Московский институт радиотехники, электроники и автоматики (МИРЭА ТУ)

Бебчик Алексей Михайлович

Раздаточный материал к диссератции

Разработка и исследование структурно-ориентированного редактора и компилятора запросов системы функционально-логического программирования

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Научный руководитель:

д.т.н., проф. Фальк Вадим Николаевич

Москва 2005

Цель, задачи и структура работы

Цель работы:
создание системы функционально-логического программирования (СФЛП), основанной на формализме направленных отношений (НО) и обладающей развитыми интерфейсными средствами построения и отладки программ.

Основные задачи:

· исследование языка FLOGOL и формальное описание его семантики;

· разработка основных принципов и метода компиляции FLOGOL-запросов;

• разработка специальной технологии ввода программ и соответствующих интерфейсных средств;

• программная реализация и интеграция в СФЛП компилятора запросов и структурно-ориентированного редактора, основанного на разработанной технологии ввода;

· тестирование созданной СФЛП на наборе типовых задач декларативного программирования.

Структура диссерации:

Глава 1: Языки и системы декларативного программирования.

Глава 2: Теория направленных отношений и язык функционально-логического программирования S-FLOGOL.

Глава 3: Компилятор запросов программ на языке S-FLOGOL.

Глава 4: Структурно-ориентированный редактор.

Научная новизна и практическая значимость

Научная новизна:

1. Предложена система ограничений языка FLOGOL, позволяющая выполнить компиляцию программы в конечную контекстно-свободную сетевую грамматику (КССГ) и значительно упрощающая реализацию языка без существенного снижения его основных выразительных возможностей.

2. На основе предложенной системы ограничений разработан язык S-FLOGOL, формально описаны его синтаксис и семантика.

3. Сформулированы основные принципы и разработан метод компиляции запросов программ на языке S-FLOGOL, опирающийся на формальное описание семантики

4. Предложена оригинальная технология дедуктивного ввода программ, позволяющая минимизировать ручной ввод, полностью исключить синтаксические ошибки и значительно снизить количество семантических ошибок при вводе программы.

Практическая значимость работы заключается в возможности использования созданной совместно с Бебчиком Ан. М. СФЛП для проведения исследовательских работ с применением формализма НО, а также для обучения студентов основам декларативного программирования.
Практическая значимость работы подтверждается использованием созданной СФЛП в учебном процессе МИРЭА и МАИ.

Апробация работы и публикации

Апробация работы

Основные результаты диссертации докладывались и обсуждались на научно-технических конференциях МИРЭА (Москва, 2002, 2003, 2004), международных конференциях «Информационные средства и технологии» (Москва, 2003, 2004), всероссийской межвузовской научно-технической конференции «Микроэлектроника и информатика – 2004» (Зеленоград, 2004), международной научно-технической конференции «Информационные технологии в науке, образовании и производстве» (Орел, 2004), международной конференции «Континуальные алгебраические логики, исчисления и нейроинформатика в науке и технике» (Ульяновск, 2004), международной научно-технической конференции «Компьютерное моделирование 2004» (Санкт-Петербург, 2004) и «Девятой Национальной конференции по искусственному интеллекту с международным участием КИИ-2004» (Тверь, 2004).

Реализованная СФЛП демонстрировалась на выставке программных продуктов «Девятой национальной конференции по искусственному интеллекту с международным участием КИИ-2004».

Публикации

Основные результаты, полученные при выполнении диссертационной работы, опубликованы в 9 печатных работах.

Язык S-FLOGOL (Small FLOGOL)

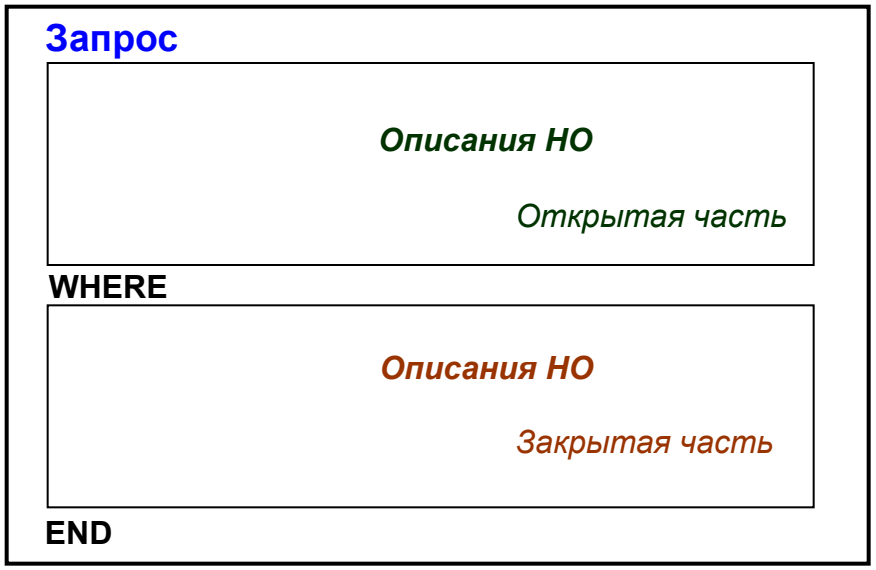
- Основан на теории направленных отношений (НО).
- Поддерживает аппликативный и композиционный стили программирования.
- Обладает развитой схемной надстройкой, включающей:
  - индексированные имена объектов,
  - условные конструкции,
  - свертки.
- Допускает динамическую типизацию объектов.
- Поддерживает параметризацию определений НО.
- Позволяет строить многомодульные программы.
- Обладает средствами ограничения области видимости НО.

Основные ограничения по сравнению с языком FLOGOL:

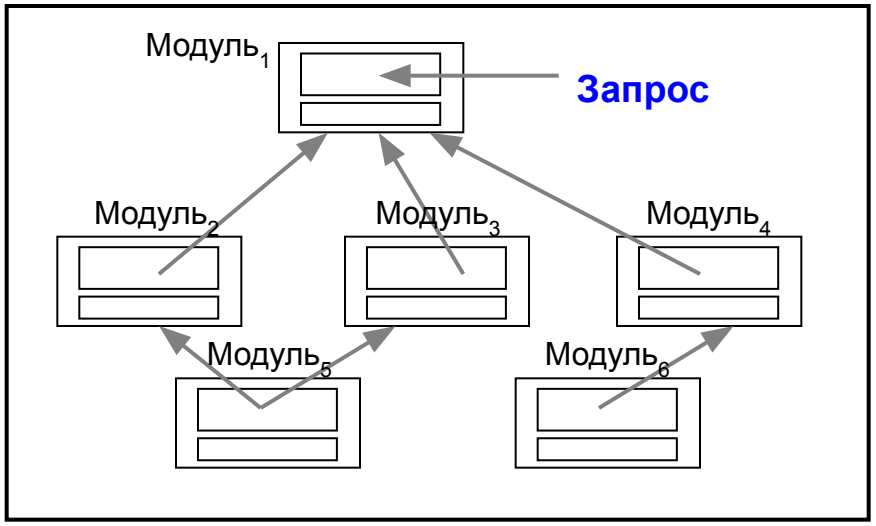
- Упрощенная модульная структура.
- Исключение пользовательских связок.
- Упрощенный механизм параметризации (компиляция).
- Отсутствие открытых интервалов в свертках (компиляция).

