

# Автоматизированная реализация механизмов композиции в предметно-ориентированных языках

Андрей Бреслав

СПбГУ ИТМО

2010

# Предметно-ориентированные языки (ПОЯ)

**Языки общего назначения  
(ЯОН/GPL)**

Позволяют решать  
*любые* задачи

Java, C/C++, C#, Ruby

**Предметно-ориентированные языки (ПОЯ/DSL)**

Позволяют решать  
некоторые задачи  
*просто*

SQL, make, YACC,  
Regular expressions

# Средства автоматизации разработки ПОЯ

## ПОЯ

- **ПОНЯТИЯ** из предметной области
- **СВЯЗИ** между ними

Для разработки достаточно генератора трансляторов:

- YACC, ANTLR, ASF+SDF

## Применение в реальных проектах

- **Среда разработки**
  - xText
  - Fujaba
  - MPS
- **Модульность и повторное использование**
  - ???

# Постановка задачи

- Автоматизировать поддержку мощных **механизмов композиции**, обеспечивающих модульность и повторное использование
  - Шаблоны
    - Аналогично макроопределениям в C или Lisp
    - **Корректность проверяется статически**
  - Аспекты
    - Аналогично механизмам в AspectJ



# Пример языка с шаблонами

```
funDef :  
  type ID '('  
    (var (',' var)*)?  
  ')';
```

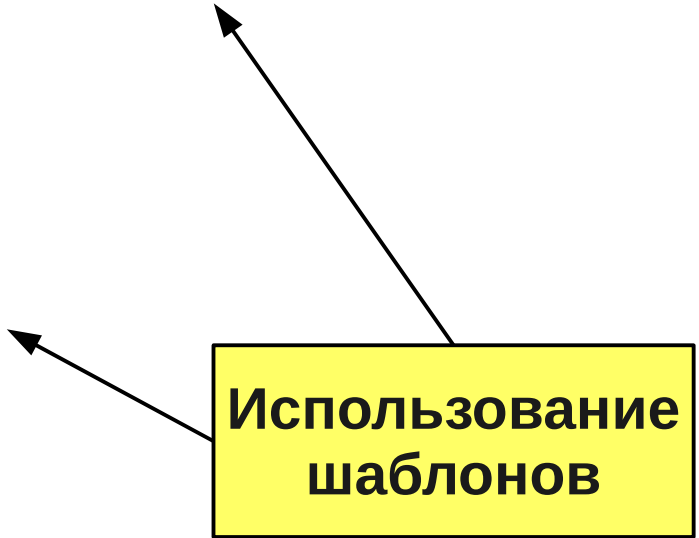
```
funDef :  
  type ID '('  
    <List var, ','>  
  ')';
```

```
funcCall :  
  ID '('  
    (expr (',' expr)*)?  
  ')';
```

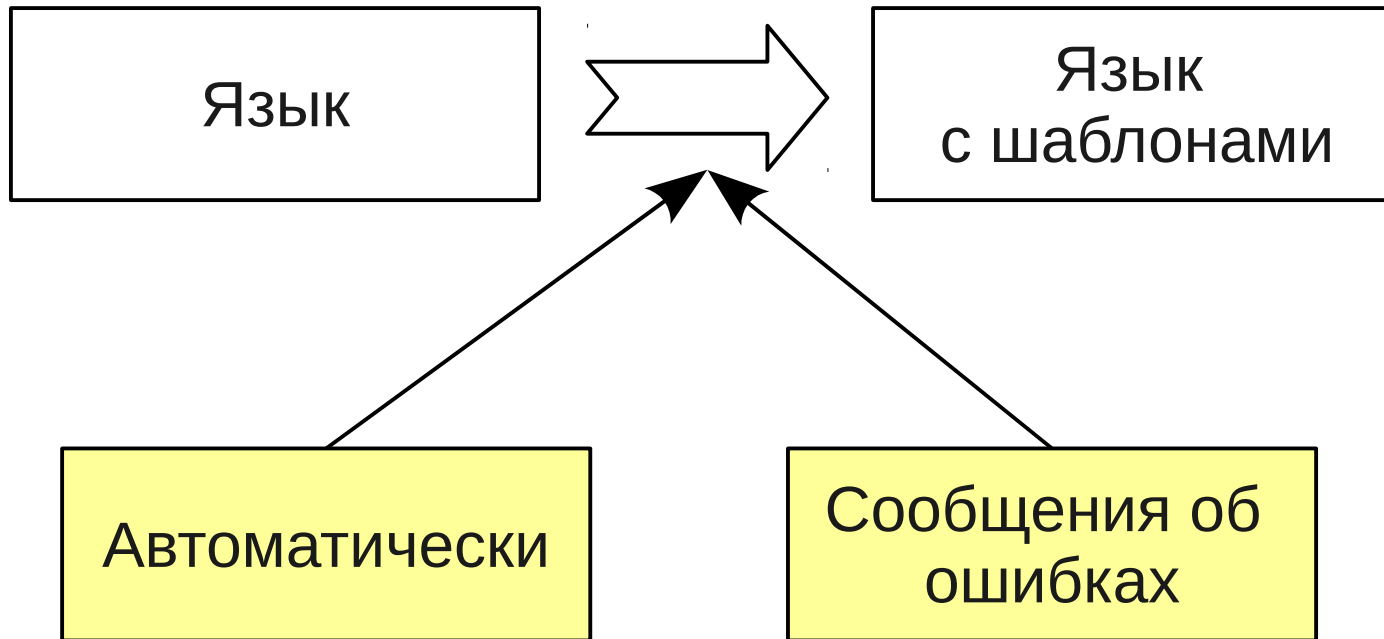
```
funcCall :  
  ID '('  
    <List expr, ','>  
  ')';
```

```
template List <item, sep> {  
  (<?item> (<?sep> <?item>)*)?  
}
```

