Проблемы реализации синтаксически сахарных конструкций в компиляторах

Михалкович С.С., мехмат ЮФУ

Синтаксически сахарные конструкции в языке программирования

- Синтаксический сахар – синтаксическая конструкция в языке программирования, дублирующая существующую
- Переводится компилятором в конструкции базового языка
- Имеет простую синтаксическую форму
- Интуитивно очевиден, упрощает восприятие
- Может скрывать сложную реализацию
- Требует семантических проверок, отсутствующих в базовом языке
- Современные языки имеют тенденцию к расширению средствами синтаксического сахара

Аналогичные системы и решения

- TXL – DSL-язык для поддержки анализа и преобразования исходных текстов программ (1985 г.)
- Stratego/XT + SDF – DSL-язык и программные инструменты для трансформации программ (1998 г.)
- Scala – язык программирования для JVM, содержит множество синтаксически сахарных расширений, реализуемых средствами языка (1998 г.)
- SugarJ – язык для Library-based расширений языка Java (2011 г.)
- SoundX – аналог SugarJ для набора языков, имеет улучшенный type-checker на уровне сахарного языка (2015 г.)
- Roslyn – для платформы .NET: C#, VB (2014 г.)

Синтаксический сахар средствами языка PascalABC.NET

Добавление элемента к списку

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

Перегрузка операции += как синтаксический сахар

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

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

Синтаксический сахар средствами языка PascalABC.NET

Добавление элемента к списку

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

Перегрузка операции += как синтаксический сахар

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

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

Пример. 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));
```

Пример. 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 переменной

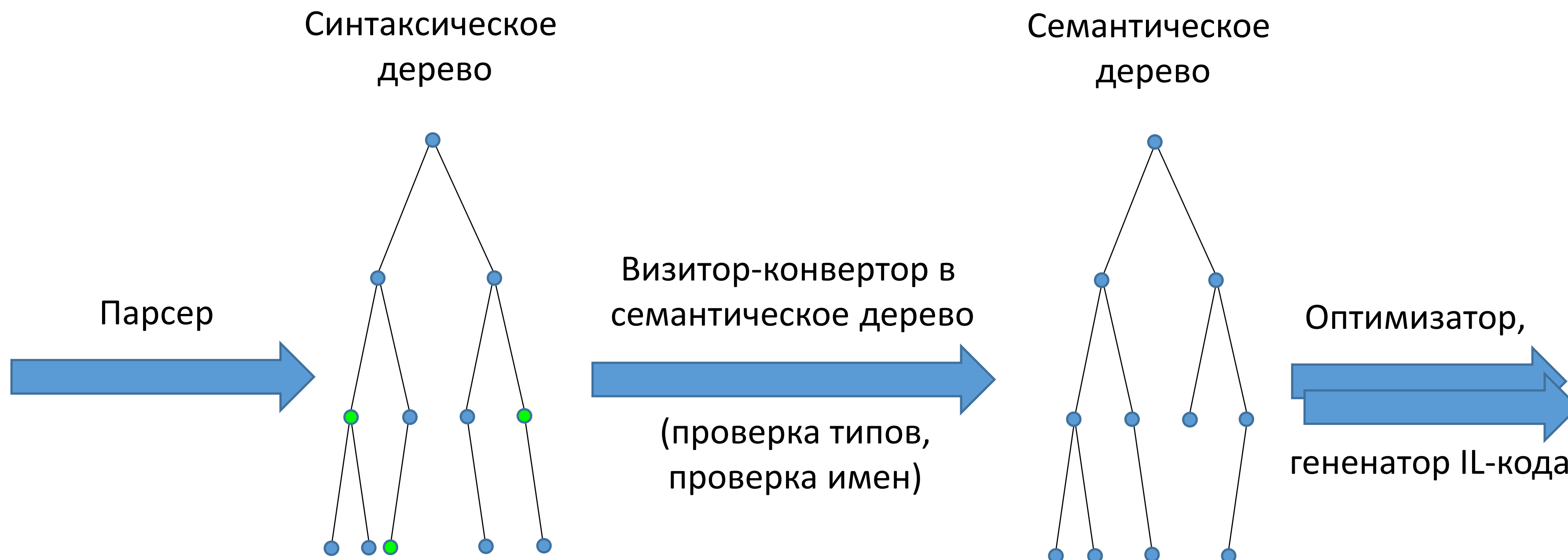
Сахарная конструкция