Программа на языке S-FLOGOL

Программный модуль:



Программа:



Пример программы:

```
MODULE Arithm =
  //Конструкторы
  (+0+:+1)Null;
  (+1+:+1)Succ;
  //Аппликативные определения
  Nat0={:Null};
  Nat1={:Succ(Null)};
  Add={Null,x:x};
  Add={Succ(x),y:Succ(@(x,y))};
  //Композиционные определения
  --- = {x:x};
  [0]Nat=Null;
  (K=1..100)
    [K]Nat=Null·(·J=1..K)Succ;
  AddC= (~Succ # ---)·@·Succ U (~Null # ---);
  //Запросы к системе
  QAdd={:Add(Nat0,Nat1)};
  QAddC=([1]Nat#[2]Nat)·AddC;
END
```

имя модуля

комментарии

Индексированные имена НО

свертка

Сетевая интерпретация программ

Программа:

OBJECT Null;

CONSTRUCTOR(1)Succ;

Nat1=Null·Succ;

Nat2=Null·Succ·Succ;

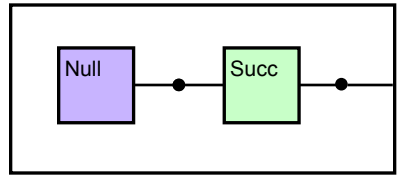
Add={Null,x:x};

Add={Succ(x),y:Succ(@(x,y))};

QUERY=(Nat1#Nat2)·Add

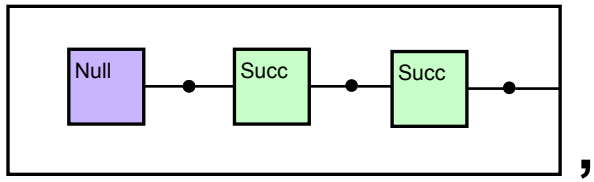
Правила КССГ $R =$

Nat1 →



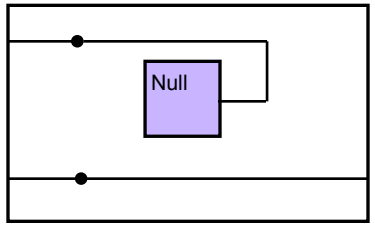
,

Nat2 →



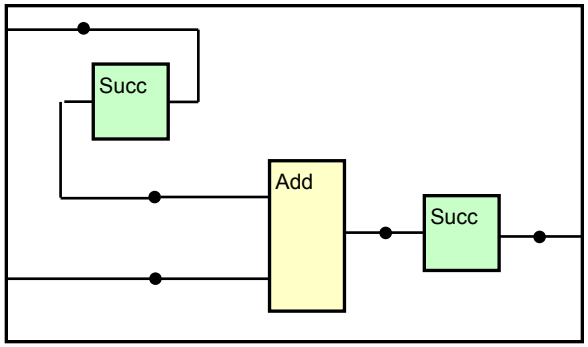
,

Add →



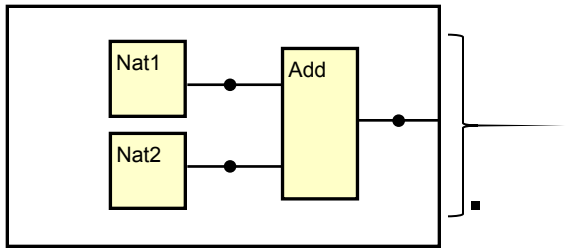
,

Add →



,

Query →



.

Базис КССГ:

$B_m = \{\text{Null}^{(0,1)}, \text{Succ}^{(1,1)}\},$

$B_н = \{\text{Nat1}^{(0,1)}, \text{Nat2}^{(0,1)}, \text{Add}^{(2,1)}, \text{QUERY}^{(0,1)}\}.$

Аксиома КССГ:

$\alpha = \text{QUERY}^{(0,1)}.$

Описания НО

Описания НО задаются *доменными выражениями* (Дом):

· Объявление НО: Дом → Спец Имя

(+0+:+1)Null
(+1+:+1)Succ

Ограничение: исключена внешняя пользовательская интерпретация НО.

· Определение НО: Дом → Спец Имя = Рел

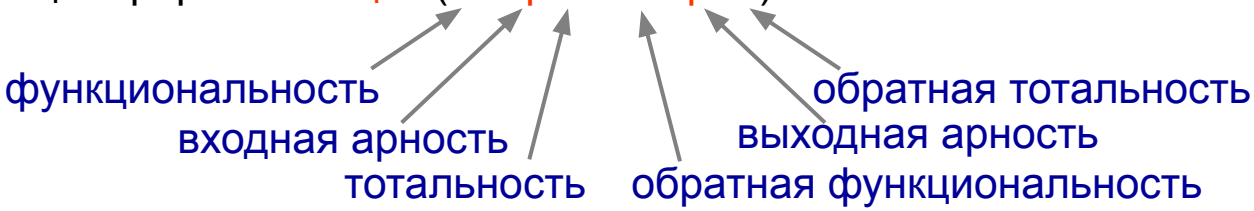
(2:1)Add = {Null,x:x}
(2:1)Add = {Succ(x),y:Succ(Add(x,y)}

· Объединение описаний: Дом → Дом ; Дом

(+0+:+1)Null;
(+1+:+1)Succ;
(2:1)Add = {Null,x:x};
(2:1)Add = {Succ(x),y:Succ(Add(x,y)}

Описания НО могут быть *специфицированы*:

Общая форма: Спец →(+ Ар + : + Ар +)



Примеры:

(+0+:+1)Null
(+1+:+1)Succ

Ключевая спецификация:

Спец →Ключ
Спец →Ключ(Ар)
Спец →Ключ(Ар:Ар)

Ключ – ключевое слово, оределяющее свойства и, возможно, арности НО.

Пример: **OBJECT** ≅ (+0+:+1)
OBJECT Null

Полиморфизм - идентичные имена, но разные спецификации:

(2:1)Add
(3:0)Add

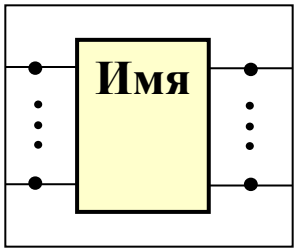
Определение НО

Определение НО задается *реляционным выражением* (Рел):

· Вызов по имени: Рел → Имя

· Рекурсивный вызов: Рел → @

· Вызов параметра: Рел → <<Ар>>



· График - аппликативная форма определения НО:

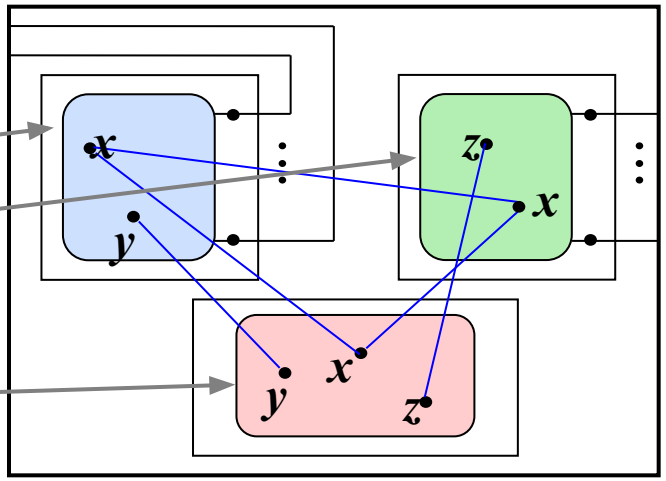
Рел → {Терм : Терм ? Формула}

Входной терм – аргументы вызова.

Выходной терм – результат вызова.

Формула – ограничения на интерпретацию переменных графика.

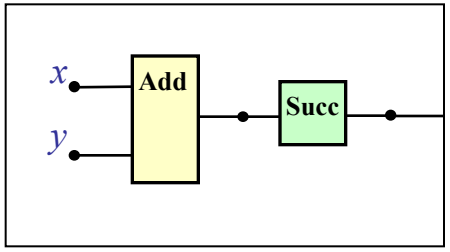
· Формула:



· Терм:

Переменная
Вызов
Терм , Терм
Терм | Терм

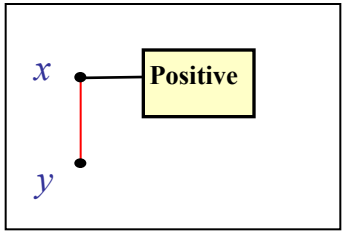
Пример: Succ(Add(x,y))



· Формула:

Терм=Терм
Терм<>Терм
Вызов
Формула & Формула
Формула | Формула

Пример: Positive(x) & x<>y



Композиционное программирование

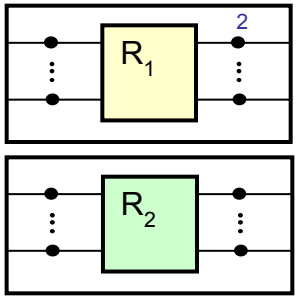
· Композиция отношений: Рел → Рел ⊗ Рел, где ⊗ ∈ {·,#,→,∇,*,∪,∩},

--- = {x:x}

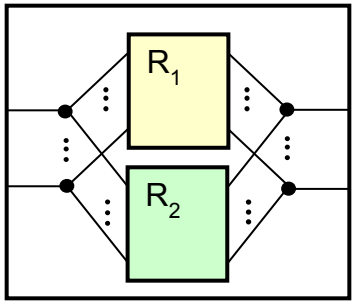
При помощи комбинаторных констант и операций композиции можно задать любое комбинаторное НО.

