFoldExpr,
  RequiresExpr,
  UnaryExpr,
  PostfixExpr,
  BinaryExpr,
  AssignExpr,
  // ....
};
```

AST Interface

```
enum class NodeKind : std::uint8_t
{
  Error,
  // ....
  // Expressions
  ImplicitVoidExpr,
  LiteralExpr,
  DeclRefExpr,
  CxxThisExpr,
  LambdaExpr,
  CxxFoldExpr,
  RequiresExpr,
  UnaryExpr,
  PostfixExpr,
  BinaryExpr,
  AssignExpr,
  // ....
};
```

```
enum class LNodeKind : std::uint8_t
{
  // ....
  TypedefMultiDecl,
  UnnamedTypedefMultiDecl,
  AnyLabelStmt,
  LinkageSpec,
  ExplicitExpr,
  GccCaseRange,
  ConversionOperatorDecl,
  MSVC_TryExceptStmt,
  MSVC_LeaveStmt,
  // ....
};
```

AST Interface

```
class PtreeBuilder
{
// ....
  template <typename Kind, Kind kind>
  [[nodiscard]] Tree *Make(NodeCollector &&) = delete;

  template <auto AstKind>
  [[nodiscard]] Tree *Make(NodeCollector &&nodes)
  {
      return Make<decltype(AstKind), AstKind>(std::move(nodes));
  }
// ....
};
```

AST Interface

```
class PtreeBuilder
{
  using NK = NodeKind;
  using Collector = NodeCollector;
public: // Expressions
  template<> Tree *Make<NK, NK::ImplicitVoidExpr>(Collector &&);
  template<> Tree *Make<NK, NK::LiteralExpr>     (Collector &&);
  template<> Tree *Make<NK, NK::DeclRefExpr>     (Collector &&);
  template<> Tree *Make<NK, NK::CxxThisExpr>     (Collector &&);
  template<> Tree *Make<NK, NK::LambdaExpr>      (Collector &&);
  template<> Tree *Make<NK, NK::CxxFoldExpr>     (Collector &&);
  template<> Tree *Make<NK, NK::RequiresExpr>    (Collector &&);
  template<> Tree *Make<NK, NK::UnaryExpr>       (Collector &&);
  template<> Tree *Make<NK, NK::PostfixExpr>     (Collector &&);
  template<> Tree *Make<NK, NK::BinaryExpr>      (Collector &&);
  template<> Tree *Make<NK, NK::AssignExpr>      (Collector &&);
// ....
};
```

AST Interface

```
class PtreeBuilder
{
  using LNK = LNodeKind;
public: // Legacy nodes
  template<> Tree *Make<LNK, LNK::GccCaseRange>           (Collector &&);
  template<> Tree *Make<LNK, LNK::ConversionOperatorDecl>(Collector &&);
  template<> Tree *Make<LNK, LNK::MSVC_TryExceptStmt>     (Collector &&);
  template<> Tree *Make<LNK, LNK::MSVC_LeaveStmt>         (Collector &&);
// ....
};
```

AST Interface

```
template<>
Tree *PtreeBuilder::Make<NK, NK::BinaryExpr>(Collector &&collector)
{
  subbuilder.Skip();
  auto op = collector.TryNextAsTree();
  subbuilder.Reset();

  if (PtreeUtil::Eq(op, ','))
    return Conv<PtreeCommaExpr>(std::move(collector));
  else if (PtreeUtil::Eq(op, ".->") || PtreeUtil::Eq(op, ".*"))
    return Conv<PtreePmExpr>(std::move(collector));
  else if (PtreeUtil::Eq(op, '='))
    return Conv<PtreeAssignExpr>(std::move(collector));

  return Conv<PtreeInfixExpr>(std::move(collector));
}
```

AST Interface

```
class Parser
{
  // ....
  template<auto Kind>
  Tree *Make(NodeCollector &&nodes)
  {
    return TreeBuilder().Make<Kind>(std::move(nodes));
  }
  // ....
};
```

Now

```
// additive-expression:
//   multiplicative-expression
//   additive-expression + multiplicative-expression
//   additive-expression - multiplicative-expression
Tree *AdditiveExpression()
{
  auto lhs = MultiplicativeExpression();

  while (Match('+', '-'))
  {
    auto collector = MakeCollector();

    collector.Add(lhs);
    collector.Add(Consume());
    collector.Add(MultiplicativeExpression>());

    lhs = Make<NodeKind::BinaryExpr>(std::move(collector));
  }
  return lhs;
}
```

Lambda "Expression"

Lambda "Expression"

```
primary-expression:
  literal
  this
  ( expression )
  id-expression
  lambda-expression
  fold-expression
  requires-expression
```

Lambda "Expression"

```
primary-expression:
  literal
  this
  ( expression )
  id-expression
  lambda-expression
  fold-expression
  requires-expression
```

Lambda "Expression"

```
primary-expression:
  literal
  this
  ( expression )
  id-expression
  lambda-expression
  fold-expression
  requires-expression

lambda-expression:
  lambda-introducer attribute-specifier-seq' lambda-declarator compound-statement
  lambda-introducer < template-parameter-list > requires-clause' attribute-
specifier-seq'
  lambda-declarator compound-statement
```

Lambda "Expression"

```
primary-expression:
  literal
  this
  ( expression )
  id-expression
  lambda-expression
  fold-expression
  requires-expression

lambda-expression:
  lambda-introducer attribute-specifier-seq' lambda-declarator compound-statement
  lambda-introducer < template-parameter-list > requires-clause' attribute-
specifier-seq'
  lambda-declarator compound-statement
```

Lambda "Expression"

```
statement:
  labeled-statement
  attribute-specifier-seq' expression-statement
  attribute-specifier-seq' compound-statement
  attribute-specifier-seq' selection-statement
  attribute-specifier-seq' iteration-statement
  attribute-specifier-seq' jump-statement
  attribute-specifier-seq' try-block
  declaration-statement
```

Lambda "Expression"

```
int Meow()
{
  return 1 + []() -> int {
    struct Oops
    {
      int operator()()
      {
        return 2;
      }
    };
    Oops o1;
    Oops o2;
    return o1() + o2();
  }();
}
```

Lambda "Expression"

```
int Meow()
{
  return 1 + []() -> int {
    struct Oops
    {
      int operator()()
      {
        return 2;
      }
    };
    Oops o1;
    Oops o2;
    return o1() + o2();
  }();
}
```

Lambda "Expression"

```
int Meow()
{
  return 1 + []() -> int {
    struct Oops
    {
      int operator()()
      {
        return 2;
      }
    };
    Oops o1;
    Oops o2;
    return o1() + o2();
  }();
}
```

Lambda "Expression"

```
int Meow()
{
  return 1 + []() -> int {
    struct Oops
    {
      int operator()()
      {
        return 2;
      }
    };
    Oops o1;
    Oops o2;
    return o1() + o2();
  }();
}
```

Lambda "Expression"

