Software Engineering Conference Russia
October 2017, St. Petersburg

Проект СЛанг: текущее состояние и перспективы

Алексей Канатов, Samsung R&D Center
Евгений Зуев, Innopolis University

Содержание

- Вступление
- Единицы компиляции, контейнеры, общие понятия
- Некоторые операторы – if & loop
- Множественное наследование и валидность обращений к элементам контейнеров
- Отсутствие нулевого указателя и инициализация атрибутов
- Константные объекты
- Основы стандартной библиотеки
- Расширенная перегрузка имен
- Расширения контейнеров
- Множественное переопределение
- Параметризация контейнеров и подпрограмм
- Пример параллельной программы
- Заключение

Вступление

- **Немного об авторах**: C++, Ada, Modula-2, Zonnon, Eiffel – всякой твари по паре ☺
- **О терминах**: ‘feature’ – подпрограмма (действие) или атрибут (данные), атрибут – переменный или константный, подпрограмма – процедура или функция; граф наследования, понятие конформности; модуль, тип, класс
- **Цель:** дать общее представление о языке, его свойствах, отличных от других популярных языков за 30 минут ☺

Виды единиц компиляции

3 вида:

- **Анонимная процедура:** простая последовательность операторов
- **“Отдельностоящая” подпрограмма:** область видимости, формальные параметры, пред- и постусловия, тело - операторы
- **Контейнер (юнит)**: поименованный набор подпрограмм, атрибутов, инвариант, базовые контейнеры
  - Может быть параметризован типом или константным выражением перечислимого типа
  - Контейнер задает тип
  - Контейнер поддерживает множественное наследование (класс)
  - Контейнер поддерживает прямое использование (модуль)

Имя контейнера

```
StandardIO.put("Hello world!\n")
routine(“ha-ha-ha”)

use StandardIO as io
routine(aString: String) is
    io.put("Test!\n")
    c is C("This is a string“)
    io.put(c.string + “ “ + aString)
end

unit C
    string: String
    init (aString: as string) is
        string := aString
    end
end
```

Новое короткое имя контейнера

‘Просто’ процедура

Контейнер

Контейнеры – 3 в 1(класс,модуль,тип)

Использование(модуль)

Клиенты получают доступ к видимым элементам модуля

Наследование (класс)

Контейнер наследует все элементы из базовых контейнеров (классов)

Типизация (тип)

Каждый контейнер определяет тип. Соответственно можно описать атрибут контейнера или локальный атрибут подпрограммы или аргумент, имеющий такой тип

```
StandardIO.put("Hello world!\n")
goo (C)

unit C extend B, ~D use B
end

goo(b: B) use D is
    D.foo
end
unit B is
    foo is
    end
end
```

Использование (модуль)

Наследование(класс)

Типизация (тип)

Использование (модуль)

Базовые примеры

Подпрограммы – процедуры и функции

- `a is ... end` // процедура без параметров;
  // () можно опускать
- `foo: T is ... end` /* функция без параметров;
  возвращает объекты конформные типу T */

Атрибуты контейнеров – переменные или константы

- `v: Type` // переменная
- `const c: Type` // константа

Подпрограммы могут иметь локальные атрибуты, которые также могут быть переменными или константами

- `v is expression` // переменная
- `const c is expression` // константа

Пример контейнера

```
unit X
        const c1: Type is someExpression
        const c2 is someExpression
        v0: Type
        v1: ?Type // v1 не инициализирована явно.
        v2 is someExpression
        v3: Type is someExpression
        foo is
                const localConstant1: Type is someExpression
                const localConstant2 is someExpression
                localVariable1: Type is someExpression
                localVariable2 is someExpression
        end
        init is
                v0 := someExpression // Присваивание
            // c1 := someExpression // Ошибка компиляции
        end
end
x is X; y is X.v0
```

Как скомпилировать программу?

