Компиляторы: синтаксический сахар и вязкая семантика

Михалкович С.С., мехмат

Современные компиляторы

- Относительная простота разработки
- DSL-языки (Domain Specific Language)
- Программист может быстро сделать компилятор «под себя»
- Последнее время – взрывное развитие новых языков и улучшение уже существующих
- Взаимопроникновение возможностей языков программирования

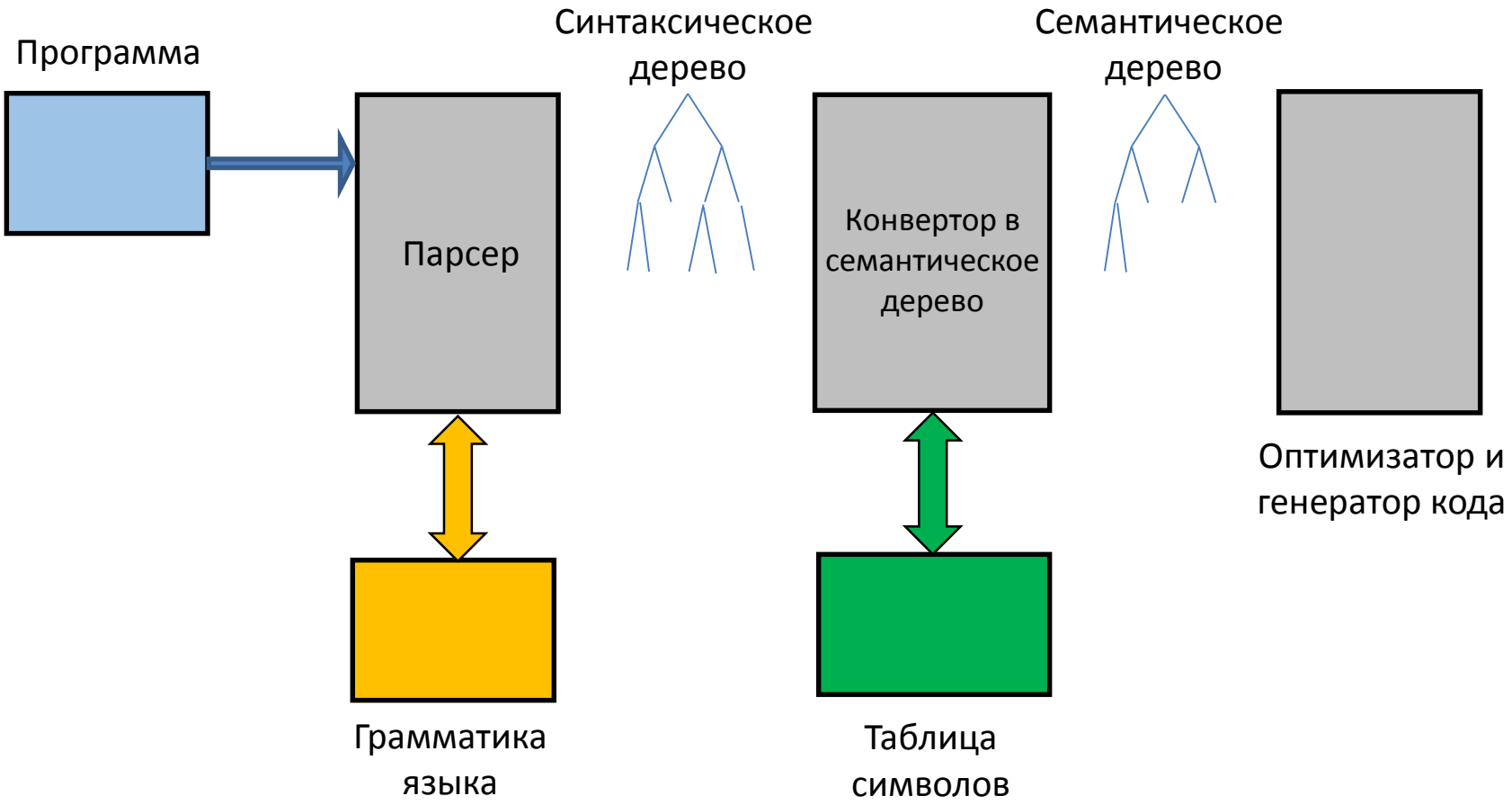
Что такое синтаксис и семантика

- Синтаксис – это правила записи конструкций языка
- Синтаксис определяется грамматикой
- Синтаксис не содержит смысла конструкций
- Семантика – смысл конструкций языка
- Есть несколько разных употреблений слова Семантика
- Для компилятора семантика – это смысл конструкции, необходимый для генерации кода

Этапы построения компилятора

- Разработка грамматики (формы Бэкуса-Наура, контекстно-свободные грамматики)
- Определение узлов синтаксического дерева
- Преобразование программы в синтаксическое дерево (парсер)
- Определение узлов семантического дерева
- Семантические проверки
- Преобразование синтаксического дерева в семантическое дерево
- Оптимизация
- Генерация кода

Архитектура компилятора



Синтаксическое и семантическое деревья

- Синтаксическое дерево существенно более просто по структуре
- Семантическое дерево по структуре может сильно отличаться от синтаксического
- Синтаксически сахарные конструкции отсутствуют в семантическом дереве
- Синтаксическое дерево гораздо проще сгенерировать, чем семантическое
- Деление на синтаксические и семантические деревья присутствует в инфраструктуре многих компиляторов (например, Roslyn для C#)

Определение грамматики и генерация синтаксического дерева

Часть файла .yacc определения парсера

```