Использование  
шаблонов

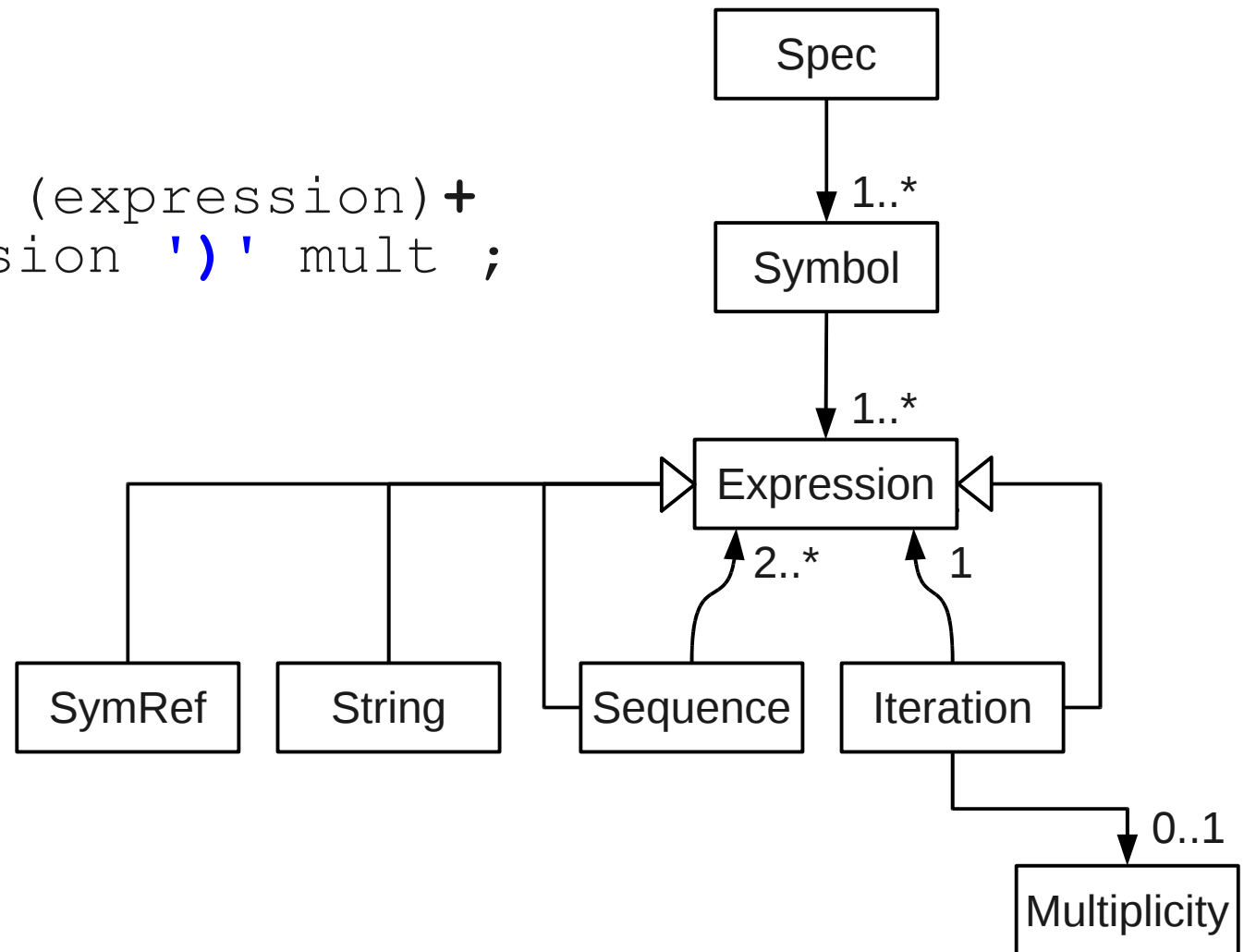


# Цель



# Грамматика исходного языка

```
spec : (symbol ';' )+ ;  
symbol : ID (':' expression)+ ;  
expression  
  : ID  
  : STRING  
  : expression (expression)+  
  : '(' expression ')' mult ;  
mult  
  : '+'  
  : '*'  
  : '?' ;
```



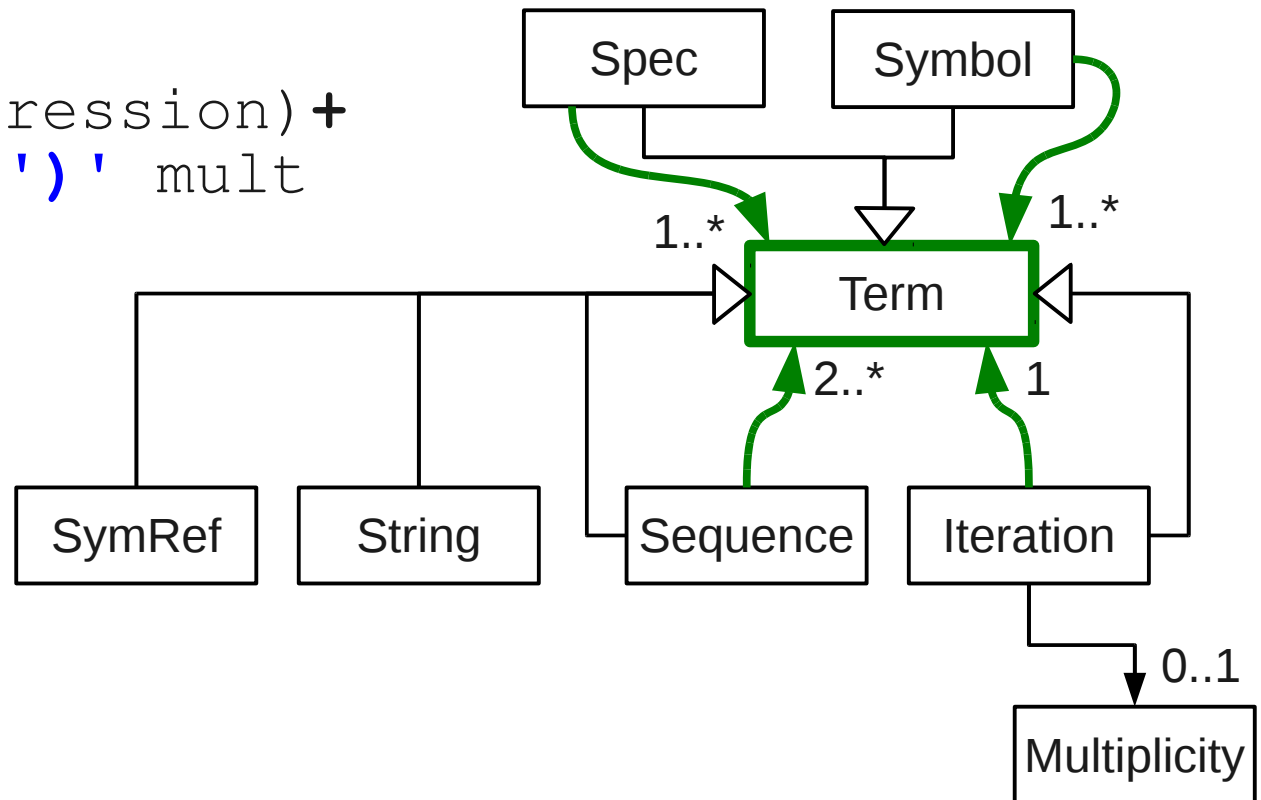
# Грамматика общего языка шаблонов

```
abstraction
    : 'template' NAME (var (',' var)*)? type?
      '{' term '}' ;
var : NAME type?;
type : ':' typeName ('?' | '*' | '+')?;
typeName
    : basicType
    : <?domainSpecificTypes> ;
basicType : 'Integer' | 'String'
           | 'Boolean' | 'Character' ;
term
    : genericTerm
    : <?domainSpecificTerms> ;
genericTerm
    : '<?' NAME '>'
    : '<' NAME (term (',' term)*)? '>' ;
```



# Добавление шаблонов в существующий язык

```
spec : (symbol ';' )+  
      : genericTerm ;  
symbol : ID (':' expression)+  
        : genericTerm ;  
expression  
  : ID  
  : STRING  
  : expression (expression)+  
  : '(' expression ')' mult  
  : genericTerm ;  
term  
  : genericTerm  
  : spec  
  : symbol  
  : expression ;
```



# Шаблоны в расширенном языке

```
funDef :  
  type ID '('  
    (var (' , ' var) *) ?  
  ')' ;
```

```
funDef :  
  type ID '('  
    <List var, ' , ' >  
  ')' ;
```

```
funcCall :  
  ID '('  
    (expr (' , ' expr) *) ?  
  ')' ;
```

```
funcCall :  
  ID '('  
    <List expr, ' , ' >  
  ')' ;
```

```
template List <item, sep> {  
    (<?item> (<?sep> <?item>) *) ?  
}
```

# Семантика разворачивания шаблонов

- Основные правила разворачивания:
  - Применение шаблона заменяется его телом с подстановкой параметров
  - Параметр заменяется фактическим аргументом
- Рекурсия запрещена. Замыкания не поддерживаются

$$\frac{\{p = e\} \subseteq \gamma}{\mathcal{I}_\gamma [ \langle ?p \rangle ] = e} \text{ var-inst}$$

$$\text{template} (T \langle p_1, \dots, p_n \rangle \{b\})$$

app-inst


$$\gamma' = \bigcup_{i=1}^n \{p_i = \mathcal{I}_\gamma [a_i]\} \quad \mathcal{I}_\gamma [ \langle T a_1, \dots, a_n \rangle ] = \mathcal{I}_{\gamma \cup \gamma'} [b]$$

$$\frac{t = \mathcal{TC} (C) @id \{r_i = rv_i, a_j = av_j\}}{\mathcal{I}_\gamma [t] = \mathcal{TC} (C) @id' \{r_i = \mathcal{I}_\gamma [rv_i], a_j = av_j\}} \text{ ds-inst}(C)$$

# Структурная корректность

```
template List <item, sep> {  
    (<?item> (<?sep> <?item>) *) ?  
}
```

**term**

  
<List var, a : expression>



(var ((**a : expression**) var) \*) ?