Сетевая интерпретация операций компопозиции НО:

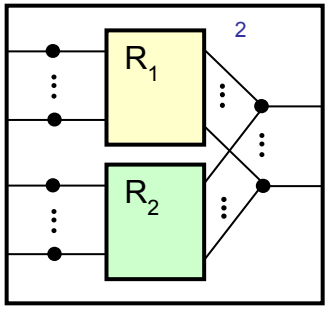
Объединение: $R_1 \cup R_2$



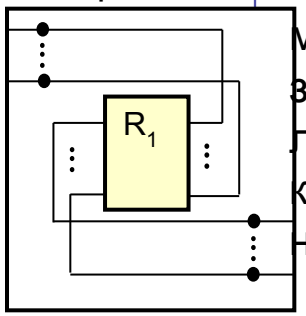
Пересечение: $R_1 \cap R_2$



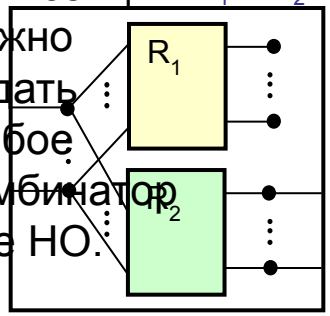
Унификация: $R_1 \nabla R_2$



Инверсия: $\sim R_1$

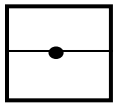


Условная: $R_1 \rightarrow R_2$

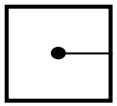


· Комбинаторные константы:

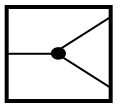
--- = {x:x}



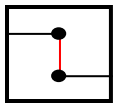
<-- = {:x}



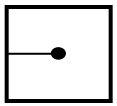
--< = {x:x,x}



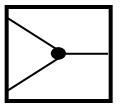
-/- = {x:y?x<>y}



--> = {x:}



>-- = {x,x:x}



При помощи комбинаторных констант и операций композиции можно задать любое комбинаторное НО.

Пример: композиционное определение НО сложения:

--- = {x:x}

Add = (~Succ # ---)·Add·Succ ∪ (~N # ---)

Параметризованные описания НО

Параметризация позволяет задавать своего рода шаблоны описаний НО.

Список параметров:

СпПар → Имя полное имя НО
СпПар → @ собственные параметры
СпПар → _ пропуск параметра
СпПар → СпПар ; СпПар объединение в список

Вызов параметризованного НО: Имя[СпПар]

Вызов параметра в определении: <<Ар>>

*Значения по умолчанию: Имя[СпПар] = Рел

Ограничения:

1) Нет групп параметров

СпПар → Имя
полное имя НО
СпПар → @
собственные параметры

3) Нет СпПар→График

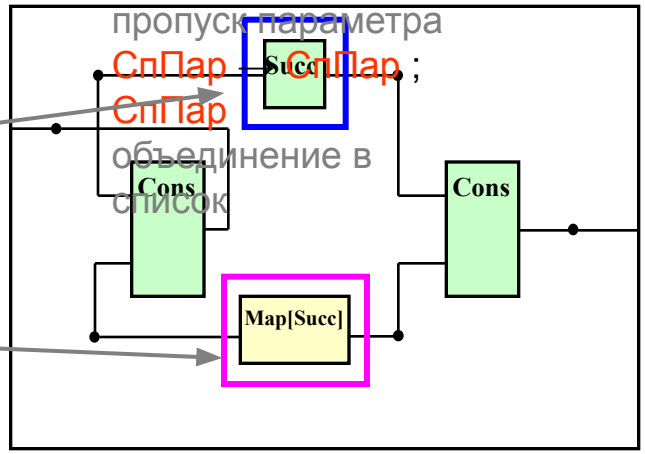
СпПар → _
пропуск параметра
СпПар → СпПар ;
СпПар
объединение в список

Пример параметризованного НО – отношение Map:

--- = {x:x};

Map[---] = {Cons(x,y): Cons(<<0>>(x),@(xs))}∪{Nil:Nil};

QUERY = Map[Succ];



Условная конструкция IF позволяет выбирать варианты определения НО.

Пример: контроль арностей входного параметра отношения Map .

Map[---] = **IF** ►<<0>> <> 1 **OR** <<0>>► <> **1** **THEN** ---
ELSE {Cons(x,y): Cons(<<0>>(x),@(xs))} ∪ {Nil:Nil} ;

QUERY = Map[(2:1)Add]; ⇨ {Cons(x,y): Cons(---(x), Map[(2:1)Add](xs))} ∪ {Nil·Nil}

Индексированные имена

Индексирование позволяет естественным образом задавать имена, семантически зависящие от некоторых кортежей целочисленных индексов.

Список индексов:
СпИнд → Ар
СпИнд → СпИнд, СпИнд

Имя переменной терма: Переменная → [СпИнд] Ид

Имя отношения: Имя → Спец [СпИнд] Ид СпПар

Массивы объектов:

[1,1]Cell= [1]Nat;
[1,2]Cell= [4]Nat;
[2,1]Cell= [2]Nat;
[2,2]Cell= [10]Nat;
QUERY= ([1,1]Cell#[1,2]Cell)·Add

Натуральные числа:

[1]Nat= Null·Succ;
[2]Nat= Null·Succ·Succ;
Add={Succ([1]x),[2]x:Succ(@([1]x,[2]x));
QUERY=[1]Nat # [2]Nat·Succ

Свертка

Свертка позволяет в компактной форме задавать объекты (и множества объектов), а также эффективно формировать сетевое определение параметризованных НО.

Выр → (Инф ИдСв = СпЗнач) Выр

Ограничение: нет пользовательских связок для реляционного выражения.

Ид – операторная переменная свертки.
Инф – инфиксная связка.
СпЗнач – список значений переменной.
Выр – выражение-операнд свертки.

(⊗ J=1..10)Выр ⇨ ⊗([1/J]Выр, ⊗([2/J]Выр,… ⊗([9/J]Выр,[10/J]Выр)…)), где

[x/J]Выр обозначает результат подстановки x вместо J в выражение Выр.

Свертка (продолжение)

Пример: натуральные числа

1) [3]Nat = Null·(· J=1..3)Succ ⇨ [3]Nat = Null·Succ·Succ· Succ

2) [0]Nat = Null;
(· J=1..100)[J]Nat = Null·(·K=1..J)Succ ⇨
[0]Nat = Null;
[1]Nat = Null·Succ;
[2]Nat = Null·Succ·Succ;
…
[100]Nat = Null·Succ·…·Succ;

Типизация

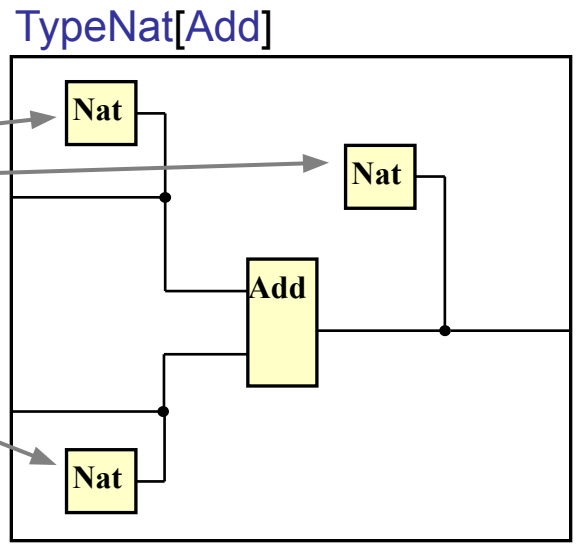
Типовое отношений может быть задана программистом при помощи так называемых типовых отношений. Типовое отношение для типа $T = (t'_1,...,t'_{n'} \to t''_1,...,t''_{n''})$, где $t'_i(t''_i)$ – сорта входных(выходных) данных, определяется как:

$TypeT = \{G_{t'_1}, \ldots, G_{t'_{n'}} : G_{t''_1}, \ldots, G_{t''_{n''}}\} \cap «0»,$

где G_t – НО арности $(0,1)$, генерирующее данные сорта t.