```
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));
```

Архитектура компилятора PascalABC.NET



- – сахарный узел
- – несахарный узел

На семантике **не остаётся сахарных узлов**

Desugaring – общая схема

- Базовый язык без расширений представлен синтаксическими узлами с типами $t_1, \ldots, t_n$.
- Сахарные расширения языка – синтаксические узлы с типами $s_1, \ldots s_k$.
- Этап **desugaring** (устранение сахара): визитор обрабатывает сахарный узел с типом `s_node` в методе visit по схеме:

```
visit(s_node sug)
{
  var desug = t_node(sug.n1,… sug.nr);
  visit(desug);
}
```

Простой пример: i++

Синтаксический сахар

```
i++;
```

Переводится в

```
i += 1;
```

В абстрактном синтаксисе:

```
inc_node(var)
```

```
assignplus_node(var, int_const(1))
```

- В какой момент делать desugaring-преобразования?
- Необходимы дополнительные семантические проверки: операция ++ не может выполняться для строк и вещественных, а += может
- В какой момент делать семантические проверки?

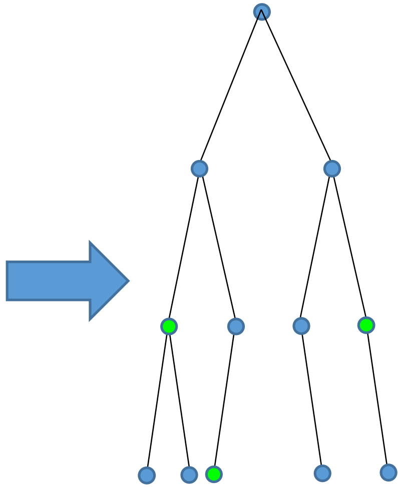
Способ 1. Семантический Desugaring

```
visit(inc_node sug)
{
  IsIntTypeOrError(sug.var.type);
  var desug = assignplus_node(sug.var, int_const(1));
  ReplaceInSynTree(sug, desug);
  visit(desug);
}
```

- Desugaring осуществляется непосредственно перед преобразованием в семантику – в визиторе-конверторе в семантическое дерево
- sug – сахарный узел
- desug – узел базового языка
- IsIntTypeOrError(sug.var.type) – семантическая проверка: тип переменной var в выражении var++ должен быть целым
- ReplaceInSynTree(sug, desug) – Замена в синтаксическом дереве сахарного узла на несахарный на случай повторного обхода синтаксического дерева

Способ 2. Синтаксический desugaring

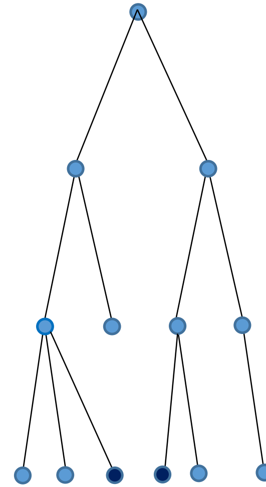
Синтаксическое дерево (сахар)



Desugaring синтаксического дерева

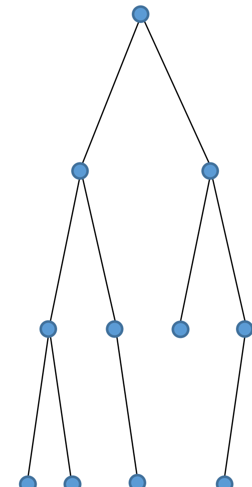
\+ вставка узлов семантических проверок

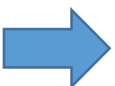
Синтаксическое дерево (без сахара)



Конвертор в семантическое дерево

Семантическое дерево





- – сахарный узел
- – несахарный узел
- – узел семантической проверки

Способ 2. Синтаксический desugaring

```
visit(inc_node sug)
{
  var check = sem_check_statement(sug);
  var desug = assignplus_node(sug.var, int_const(1));
  ReplaceInSynTree(sug, SeqStatements(check,desug));
  visit(sug.var);
}
```

- Desugaring осуществляется в визиторе, преобразующем синтаксическое дерево. Наиболее чистый способ, т.к. синтаксические действия не переносятся на семантический уровень.
- sem_check_statement – конструирование специального узла семантических проверок
- ReplaceInSynTree(sug, desug) – замена в синтаксическом дереве сахарного узла на последовательность узлов, содержащую узел семантических проверок и заменяющий несахарный узел

Пример: кортежное присваивание

Синтаксический сахар

```
var t := (123,'str');
(a,b) := t;
```

Переводится в

```
semantic_check()
var t := (123,'str');
var #t1 := t;
a := #t1.Item1;
b := #t1.Item2;
```

- Используется синтаксический desugaring как более «чистый».
- Семантические проверки – в специальном узле:
  1. #t1 имеет тип Tuple
  2. Кортеж #t1 содержит элементов не меньше, чем количество переменных в левой части
- Если «забыть» сделать проверку 1, то – плохое сообщение об ошибке: «Item1 не объявлен в типе ...». Понятие «вязкой семантики»
- Другие семантические проверки
  - несоответствия типов
  - отсутствия описаний переменных в левой части

делать не надо (!) – это берёт на себя базовый язык.

Срезы. Использование библиотечных функций

Синтаксический сахар

