FLIDE

FLOGOL Integrated Development Environment

Система
функционально-логического
программирования
на языке S-FLOGOL

Московский институт радиотехники, электроники и автоматики (МИРЭА ТУ)

Бебчик Алексей Михайлович

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

Цель работы:
создание системы функционально-логического программирования (СФЛП) на основе формализма направленных отношений (НО), предназначенной для решения задач искусственного интеллекта и для учебных целей .

Основные задачи:
· исследование языка FLOGOL и формальное описание его семантики,
· разработка основных принципов и метода компиляции FLOGOL-запросов,
· разработка специальной технологии и интерфейсных средств ввода программ,
· программная реализация и интеграция в СФЛП компилятора запросов и структурно-ориентированного редактора FLOGOL-программ.

Московский институт радиотехники, электроники и автоматики (МИРЭА ТУ)

Бебчик Алексей Михайлович

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

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.

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:

MODULE Имя (Имя$_1$, …, Имя$_n$) =

Модуль$_3$

Описания НО

Открытая часть

WHERE

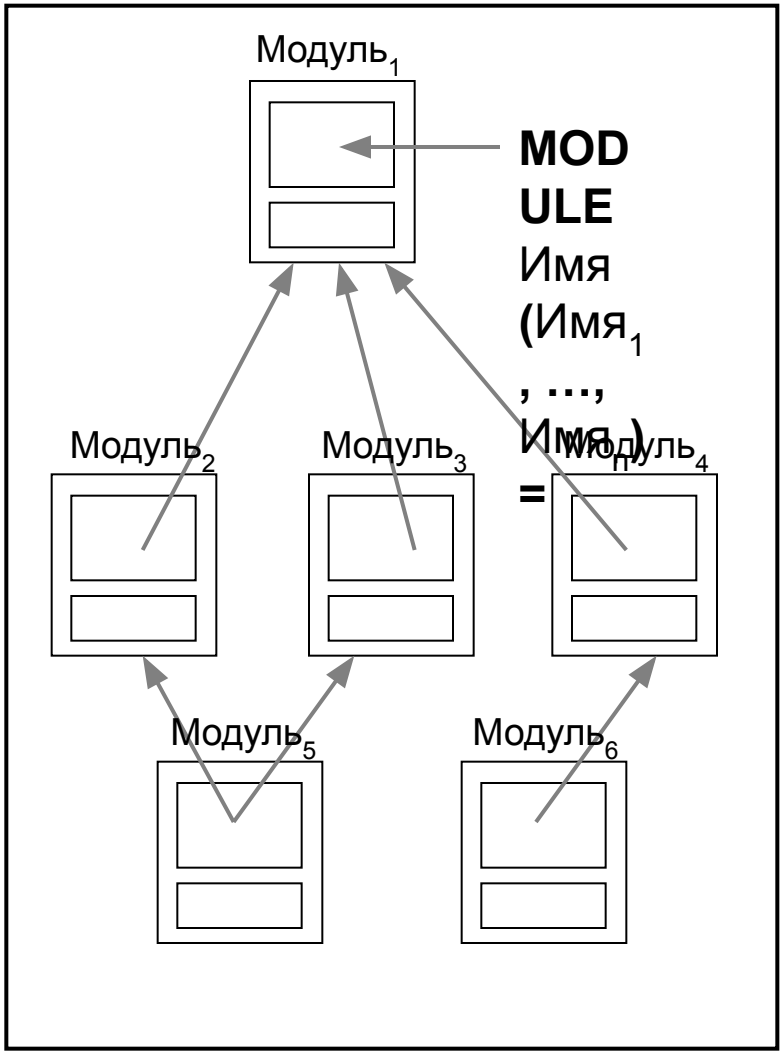
Модуль$_5$

Описания НО

Закрытая часть

END

Программа:



Пример программы (модуль “Arithm”)

```
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

описания НО

Сетевая интерпретация программ

Программа:

OBJECT Null;

CONSTRUCTOR(1)Succ;

Nat1=Null·Succ;

Nat2=Null·Succ·Succ;

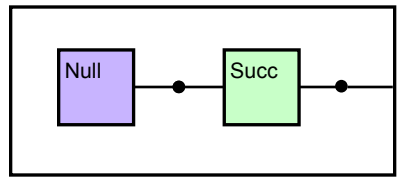
Add={Null,x:x};

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

QUERY=(Nat1#Nat2)·Add

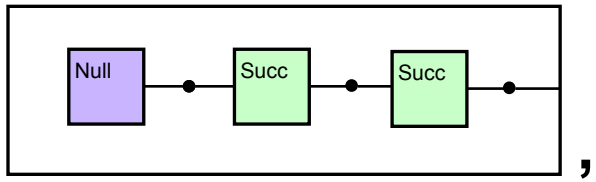
Правила КССГ $R =$

Nat1 →



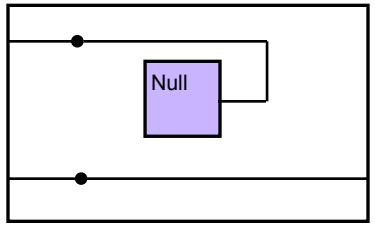
,

Nat2 →



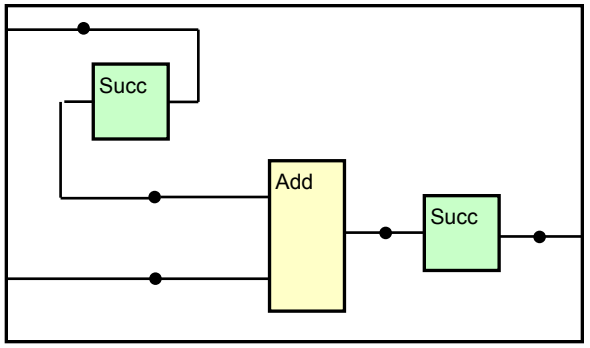
,

Add →



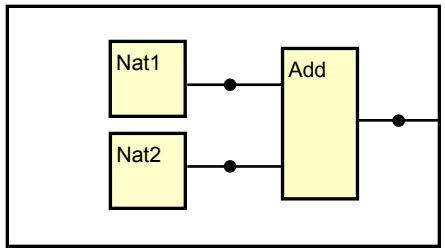
,

Add →



,

Query →



.

Базис КССГ:

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

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

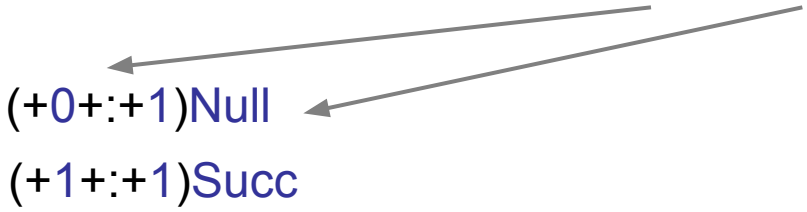
Аксиома КССГ:

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

Описания НО

Описания НО задаются *доменными выражениями* (Дом):

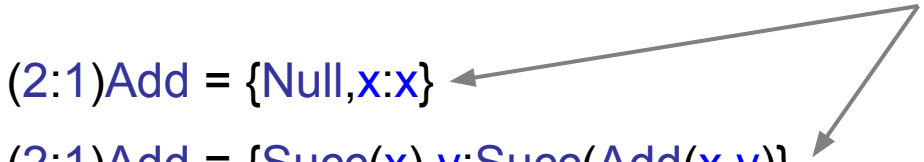
· Объявление НО: Дом → Спец Имя



Ограничение

Исключена внешняя пользовательская интерпретация.

· Определение НО: Дом → Спец Имя = Рел



· Объединение описаний НО: Дом → Дом ; Дом

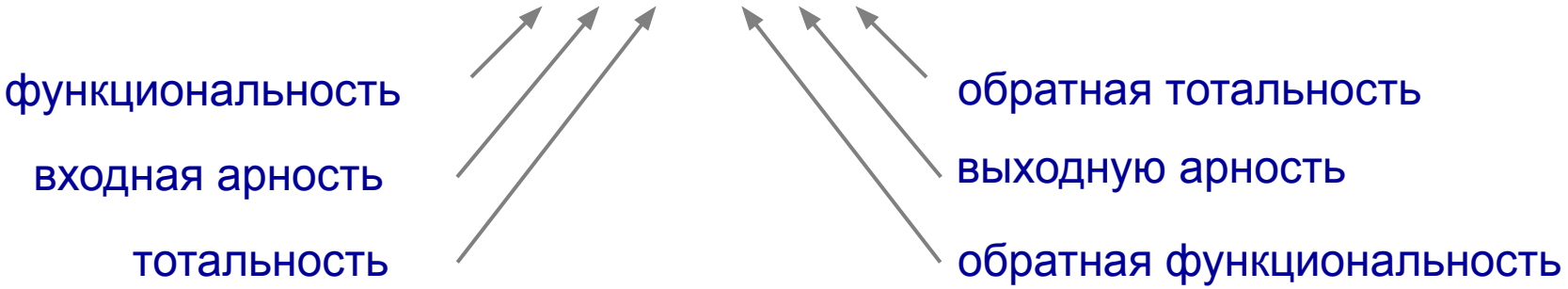
(+0+:+1)Null;
(+1+:+1)Succ;
(2:1)Add = {Null,x:x};
(2:1)Add = {Succ(x),y:Succ(Add(x,y)}

⇔

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

Спецификация НО

Общая форма: Спец →(+ Ар + : + Ар +)



Ключевая спецификация:

Ключевое слово	Спецификация
OBJECT	(+0+:+1)
CONSTRUCTOR(*n*)	(+*n*+:+1)
OPERATOR	(1+:1)
PREDICATE(*n*)	(+*n*:0)
FUNCTION(*n*)	(+*n*+:1)

Примеры:

(+0+:+1)Null

OBJECT Null

(+1+:+1)Succ

CONSTRUCTOR(1)Succ

Полиморфизм: (2:1)Add

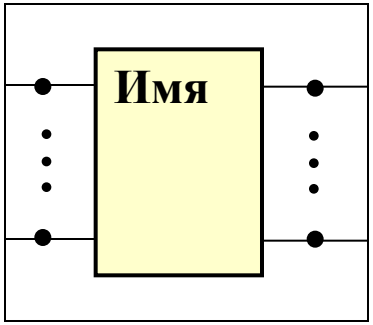
(3:0)Add

Неявная спецификация определений: (2:1)Add={Succ(x),y:Succ(Add(x,y)} ∪ {Nil,x:x}

Определение НО

Определение НО задается *реляционным выражением* (Рел):