Пример типизации НО сложения:

```
//Генератор натуральных чисел
Nat = Null ∪ @·Succ;
//Отношение сложения
Add = {Null,x:x};
Add = {Succ(x),y:Succ(@(x,y))};
//Типиовое отношение арности (2,1)
(2:1)TypeNat = {Nat,Nat:Nat} ∩<<0>>;
//Типизированное отношение сложения
TypedAdd = TypeNat[Add]
```



Описание семантики языка (начало)

Семантика языка описана с точностью до преобразования в КССГ, индуцированную выделенным в программе запросом к системе.

Принципы описания:

- Φ_{φ}□ …□ – интерпретация синтаксической конструкции языка при интерпретации φ контекста ее переменных.
- $\varphi = < \xi, Name >$ – контекст вызова НО, где ξ – контекст сверки, $Name$ – имя вызываемого НО.
- $\xi = v_1 v_2 ... v_n, n \geq 0$ – значения операторных переменных свертки (v_i), в области действия которых находится описываемая конструкция.
- $Name = < Spec, IndList, Id, Cons, Prms >$ – спецификатор, список индексов, идентификатор признак НО – конструктора, список параметров вызова.
- Контекст, имя вызова, список индексов и список параметров записываются в виде *списков*. Для работы со списками используются функции *elm*(*Список*, *Номер*) – выборка элемента списка с указанным номером, *append(Список, Список)* – конкатенация списков, пустой список обозначается [ ].

Область интерпретации выражений:

Тип	Область интерпретации	Обозначение	По умолчанию
Дом	Множество правил КССГ	R	∅
Рел	Множество сетей, Множество правил	S, R	∅, ∅
СпПар	Список имен НО	$Prms$	[ ]
Терм	Множество сетей, Множество правил	S, R	$\{S0^{(0,0)}\}$, ∅
Формула	Множество сетей, Множество правил	S, R	$\{S0^{(0,0)}\}$, ∅
Ар	Множество натуральных чисел	Nat	0
Лог	$\{True, False\}$	$Bool$	$False$
СпИнд	[ ]	$IntList$	[ ]
СпЗнач	[ ]	$ValList$	[ ]

Описание семантики языка (окончание)

Семантика определения НО в доменном выражении:

$\Phi_\varphi\square Спец\ e_1\ Ид\ e_2 = e_3\square = R$, где $e_1 \in e(СпИнд), e_2 \in e(СпПар),\ e_3 \in e(Рел),$

$Name_0 = elm(\varphi,2),$ имя вызываемого НО

$Name_{call} = [elm(Name_0,1), elm(Name_0,2), elm(Name_0,3)],$ имя вызываемого НО без параметров

$Name_{dom} = [\ \Phi_\varphi\ \square Спец\ \square, \Phi_\varphi\ \square e_1\ \square, k(Ид)],$ имя определяемого НО

$Name = unify(Name_{call}, Name_{dom}),$ унификация имен вызываемого и определяемого НО

$Prms = joinPrms(elm(\varphi,5),$ подстановка значений по умолчанию в параметры вызова

$[S, R_1] = \Phi_{\varphi'}\square e_3\ \square, \varphi' = [elm(\varphi,1), append(Name, Prms)],$ формирование сети вызываемого НО и индуцированных сетевых определений

$R_2 = makeR(Name, S),$ формирование сетевого определения

$R = R_1 \cup R_2.$ формирование результата

Семантика объявления НО в доменном выражении:

$\Phi_\varphi\square Спец\ e_1\ Ид\ \square = \{[Name, makeS1(Name)]\}$, где $e_1 \in e(СпИнд),$ *makeS*1 – функция, формирующая одноэлементную сеть, содержащую элемент НО с именем *Name*.

$Name_0 = elm(\varphi,2),$

$Name_{call} = [elm(Name_0,1), elm(Name_0,2), elm(Name_0,3)],$ имя вызываемого НО

$Name_{dom} = [\ \Phi_\varphi\ \square Спец\ \square, \Phi_\varphi\ \square e_1\ \square, k(Ид)],$ имя определяемого НО

$Name = unify(Name_{call}, Name_{dom}).$ унификация имен вызываемого и определяемого НО

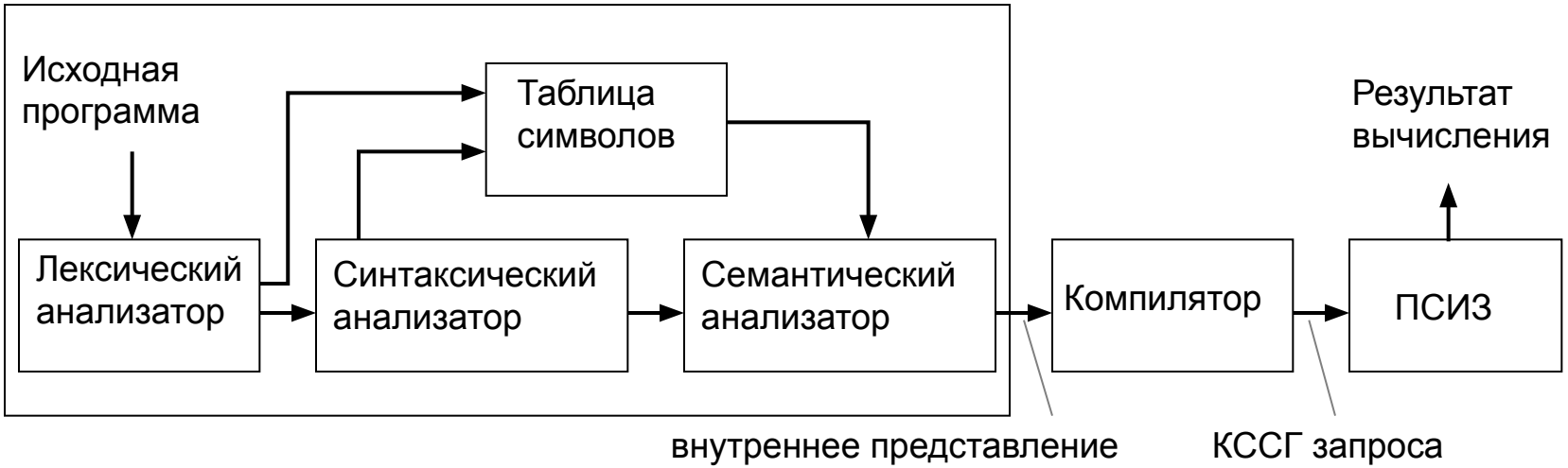
Семантика объединения описаний НО:

$\Phi_\varphi\square e_1\ ;\ e_2\square = \Phi_\varphi\ \square e_1\ \square \cup \Phi_\varphi\ \square e_2\ \square$, где $e_1, e_2 \in e(Дом),$ объединение множеств правил КССГ

Реализация языка S-FLOGOL

Смешанная форма = компиляция + интерпретация

Структурно-ориентированный редактор (специальная технология ввода)



Компиляция – генерация контекстно-свободной сетевой грамматики (КССГ), эквивалентной сетевой интерпретации S-FLOGOL-программы с выделенным запросом к системе.