Виды точек входа:

- Анонимная процедура: Первый оператор есть точка входа
- Процедура
- Процедура инициализации некоторого контейнера

Глобальный контекст:

- Все контейнеры верхнего уровня (невложенные) и все отдельностоящие подпрограммы взаимно доступны
- Конфликт имен разрешается вне самого языка – файл описания проекта

```
StandardIO.put("Hello world!\n")
routine ((“ha-ha-ha”))

foo(strings: Array[String]) is
   ...
end

unit C
    init is ... end
end
```

Source 1:

```
foo is end
unit A is
   foo is ... end
end
```

Source 2:

```
goo is ... end
```

Source 3:

```
foo
goo
a is A
a.foo
```

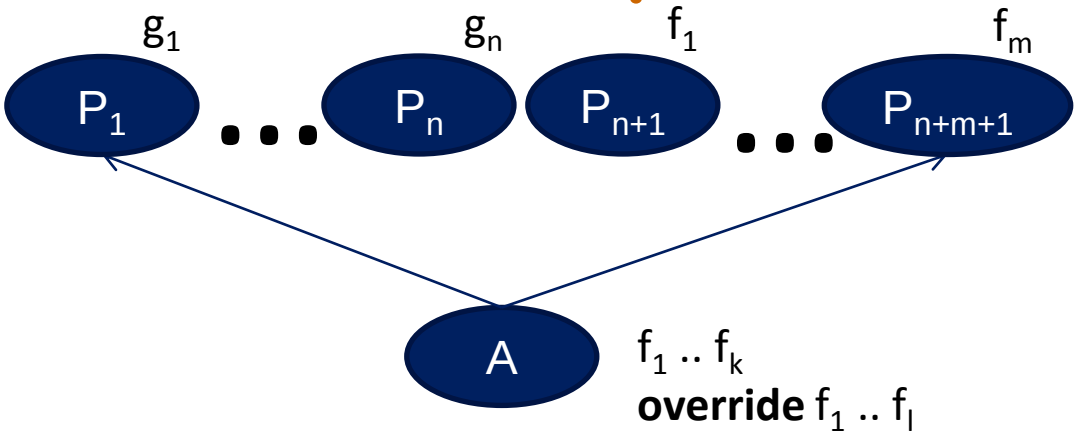
Операторы - if & loop

- Только один условный оператор и один цикл
- Но есть две формы условного оператора
- И 3 формы циклов

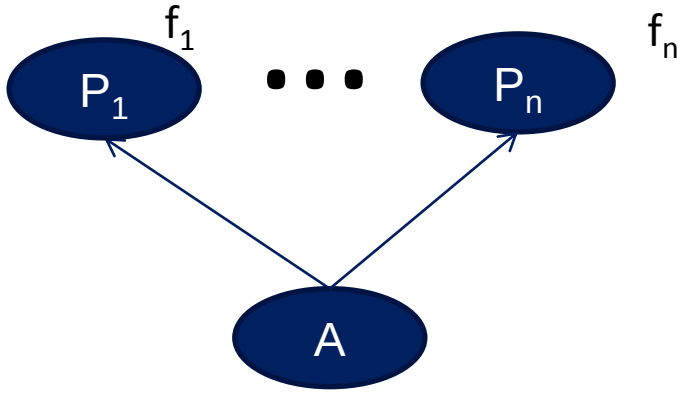
```
if condition then
    thenAction
else
    elseAction
end

if a is
   T1: action1 // где T1 это тип
   E2: action2 // где E2 это выражение
   else action3
end

while index in 1..10 loop // цикл пока, 0+
   body
end

loop
   body
while condition end // цикл пока, 1+

loop // вечный цикл
   body
end
```

Наследование и валидность обращений

- **Переопределение в контейнере:**
  - Если g_i идентично g_j тогда только одна копия g наследуется
  - g_1 .. g_n наследуются как есть
  - f_1 .. f_k впервые определены в A
  - $l \le m$, пусть f_1 .. f_l переопределяют набор f_1 .. f_m исходя из конформности сигнатур, тогда оставшиеся из f_1 .. f_m наследуются как есть



- **Переопределение при наследовании**:
  - f_i переопределяет f_1 .. f_k , где $k < n$, на основании конформности сигнатур
  - Тогда в A будет f_1 .. f_{n-k+1} элементов