```
var a := Arr(1,2,3,4,5);
var b := a[1:5:2];
```

Переводится в

```
var a := Arr(1,2,3,4,5);
var b := a.SystemSlice(1,5,2);
```

- Используется синтаксический desugaring (способ 2)
- Функция SystemSlice определена в стандартном модуле как метод расширения массива a. SystemSlice – достаточно сложная (50 строк кода на PascalABC.NET). Генерировать код в виде дерева – неэффективно
- SystemSlice не должна содержать сахарных конструкций во избежание зацикливания
- Семантические проверки:
  - a – массив, список или строка
  - параметры слайса – целые
- Особенность: это синтаксический сахар уровня выражения (нельзя заменить синтаксический узел на несколько узлов). Он заменяется специальным узлом, хранящим несахарный узел и алгоритм семантических проверок.

Типы кортежей

Синтаксический сахар

```
var t: (integer,string);
var d: (Mon,Tue,Wed,Thi,Fri);
```

Переводится в

```
var t: Tuple<integer,string>;
var d: (Mon,Tue,Wed,Thi,Fri);
```

- Синтаксис типов кортежей (integer,real) конкурирует с синтаксисом перечислимого типа (Mon,Tue,Wed,Thi,Fri), уже имеющимся в базовом языке Паскаль
- На синтаксическом уровне сложно отличить перечислимый тип от типа кортежа
- Поэтому используется способ 1 семантического desugaringa (синтаксическое дерево строится «на лету» непосредственно в конверторе перевода в семантику)
- При парсинге (T1, T2) генерируется узел enum_or_tuple_type_node, который после семантических проверок на этапе семантики заменяется либо на enum_type_node либо на tuple_type_node

Оператор yield и «взрыв синтаксиса»

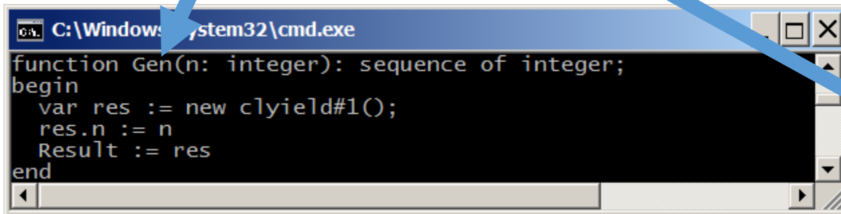
Синтаксический сахар

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

Оператор yield и «взрыв синтаксиса»

Синтаксический сахар

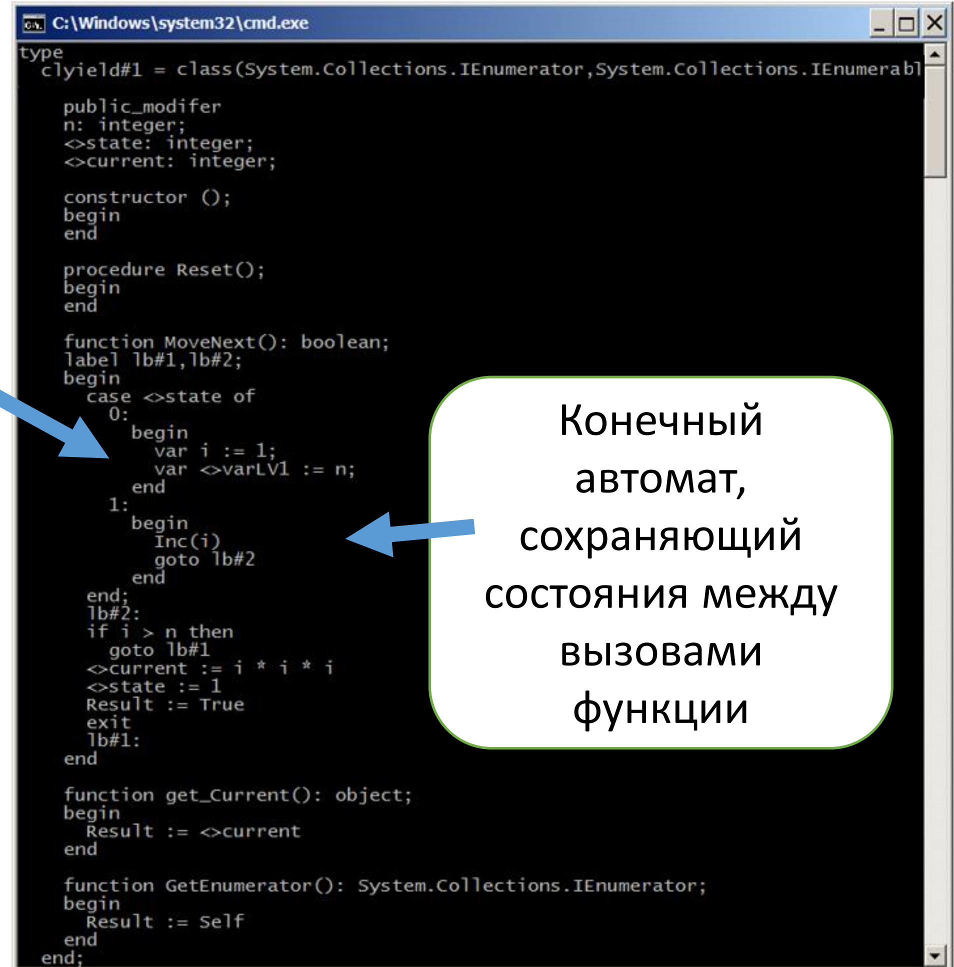
```
function Gen(n: integer):
  sequence of integer;
begin
  for var i:=1 to n do
    yield i * i * i
end;
```