- Необходимо
  - гарантировать корректность разворачивания
  - корректно позиционировать сообщения об ошибках

# Типы в определениях шаблонов

```
template List <  
  item : Expression,  
  sep  : Expression>  
  : Expression {  
    (<?item> (<?sep> <?item>)*)?  
  }
```

```
<List  
  var,          // : SymRef  
  a : expression // : Symbol  
> // : Expression
```

- Система типов контролирует структуру результата разворачивания
- Типизируются шаблонные выражения
- Типы автоматически порождаются по целевой мета-модели

$$\frac{}{\Gamma \cup \{v : \tau\} \vdash \langle ?v \rangle : \tau} \text{ var}$$

$$\Gamma \cup \bigcup_{i=1}^n \{p_i : \tau_i\} \vdash b : \sigma$$

$$\frac{}{\Gamma \vdash \mathbf{template}(T \langle p_1 : \tau_1, \dots, p_n : \tau_n \rangle : \sigma \{b\})} \text{ abstr}$$

$$\frac{\Gamma \vdash \mathbf{template}(T \langle p_1 : \tau_1, \dots, p_n : \tau_n \rangle : \sigma \{b\}) \quad \forall i : [1 : n]. \Gamma \vdash a_i : \tau_i}{\Gamma \vdash \langle T a_1, \dots, a_n \rangle : \sigma} \text{ appl}$$

$$\frac{}{\Gamma \vdash \mathbf{NULL} : \tau^?} \text{ null} \quad \frac{\Gamma \vdash x : \tau}{\Gamma \vdash x : \tau^?} \text{ relax} \quad \frac{\Gamma \vdash x : \tau^+}{\Gamma \vdash x : \tau^*} \text{ relax}^+$$

$$\frac{\Gamma \vdash x : \tau \quad \tau \preceq \sigma}{\Gamma \vdash x : \sigma} \text{ subtype}$$

$$\frac{}{\Gamma \vdash [] : \tau^*} \text{ elist} \quad \frac{\forall i \in [1 : n]. \Gamma \vdash \mathbf{Item}(t_i, \tau)}{\Gamma \vdash [t_1, \dots, t_n] : \tau^+} \text{ list}$$

$$\frac{}{\Gamma \vdash \{\} : \tau^*} \text{ eset} \quad \frac{\forall i \in [1 : n]. \Gamma \vdash \mathbf{Item}(t_i, \tau)}{\Gamma \vdash \{t_1, \dots, t_n\} : \tau^+} \text{ set}$$

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathbf{Item}(t, \tau)} \text{ item} \quad \frac{\Gamma \vdash v : \tau^*}{\Gamma \vdash \mathbf{Item}(\langle ?v \rangle, \tau)} \text{ item}^* \quad \frac{\Gamma \vdash v : \tau^+}{\Gamma \vdash \mathbf{Item}(\langle ?v \rangle, \tau)} \text{ item}^+$$

$$\frac{\begin{array}{l} x = \mathcal{TC}(C) @id \{r_i = v_i; a_j = v'_j\} \\ \mathcal{TM} \Vdash x : \mathcal{TC}(C) \end{array} \quad \begin{array}{l} r_i = \mathcal{TR}(\rho_i : \tau_i) \\ \Gamma \vdash v_i : \tau_i \end{array}}{\Gamma \vdash x : C} \text{ ds-type}(C)$$

# Свойства системы типов

## Теорема (О сохранении типов)

*Если среда  $\gamma$  согласована с контекстом  $\Gamma$  и  $\Gamma \vdash e : \tau$ , то  $\Gamma \vdash \mathcal{I}_\gamma [e] : \tau$ . Другими словами, преобразование  $\mathcal{I} [\bullet]$  сохраняет типы.*

## Теорема (О нормализации)

*Если среда  $\gamma = \cup \{p_i = e_i\}$  согласована с контекстом  $\Gamma$ , все  $e_i$  имеют нормальную форму и  $\Gamma \vdash e : \tau$ , то результат вычисления  $\mathcal{I}_\gamma [e]$  имеет нормальную форму.*

# Вывод типов

- Часто типы в шаблонах можно не указывать
- Типы определяются автоматически из тела шаблона
- Возможность вывода тесно связана с однозначностью грамматики
- Алгоритм вывода является корректным, но не полным
  - Иногда вывести тип невозможно, и требуется его явное указание



# Применения шаблонов грамматик: параметризованные модули

```
template SMMain<
    event, eventRef, commandRef
> : Symbol+
{
    system : <?event>* stateMachine;
    stateMachine :
        'statemachine' NAME '{'
            state*
        '}' ;
    state :
        'state' NAME '{'
            do? (transition ';' ) *
        '}' ;
    do : 'do' block;
    transition : 'on' <?eventRef> 'goto' stateRef;
    stateRef : NAME;
    block : '{' ( <?commandRef> ';' ) * '}' ;
}
```

# Промежуточный итог

- Механическая процедура для расширения языка механизмом шаблонов
  - Синтаксис
  - Семантика
  - Система типов
    - Гарантии корректности результатов
    - Диагностика ошибок до разворачивания
- Применения для языка описания грамматик
  - Шаблоны выражений
  - Параметризованные модули

# Аспектно-ориентированное программирование

- «Инвазивная композиция ПО»
- Незнание (Obliviousness)
  - Расширение программ, которые не были специально для этого спроектированы
- Квантификация (Quantification)
  - Применение одного и того же расширения в нескольких точках
    - Множество точек применения описывается компактным предикатом

# Появление и распространение

- AspectJ – аспектно-ориентированное расширение Java
  - Те же идеи используются в популярных платформах: Spring, Equinox и т. д.
- Aspect.NET – аспекты для платформы .NET
- Аспекты в формальных грамматиках
  - JastAdd
  - AspectG (ANTLR)
  - Silver
  - LISA
  - Rats!