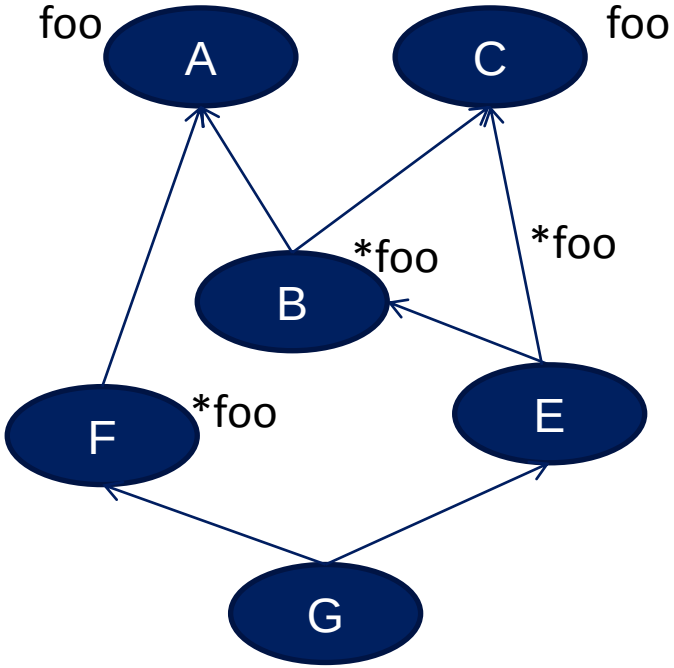


- **Валидность обращения**
  - Обращение валидно, если компилятор может его однозначно интерпретировать!
  - Существует только одна f в A с сигнатурой (T_1..T_n) которой конформен кортеж (ET_1..ET_n)

```
// P1..Pn – base units for A
// E1..En – expressions of types Eti

a is A
a.f(E1, .. En)
// Is it a valid feature call?
```

Наследование и валидность обращений

- Общий подход: множественное наследование с перегрузкой имен, наличием конфликтующих версий, множественным переопределением и проверкой однозначности обращений
- Обязательные проверки для графа наследования:
  - Отсутствие циклов
  - Полиморфные конфликты версий разрешены ‘select’




```
abstract unit A
   foo (T) is abstract
end

unit C
   foo (T) is end
end

unit B extend A, C
   override foo (T) is end
end

unit E extend C, B
   override C.foo
end

unit F extend A
   override foo (T1) is end
end

unit G extend F, E
   use E.foo
end
```

Отсутствие нулевого указателя и инициализация атрибутов

Основные постулаты:

- Каждая сущность должна получить значение до первого обращения к ее подпрограммам или атрибутам
- Если надо определить сущность без значения, то нет возможности обращаться к ее подпрограммам или атрибутам
- Должен быть механизм проверки типа сущности и если тип определен, то можно обращаться к ее элементам
- Сущность, описанная с возможностью не иметь значение, может его потерять
- Опасные присваивания запрещены
- Поддерживаются типы-значения
- Пустого указателя просто нет!

```
e1 is 5 // Тип e1 задается типом константы 5
e2: Type is Expression
  // Тип Expression конформен типу Type
unitAttr: Type
  // init должен задать значение атрибуту
  // unitAttr

entity: ?A // У entity нет значения!!!
entity.foo // Ошибка компиляции

if entity is A then
   // Если тип entity A или производный от A,
   // то можно обращаться к подпрограммам
   // и атрибутам
   entity.foo
end

? entity // удалить значение.


a: A is entity // Ошибка компиляции

i: ?Integer
i := i + 5 // Ошибка компиляции
if i is Integer then i := i + 5 end
```

Константные объекты

- Каждый контейнер может определить все известные константы используя **const is**
- `Integer.1` – это валидный константный объект типа `Integer`
- Для упрощения доступа можно импортировать константные объекты, используя **use const**