- Вызов по имени: Рел → Имя
- Рекурсивный вызов: Рел → @
- Вызов параметра: Рел → <<Ар>>



- График (аппликативная форма):

Рел → {Терм : Терм ? Формула}

Входной терм – аргументы вызова.

Выходной терм – результат вызова.

Формула – ограничения на интерпретацию переменных графика.

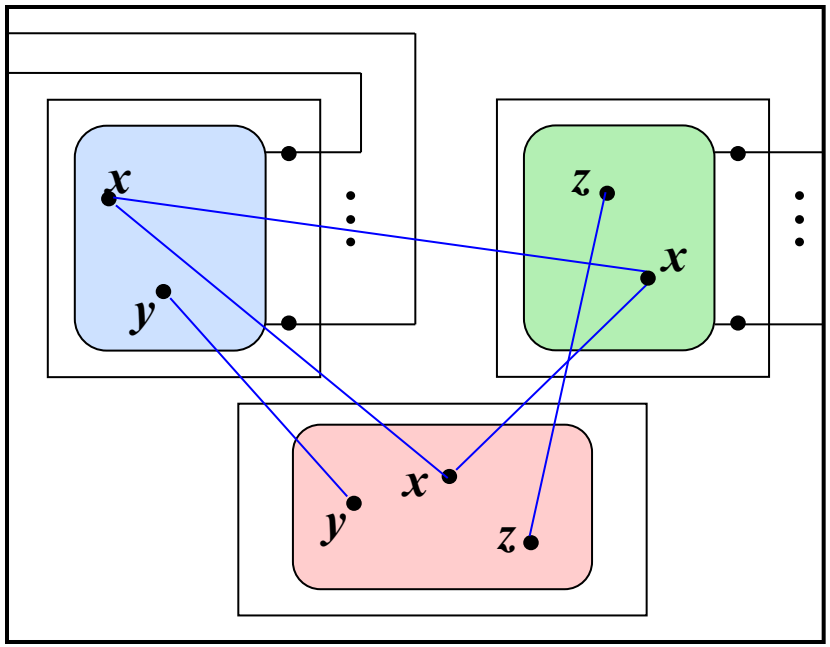


График: терм и формула

· Терм:

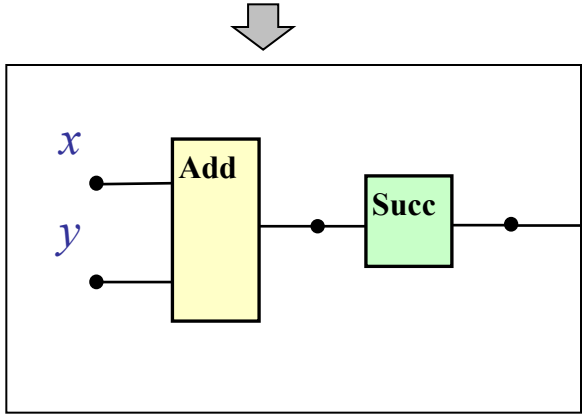
Переменная

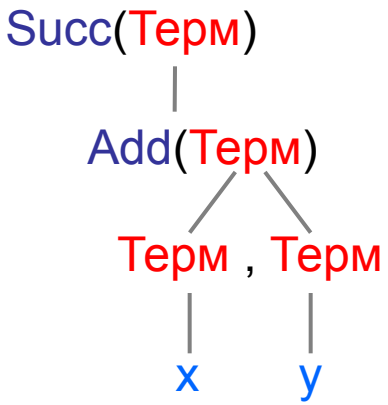
Имя(Терм)

Терм , Терм

Терм | Терм

Пример: Succ(Add(x,y))





· Формула:

Терм=Терм

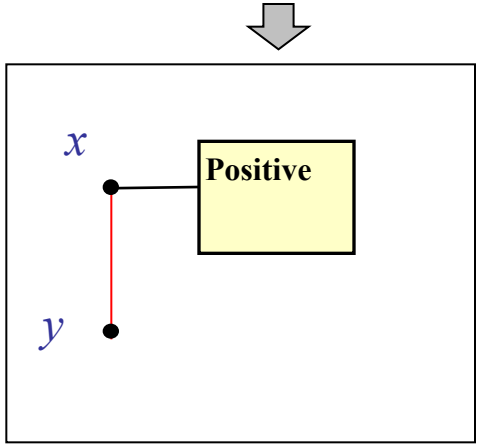
Терм<>Терм

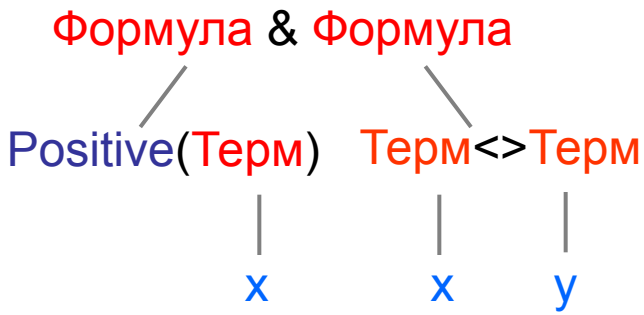
Имя(Терм)

Формула & Формула

Формула | Формула

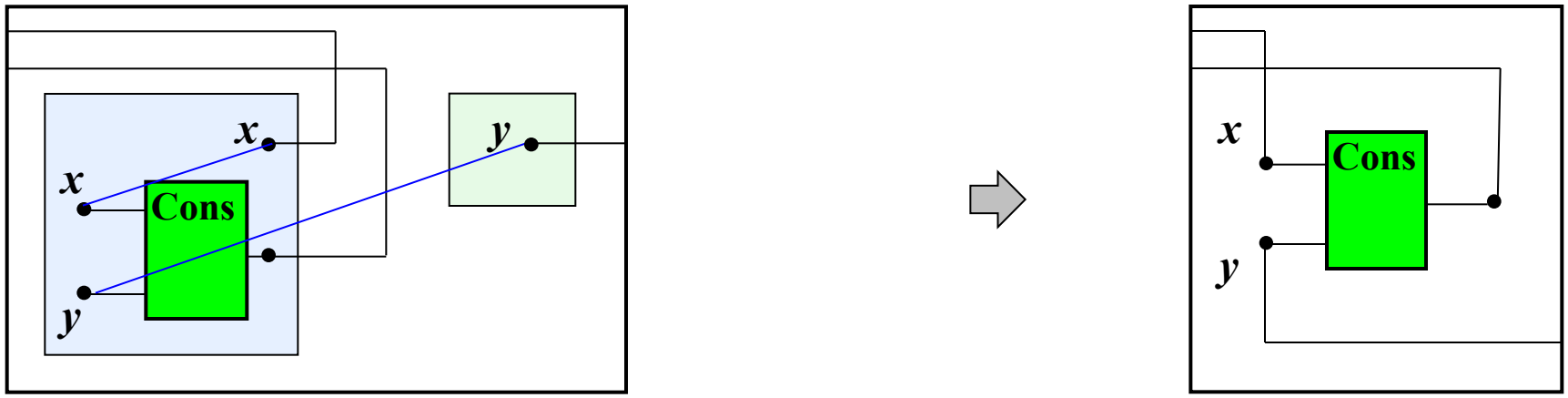
Пример: Positive(x) & x<>y



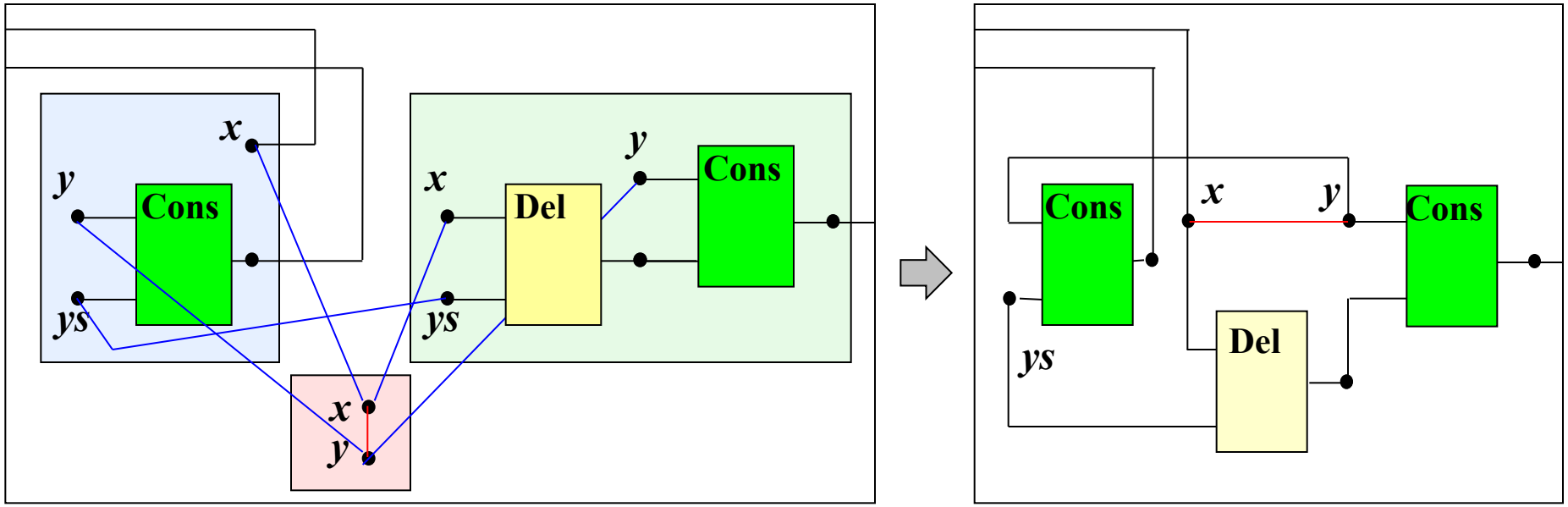


Пример. Определение НО «Del»

Del = {x,Cons(x,y):y}



Del = {x,Cons(y,ys):Cons(y,Del(x,ys))?x<>y}

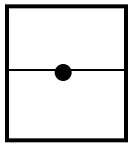


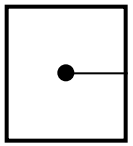
Композиционное программирование

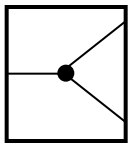
· Композиция отношений: Рел → Рел ⊗ Рел, где ⊗ ∈ {·,#,→,∇,*,∪,∩}

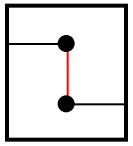
· Инверсия отношения: Рел → ~Рел

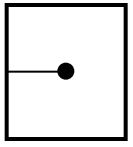
· Комбинаторные константы:

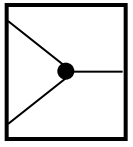
--- = {x:x} 

<-- = {:x} 

--< = {x:x,x} 

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

--> = {x:} 

>-- = {x,x:x} 

Пример композиционного определения НО сложения:

OBJECT Null;

CONSTRUCTOR(1) Succ;

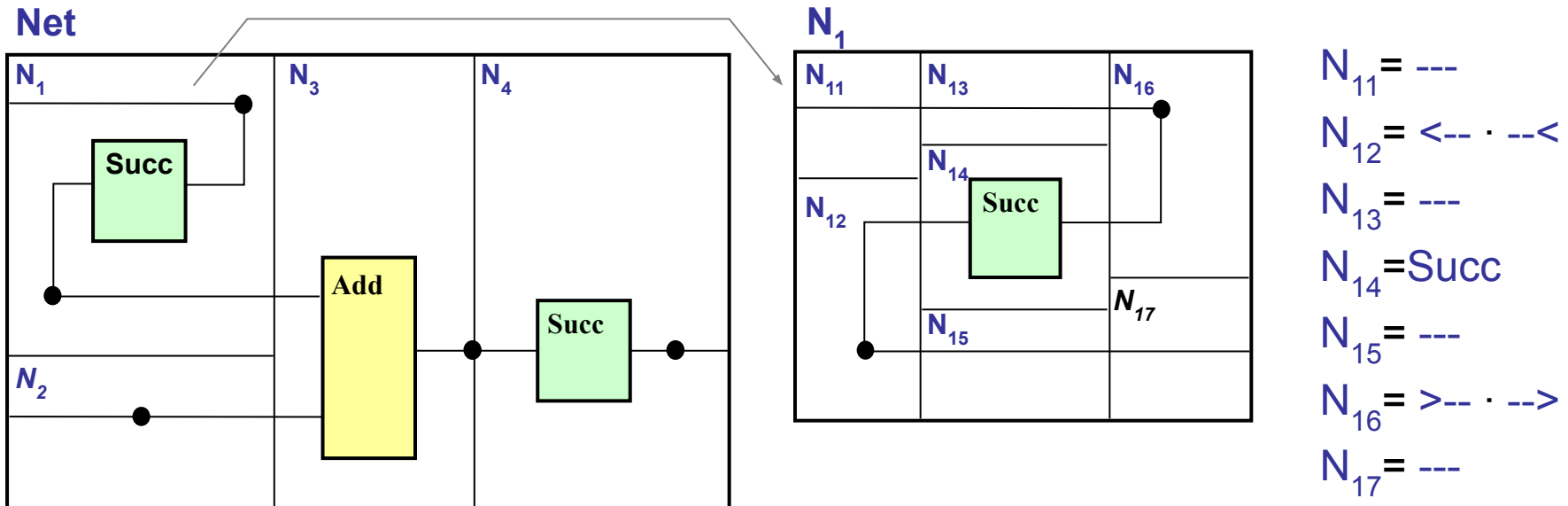
Add = (--- # Null # ---)·(>-- # ---)·(>-- # ---);

Add = ((--- # <-- · --<)·(--- # Succ # ---)·(>-- · --> # ---)# ---)·Add·Succ;

или Add = (~Succ # ---)·Add·Succ **U** (~N # ---);

Композиционное программирование (продолжение)

Преобразование сети:



N_{11}= ---