assignment
  : var_reference assign_operator expr
  {
    $$ = new assign($1 as addressed_value, $3, $2.type, @$);
  }
  | tkRoundOpen variable tkComma variable_list tkRoundClose assign_operator expr
  {
    if ($6.type != Operators.Assignment)
    parsertools.AddErrorFromResource("ONLY_BASE_ASSIGNMENT_FOR_TUPLE",@6);
    ($4 as addressed_value_list).variables.Insert(0,$2 as addressed_value);
    ($4 as addressed_value_list).source_context = LexLocation.MergeAll(@2,@3,@4);
    $$ = new assign_tuple($4 as addressed_value_list, $7, @$);
  }
  ;

if_stmt
  : tkIf expr_l1 tkThen unlabelled_stmt
  {
    $$ = new if_node($2, $4 as statement, null, @$);
  }
  | tkIf expr_l1 tkThen unlabelled_stmt tkElse unlabelled_stmt
  {
    $$ = new if_node($2, $4 as statement, $6 as statement, @$);
  }
  ;
```

Семантика – примеры ошибок

Семантические ошибки

```
var
  a,b,a: integer;
  d: integer;
  b: boolean;
  r: real;
begin
  d := r;
  foreach var x in d do
    Print(x);
end.
```

Проблема расширения языка

- Язык уже создан, но в следующих версиях хочется язык расширить (пользователи требуют)
- Как это сделать безболезненно (более или менее)?
- Новые библиотеки – это не расширение языка
- Новые синтаксические конструкции упрощают жизнь программиста
- Новые синтаксические конструкции хорошо бы сводить к старым – известным

Что такое синтаксический сахар

- Синтаксический сахар – это синтаксические конструкции языка, которые строятся на базе других, уже реализованных, синтаксических конструкций
- Иногда синтаксический сахар опирается на библиотечные функции или классы
- Для реализации большинства синтаксически сахарных конструкций необходимы семантические проверки

Почему – «вязкая семантика»?