```
int Meow()
{
  return 1 + []() -> int {
    struct Oops
    {
      int operator()()
      {
        return 2;
      }
    };
    Oops o1;
    Oops o2;
    return o1() + o2();
  }();
}
```

Declarations are jerks

```
int Meow(int);
int Meow( 2 );
```

```
. int Meow(2);
```

```
declaration:
 .name-declaration
  special-declaration
```

```
. int Meow(2);
```

```
name-declaration:
 .block-declaration
  nodeclspec-function-declaration
  function-definition
  deduction-guide
  empty-declaration
```

```
. int Meow(2);
```

```
block-declaration:
 .simple-declaration
```

. int Meow(2);

```
simple-declaration:
                          .decl-specifier-seq init-declarator-list' ;
  attribute-specifier-seq  decl-specifier-seq init-declarator-list  ;
  attribute-specifier-seq' decl-specifier-seq ref-qualifier' [
identifier-list ] initializer ;
```

. int Meow(2);

decl-specifier-seq -> ... -> simple-type-specifier -> int

```
Tree *SimpleTypeSpecifier()
{
  if Match(tk::Int, tk::Char, ....)
  {
    return Make<Trivia::BuiltinType>(Consume());
  }
  return nullptr;
}
```

. int Meow(2);

```
simple-declaration:
                          .decl-specifier-seq init-declarator-list' ;
  attribute-specifier-seq  decl-specifier-seq init-declarator-list  ;
  attribute-specifier-seq' decl-specifier-seq ref-qualifier' [
identifier-list ] initializer ;
```

int -> simple-type-specifier -> ... -> decl-specifier-seq

int . Meow(2);

```
simple-declaration:
                          decl-specifier-seq .init-declarator-list' ;
  attribute-specifier-seq  decl-specifier-seq  init-declarator-list  ;
  attribute-specifier-seq' decl-specifier-seq  ref-qualifier' [
identifier-list ] initializer ;
```

```
int . Meow(2);
```

```
simple-declaration:
  decl-specifier-seq .init-declarator-list ;
```

```
int . Meow(2);
```

```
init-declarator-list:
  .init-declarator
   init-declarator-list , init-declarator
```

```
int . Meow(2);
```

```
init-declarator:
  .declarator initializer'
```

```
int . Meow(2);
```

```
declarator:
  ptr-declarator // no matches
 .noptr-declarator parameters-and-qualifiers
```

int . Meow(2);

```
declarator:
  ptr-declarator // no matches
 .noptr-declarator parameters-and-qualifiers
```

Meow -> unqualified-id -> id-expression -> ... -> declarator-id -> noptr-declarator

```
int Meow . (2);
```

```
declarator:
  ptr-declarator // no matches
  noptr-declarator .parameters-and-qualifiers
```

int Meow . (2);

```
parameters-and-qualifiers:
 .( parameter-declaration-clause ) cv-qualifier-seq'
   ref-qualifier' noexcept-specifier' attribute-specifier-seq'
```

int Meow (. 2);

```
parameters-and-qualifiers:
  (.parameter-declaration-clause ) cv-qualifier-seq'
   ref-qualifier' noexcept-specifier' attribute-specifier-seq'
```

int Meow (. 2);

```
parameters-and-qualifiers:
  (.parameter-declaration-clause ) cv-qualifier-seq'
   ref-qualifier' noexcept-specifier' attribute-specifier-seq'
```

parameter-declaration-clause -> SyntaxError

1. **decl-specifier-seq declarator-id (expression-list) ;**

```
int Meow( 2 ); // variable declaration
```

2. **decl-specifier-seq declarator-id (parameter-declaration-clause) ;**