N_{12}= <-- · --<

N_{13}= ---

N_{14}=Succ

N_{15}= ---

N_{16}= >-- · -->

N_{17}= ---

$Net=(N_1 \# N_2)\cdot N_3 \cdot N_4$

$N_1=(N_{11} \# N_{12})\cdot(N_{13} \# N_{14} \# N_{14})\cdot(N_{16} \# N_{17})$

N_2= ---

N_3=Add

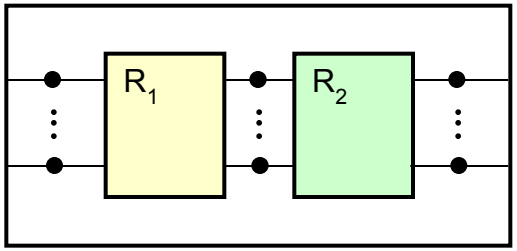
N_4=Succ

Net = ((--- # <-- · --<)·(--- # Succ # ---)·
(>-- · --> # ---)# ---)·Add·Succ

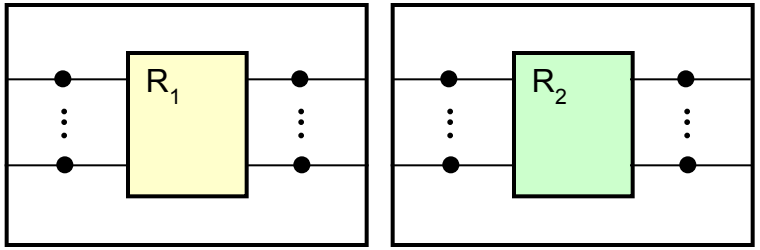
или Net = (~Succ # ---)·Add·Succ

Операции композиции НО

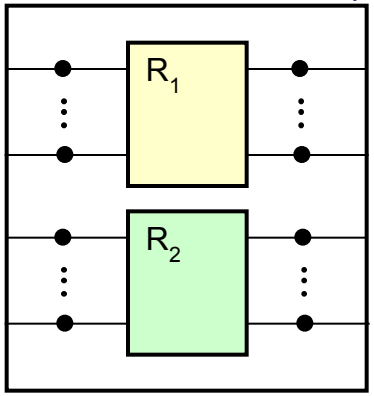
Последовательная: $R_1 \cdot R_2$



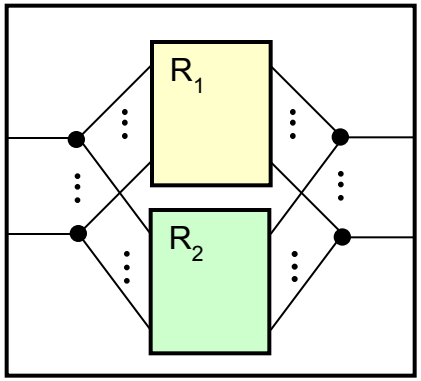
Объединение: $R_1 \cup R_2$



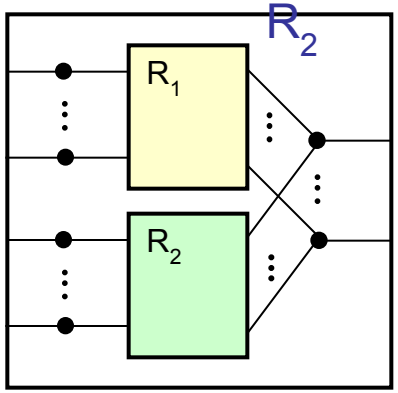
Параллельная: $R_1 \# R_2$



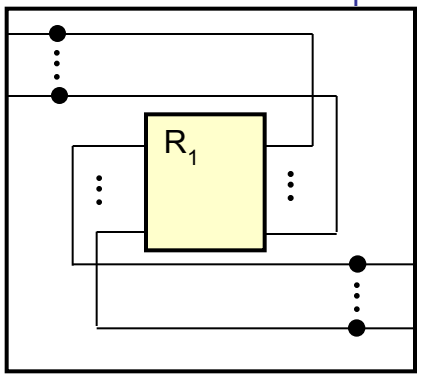
Пересечение: $R_1 \cap R_2$



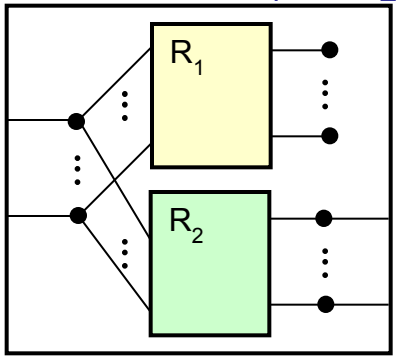
Унификация: $R_1 \nabla R_2$



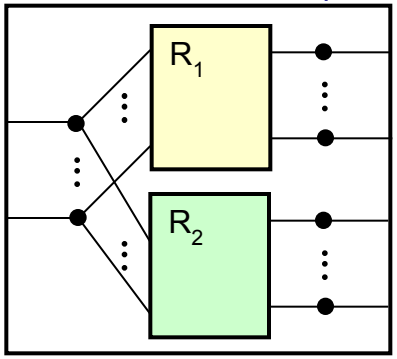
Инверсия: $\sim R_1$



Условная: $R_1 \rightarrow R_2$



Конкатенация: $R_1 * R_2$



Параметризованные описания НО

Список параметров:

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

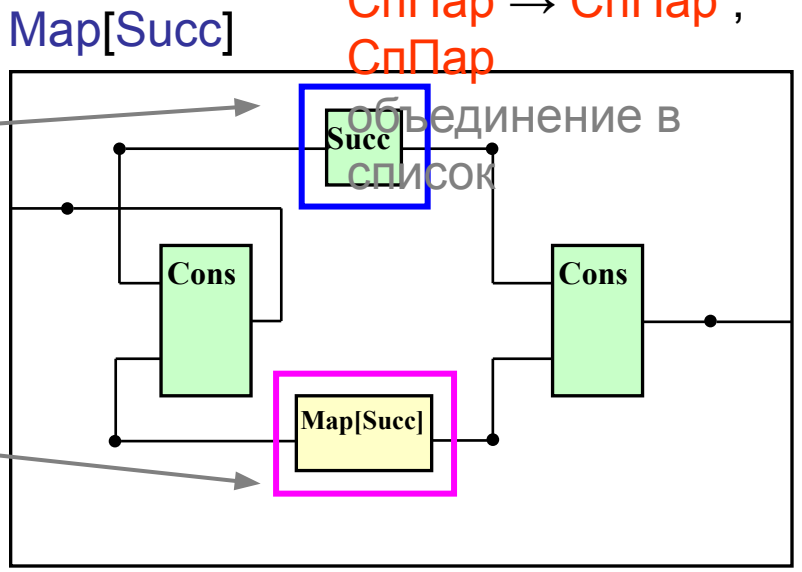
Вызов параметризованного НО: Имя[СпПар]

Вызов параметра в определении: <<Ар>>

*Значения по умолчанию: Имя[СпПар] = Рел

Пример параметризованного НО:

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



Ограничения:

1) Нет групп параметров
СпПар → Имя
полное имя НО
3) Нет СпПар→График
СпПар → @
собственные параметры
СпПар → _
пропуск параметра
СпПар → СпПар ; СпПар
объединение в список

Условная конструкция IF

Выр → IF Лог THEN Выр ELSE Выр

Логическое выражение: Лог → Ар АрСр Ар, где АрСр ∈ {=, <>, >, <, >=, <=};

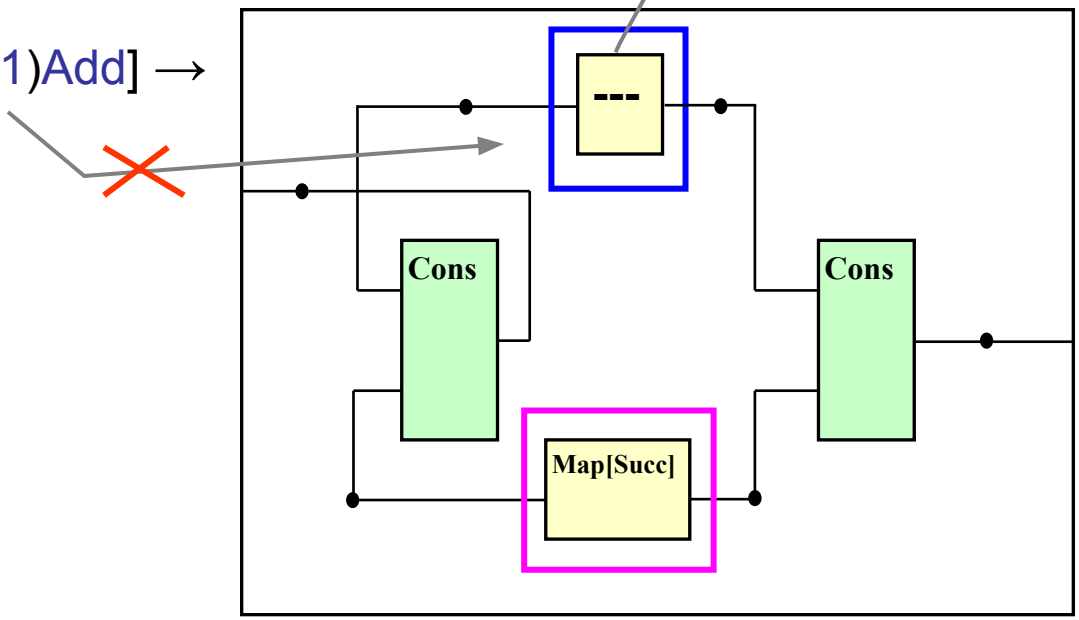
Лог → Ар ЛогСр Ар, где ЛогСр ∈ {AND,OR,XOR}.

Пример (контроль арностей параметра):

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

QUERY = Map[(2:1)Add];
```