Интерпретация – вычисление КССГ подсистемой исполнения запросов (ПСИЗ).

Стадии компиляции:

- Предварительная: завершение формирования и дополнительная обработка внутреннего представления программы;
- основная: формирование и вычисление логической программы-компилятора (ЛПК), результатом которого является КССГ запроса;
- заключительная: обработка полученной КССГ запроса, запись КССГ в сетевой модуль СФЛП и передача в графический редактор.

Предварительная стадия компиляции

· Дополнение не полностью введенных элементов программы соответствующими значениями по умолчанию:

CONSTRUCTOR(Ap) Succ

СпИнд Add = {Succ(x),y:Succ(@(x,y))}

Add = ИмяОтн

⇨

CONSTRUCTOR(0) Succ

Add = {Succ(x),y:Succ(@(x,y))}

Add = EmptyRel

· Первичный семантический анализ:

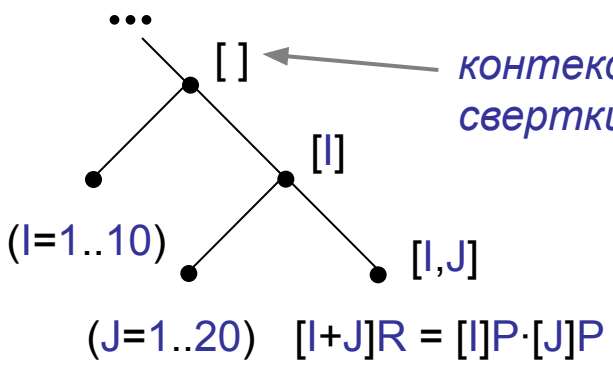
- контроль вызова не объявленных НО,
- чистка грамматики (исключение не используемых НО).

· Индексирование вхождений операторных переменных сверток:

Программа

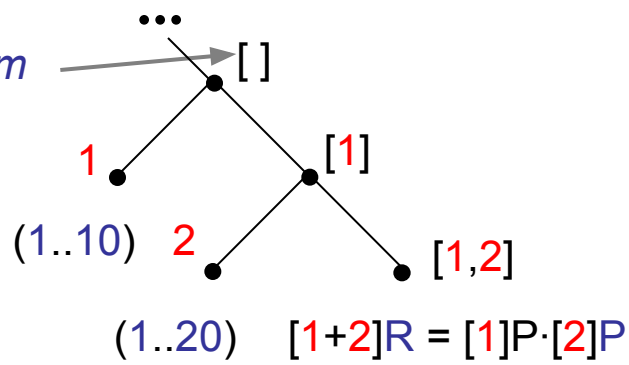
```
…
(I=1..10)
   (J=1..20)
      [I+J]R = [I]P·[J]P
…
```

Дерево



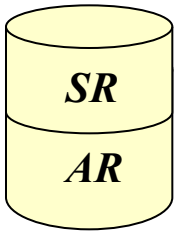
контекст свертки

Индексированное дерево



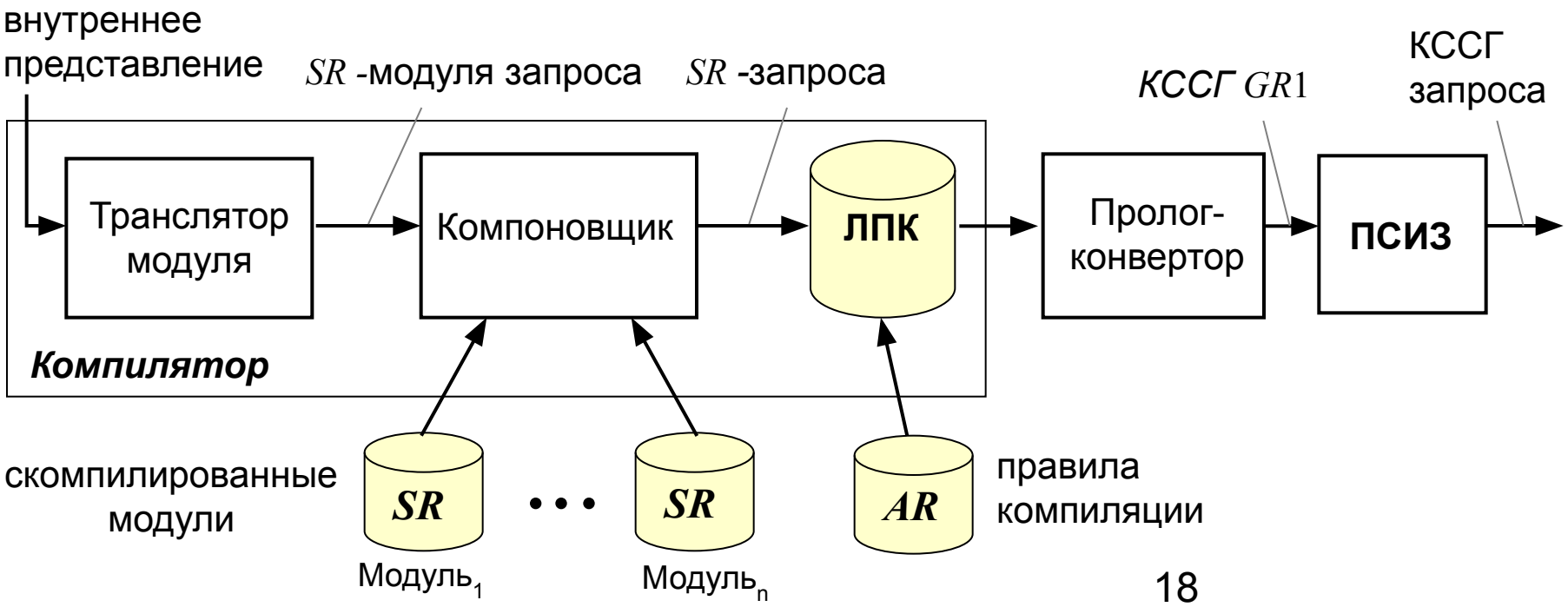
Основная стадия компиляции

Формирование КССГ выполняется логической программой-компилятором (ЛПК), база данных которой содержит:



- *SR* -правила (*SpecialRules*) - база описаний НО.
- *AR* -правила (*AdditionalRules*) - управляющие и вспомогательные правила компиляции.

Схема выполнения основной стадии:



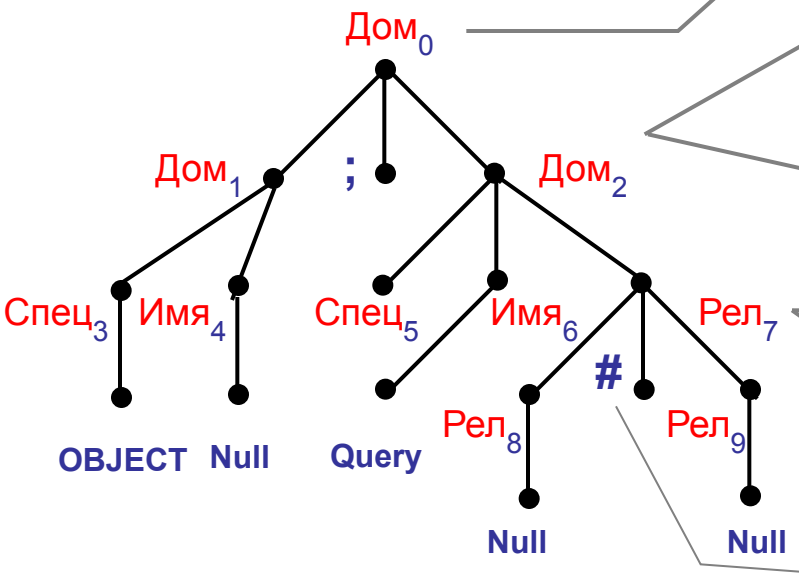
База описаний НО (*SR*-правила)

Для каждой синтаксической конструкции программы на основании семантических функций трансляции строится правило на языке Пролог и добавляется во множество SR-правил базы описаний НО.