- Семантические проверки при реализации синтаксического сахара делаются не в одной точке, а «размазаны» по этапам компиляции
- Если семантическую проверку удалось сделать на раннем этапе – это удача
- Большинство семантических проверок откладывается до самого позднего этапа генерации семантического дерева
- Одна и та же семантическая ошибка может проверяться на разных уровнях. На последних уровнях диагностика ошибок лучше.
- Итак – семантическая ошибка «вязнет» в семантических проверках. Если пропустить семантическую проверку на раннем уровне, надо чтобы ошибка «увязла» на позднем уровне.

О вреде сахара

О вреде сахара

Полная ерунда!

Примеры синтаксического сахара

Тип последовательности sequence of T

- sequence of T – синоним интерфейса System.Collections.Generic.IEnumerable<T>
- При обработке синтаксического узла sequence_type он переводится в другой синтаксичекий узел template_type_reference, который затем и обрабатывается
- Семантические проверки отсутствуют!!! (повезло)

```
public override void visit(SyntaxTree.sequence_type _sequence_type)
{
    // SSM 11/05/15 sugared node
    var l = new List<ident>();
    l.Add(new ident("?System"));
    l.Add(new ident("Collections"));
    l.Add(new ident("Generic"));
    l.Add(new ident("IEnumerable"));
    var tr = new template_type_reference(new named_type_reference(l),
        new template_param_list(_sequence_type.elements_type, _sequence_type.elements_type.source_context),
        _sequence_type.source_context);
    visit(tr);
}
```

Конструкция i++

Синтаксический сахар

```
var i := 5;
begin
  i++
end.
```

После desugaring

```
var i := 5;
begin
  i += 1
end.
```

- Семантическая проверка: **i должно быть целым**
- Алгоритм: генерация синтаксического поддерева i += 1 и по нему – генерация семантического поддерева
- Когда её сделать? Перед генерацией семантического узла i += 1
- Что будет если её не сделать?
- Важность ранней диагностики ошибок

Синтаксический сахар средствами языка

- Ряд синтаксических конструкций реализуется средствами языка

Добавление элемента к списку

```
var l := new List<integer>;
begin
  l.Add(777)
end.
```

Перегрузка операции и метод расширения

```
function List<T>.operator+=
  (var Self: List<T>; x: T): List<T>;
begin
  Self.Add(x);
  Result := Self;
end;

var l := new List<integer>;
begin
  l += 777
end.
```

Лирическое отступление. Когда проверяется семантика?

- Семантика может определяться синтаксисом (парадокс!)
- Легковесную семантику можно накапливать при синтаксическом разборе (например, имена и пространства имен)
- Сложная семантика проверяется в последний момент – при генерации семантического узла нужной конструкции. В этот момент – известно ВСЁ. Но код генерации семантики – тяжеловесный, и добавление семантических проверок еще его утяжеляет.
- На семантические проверки может уходить 50-80% этапа генерации семантического дерева.

for через while

Цикл for

```
for var i:=1 to 10 do
  operator
```

Замена на while

```
begin
  var i := 1;
  var t := 10;
  // semantic_check("eq_types",t,i)
  while i<>t do
  begin
    operator
    Inc(i)
  end;
end;
```

- Можно заменить for на while прямо в момент генерации синтаксического дерева
- В определенную точку надо вставить синтаксический узел semantic_check, проверяющий равенство типов i и t
- При обходе узла semantic_check в процессе построения семантического дерева типы i и t уже известны. Поэтому осуществляется нужная проверка и при необходимости генерируется ошибка
- Лишние begin-end можно убрать на этапе когда синтаксическое дерево будет полностью построено – до начала построения семантического дерева

Автовывод типа

Описание с автовыводом типа

```
var a := 1;
```

Desugaring

```
var a: same_type(1) := 1;
```

- Можно это сделать на этапе построения синтаксического дерева
- Необходимо использовать синтаксический узел same_type, вычисляющий тип на этапе построения семантического дерева