```
function Gen(n: integer): sequence of integer;
begin
  var res := new clyield#1();
  res.n := n
  Result := res
end
```

При desugaringe происходит «взрыв синтаксиса»: генерируются части синтаксического дерева, эквивалентные 70 строкам кода

Переводится в



```
type
  clyield#1 = class(System.Collections.IEnumerator,System.Collections.IEnumerabl

    public_modifer
    n: integer;
    <>state: integer;
    <>current: integer;

    constructor ();
    begin
    end

    procedure Reset();
    begin
    end

    function MoveNext(): boolean;
    label lb#1,lb#2;
    begin
      case <>state of
        0:
          begin
            var i := 1;
            var <>varLV1 := n;
          end
        1:
          begin
            Inc(i)
            goto lb#2
          end
      end;
      lb#2:
      if i > n then
        goto lb#1
      <>current := i * i * i
      <>state := 1
      Result := True
      exit
      lb#1:
    end

    function get_Current(): object;
    begin
      Result := <>current
    end

    function GetEnumerator(): System.Collections.IEnumerator;
    begin
      Result := Self
    end
  end;
```

Оператор yield и взрыв синтаксиса (2)

- Оператор yield реализован с помощью синтаксического desugaring
- Причина: при генерации кода для внешних функций и классов необходимо несколько раз переключать контекст. При семантическом desugaringe делать это гораздо сложнее
- Основная проблема реализации – захват всех переменных в теле функции с yield и их классификация (локальные, глобальные, поля класса, внешние имена). После захвата переменные некоторых видов переименовываются по дереву
- Анализ вида имени производится на синтаксическом уровне специальными визиторами, накапливающими легковесную семантическую информацию об этих именах

Оператор yield и взрыв синтаксиса (3)

- Семантическое действие, которое нельзя выполнить на синтаксическом уровне – автовывод типа переменной

Код в теле функции

```
var a := sin(1) + 1;
```

Переводится в поле класса

```
type #clYield1 = class
    a: auto_type;
```

В теле функции остаётся

```
a := sin(1) + 1;
```

- Тип **auto_type** – семантический, меняется на тип выражения из правой части при первом присваивании.
- Это достигается модификацией кода метода visit(assign_node) визитора-конвертора в семантическое дерево.

Сравнение синтаксического и семантического desugaring

- Синтаксический desugaring – наиболее чистый, основные действия делаются на синтаксическом уровне
- При синтаксическом desugaringe для семантических проверок генерируются специальные проверочные узлы, при обходе которых на этапе семантики осуществляются семантические проверки
- Синтаксический desugaring затруднён или невозможен когда необходимо генерировать различный код в зависимости от семантической информации (например, от вида некоторой переменной: локальная, глобальная, поле)
- При синтаксическом desugaringe может накапливаться некоторая легковесная семантика (имена и пространства имен, вид имени) для последующих преобразований на уровне синтаксиса
- Семантический desugaring значительно затруднён при необходимости переключать контекст (генерация desugared-кода на различных уровнях: локальном, глобальном, уровне класса)
- В обоих случаях необходимо изменять глобальное синтаксическое дерево (по разным причинам)

Спасибо за внимание!

О вреде сахара

О вреде сахара

Полная ерунда!