Программа:

OBJECT Null;
QUERY = Null#Null

Внутреннее представление:



SR-БД:

корневой домен

```
дом_0(Контекст, Правило):-
      дом_1(Контекст, Правило).
дом_0(Контекст, Правило):-
      дом_2(Контекст, Правило).
дом_1(Контекст, Правило):-
      спец_3(Контекст, Спец),
      одноэлСеть(Спец, 'Null', Сеть),
      Правило = ['Конс', 'Query', Сеть].
спец_3(Контекст, сСпец(+,0,+,+,1,-)).
дом_2(Контекст, Правило):-
      спец_5(Контекст,Спец),
      рел_7(Контекст, Сеть)
      Правило = ['Прав', 'Query', Сеть].
рел_7(Контекст, Сеть):-
      рел_8(Контекст, Сеть1),
      рел_9(Контекст, Сеть2),
      послКомп(Сеть1,Сеть2,Сеть).
рел_8(Контекст, Сеть):-
      одноэлСеть(Спец, 'Null', Сеть).
рел_9(Контекст, Сеть):-
      одноэлСеть(Спец, 'Null', Сеть).
```

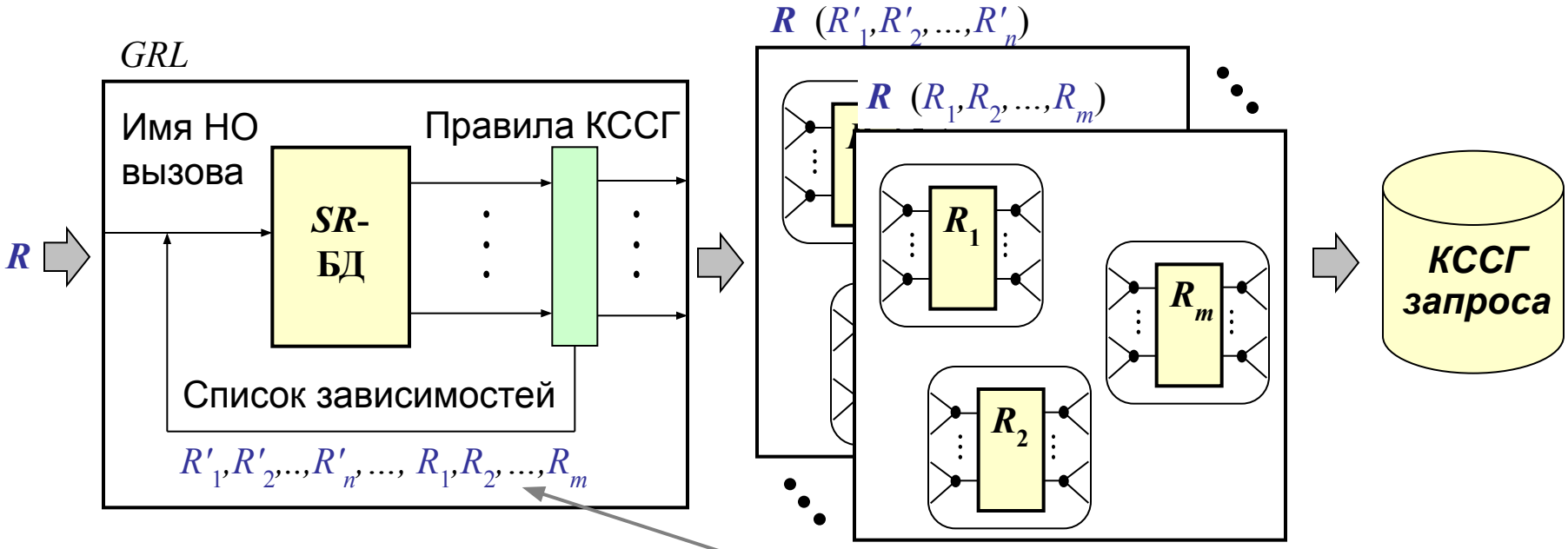
Контекст (вызова НО):
- список значений переменных свертки;
- имя вызываемого отношения.

Управление компиляцией (*AR*-правила)

Конструктивное построение КССГ:

1) получить сетевое определения НО с указанным именем,
2) выполнить шаг 1) для всех имен НО, индуцированных этим определением.

Схема формирования КССГ (*AR*-правило *GRL*):



Формирование КССГ:

- начинается с шага 1) для НО запроса,
- завершается, когда список зависимостей пуст.

Запрос на получение определения НО осуществляется вызовом *SR*-правила корневого доменного выражения с контекстом, содержащим имя запрашиваемого НО.

Формирование сети

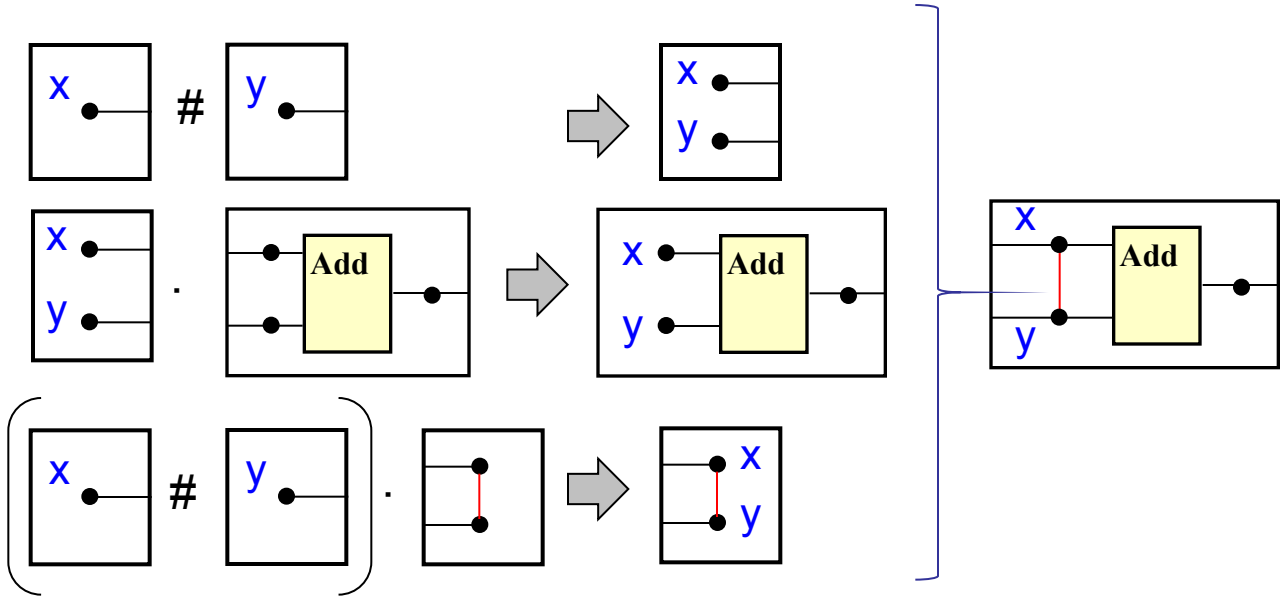
· Формирование графика

Пример: {x, y: Add(x,y)?x<>y}

входной терм = x, y

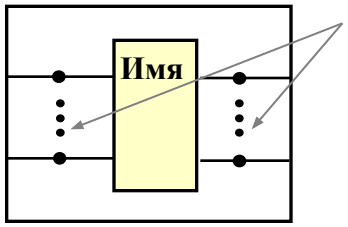
выходной терм = Add(x,y)

формула = x <> y



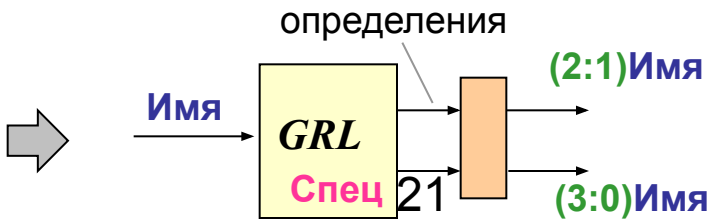
· Формирование вызова НО

Проблема:
имя вызываемого и определяемого НО, как правило, не специфицировано.