```
int Meow(int); // function declaration
```

Grammars

Grammars

Regular

```
\#([a-fA-F]|[0-9]){3, 6}
```

Grammars

Regular

```
\#([a-fA-F]|[0-9]){3, 6}
```

Context-Free

```
C++, C#, Java, ....
```

Grammars

Regular

```
\#([a-fA-F]|[0-9]){3, 6}
```

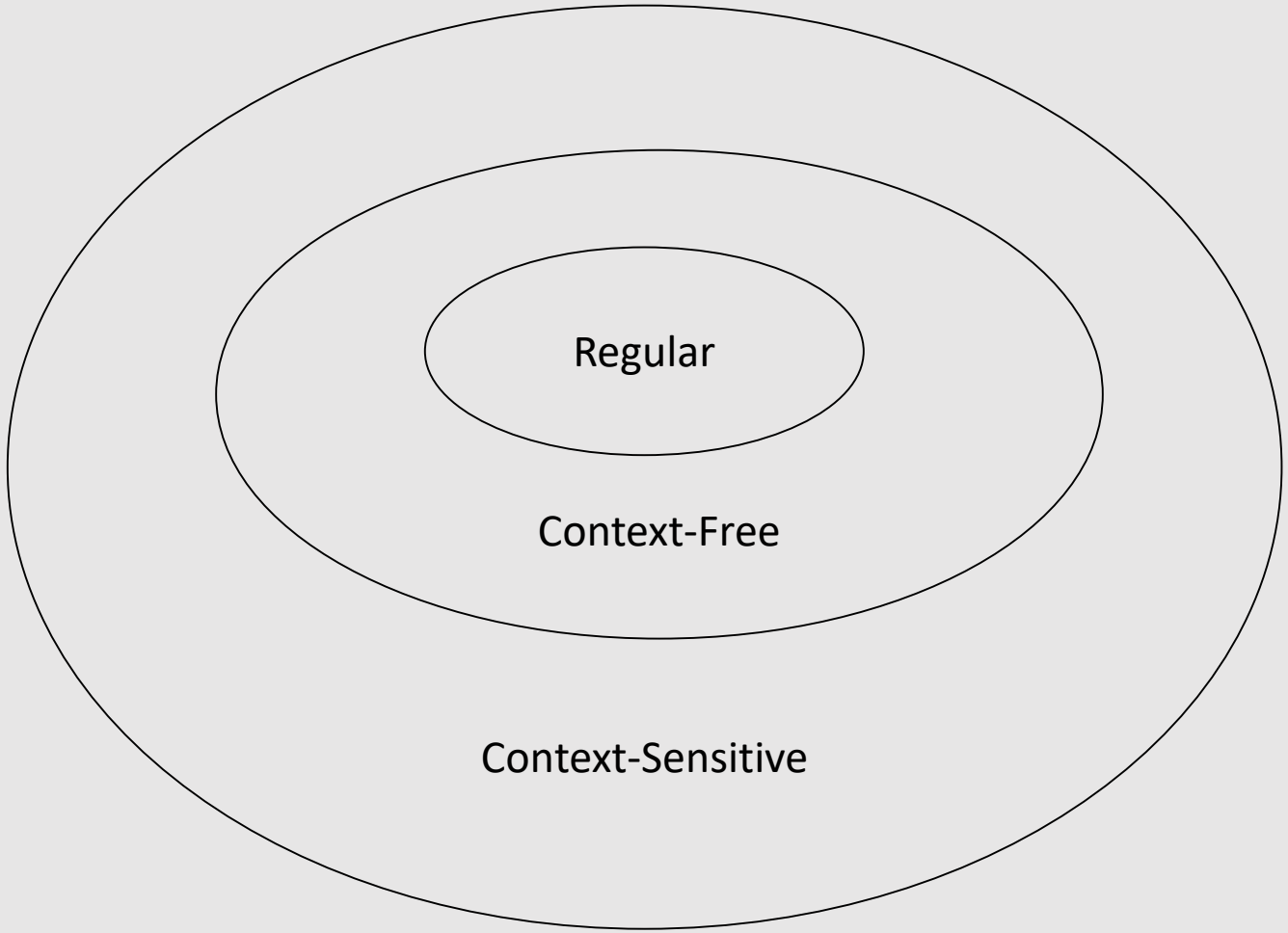
Context-Free

C++, C#, Java, ...

Context-Sensitive

Bach language (aabccb, baabcaccb), palindromes

Grammars



Grammars

Context-Free

C++, C#, Java, ...

Context-Free Parsers

Top-Down

Bottom-Up

Context-Free Parsers

Top-Down

Bottom-Up

LL(1) — Generated LL table

Recursive descent — Handwritten

Context-Free Parsers

Top-Down	
LL(1)	Generated LL table
Recursive descent	Handwritten

Bottom-Up	
SLR(K)	Generated (big LR table)
LR(1)	Generated (big LR table)
LALR(1)	Generated (small LALR table)

Context-Free Parsers

Top-Down		Bottom-Up	
LL(1)	Generated LL table	SLR(K)	Generated (big LR table)
Recursive descent	Handwritten	LR(1)	Generated (big LR table)
LL(*)	Handwritten, $O(n^3)$	LALR(1)	Generated (small LALR table)

Context-Free Parsers

Recursive descent

`Handwritten`

$\forall G \quad A \rightarrow a \mid b$

Context-Free Parsers

Recursive descent

`Handwritten`

$\forall \mathrm{G} \quad \mathrm{A} \rightarrow \mathrm{a} \mid \mathrm{b}$

$(1)\ FIRST(a) \cap FIRST(b) = \emptyset$

Context-Free Parsers

Recursive descent

`Handwritten`

$\forall$ G A $\rightarrow$ a | b

(1) $FIRST(a) \cap FIRST(b) = \emptyset$

(2) e $\in$ FIRST(a) $\Rightarrow$ $FIRST(a) \cap FOLLOW(A) = \emptyset$

Context-Free Parsers

Recursive descent

`Handwritten`

$\forall$ G A $\rightarrow$ a | b

(1) $FIRST(a) \cap FIRST(b) = \emptyset$

(2) e $\in$ FIRST(a) $\Rightarrow$ $FIRST(a) \cap FOLLOW(A) = \emptyset$

There are no overlapping prefixes

There is no left recursion

There is no left recursion

```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```

There is no left recursion

```
// expression:
//   assignment-expression
//   expression , assignment-expression
Tree *Expression()
{
    auto expression = AssignmentExpression();

    while(Match(','))
    {
      auto comma = Consume();
      auto rhs = AssignmentExpression();
      expression = Make<PtreeCommaExpr>(expression, comma, rhs);
    }
    return expression;
}
```

There are no overlapping prefixes

1. **decl-specifier-seq declarator-id (expression-list) ;**
   ```
   int Meow( 2 ); // variable declaration
   ```

2. **decl-specifier-seq declarator-id (parameter-declaration-clause) ;**
   ```
   int Meow(int); // function declaration
   ```

Context-Free Parsers

Recursive descent

`Handwritten`

$\forall$ G A $\rightarrow$ a | b

(1) $FIRST(a) \cap FIRST(b) = \emptyset$

(2) e $\in$ FIRST(a) $\Rightarrow$ $FIRST(a) \cap FOLLOW(A) = \emptyset$

There are no overlapping prefixes

There is no left recursion

Context-Free Parsers

Recursive descent — Handwritten

Predictive

$\forall$ G A $\rightarrow$ a | b

(1) $FIRST(a) \cap FIRST(b) = \emptyset$ — There are no overlapping prefixes

(2) e $\in$ FIRST(a) $\Rightarrow$ $FIRST(a) \cap FOLLOW(A) = \emptyset$ — There is no left recursion

Backtracking

Exponential in some cases

Most vexing parse

```
struct Paws {};
struct Cat  { Cat(...); };

void Meow()
{




}
```

Most vexing parse

```
struct Paws {};
struct Cat  { Cat(...); };

void Meow()
{
    Cat mrr    (Paws);       // function decl
    Cat mrre   (Paws());     // function decl




}
```

Most vexing parse

```
struct Paws {};
struct Cat  { Cat(...); };

void Meow()
{
    Cat mrr    (Paws);        // function decl
    Cat mrre   (Paws());      // function decl
    Cat mrrmeow((Paws()));    // variable decl



}
```

Most vexing parse

```
struct Paws {};
struct Cat  { Cat(...); };

void Meow()
{
    Cat mrr    (Paws);        // function decl
    Cat mrre   (Paws());      // function decl
    Cat mrrmeow((Paws()));    // variable decl

    Cat mrrnmeow(Paws(), Paws(), Paws(),  Paws() ); // function decl
    Cat mrrrmeow(Paws(), Paws(), Paws(), (Paws())); // variable decl
}
```

Most vexing parse

```
struct Paws {};
struct Cat  { Cat(...); };

void Meow()
{
    Cat mrr     (Paws);         // function decl
    Cat mrre    (Paws());       // function decl
    Cat mrrmeow((Paws()));      // variable decl

    Cat mrrnmeow(Paws(), Paws(), Paws(),  Paws() ); // function decl
    Cat mrrrmeow(Paws(), Paws(), Paws(), (Paws())); // variable decl
}
```

ambiguous

disambiguation

So, Backtracks

int Meow (. 2);

```
parameters-and-qualifiers:
  (.parameter-declaration-clause ) cv-qualifier-seq'
   ref-qualifier' noexcept-specifier' attribute-specifier-seq'
```

```
int Meow ( . 2);
```

```
parameters-and-qualifiers:
  (.parameter-declaration-clause ) cv-qualifier-seq'
   ref-qualifier' noexcept-specifier' attribute-specifier-seq'
```

```
int Meow .(2);
```

```
init-declarator:
 declarator .initializer'
```

```
int Meow .(2);
```

```
initializer:
 .brace-or-equal-initializer
  ( expression-list )
```

```
int Meow .(2);
```

```
initializer:
  brace-or-equal-initializer
 .( expression-list )
```

```
int Meow (.2);
```