# Общая схема аспекта

- Аспект – набор *аспектных правил*
- Каждое правило включает
  - Срез (point-cut)
    - Поискový запрос, описывающий точки применения
    - Обычно использует язык образцов, похожий на регулярные выражения
      - AspectJ: **call**(int Hash\*.size\*(..))
  - Совет
    - Описывает изменения и их положение в точке применения
      - AspectJ: **after call**(int Hash\*.size\*(..)) {Log.write("size() called");}

# Пример использования аспектов в грамматиках

- Задача:
  - Описать отличия PostgreSQL от SQL92
  - После DISTINCT может идти предложение ON
- Срез:
  - `setQuantifier : ?d=DISTINCT | ...;`
- Совет:
  - `after ?d : (ON '(' <List expression, ','> ')')?;`

# Составляющие языка аспектов

- Язык срезов
  - Использует те же базовые понятия, что и язык шаблонов
  - Добавляет подстановочные знаки (Wildcards)
    - Соответствуют любому выражению данного типа
- Язык советов
  - Операции **before, after, instead, remove**
  - **instead** ?a : <T <?a>>
    - Заменяет все вхождения переменной результатом разворачивания шаблона

# Примеры срезов

- **expr**: term ( ("+" | "-") term)\* ;
  - Точное совпадение
- **expr**: <?:Production+> ;
  - Подстановочный знак (ПЗ) для продукций
- <?:Symbol>: term <?:Sequence> ;
  - ПЗ для последовательностей
- <?:Symbol>: <?t:Symbol> ( ("+" | "-") ?t)\* ;
  - ПЗ Для символов и переменная



# Примеры срезов

- **expr**: term ( ("+" | "-") term)\* ;
  - Точное совпадение
- **expr**: {...} ;
  - Подстановочный знак (ПЗ) для продукций
- **#**: term .. ;
  - ПЗ для последовательностей
- **#**: ?t=# ( ("+" | "-") ?t)\* ;
  - ПЗ Для символов и переменная

# Семантика аспектов

- Сопоставление с образцом
  - Дана модель, найти в ней все объекты, соответствующие образцу
  - NP-полная задача в случае наличия неупорядоченных коллекций и переменных
- Применение советов
  - Замена вхождений переменных новыми объектами
  - Использует разворачивание шаблонов

# Пример (PostgreSQL)

- Аспектное правило

- `tableExpression : ?f=fromClause .. ;`  
– `instead ?f : <?f>;`

- Правило грамматики

- `tableExpression`  
: `fromClause whereClause?`  
`groupByClause? HavingClause?;`

- Результат

- `tableExpression`  
: `fromClause? whereClause?`  
`groupByClause? havingClause?;`

$$\frac{\Upsilon(v) = [e] \quad e \cong x}{x \text{ match}_{\Upsilon} \langle ?v \rangle = \Upsilon \uplus \{v \mapsto [x]\}} \text{ match-var-e}$$

$$\frac{\overline{\Upsilon}(v) = \langle P \rangle}{x \text{ match}_{\Upsilon} \langle ?v \rangle = (x \text{ match}_{\Upsilon} P) \dot{\cup} \{v \mapsto [x]\}} \text{ match-var-P}$$

$$\frac{\text{template}(T \langle p_1, \dots, p_n \rangle \{B\}) \quad \Upsilon' = (\biguplus_i \{p_i \mapsto \langle a_i \rangle\}) \uplus \Upsilon}{x \text{ match}_{\Upsilon} \langle T a_1, \dots, a_n \rangle = x \text{ match}_{\Upsilon'} B} \text{ match-app}$$

$$\frac{x = C@id \{f_i = v_i\}}{x \text{ match}_{\Upsilon} \langle ? : C \rangle = \Upsilon} \text{ match-wc} \quad \frac{x \text{ match}_{\Upsilon} \langle ? : C \rangle = \Upsilon}{x \text{ match}_{\Upsilon} \langle ? : C^? \rangle = \Upsilon} \text{ match-wc-?}$$

$$\frac{}{NULL \text{ match}_{\Upsilon} \langle ? : C^? \rangle = \Upsilon} \text{ match-null}$$

$$\frac{x = C@id \{f_i = v_i\} \quad v_i \text{ match}_{\Upsilon} P_i = \Upsilon_i}{x \text{ match}_{\Upsilon} \mathcal{TC}(C) @id' \{f_i = P_i\} = (\biguplus_i \Upsilon_i) \uplus \Upsilon} \text{ match-ds}$$

$$\frac{x = [x_1, \dots, x_n] \quad P = [P_1, \dots, P_m]}{x \text{ match}_{\Upsilon} P = matchList_{\Upsilon}(x, P)} \text{ match-list}$$

$$\frac{x = \{x_1, \dots, x_n\} \quad P = \{P_1, \dots, P_m\}}{x \text{ match}_{\Upsilon} P = matchSet_{\Upsilon}(x, P)} \text{ match-set}$$

$$\frac{x \text{ — значение примитивного типа}}{x \text{ match}_{\Upsilon} x = \Upsilon} \text{ match-prim}$$

# Семантика применения советов

Пусть  $P \text{ match } e = \Upsilon \neq \perp$ ,  
 $V = \{\langle v_i, t_i \rangle \mid i = 1..m\}$ ,  $\Upsilon(v_i) = [e_1^i, \dots, e_{n_i}^i]$ , тогда

$$\mathcal{R} @ e = \left( \bigsqcup_{i=1}^m \bigsqcup_{j=1}^{n_i} e_j^i \mapsto \mathcal{I}_{\overline{\Upsilon}}[t_i] \right) \triangleright e$$

# Структурная корректность в аспектах

- При замене необходимо гарантировать соблюдение структурных ограничений
- Используется та же система типов, что и для шаблонов, с небольшим расширением:

$$\frac{}{\vdash \langle ? : \tau \rangle : \tau} \text{ wcard}$$

$$\frac{\mathcal{R} = \langle p, T, V \rangle \quad \Gamma(p) \vdash v : \tau \quad \Gamma(p) \vdash t : \sigma \quad \sigma \preceq \tau}{(\text{instead } \langle ?v \rangle : t) \in Allowed(\mathcal{R})} \text{ aspect}$$

**Теорема 1.7.** Если  $\mathcal{R}$  таково, что  $V \subseteq Allowed(\mathcal{R})$ , то в результате применения  $\mathcal{R}@e$  не нарушаются структурные ограничения, накладываемые мета-моделью.

# Итог

- Автоматическое расширение языков механизмами композиции
  - Шаблоны
  - Аспекты
- Гарантии структурной корректности
  - Обнаружение ошибок до применения шаблонов/аспектов
- Применение к языку описания формальных грамматик
  - Шаблоны выражений
  - Параметризованные модули
  - Описание диалектов
  - ...





# Шаблоны (типизированные макроопределения)

```
VAR x = 2; VAR y = 5;  
VAR xy = 0;  
VAR trigger = 0;  
loop:  
    xy = xy + y;  
    x = x - 1;  
    trigger = 1;  
IF x = 0 THEN GOTO loop;
```

```
VAR x = 2; VAR y = 5;  
VAR xy = 0;  
$UNTIL <x = 0> <  
    xy = xy + y;  
    x = x - 1;  
>;
```