Map[(2:1)Add] →



Индексированные имена

Имя отношения:

Имя → Спец [СпИнд] Ид СпПар

Имя переменной терма:

Переменная → [СпИнд] Ид

Список индексов:

СпИнд → Ар

СпИнд → СпИнд, СпИнд

Натуральные числа:

[0]Nat= Null;

[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

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

Свертка

Выр → (Инф ИдСв = СпЗнач) Выр

Ид – операторная переменная свертки.

Инф – инфиксная связка.

СпЗнач – список значений переменной.

Выр – выражение-операнд свертки.

Ограничение

Нет пользовательских связок для реляционного выражения.

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

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

Примеры:

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;

Свертка: моделирование связок

Операции над массивами объектов:

...
[1,1]Cell= [1]Nat;
[1,2]Cell= [4]Nat;
[2,1]Cell= [2]Nat;
[2,2]Cell= [10]Nat;

пользовательская связка

FLOGOL:
SummCells = (ADD I = 1..2)(ADD J = 1..2)[I,J]Cell

комбинирующая форма

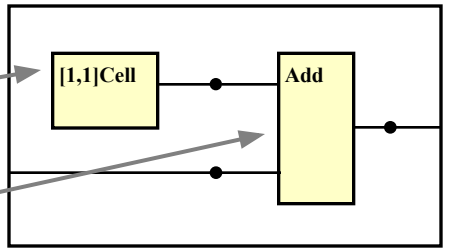
F = (<<0>># ---)·<<1>>;

SummCells = Null·(·I = 1..2)(·J = 1..2)F[[I,J]Cell;Add];

<<0>>

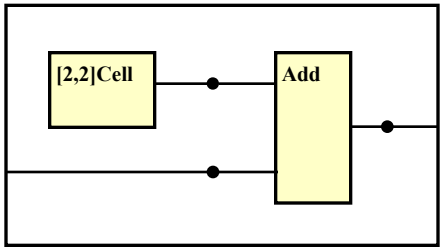
<<1>>

F[[1,1]Cell;Add]

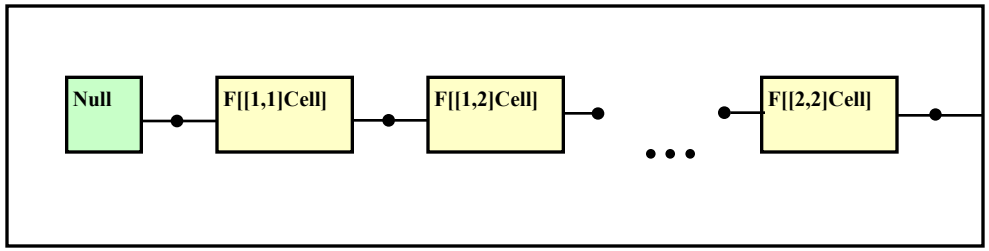


...

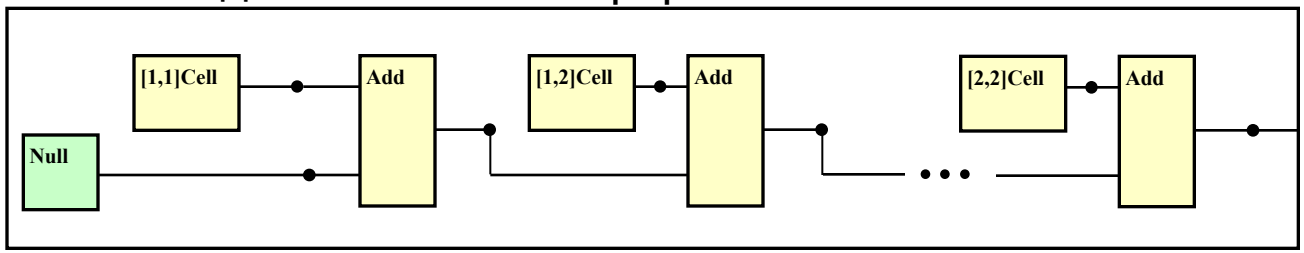
F[[2,2]Cell;Add]



SummCells



после подстановки комб. форм



Типизация

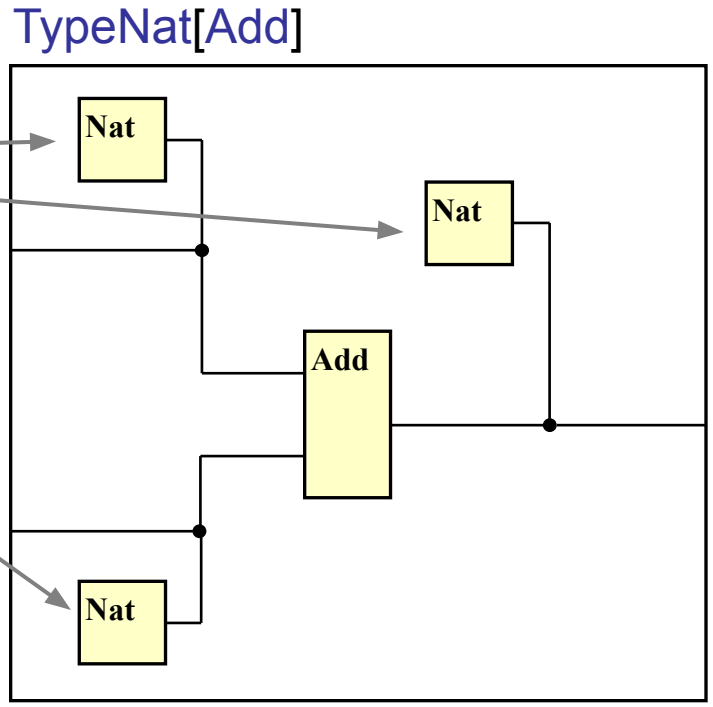
Типовое отношение для типа $T = (t'_1,\ldots,t'_{n'} \to t''_1,\ldots,t''_{n''})$, где $t'_i(t''_i)$ – сорта входных(выходных) данных, определяется как:

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

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

Пример (типизация 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

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 и имя вызова записываются в виде списков.
  elm(*Список*, *Номер*) – выборка элемента списка с указанным номером,
  append(Список, Список) – конкатенация списков,
  [ ] – пустой список.

Семантика основных конструкций

Область интерпретации выражений:

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

Семантика определения НО в доменном выражении:

Φ_{φ} □*Спец* e_1 *Ид* $e_2 = e_3$ □ $= R$, 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

$Name_{dom} = [\Phi_{\varphi}$ □*Спец*□ $, \Phi_{\varphi}$ □e_1 □$, k(Ид)],$ 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

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

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

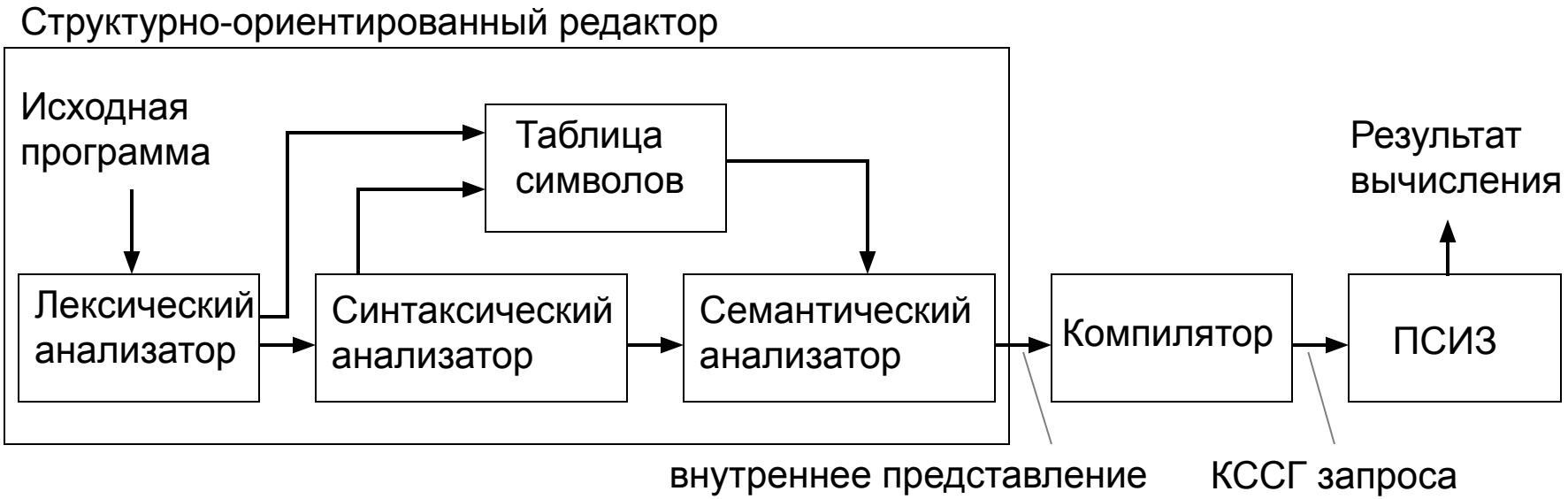
$R = R_1 \cup R_2.$ формирование результата

Реализация языка S-FLOGOL

Компилятор запросов

Реализация языка S-FLOGOL

Смешанная форма = компиляция + интерпретация



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).

Оригинальная технология ввода позволяет:

· исключить лексический и синтаксический анализ,

· обеспечить синтаксическую корректность и однозначность разбора,

· выполнить первичный семантический анализ.