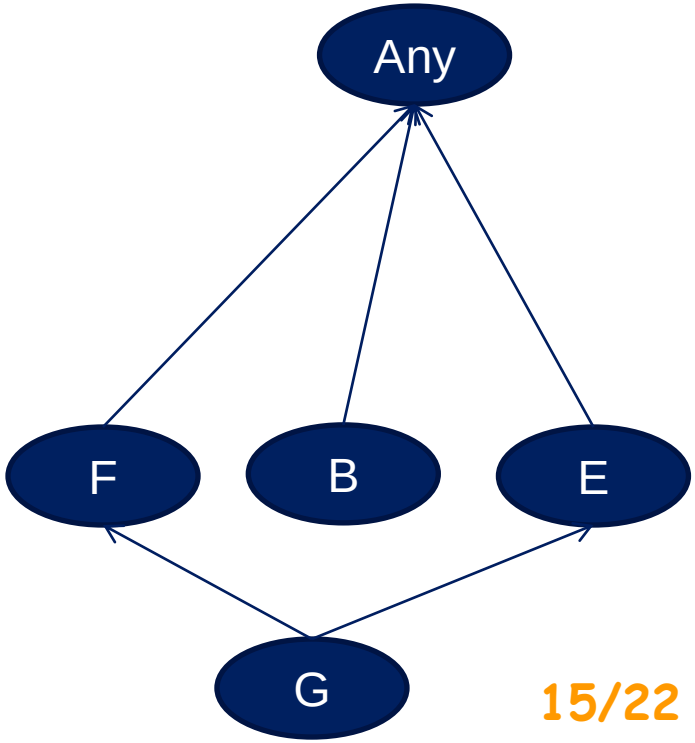
```
val unit Integer
     extend Integer[Platform.IntegerBitsCount]
   ...
end
val unit Integer [BitsNumber: Integer]
                    extend Numeric, Enumeration
   const minInteger is - (2 ^ (BitsNumber - 1))
   const maxInteger is 2 ^ (BitsNumber - 1) - 1
   const is
       // Этот набор константных объектов задает
       // все возможные целые константы
       minInteger .. maxInteger
   end
   init is
      data := Bit[BitsNumber]
   end
   hidden data: Bit[BitsNumber]
invariant
   BitsNumber > 0
end
abstract unit Any
     use const Integer, Real, Boolean, Character,
               Bit, String
   ...
end
```

Константные объекты

```
unit WeekDay
   const is Monday, Tuesday, Wednesday, Thursday,
            Friday, Saturday, Sunday
   end
end
use const WeekDay foo(Monday)
foo (day: WeekDay) is
   if day is
      Monday .. Friday: StandardIO.put (“На работу – день рабочий!\n”)
      Saturday, Sunday: StandardIO.put (“Выходной!!!!!\n”)
   end
end
unit A
   const is
       a1.init, a2.init(T), a3.init(T1, T2)
   end
   init is ... end
   init (arg: T) is ... end
   init (arg1: T1; arg2: T2) is ... end
end
const x is A.a1
y is A.a2
```

Основы стандартной библиотеки: все задано явно

```
abstract unit Any use const Integer, Real, Boolean, Character, Bit, String is
   = (that: ? as this): Boolean is foreign
   final /= (that: ? as this): Boolean is return not ( this = that) end
   = (that: as this): Boolean is foreign
   final /= (that: as this): Boolean is return not ( this = that) end
   == (that: ? as this): Boolean is foreign
   final /== (that: ? as this): Boolean is return not ( this == that) end
   == (that: as this): Boolean is foreign
   final /== (that: as this): Boolean is return not ( this == that) end
   hidden := (that: ? as this) is foreign
   hidden := (that: as this) is foreign
   toString: String is foreign
   sizeof: Integer is foreign ensure return >= 0 end
end // Any
unit System is
   clone (object: Any): as object is foreign
   deepClone (object: Any): as object is foreign
end // System
unit Platform is
   const IntegerBitsCount is 32
   const RealBitsCount is 64
   const CharacterBitsCount is 8
   const BooleanBitsCount is 8
   const PointerBitsCount is 32
   const BitsInByteCount is 8
end // Platform
```