? необходимо знать арность

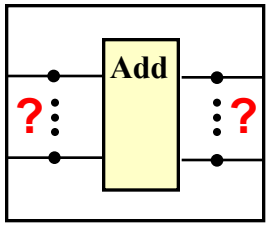
Решение:
получить определение вызываемого НО, извлечь информацию о его спецификации.



Оптимизация получения спецификаторов

· Исключение дублирующих спецификаторов.

Вызов НО Add:

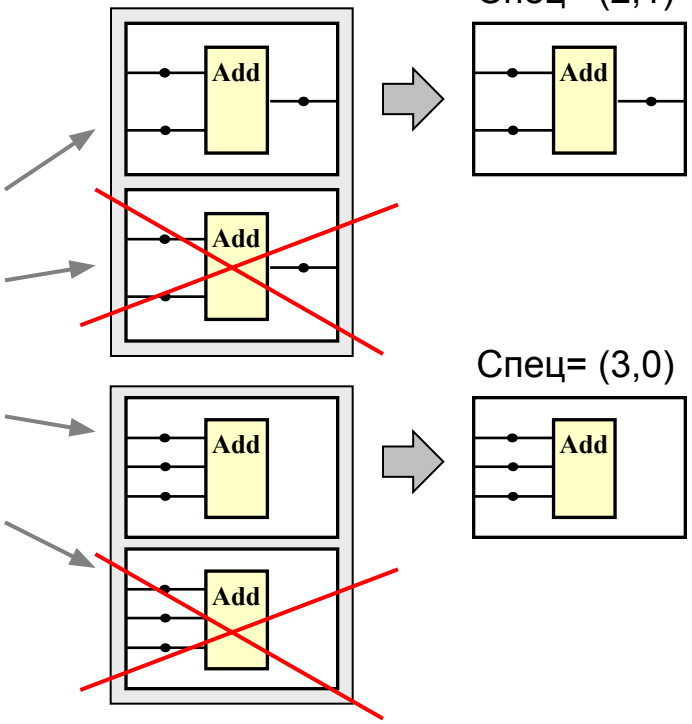


Add={Null,x:x}

Add={Succ(x),y:Succ(Add(x,y))}

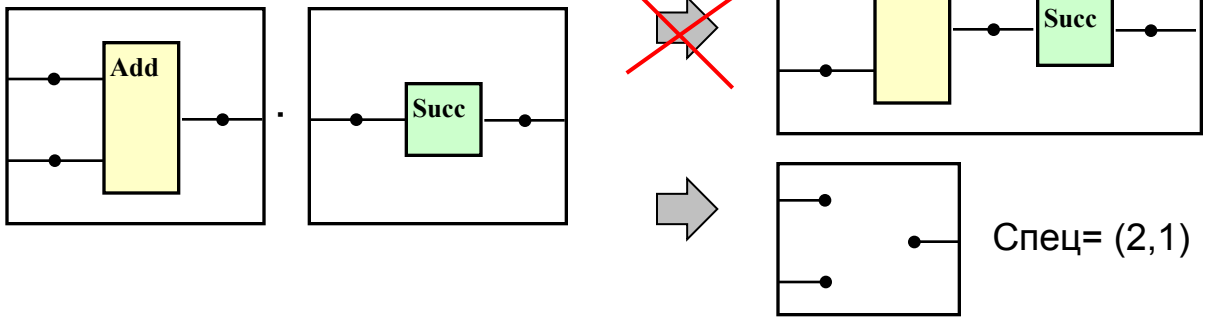
Add={Null,x,x:}

Add={Succ(x),y,Succ(Add(x,y)):}



· Частичное выполнение композиций НО.

Спец? (Add·Succ):



SimpleNet – язык текстового описания КССГ

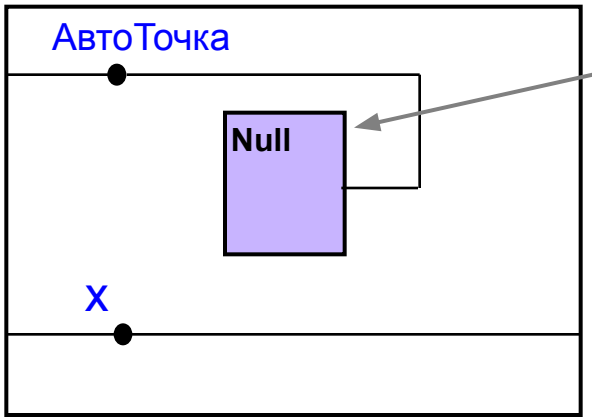
Причина создания языка: чрезвычайно сложный для конструктивного построения формат внутреннего представления КССГ в ПСИЗ.

Основные грамматические правила:

Сеть → сСеть(Имя,СпЭл,СпТоч,СпДуг),
Имя → сИмя(Спец,СпИнд,Ид,СпПар,ТипОпр) ,
Спец → сСпец(Ф,ВхАр,Т,Оф,ВхАр,От),
СпЭл → {Эл}+ , Эл → сЭл (НомЭл,Спец,Имя) ,
СпТоч → {Точ} , Точ → сТоч (НомТоч,СпИнд,Ид) ,
СпДуг → {Дуга} , Дуга → сДуга (НомДуги,Тип,НомЭл,НомТоч).

Сеть – универсальная структура данных, содержащая результат интерпретации доменного и реляционного выражения, терма и формулы.

Пример. Сеть НО Add:



SimpleNet (*SN-*) представление:

```
сСеть(
[
   сЭл([276],сСпец(-,2,-,-,1,-),сИмя(сСпец(-,2,-,-,1,-),Nil,Add,Nil,Прав)),
   сЭл([280],сСпец(-,0,-,-,1,-),сИмя(сСпец(-,0,-,-,1,-),Nil,Null,Nil,Конс))],
[
   сТоч([2,1,[280]],Nil,АвтоТочка),
   сТоч([[276],Nil,x],Nil,...)
],
[
   сДуга([1,2,1,[280]],Вх,[276],1,[2,1,[280]]),
   сДуга([2,285],Вх,[276],2,[[276],Nil,x]),
   сДуга([2,289],Вых,[276],1,[[276],Nil,x]),
   сДуга([2,2,1,[280]],Вых,[280],1,[2,1,[280]])
])
```

Заключительная стадия компиляции

· Преобразование КССГ из *SN*-формата в формат КССГ (выполняется в ПСИЗ).

· Редукция имен – исключение не значимых для КССГ элементов имени.

имя НО после компиляции:

(-,0,-,-,1,-),Nil,Null,Nil,Конс

(-,2,-,-,1,-),Nil,Add,Nil,Прав

(-,3,-,-,0,-),Nil,Add,Nil,Прав

(-,1,-,-,1,-),Nil,Map,Succ,Прав

· Редукция имен – исключение не значимых для КССГ элементов имени.

Null

(-,2,-,-,1,-)Add

(-,3,-,-,0,-)Add

Map[Succ]

· Установка интерпретации системных типов данных и отношений.

(-,3,-,-,0,-),Nil,124,Nil,Конс → системный тип – натуральное число

(-,3,-,-,0,-),Nil,$$Add,Nil,Конс → системное НО – сложение нат. чисел

· Сохранение полученной КССГ и ее отображение пользователю.

КССГ сохраняется в сетевой модуль текущего проекта и открывается графическим редактором для:

– просмотра и возможной корректировки результата компиляции,

– выполнения запроса.

Технология дедуктивного ввода программ

Технология позволяет:

· Минимизировать ручной ввод (идентификаторы, константы).
· Исключить лексический и синтаксический анализ программы.
· Снизить число семантических ошибок ввода.
· Повысить читаемость текста программы.
· Использовать математические и др. специальные символы.
· Обеспечить переносимость на различные естественные языки.

Недостатки:

· Необходимость введения в грамматику дополнительных атрибутов.
· Несколько сниженная скорость ввода программ.