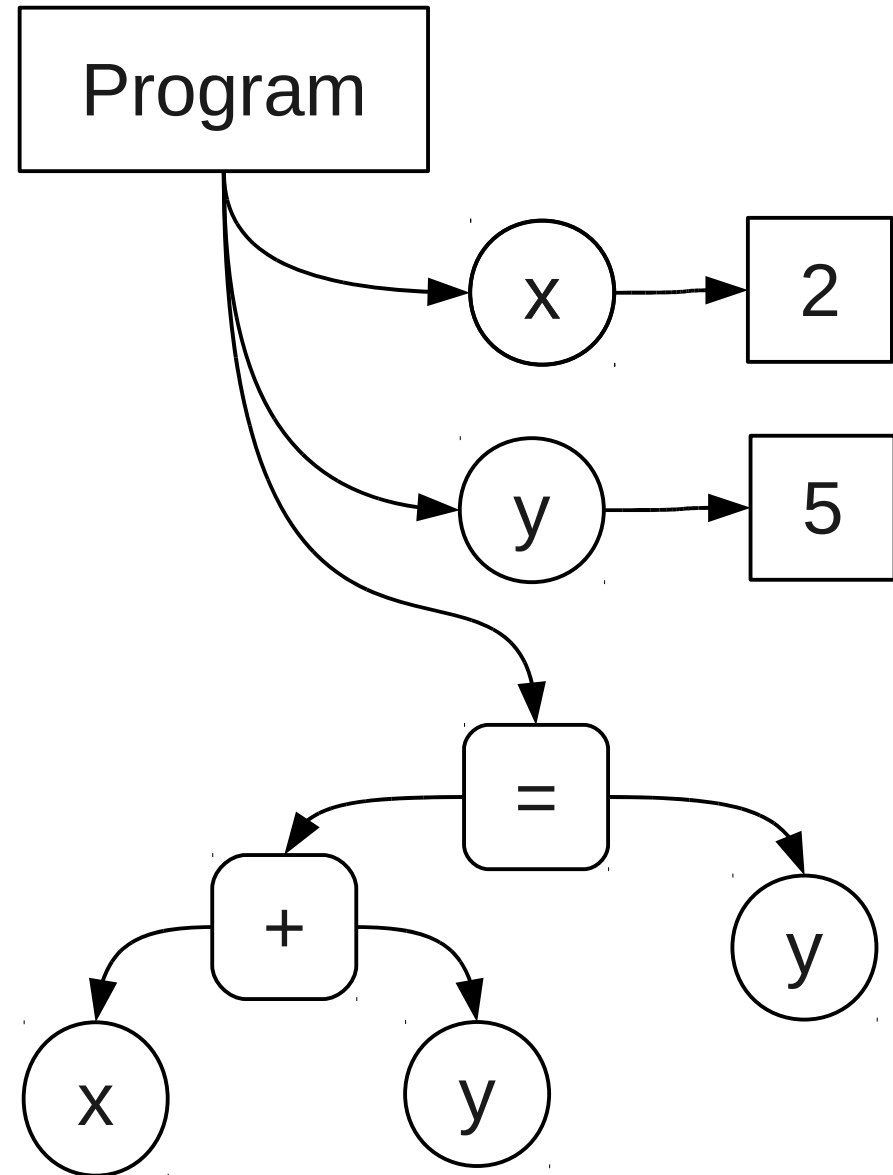
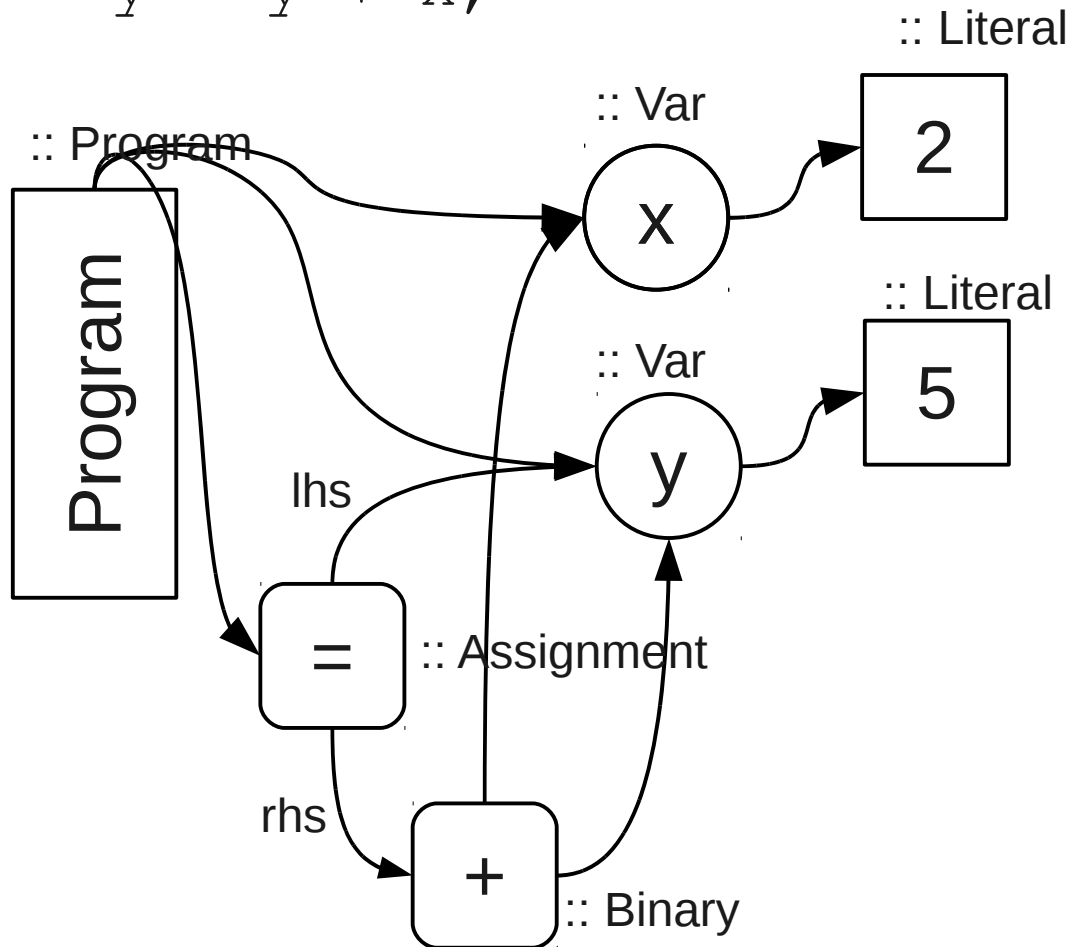
Расширенный  
язык



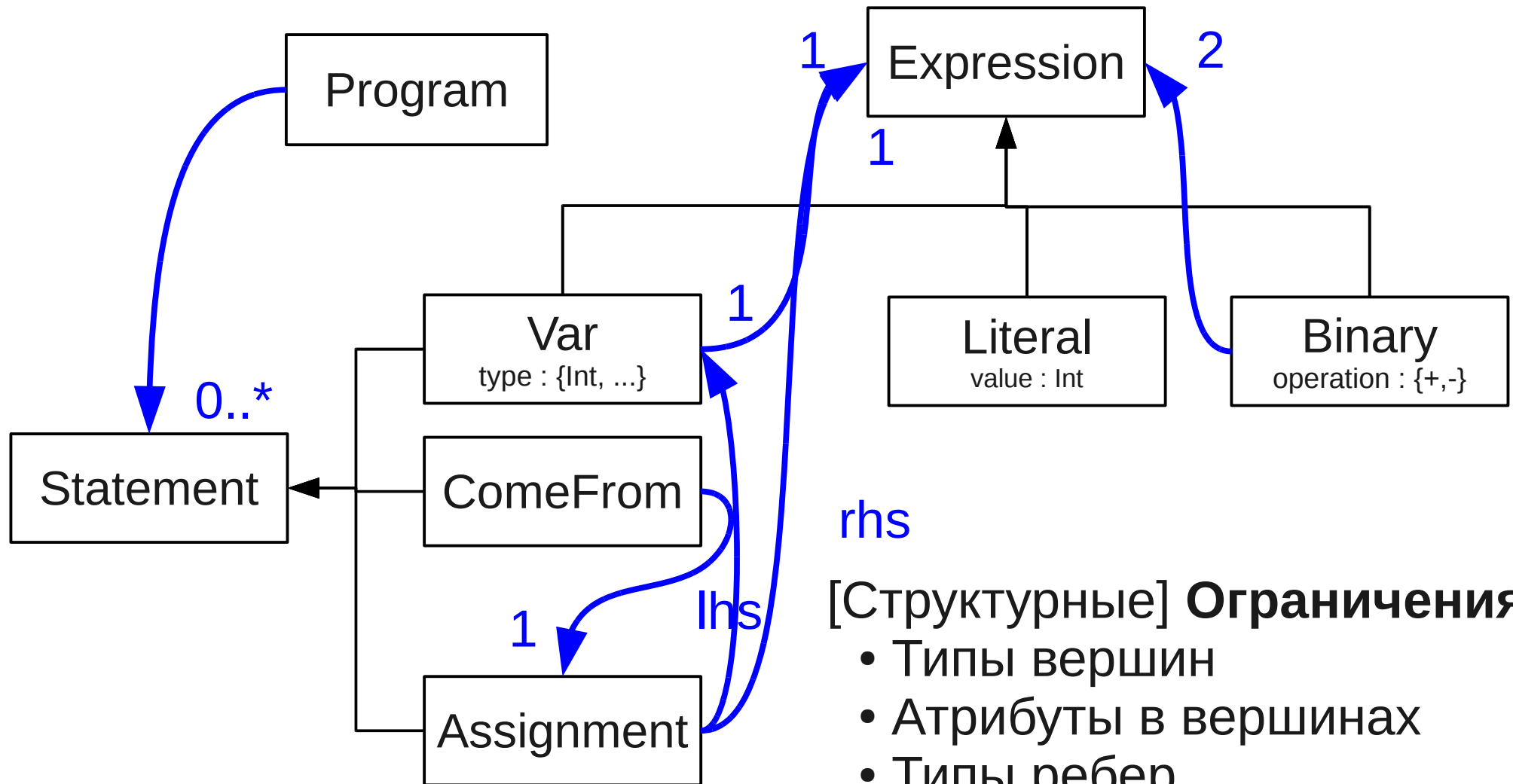
```
template $UNTIL <condition> <body> {  
    VAR trigger = 0;  
    loop:  
        <body>  
        trigger = 1;  
    IF <condition> THEN GOTO loop;  
}
```