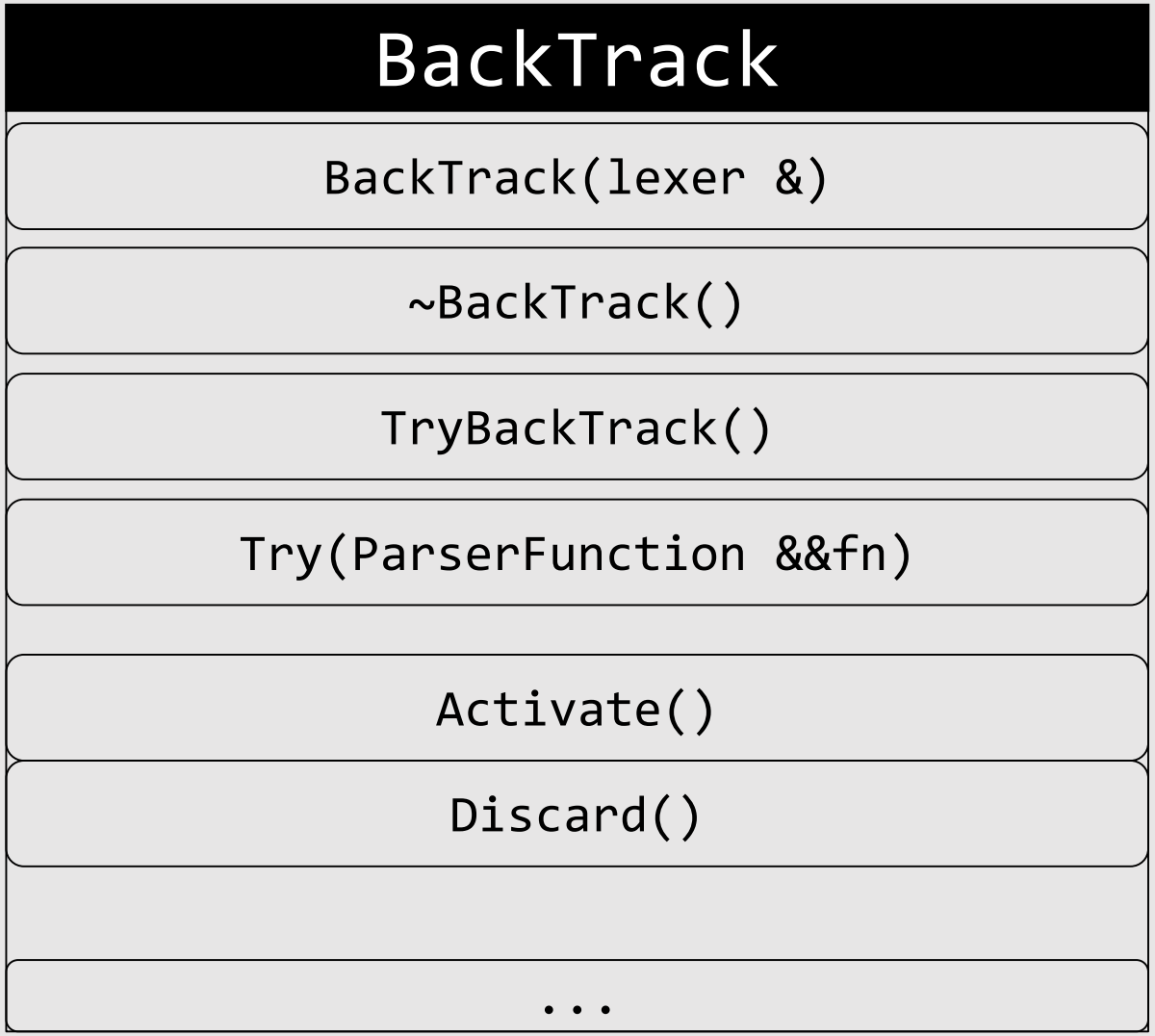
```
initializer:
  brace-or-equal-initializer
  ( .expression-list )
```

int Meow (2.);

```
initializer:
  brace-or-equal-initializer
  ( expression-list .)
```

int Meow (2).;

```
initializer:
  brace-or-equal-initializer
  ( expression-list ).
```



Saves lexer position

Calls TryBackTrack

Backtracks if activated

Invokes fn and tries backtrack

```
// noptr-declarator:
//   ....
//   noptr-declarator parameters-and-qualifiers
Parser::Tree *Parser::NoptrDeclarator()
{
  // ....
  collector.TryAdd(DeclaratorId());
  while(true)
  {
    auto &&backTrack = MakeBackTrack();
    if (auto paramsAndQuals = ParametersAndQualifiers()) // activates backtrack if error
    {
      collector.Add(paramsAndQuals, addOptions);
      continue;
    }
    break;
  } // backTack.TryBackTrack();

  return TreeBuilder().Promote(std::move(collector));
}
```

```
// noptr-declarator:
//   ....
//   noptr-declarator parameters-and-qualifiers
Parser::Tree *Parser::NoptrDeclarator()
{
  // ....
  builder.TryAdd(DeclaratorId());
  while(true)
  {
    if (auto paramsAndQuals = MakeBackTrack().Try(&Parser::ParametersAndQualifiers))
    {
      builder.Add(paramsAndQuals, addOptions);
      continue;
    }
    break;
  }

  return m_treeBuilder.Promote(std::move(builder));
}
```

```
// noptr-declarator:
//   ....
//   noptr-declarator parameters-and-qualifiers
Parser::Tree *Parser::NoptrDeclarator()
{
  // ....
  builder.TryAdd(DeclaratorId());
  while(true)
  {
    if (auto paramsAndQuals = Try(&Parser::ParametersAndQualifiers))
    {
      builder.Add(paramsAndQuals, addOptions);
      continue;
    }
    break;
  }

  return m_treeBuilder.Promote(std::move(builder));
}
```

Profit!

Almost

Almost

Recursive descent

`Handwritten`

Predictive

$\forall$ G A $\rightarrow$ a | b, a $\rightarrow$ e

(1) $FIRST(a) \cap FIRST(b) = \emptyset$ — There is no overlapping prefixes

(2) $FIRST(a) \cap FOLLOW(A) = \emptyset$ — There is no left recursion

Backtracking

Exponential in some cases

Exponential in some cases

```
class NoMempool {};

template<size_t SIZE, class OTHER>
class MempoolList
{
public:
  MempoolList() {}
};
```