Стадии компиляции

· Предварительная стадия

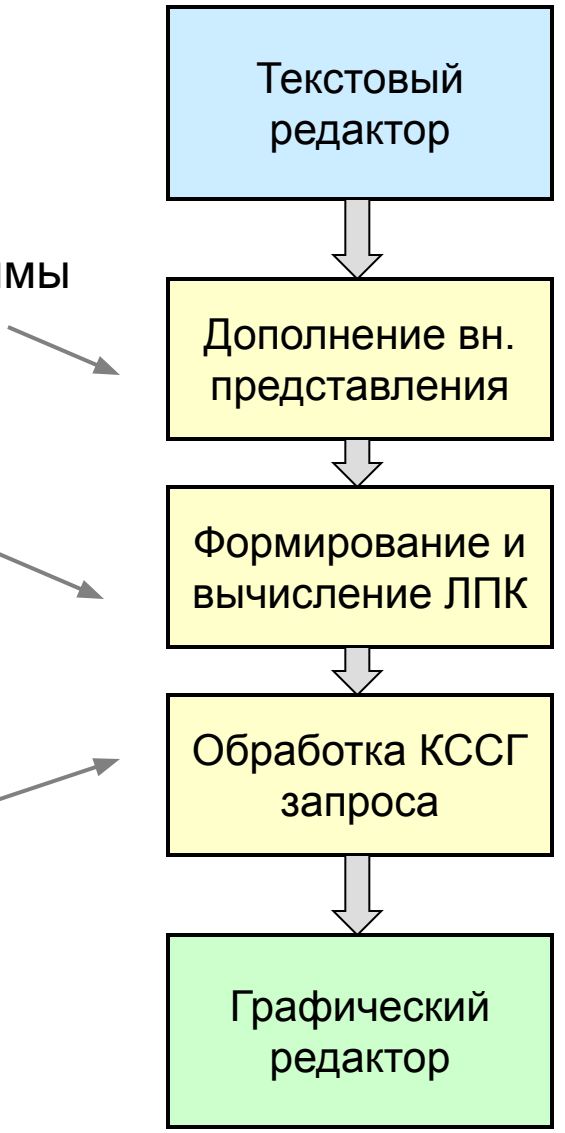
Завершение формирования и дополнительная обработка внутреннего представления программы

· Основная стадия

Формирование и вычисление логической программы-компилятора (ЛПК), результатом которого является КССГ запроса

· Заключительная стадия

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:

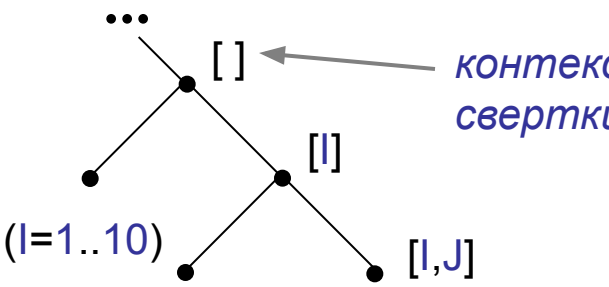
- контроль вызова не объявленных НО,
- чистка грамматики (исключение не используемых НО).

· Индексирование вхождений операторных переменных сверток:

Программа

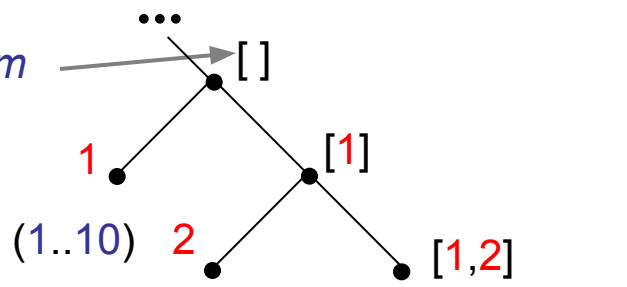
```
…
(I=1..10)
   (J=1..20)
      [I+J]R = [I]P·[J]P
…
```

Дерево



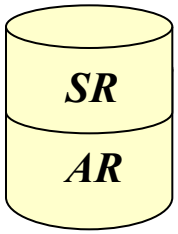
контекст свертки

Индексированное дерево



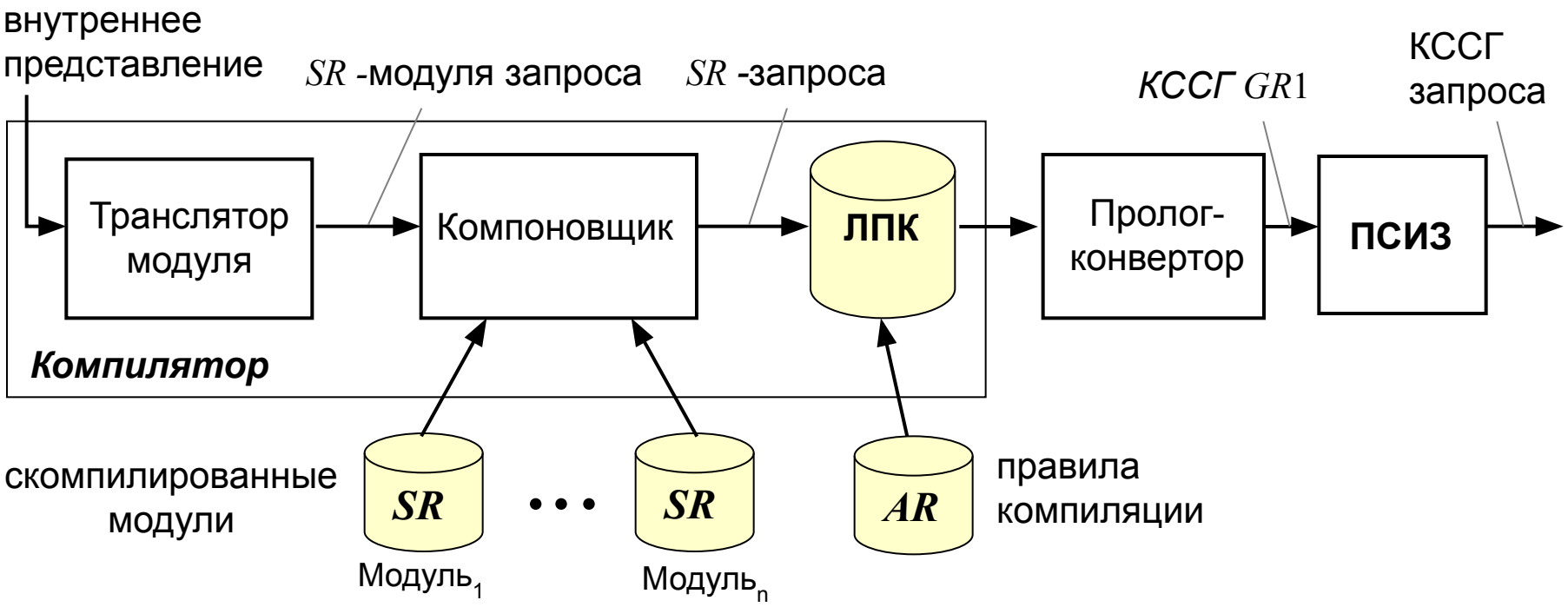
Основная стадия компиляции

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:

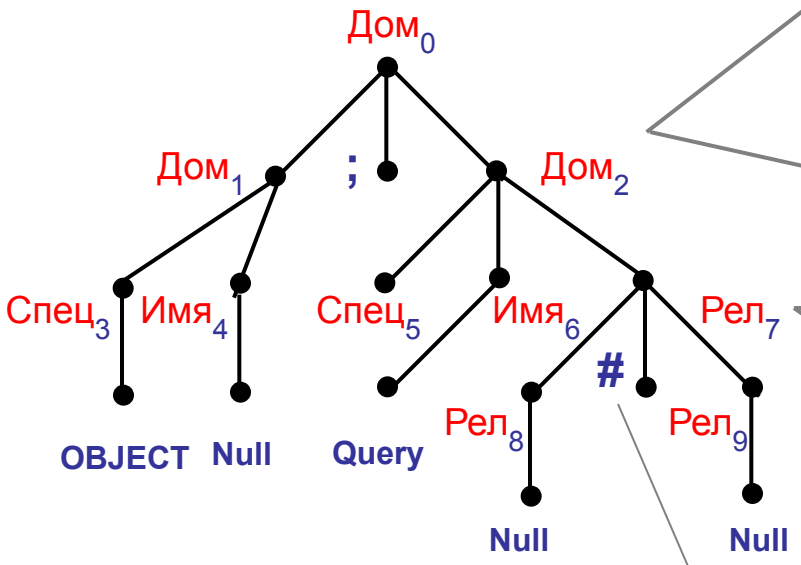


База описаний НО (*SR*-правила)

Программа:

OBJECT Null;
QUERY = Null#Null

Внутреннее представление:



SR-БД:

```
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).
```

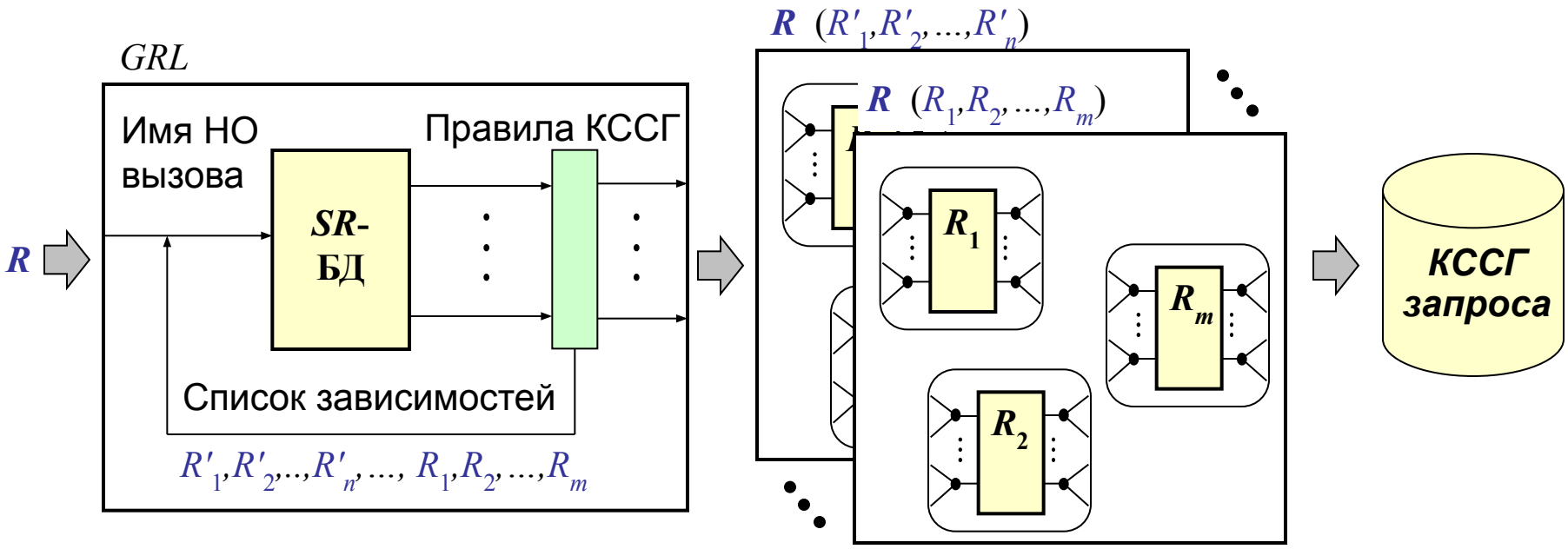
Контекст вызова:
- список значений переменных свертки;
- имя вызываемого отношения.

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.

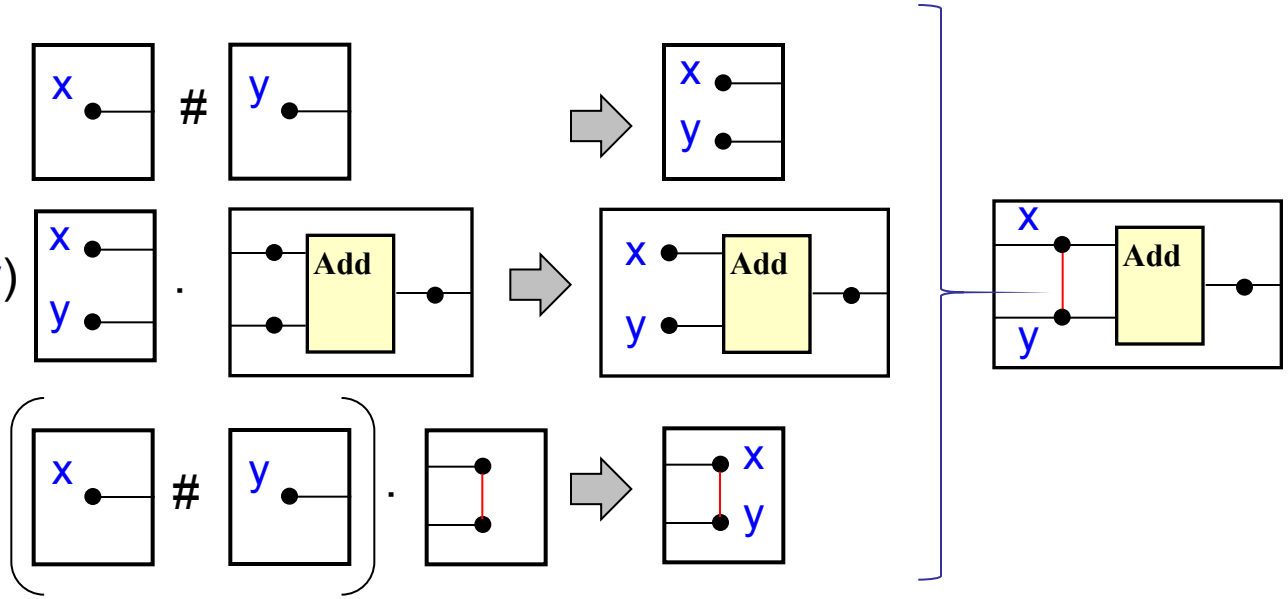
Формирование сети

· Формирование графика Пример: {x, y: Add(x,y)?x<>y}

Входной терм = x, y

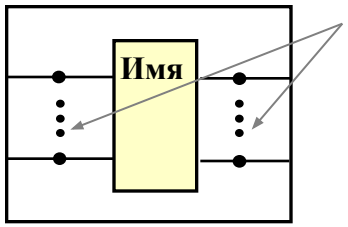
Выходной терм = Add(x,y)

Формула = x <> y



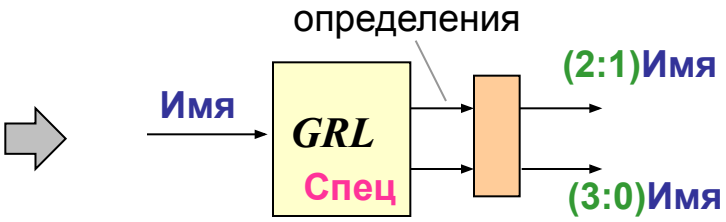
· Формирование вызова НО

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



? необходимо знать арность

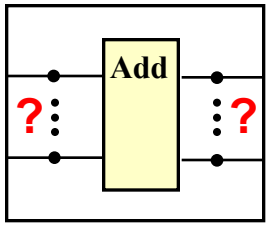
Решение:
получить определение вызываемого НО, извлечь информацию о его спецификации.



Оптимизация получения спецификаторов

· Исключение дублирующих спецификаторов.

Вызов НО Add:

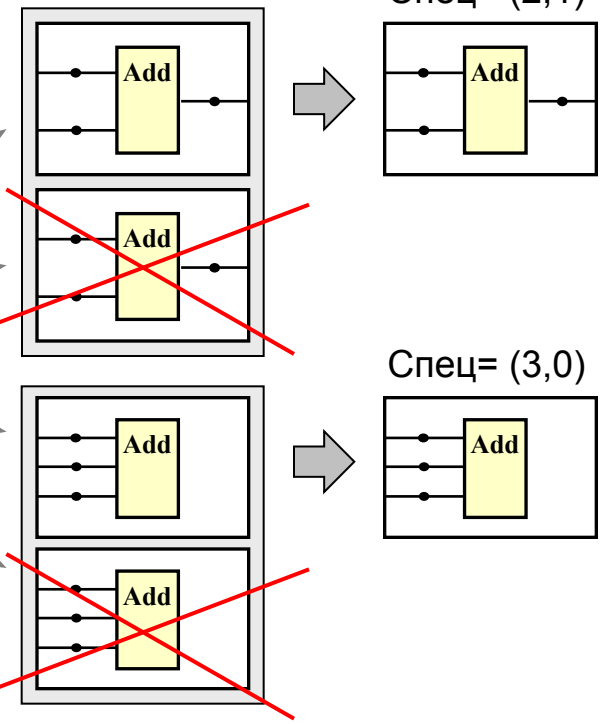


Add={Null,x:x}

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

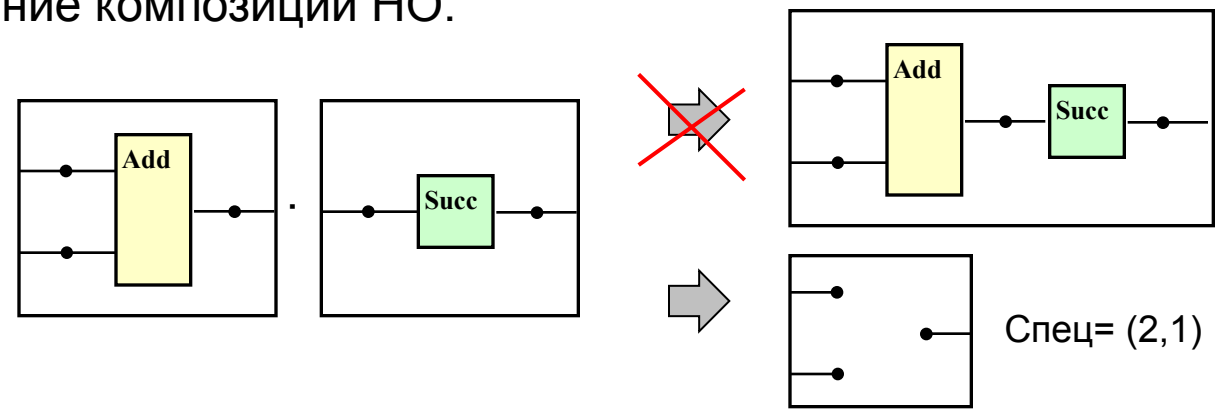
Add={Null,x,x:}

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



· Частичное выполнение композиций НО.