# Представления программы

```
VAR x = 2;  
VAR y = 5;  
y = y + x;
```



# Описание графа

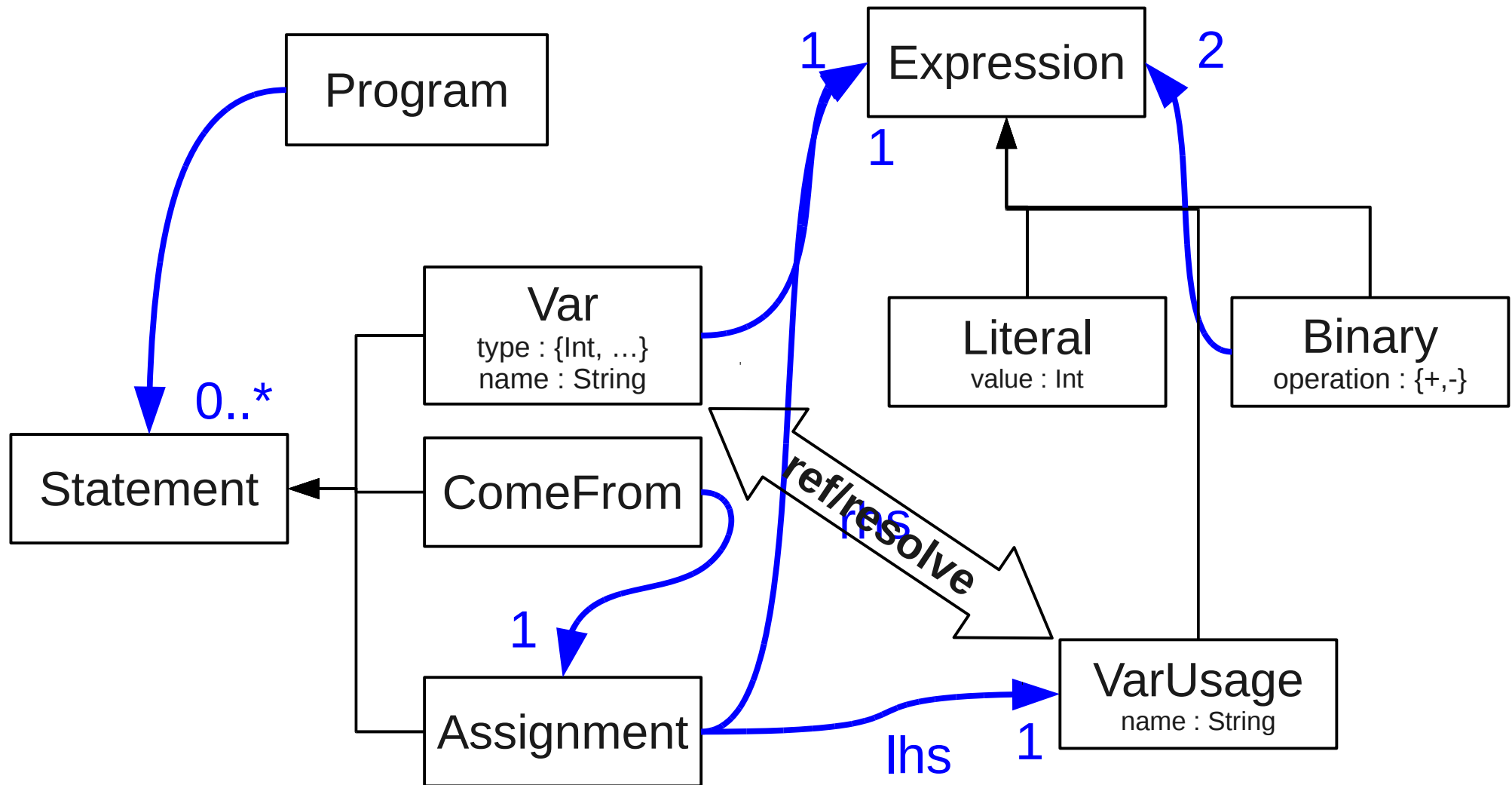


`rhs`

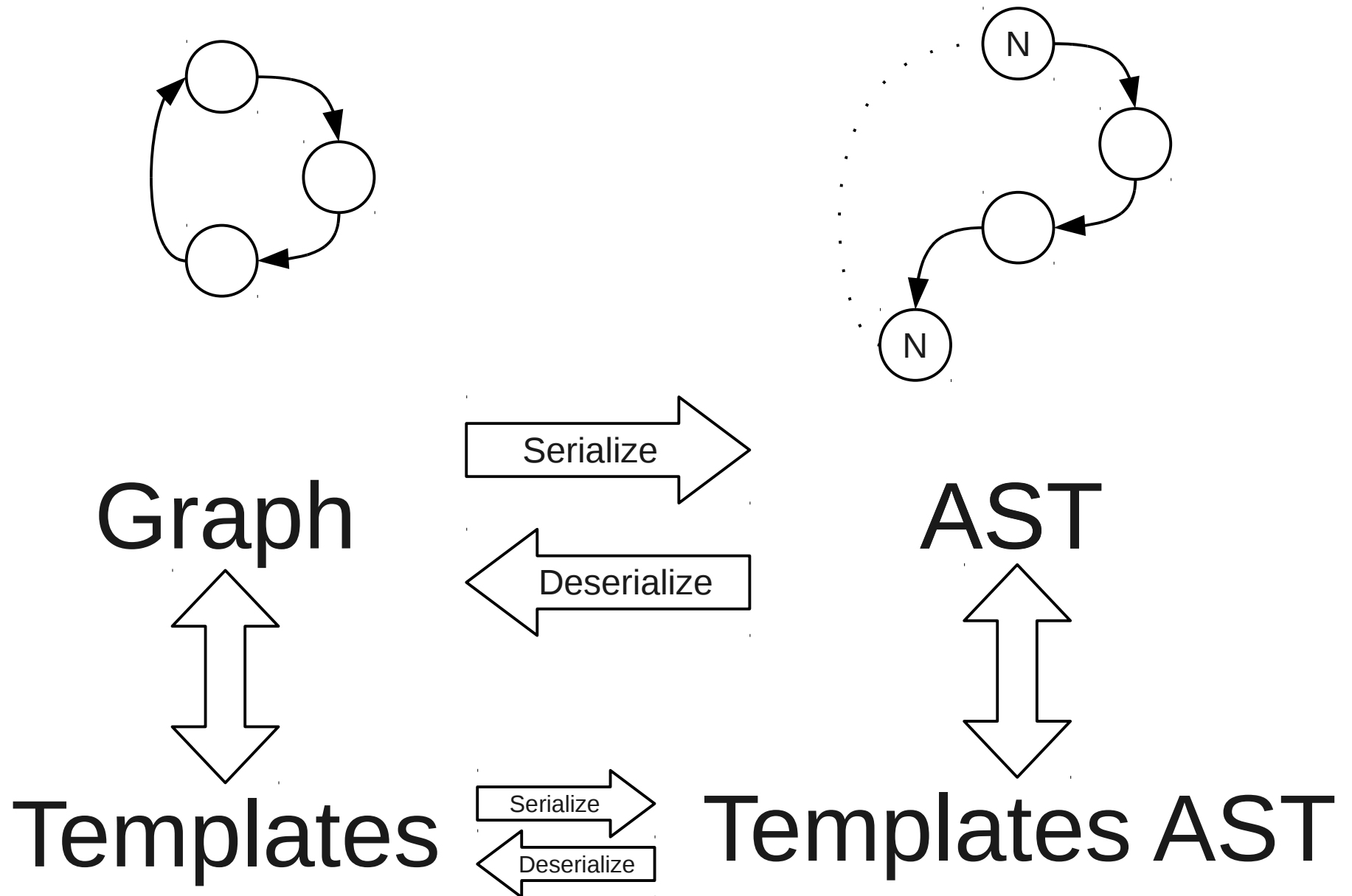
[Структурные] **Ограничения:**

- Типы вершин
- Атрибуты в вершинах
- Типы ребер
- Атрибуты на ребрах
- Типизация концов ребер