Exponential in some cases

```
typedef class MempoolList<  16,
      MempoolList<  24,
      MempoolList<  32,
      MempoolList<  40,
      MempoolList<  64,
      MempoolList<  72,
      ....
      MempoolList< 320,
      MempoolList< 512,
      MempoolList< 704,
      MempoolList<1024,
      MempoolList<2048, NoMempool> > > > .... > > > > > AllPoolsType;
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
      M<  24,
        M<  32,
          M<  40,
            M<  64,
              M<  72,
                M< 320,
                  M< 512,
                    M< 704,
                      M<1024,
                        M<2048, T>
                       >
                     >
                   >
                 >
               >
             >
           >
         >
       >
     > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
      M<  24,
        M<  32,
          M<  40,
            M<  64,
              M<  72,
                M< 320,
                  M< 512,
                    M< 704,
                      M<1024,
                        M<2048, T>
                       >
                     >
                   >
                 >
               >
             >
           >
         >
       >
     > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
      M<  24,
        M<  32,
          M<  40,
            M<  64,
              M<  72,
                M< 320,
                  M< 512,
                    M< 704,
                      M<1024,
                        M<2048, T>
                       >
                     >
                   >
                 >
               >
             >
           >
         >
       >
     > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

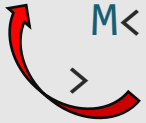
```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```



```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

Exponent

Context-Free Parsers

Top-Down	
LL(1)	Generated LL table
Recursive descent	Handwritten
LL(*)	Handwritten, $O(n^3)$

Bottom-Up	
SLR(K)	Generated (big LR table)
LR(1)	Generated (big LR table)
LALR(1)	Generated (small LALR table)

Context-Free Parsers

Top-Down	
LL(1)	Generated LL table
Recursive descent	Handwritten
LL(*)	Handwritten, $O(n^3)$

Bottom-Up	
SLR(K)	Generated (big LR table)
LR(1)	Generated (big LR table)
LALR(1)	Generated (small LALR table)

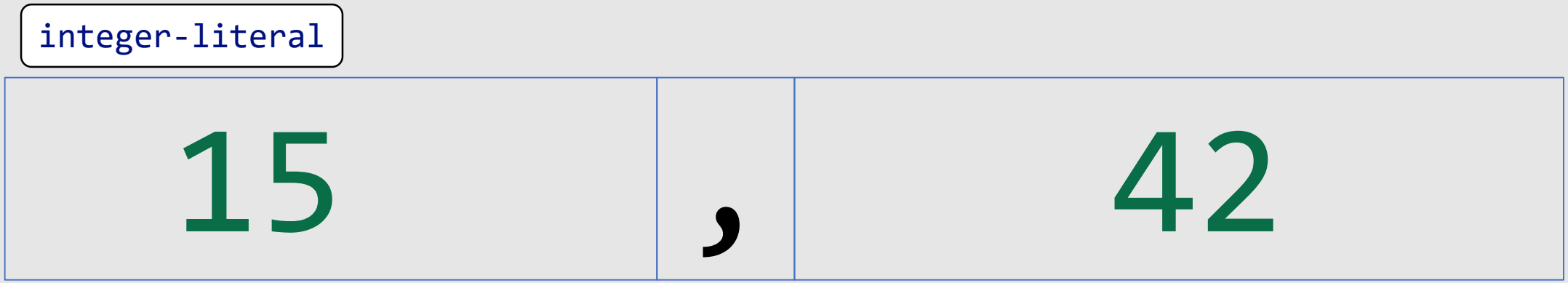
Bottom-Up

15	,	42

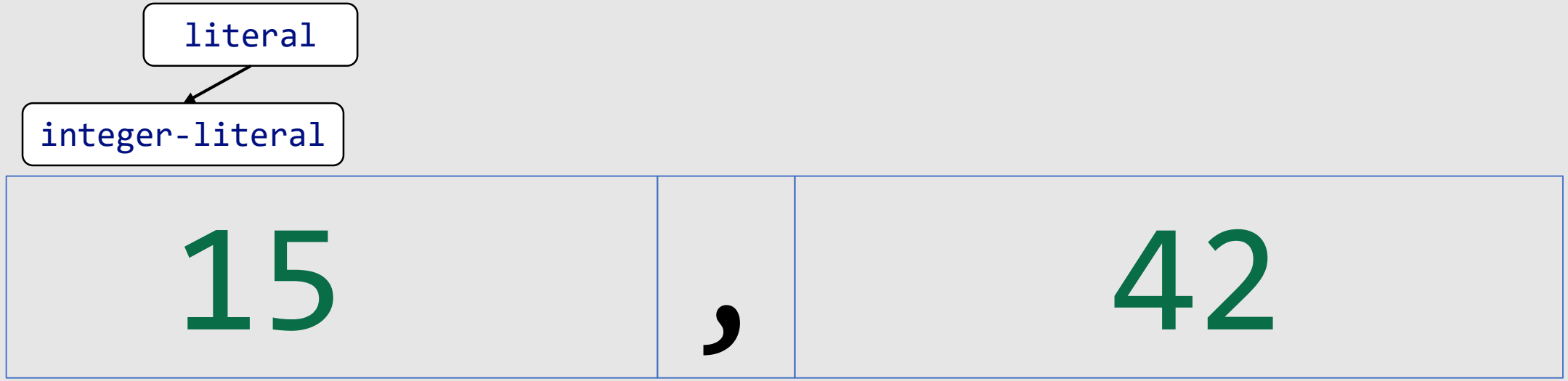
Bottom-Up

15	,	42

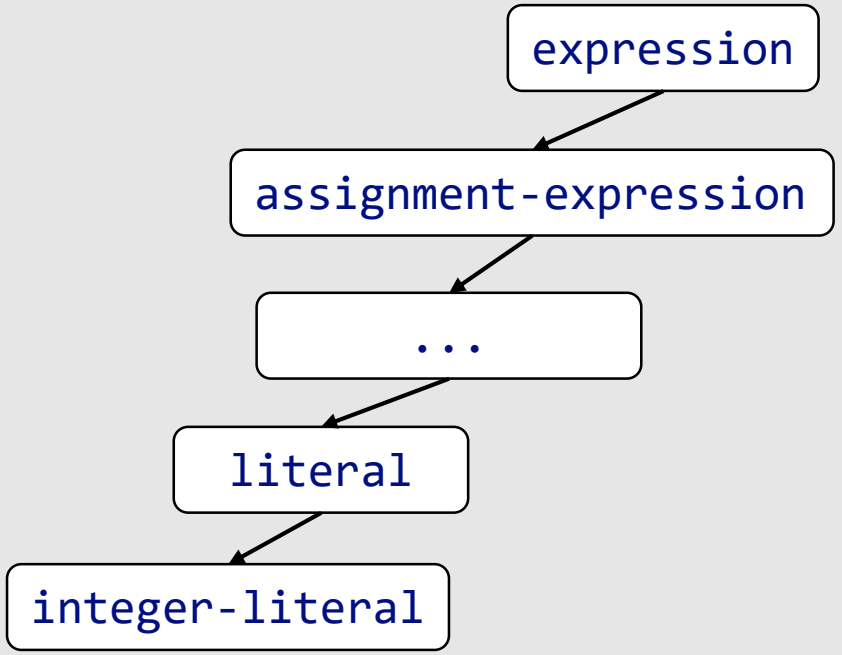
Bottom-Up



Bottom-Up

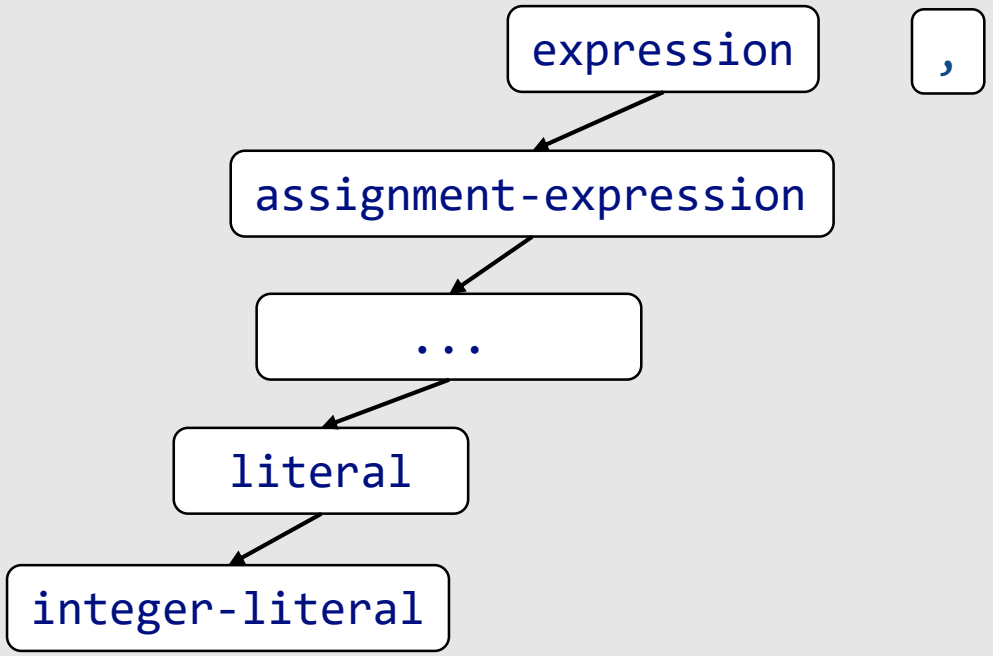


Bottom-Up



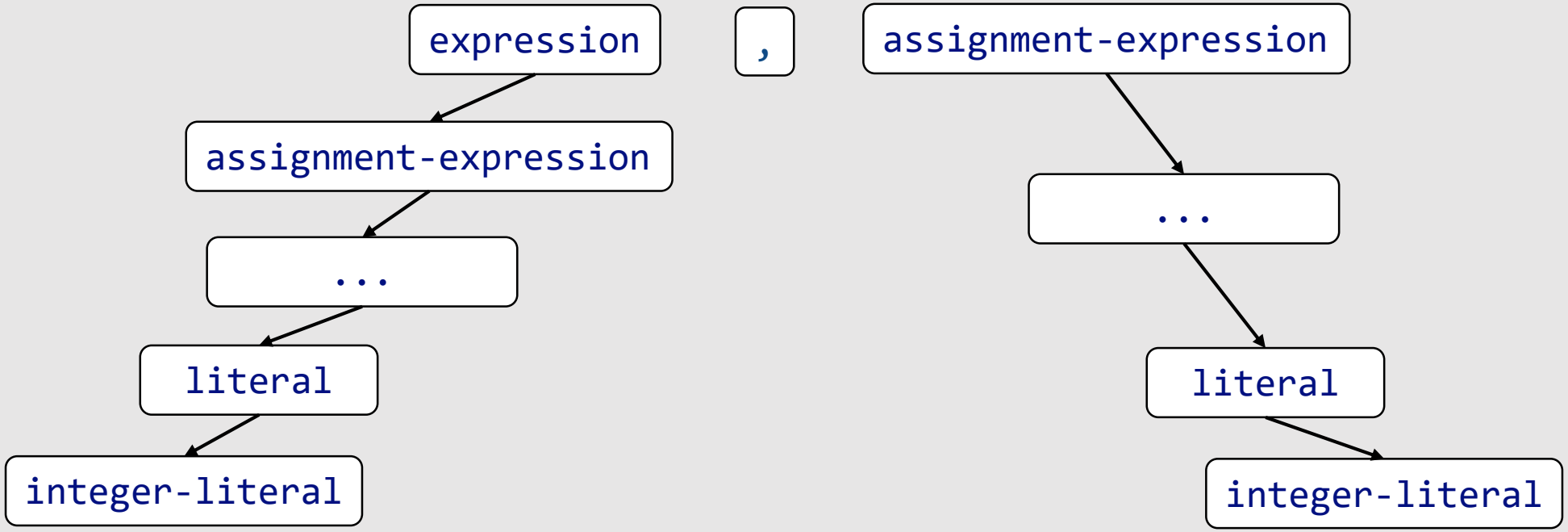
15	,	42

Bottom-Up



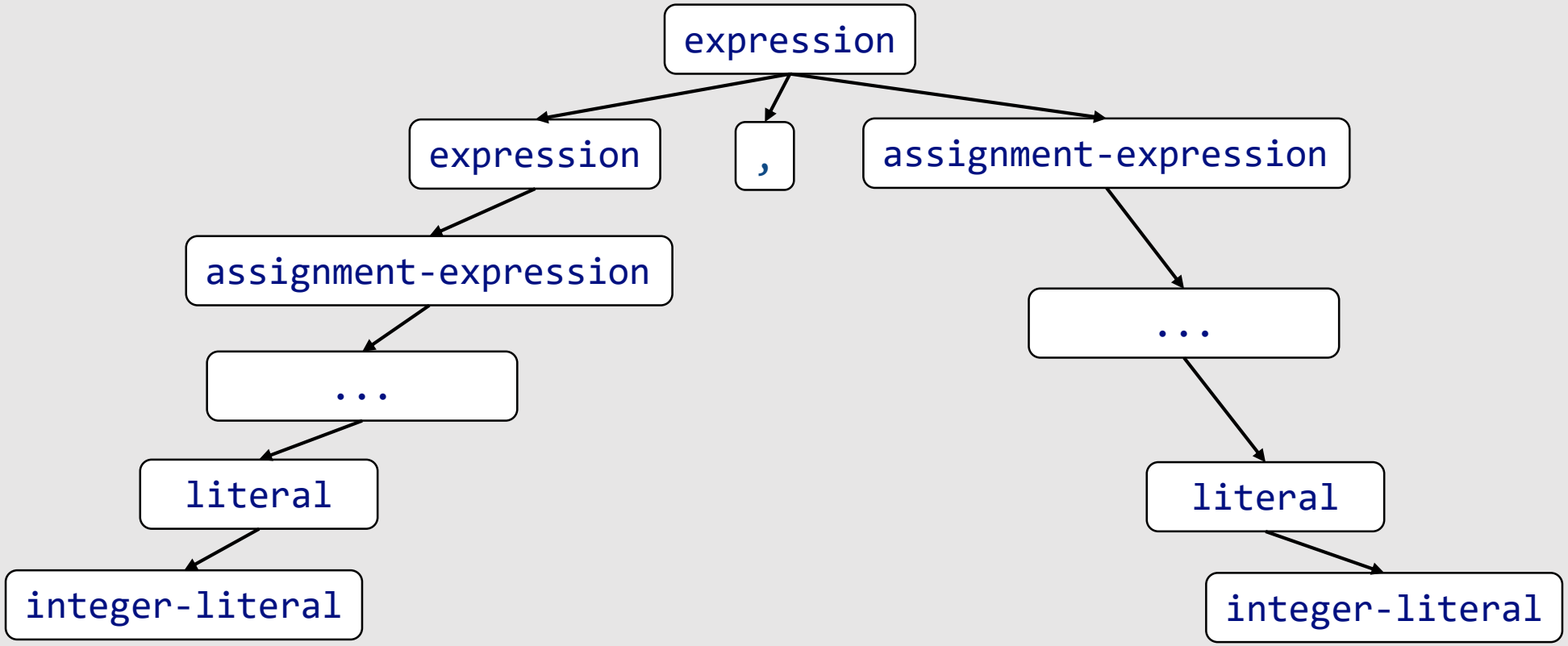
15	,	42

Bottom-Up



15	,	42

Bottom-Up



15	,	42

Bottom-Up