Правило в парсере

```
vardef
  : tkVar ident tkColon maybetype tkAssign expr
    {
      if ($4 == null)
        $4 = new same_type($6);
      $$ = new var_def_statement($2,$4,$6);
    }

maybetype
  : type { $$ = $1; }
  |      { $$ = null; }
  ;
```

```
public override void visit(SyntaxTree.same_type_node st) // SSM 22/06/13
{
    expression_node en = convert_strong(st.ex);
    if (en == null)
        AddError(get_location(st.ex),"CANNOT_EVALUATE_FUNCTION_TYPE");
    return_value(en.type);
}
```

Короткие определения функций

Сахарная конструкция

```
function Hypoth(a,b: real) :=
  Sqrt(a*a + b*b);
```

Desugaring

```
function Hypoth(a,b: real):
  same_type(Sqrt(a*a + b*b));
begin
  Result := Sqrt(a*a + b*b);
end;
```

```
// SSM 20.07.13 если это - узел с коротким определением функции без типа возвращаемого значения,
var fh = (_procedure_definition.proc_header as SyntaxTree.function_header);
if (fh != null && fh.return_type == null)
{
    var bl = _procedure_definition.proc_body as SyntaxTree.block;
    if (bl != null && bl.program_code != null)
    {
        var ass = bl.program_code.subnodes[0] as SyntaxTree.assign;
        if (ass != null)
        {
            var typ = SyntaxTreeBuilder.BuildSameType(ass.from);
            fh.return_type = typ;
        }
    }
}
//\ SSM
```

Лямбда-выражения

Сахарная конструкция

```
var f: real->real;
f := x->x*x;
Println(f(5));
```

Desugaring

```
function Anonym#1(x: real): real;
begin
  Result := x*x;
end;

...

var f: real->real;
f := Anonym#1;
Println(f(5));
```

Лямбда-выражения: захват

Сахарная конструкция

```
var f: real->real;
var a := 2;
f := x->x*a; // a захватывается
a := 3;
Println(f(5));
```

Desugaring

```
type AnonClass#1 = class
  a := 2;
  function Anonym#1(x: real): real;
  begin
    Result := x*a;
  end;
end;
...
var f: real->real;
var #c1 := new AnonClass#1;
f := #c1.Anonym#1;
#c1.a := 3;
Println(f(5));
```

Кортежное присваивание

Сахарная конструкция

```
var a := 5;
var b := 3;
(a,b) := (b,a);
```

Desugaring

```
var a := 5;
var b := 3;
var #t1 := System.Tuple.Create(b,a);
a := #t1[0];
b := #t1[1];
```

Конструкция (b,a) превращается в кортеж Tuple

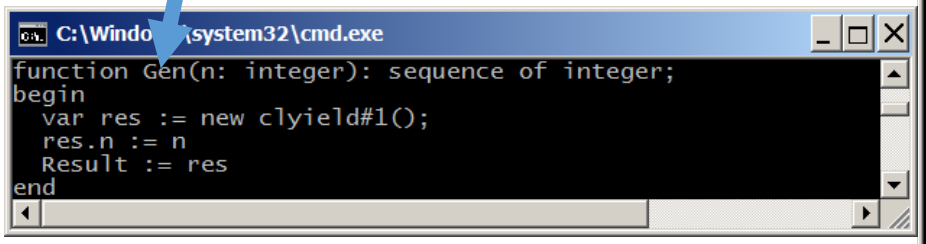
Присваивание (a,b) := (b,a) распадается на пару присваиваний

```
public override void visit(SyntaxTree.assign_tuple asstup)
{
    // Проверить, что справа - Tuple
    var expr = convert_strong(asstup.expr);

    var n = asstup.vars.variables.Count();
    if (n > t.GetGenericArguments().Count())
        AddError(get_location(asstup.vars), "TOO_MANY_ELEMENTS_ON_LEFT_SIDE_OF_TUPLE_ASSIGNMRNT");

    var tname = "#temp_var" + UniqueNumStr();

    var tt = new var_statement(new ident(tname), new semantic_addr_value(expr)); // тут semantic_ad

    var st = new statement_list(tt);
    for (var i = 0; i < n; i++)
    {
        var a = new assign(asstup.vars.variables[i], new dot_node(new ident(tname),
            new ident("Item" + (i + 1).ToString())), Operators.Assignment,
            asstup.vars.variables[i].source_context);
        st.Add(a);
    }
    visit(st);
}
```