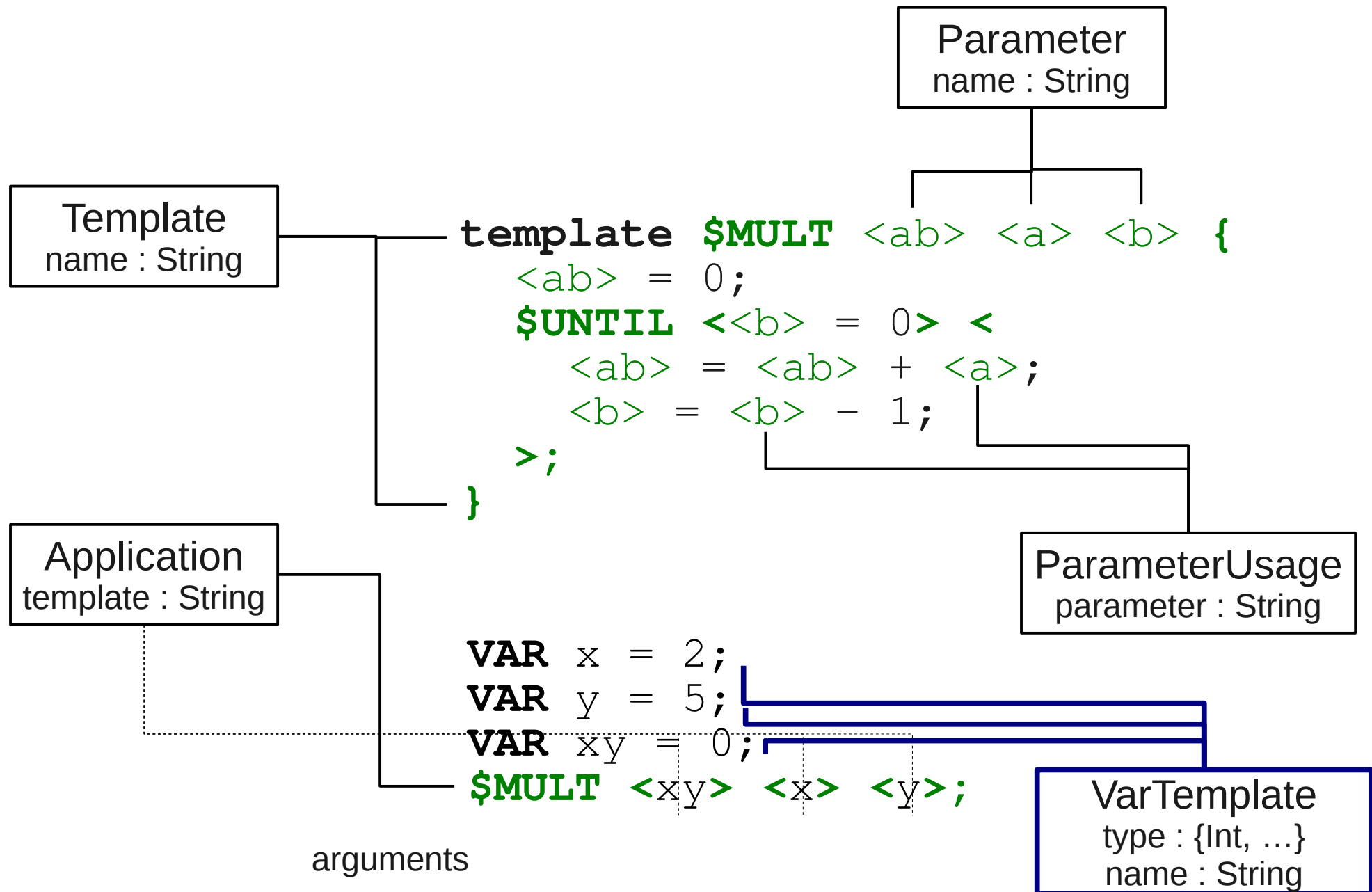
# Describing tree structure



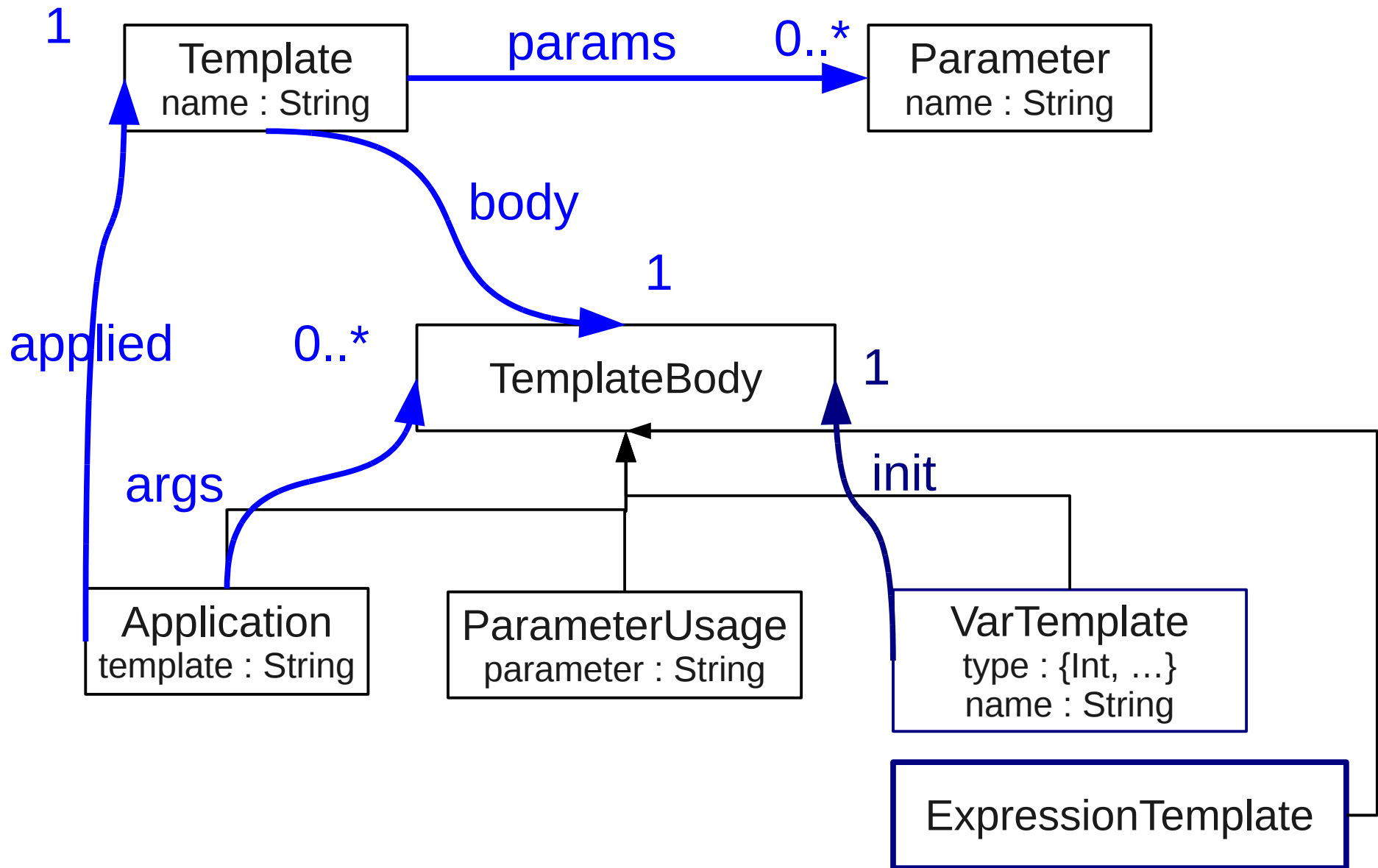
# Схема преобразования



# Описание