```
E:
  E + T
  E - T
  T
```

```
T:
  T * F
  T / F
  F
```

```
F:
  (E)
  id
```

Input: id * id

Bottom-Up

```
E:
  E + T
  E - T
  T
```

```
T:
  T * F
  T / F
  F
```

```
F:
  (E)
  id
```

Input: id * id

LL: E => T => T * F => F * F => id * F => id * id

Bottom-Up

```
E:
  E + T
  E - T
  T
```

```
T:
  T * F
  T / F
  F
```

```
F:
  (E)
  id
```

Input: id * id

LL: E => T => T * F => F * F => id * F => id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

Bottom-Up

```
E:
  E + T
  E - T
  T
```

```
T:
  T * F
  T / F
  F
```

```
F:
  (E)
  id
```

Input: id * id

LL: E => T => T * F => F * F => id * F => id * id

LR: E => T => T * F => T * id => F * id => id * id

Bottom-Up

```
E:
  E + T
  E - T
  T
```

```
T:
  T * F
  T / F
  F
```

```
F:
  (E)
  id
```

Input: id * id

LL: E => T => T * F => F * F => id * F => id * id

LR: E => T => T * F => T * id => F * id => id * id

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
  E + T
  E - T
  T

T:
  T * F
  T / F
  F

F:
  (E)
  id
```

#	Sentence form	Base	Reduce Action
1	$id_1 * id_2$		

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
  E + T
  E - T
  T

T:
  T * F
  T / F
  F

F:
  (E)
  id
```

#	Sentence form	Base	Reduce Action
1	id_1 * id_2	id_1	

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
   E + T
   E - T
   T

T:
   T * F
   T / F
   F

F:
   (E)
   id
```

#	Sentence form	Base	Reduce Action
1	id_1 * id_2	id_1	F => id
2	F * id_2	F	

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
   E + T
   E - T
   T

T:
   T * F
   T / F
   F

F:
   (E)
   id
```

#	Sentence form	Base	Reduce Action
1	id_1 * id_2	id_1	F => id
2	F * id_2	F	T => F

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
   E + T
   E - T
   T

T:
   T * F
   T / F
   F

F:
   (E)
   id
```

#	Sentence form	Base	Reduce Action
1	id_1 * id_2	id_1	F => id
2	F * id_2	F	T => F
3	T * id_2	id_2	F => id

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
  E + T
  E - T
  T
```

```
T:
  T * F
  T / F
  F
```

```
F:
  (E)
  id
```

#	Sentence form	Base	Reduce Action
1	id_1 * id_2	id_1	F => id
2	F * id_2	F	T => F
3	T * id_2	id_2	F => id
4	T * F	T * F	T => T * F

Input: id * id

LR: id * id <= F * id <= T * id <= T * F <= T <= E

```
E:
  E + T
  E - T
  T

T:
  T * F
  T / F
  F

F:
  (E)
  id
```

#	Sentence form	Base	Reduce Action
1	id_1 * id_2	id_1	F => id
2	F * id_2	F	T => F
3	T * id_2	id_2	F => id
4	T * F	T * F	T => T * F

Let's Join LL and LR

Parser
LookAhead()
Consume()
Match(...)
TryConsume(...)
ParseExpression
ParseAssignmentExpression
ParseAdditiveExpression
...

LRBase

Let's Join LL and LR

Parser
LookAhead()
Consume()
Match(...)
TryConsume(...)
ParseExpression
ParseAssignmentExpression
ParseAdditiveExpression
...

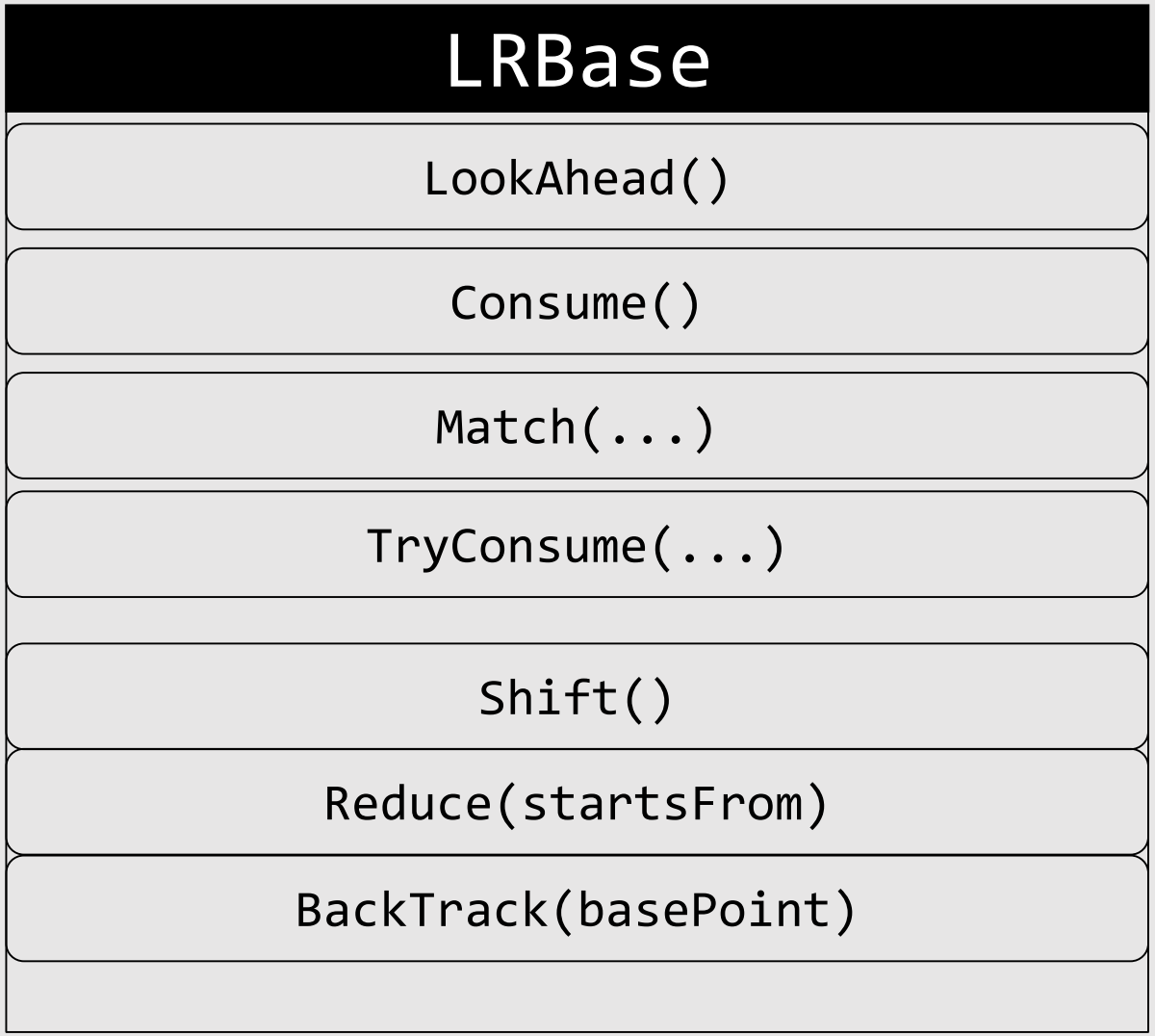
LRBase
LookAhead()
LookAhead()
Match()
TryConsume(...)

Let's Join LL and LR

Parser
LookAhead()
Consume()
Match(...)
TryConsume(...)
ParseExpression
ParseAssignmentExpression
ParseAdditiveExpression
...

LRBase
LookAhead()
LookAhead()
Match()
TryConsume(...)
Shift()
Reduce(startsFrom)
BackTrack(basePoint)

Let's Join LL and LR



Reads the top item from base stack

Pops the top item

Compares arg with top

Match + Consume

Pushes items to the stack

Reduces range of items to non-term

Activates LRBase lexer mode

Products