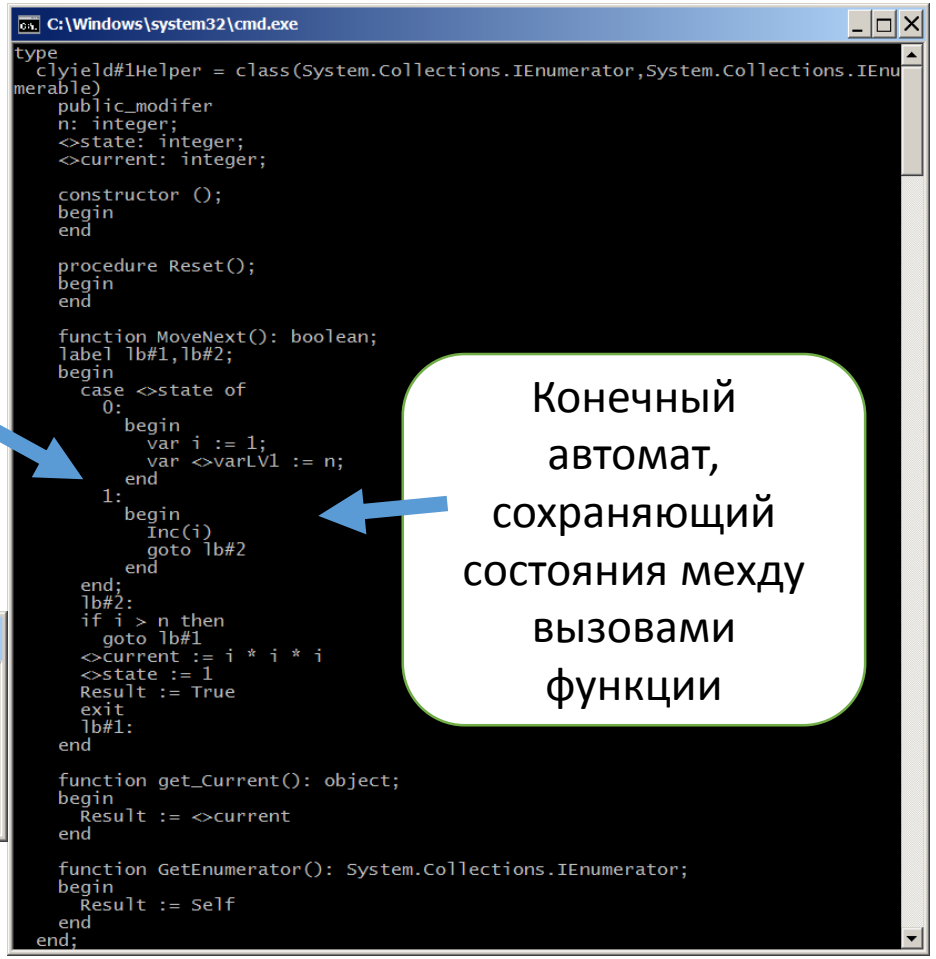
Конструкция yield. Взрыв синтаксиса

Сахарная конструкция

```
function Gen(n: integer):
  sequence of integer;
begin
  for var i := 1 to n do
    yield i*i*i;
end;

begin
  foreach var x in Gen(10) do
    Print(x);
end.
```



Desugaring



Конечный автомат, сохраняющий состояния между вызовами функции

Компиляторы: синтаксический сахар и вязкая семантика