Спец? (Add·Succ):



SimpleNet – язык текстового описания КССГ

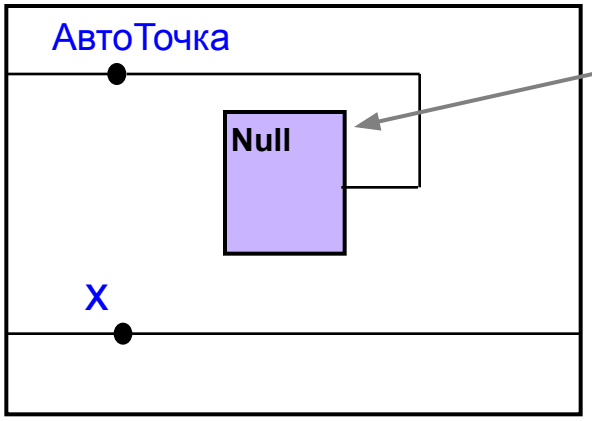
Причина создания языка: чрезвычайно сложный для конструктивного построения формат внутреннего представления КССГ в ПСИЗ.

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]],Bx,[276],1,[2,1,[280]]),
   сДуга([2,285],Bx,[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.

Интерфейсные средства

Структурно-ориентированный текстовый редкатор

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.

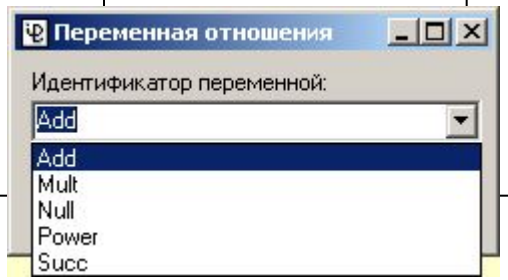
Текст программы	Альтернатива	Правила грамматики
1) **MODULE** Common= Дом **END**	*Определение*	Дом→Спец [СпИнд] Ид [СпПар]=Рел
	Конструктор	Дом→ Спец [СпИнд] Ид
	Объединение	Дом→Дом ; Дом
2) **MODULE** Common= Спец [СпИнд]Ид[СпПар]=Рел **END**	Вызов	Рел→ИмяОтн
	График	Рел→{Терм:Терм?Формула}
	Композиция	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

Специализированный ввод отдельных конструкций

Фрагмент текста программы:	Операция:	Правила грамматики:
Выражения		
1) … [1]Nat=Null 2) … [1]Nat=Null·Рел	· , *, ∇, → , #, ∪, ∩, ~	Ид→Рел РелИнф Рел РелИнф→{ · , *, ∇, → ,#, ∪, ∩, ~ }
Опциональные элементы		
1) … (2:1) [СпИнд]Add[СпПар]=Рел 2) … (2:1) Add[СпПар]=Рел 3) … (2:1) [СпИнд]Add[СпПар]=Рел	*скрыть* *отобразить опции* *восстановить*	Дом→Спец [СпИнд]Ид[СпПар]=Рел *опциональные элементы*
Обработка списков		
1) **MODULE** Cmn(System)= … 2) **MODULE** Cmn(System,ИмяМод)= …	*добавить* *удалить* *в начало* *в конец*	Модуль→ИмяМод(ИмяМод)= … *список с разделителем ‘,’*

Ввод идентификаторов и числовых констант

Текст программы	Альтернатива	Правила грамматики
1) …Спец Ид=Рел	*Раскрыть*	Дом→Спец [СпИнд] Ид [СпПар]=Рел
2) …Спец Add=Рел		*Элемент ручного ввода* Ид→ПравилоДляИд

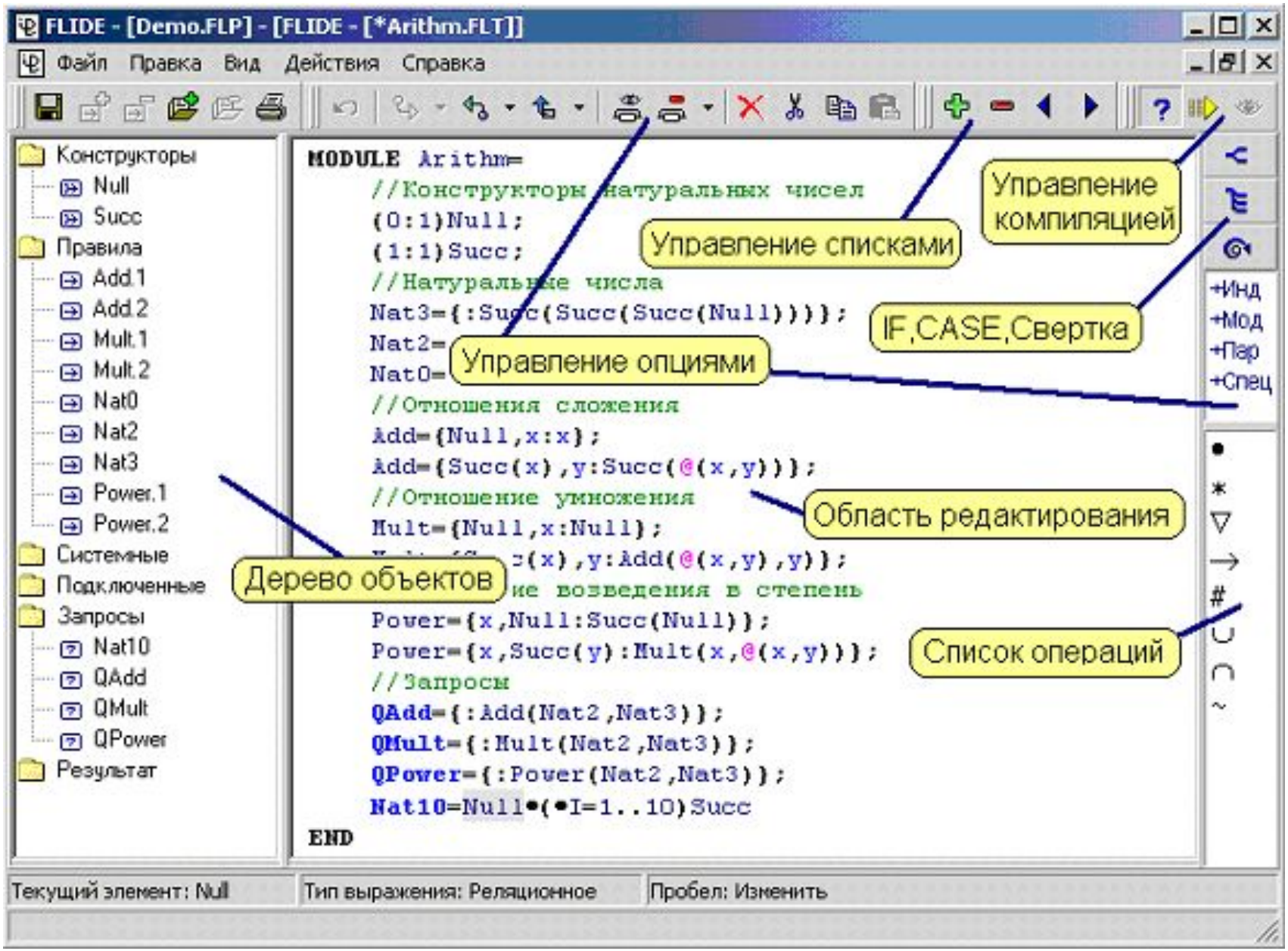


Также разработаны и реализованы:

- 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)
        };
```

Общий вид окна редактора



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.

По теме диссертации опубликовано 9 печатных работ.

Спасибо за внимание!