```
class Parser
{
  // ....
  auto LookAhead(tk token)
  {
    if (m_base.Activated())
    {
      return m_lexer.At(m_base.LexerPosition());
    }
    return m_lexer.At(m_lexer.Position());
  }
  // ....
};
```

Products

```
class Parser
{
  // ....
  auto LookAhead(tk token)
  {
    if (m_base.Activated())
    {
      return m_lexer.At(m_base.LexerPosition());
    }
    return m_lexer.At(m_lexer.Position());
  }
  // ....
};
```

Products

```
class Parser
{
  // ....
  auto Consume()
  {
    if (m_base.Activated())
    {
      auto res = m_base.At(m_base.Position());
      m_base.Advance();
      return res;
    }
    return m_lexer.Advance();
  }
  // ....
};
```

Products

```
class Parser
{
  // ....
  auto Consume()
  {
    if (m_base.Activated())
    {
      auto res = m_base.At(m_base.Position());
      m_base.Advance();
      return res;
    }
    return m_lexer.Advance();
  }
  // ....
};
```

Products

```
enum class Products : std::uint16_t
{
  PrimaryExpression,
  IdExpression,
  UnqualifiedId,
  QualifiedId,
  NestedNameSpecifier,
  ....
  AndExpression,
  ExclusiveExpression,
  InclusiveExpression,
  LogicalAndExpression,
  LogicalOrExpression,
  ThrowExpression,
  AssignmentExpression,
  AssignmentOperator,
  Expression,
  // ....
}
```

```
class Parser
{
  // ....
  template<Products Product>
  Tree *ParseImpl() = delete;
  // ....
};
```

Products

```
template<> Tree *ParseImpl<Products::PrimaryExpression>();
template<> Tree *ParseImpl<Products::IdExpression>();
template<> Tree *ParseImpl<Products::UnqualifiedId>();
template<> Tree *ParseImpl<Products::QualifiedId>();
template<> Tree *ParseImpl<Products::NestedNameSpecifier>();
template<> Tree *ParseImpl<Products::LambdaExpression>();
template<> Tree *ParseImpl<Products::LambdaIntroducer>();
template<> Tree *ParseImpl<Products::LambdaDeclarator>();
template<> Tree *ParseImpl<Products::LambdaSpecifier>();
template<> Tree *ParseImpl<Products::LambdaSpecifierSeq>();
template<> Tree *ParseImpl<Products::LambdaCapture>();
template<> Tree *ParseImpl<Products::CaptureDefault>();
template<> Tree *ParseImpl<Products::CaptureList>();
template<> Tree *ParseImpl<Products::Capture>();
template<> Tree *ParseImpl<Products::SimpleCapture>();
template<> Tree *ParseImpl<Products::InitCapture>();
template<> Tree *ParseImpl<Products::FoldExpression>();
```

Products

```
class Parser
{
  // ....
  template<Products Product>
  ParsingResult Parse()
  {
    if (auto restored = LRBase().TryConsume(Product))
    {
      return restored;
    }
    // ....
  }
  // ....
};
```

Products

```
// ....
auto lrPos  = LRBase().LexerPosition();
auto lexPos = LexerPosition();

if (auto res = ParseImpl<Product>()) // <==
{
  if (IsBackTrackActivated())
  {
    return res;
  }

  Solver().Reduce(Product, res, lexPos, lrPos);
  return res;
}

return nullptr;
// ....
```

Products

```
class Parser
{
  // ....
  template<Products Product>
  ParsingResult TryParse()
  {
    // ....
    if (bt->Activated())
    {
      bt->DoBackTrack();
      Solver().BackTrackTo(pos);
      res = nullptr;
    }
 }
  // ....
};
```

```
typedef
class M<  16,
      M<  24,
        M<  32,
          M<  40,
            M<  64,
              M<  72,
                M< 320,
                  M< 512,
                    M< 704,
                      M<1024,
                        M<2048, T>
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          M<2048, T>
                          >
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        M<1024,
                          simple-template-id
                        >
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
        M<  24,
          M<  32,
            M<  40,
              M<  64,
                M<  72,
                  M< 320,
                    M< 512,
                      M< 704,
                        simple-template-id
                      >
                    >
                  >
                >
              >
            >
          >
        >
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class M<  16,
      simple-template-id
      > APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class simple-template-id APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

```
typedef
class simple-template-id APT;
```

```
simple-type-specifier:
  nested-name-specifier' type-name
  nested-name-specifier' simple-template-id
  nested-name-specifier' template-name

nested-name-specifier:
  type-name ::
  namespace-name ::
  simple-template-id ::
```

Q&A