Основы стандартной библиотеки: все задано явно

```
val unit Boolean extend Enumeration is
        const is false.init (0), true.init (1) end
        override < (other: as this): Boolean => not this => other
        override = (other: as this): Boolean => this.data = other.data
        succ: as this => if this then false else true
        pred: as this => if this then false else true
        override const first is false
        override const last is true
        const count is 2
        ord: Integer => if this then 1 else 0
        override sizeof: Integer => Platform.BooleanBitsCount / Platform.BitsInByteCount
        & alias and (other: as this): Boolean =>
                if this then if other then true else false else false
        | alias or (other: as this): Boolean =>
                if this = false then if other then true else false else true
        ^ alias xor (other: as this): Boolean =>
                if this then if other then false else true else if other then true else false
        => alias implies (other: as this): Boolean => not this or other
        ~ alias not : Boolean => if this then false else true
        toInteger: Integer => if this then 1 else 0
        init (value: as this) is data := value.data end
        init is data := 0xb end
        hidden init (value: Integer) require value in 0..1 is data := value end
        hidden data: Bit [Platform.BooleanBitsCount]
invariant
        this and this = this /// idempotence of 'and'
        this or this = this /// idempotence of 'or'
        this and not this = false /// complementation
        this or not this = true /// complementation
end // Boolean
```

Расширенная перегрузка имен

Два контейнера с одним именем различны, если они различаются настройками

```
i1: Integer is 5
i2: Integer[8] is 5

s1: String[3] is “123”
S2: String is “123”

a1: Array[Integer,3]
          is (1, 2, 3)
a2: Array[Integer]
          is (1, 2, 3)
a3: Array[Integer,(6,8)]
          is (1, 2, 3)
```

```
val unit Integer
   extend Integer [Platform.IntegerBitsCount]
  ...
end
val unit Integer[BitsNumber: Integer]
  ...
end
abstract unit AString // String abstraction
  ...
end
unit String[N:Integer]extend AString, Array[Character,N]
     // Fixed length string
  ...
end
unit String extend AString // Variable length String
  ...
end
abstract unit AnArray[G] // One-dim array abstraction
  ...
end
// Static one-dimensional array
unit Array[G->Any init(),N:Integer|(Integer,Integer)]
       extend AnArray[G]
  ...
end
// Dynamic one-dimensional array
unit Array[G->Any init()] extend AnArray [G]
  ...
end
```

Расширения контейнеров

- Все исходные файлы могут раздельно компилироваться
- “Умный линкер” необходим для корректной обработки создания объектов
- Валидность исходного файла 4 определяется как собрана программа на его основе, что включено.

Исходный файл 1:

```
unit A
   foo is
     local is A
   end
end
```

Исходный файл 2:

```
extend unit A
   goo is ... end
end
```

Исходный файл 3:

```
extend unit A extend B
   override too is ... end
end
unit B
   too is ... end
end
```

Исходный файл 4:

```
a is A
a.too
a.foo
a.goo
```

Множественное переопределение

Пусть имеется полиморфный массив :

```
a: Array[Figure] is (Circle (5),
Triangle(1,4,7), Rectangle(4, 5),
Circle(10))
```

Надо проверить, вписаны ли все фигуры друг в друга в той последовательности, как они перечислены. Основной алгоритм может выглядеть следующим образом:

```
flag is true; while pos in 1 .. a.count loop
   if pos< a.count and then
      not a(pos).inscribedInto(a(pos +1))
   then
      flag := false
      break
   end
end
```

Проблема заключается в корректном определении сигнатур и тел функций inscribedInto в контейнерах Circle, Triangle, Rectangle. Непосредственное решение приводит к необходимости полного или частичного перебора типов контейнеров в иерархии наследования

```
abstract unit Figure is
   inscribedInto(other: Figure): Boolean
           is abstract
end
unit Circle extend Figure is
  override inscribedInto(other: Figure):
                                  Boolean is
    if other is
      Circle: // Круг вписан в круг
      Triangle: // Круг вписан в треугольник
      Rectangle:/* Круг вписан в
прямоугольник*/
    else  // неизвестная фигура
      ...
    end
  end
end // Circle
```

```
unit Circle extend Figure is
 override inscribedInto(other: Circle): Boolean is
   ...
 end

 override inscribedInto(other: Figure): Boolean
                                       =>false
 end
end
```

Расширение контейнера Circle выглядит таким образом (этот код может располагаться в другом исходном файле).

```
extend unit Circle is
 override inscribedInto (other: Rectangle): Boolean
 is
   ...
 end

 override inscribedInto(other: Triangle):Boolean is
   ...
 end
end
```

Аналогично поступаем и с другими контейнерами. В результате при обработке вызова вида `a(pos).inscribedInto(a(pos+1))`