Возможные специальные области применения:

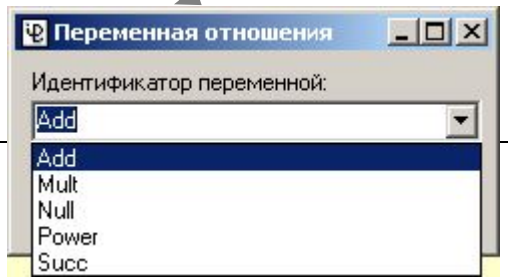
· Учебные приложения и скриптовые языки.
· Блокнотные КПК, не обладающие полноразмерной клавиатурой.
· Приложения для людей с ограниченными возможностями.

Основа технологии

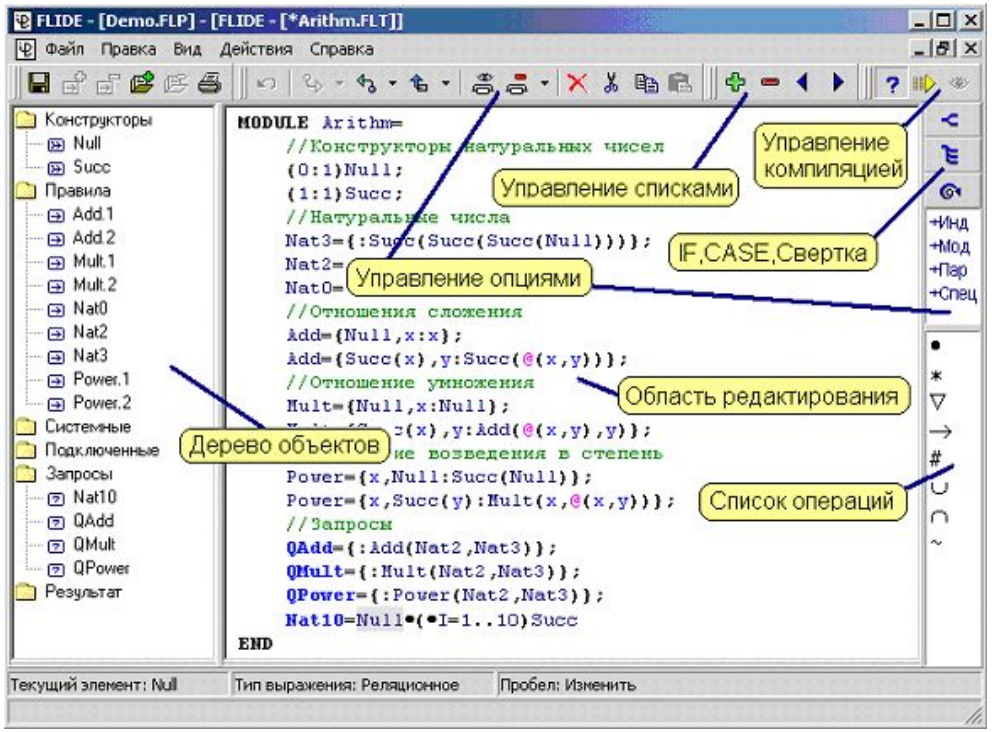
Программа – выводимая из грамматики цепочка символов.
Шаг ввода – применение правила к нетерминальному символу цепочки.
Исходное состояние – начальный нетерминальный символ грамматики.

Основные технологические приемы ввода

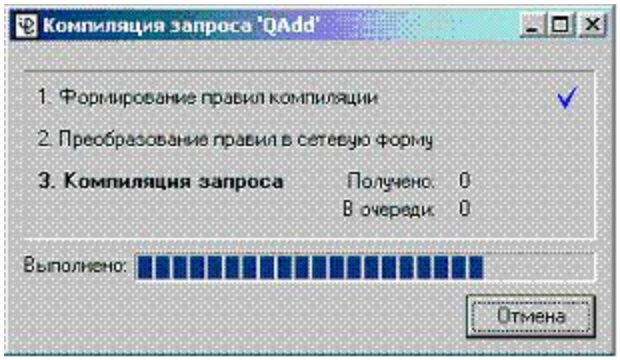
Текст программы	Альтернатива	Правила грамматики
1) **MODULE** Common= Дом **END**	*Определение*	Дом→Спец [СпИнд] Ид [СпПар]=Рел
	Конструктор	Дом→ Спец [СпИнд] Ид
	Объединение	Дом→Дом ; Дом
2) **MODULE** Common= Спец [СпИнд]Ид[СпПар]=Рел **END**	Вызов	Рел→ИмяОтн
	График	Рел→{Терм:Терм?Формула}
	Композиция	Рел→Рел РелИнф Рел
3) **MODULE** Common= Спец [СпИнд]Ид[СпПар]= {Терм:Терм?Формула} **END**	Переменная	Терм→ [СпИнд] ИдТ
	Вызов	Терм→ИмяОтн(Терм)
	Соединение	Терм→Терм , Терм
Выражения		
1) … [1]Nat=Null 2) … [1]Nat=Null·Рел	·, *, ∇, →, #, ∪, ∩, ~	Ид→Рел РелИнф Рел РелИнф→{ ·, *, ∇, →,#, ∪, ∩, ~ }
Ручной ввод		
1) …Спец Ид=Рел	*Раскрыть*	Дом→Спец [СпИнд] Ид [СпПар]=Рел
2) …Спец Add=Рел		*Элемент ручного ввода* Ид→ПравилоДляИд

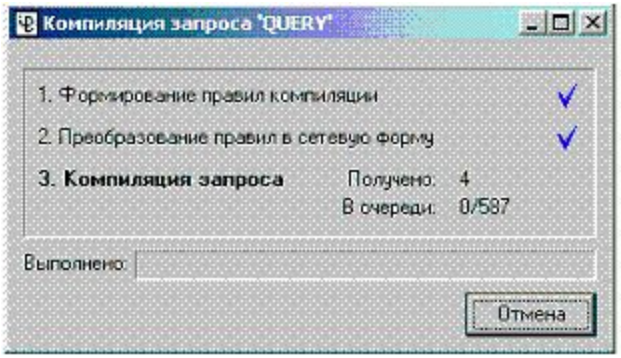


Общий вид окна редактора



Форма компиляции





Также разработаны и реализованы

- Алгоритм автоматического структурирования текста программы.
- Схема стилистического оформления текста.
- Управление детализацией отображения текста.
- Интерфейсные средства поддержки ТДВП.
- Выполнение операций по горячим клавишам.
- Типизированные буферы обмена.
- Многоуровневый откат.

```
MODULE RelComp=
    (0:1)Null;
    (1:1)Succ;
    (2:1)F;
    SecComp=
        {~«0»((,I=1..«0»>)[I]x):
            «1»((,I=1..«0»>)[I]x)
        };
    ParComp=
        {(,I=1..><0»)[I]x,(,J=1..><1»)[J]y:
            «0»((,I=1..«0»>)[I]x),
            «1»((,J=1..«1»>)[J]y)
        };
    ConcComp=
        {(,I=1..><0»)[I]x:
            «0»((,I=1..><0»)[I]x),
            «1»((,I=1..><1»)[I]x)
        };
```

Результаты работы

1. Предложена система ограничений языка FLOGOL, позволяющая выполнять компиляцию в конечную КССГ и значительно упрощающая реализацию.

2. На основе системы ограничений разработан язык S-FLOGOL, формально описаны его синтаксис и семантика.

3. Разработан метод компиляции запросов S-FLOGOL-программ. На основании формального описания семантики определены функции трансляции для всех синтаксических конструкций языка.

4. Предложены оптимизационные модификации, позволившие существенно сократить время компиляции запросов.

5. Предложена оригинальная технология дедуктивного ввода программ (ТДВП), позволяющая полностью исключить синтаксические ошибки и значительно сократить количество семантических ошибок при вводе программ.

6. Разработаны архитектура и интерфейсные средства структурно-ориентированного редактора, реализующего ТДВП.

7. Выполнена программная реализация структурно-ориентированного редактора и компилятора запросов и их интеграция в СФЛП.