будут вызываться функции не просто в соответствии с динамическим типом a(pos), а еще с учетом динамического типа аргументов. В этом и заключается смысл понятия двойной диспетчеризации, которая представлена в самом языке программирования как возможностью множественного переопределения одной подпрограммы

Параметризация подпрограмм

```
x1 is factorial1[Integer](3)
   // обращение к factorial1 будет выполнено
   // при выполнении программы

x2 is factorial2[3]
   // Этот вызов будет вычислен при компиляции

factorial1[G->Numeric](x: G): G is
   if x is
      x.zero, x.one: return x.one
   else
      return x * factorial1(x - x.one)
   end
end

factorial2[x:Numeric]: as x is
   if x is
      x.zero, x.one: return x.one
   else
      return x * factorial2[x - x.one]
   end
end
```

Пример параллельной программы

```
philosophers is (concurrent Philosopher ("Aristotle"), concurrent Philosopher ("Kant"), concurrent Philosopher ("Spinoza"), concurrent Philosopher ("Marx"), concurrent Philosopher ("Russell"))
forks is (concurrent Fork (1), concurrent Fork (2), concurrent Fork (3), concurrent Fork (4), concurrent Fork (5))
check
   philosophers.count = forks.count or else philosophers.count = 1 and then forks.count = 2
   /* Задача валидна, если число вилок совпадает с числом философов или, если философ - один, то ему просто нужны две вилки*/
end
loop /// Пусть философы едят бесконечно. Возможен и иной алгоритм симуляции …
   while seat in philosophers.lower .. philosophers.upper loop
      StandardIO.put ("Философ '" + philosophers (seat).name + "' готов есть\n")
      eat (philosophers (seat), forks (seat), forks (if seat = philosophers.upper then forks.lower else seat + 1)
   end
end
eat (philosopher: concurrent Philosopher; left, right: concurrent Fork) is
   /* Процедура - eat с тремя параллельными параметрами, вызов которой и образует критическую секцию параметризованную ресурсами, которые находятся в эксклюзивном доступе для этой секции */
   StandardIO.put (" Философ '" + philosopher.name + "' ест вилками №" + left.id + " и №" + right.id + "\n")
end
unit Philosopher is
   name: String
   init (aName: as name) is name := aName end
end
unit Fork is
   id: Integer
   init (anId: as id) is id := anId end
end
```

Заключение

Представлены

- Основные концепции языка СЛанг (контейнеры, отдельностоящие подпрограммы, некоторые операторы, подход к построению программ)
- Множественное наследование с конфликтами и множественным переопределением
- Расширенная перегрузка имен
- Базовые контейнеры
- Расширения контейнеров
- Решение проблемы инициализации атрибутов

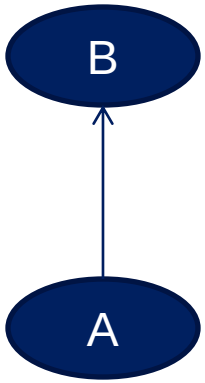
Статус

- В настоящий момент полностью проработан синтаксис языка, семантика всех конструкций, идет работа над строгим описанием правил валидности конструкций, а также ведется разработка компилятора переднего плана (front-end) и генераторов кода для нескольких программных и аппаратных платформ, среди которых .NET, LLVM, Эльбрус и др. Проводятся подготовительные работы к раскрутке компилятора (bootstrapping): к переносу реализации на собственный входной язык.
- Ожидается, что можно будет приступить к разработке приложений для операционных систем Android, Linux, Windows и ОС Эльбрус по готовности всего компилятора и прикладных библиотек, работа над которыми также ведется.

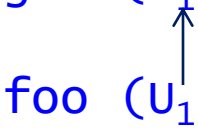
СПАСИБО! ВОПРОСЫ?

Конформность

1. Контейнер A конформен контейнеру B, если есть путь в графе наследования от A к B.
2. Сигнатура foo конформна сигнатуре goo, если каждый тип сигнатуры foo конформен соответствующему типу сигнатуры goo.




```
unit B
end

unit A extend B
end
```



Если для любого i из $1 .. N$ U_i конформен T_i