шаблонов

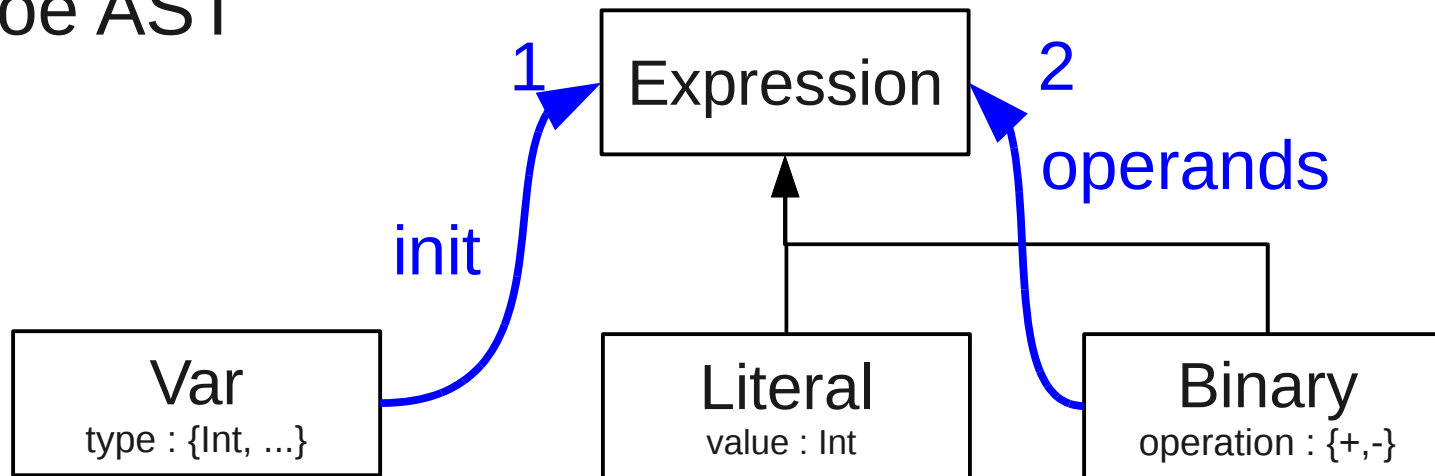


# Шаблон

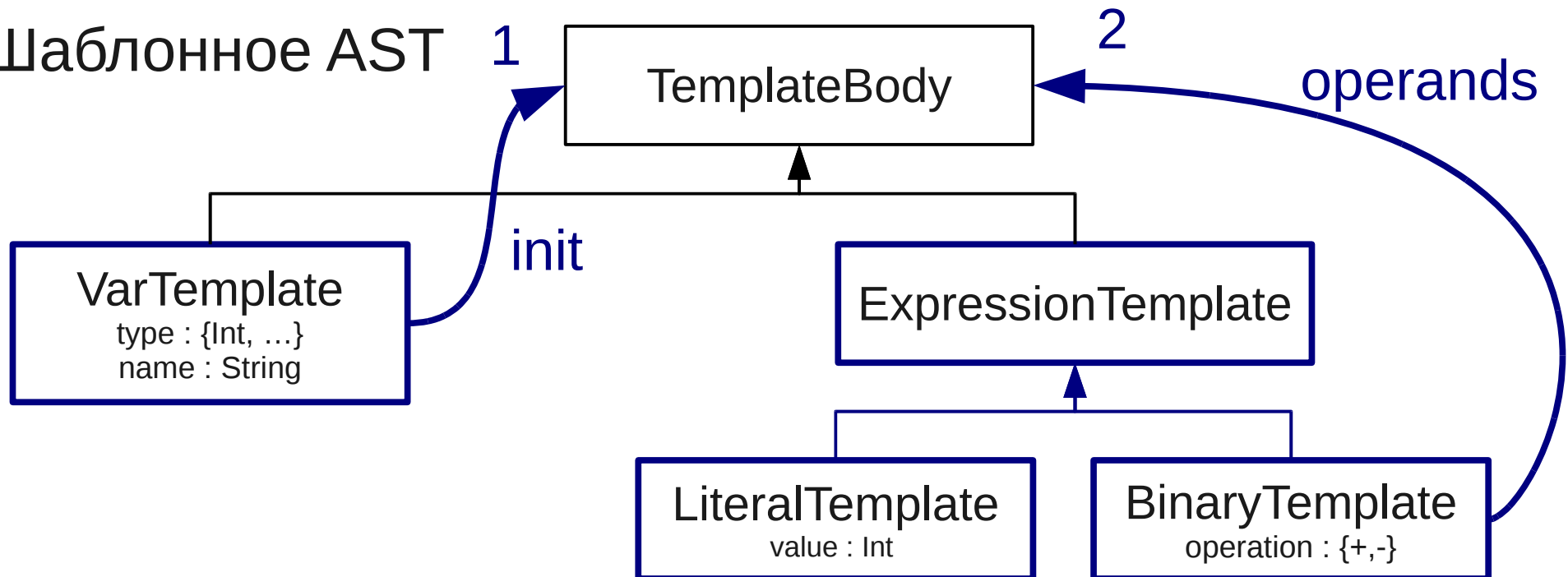


# Специфичные шаблоны

Исходное AST

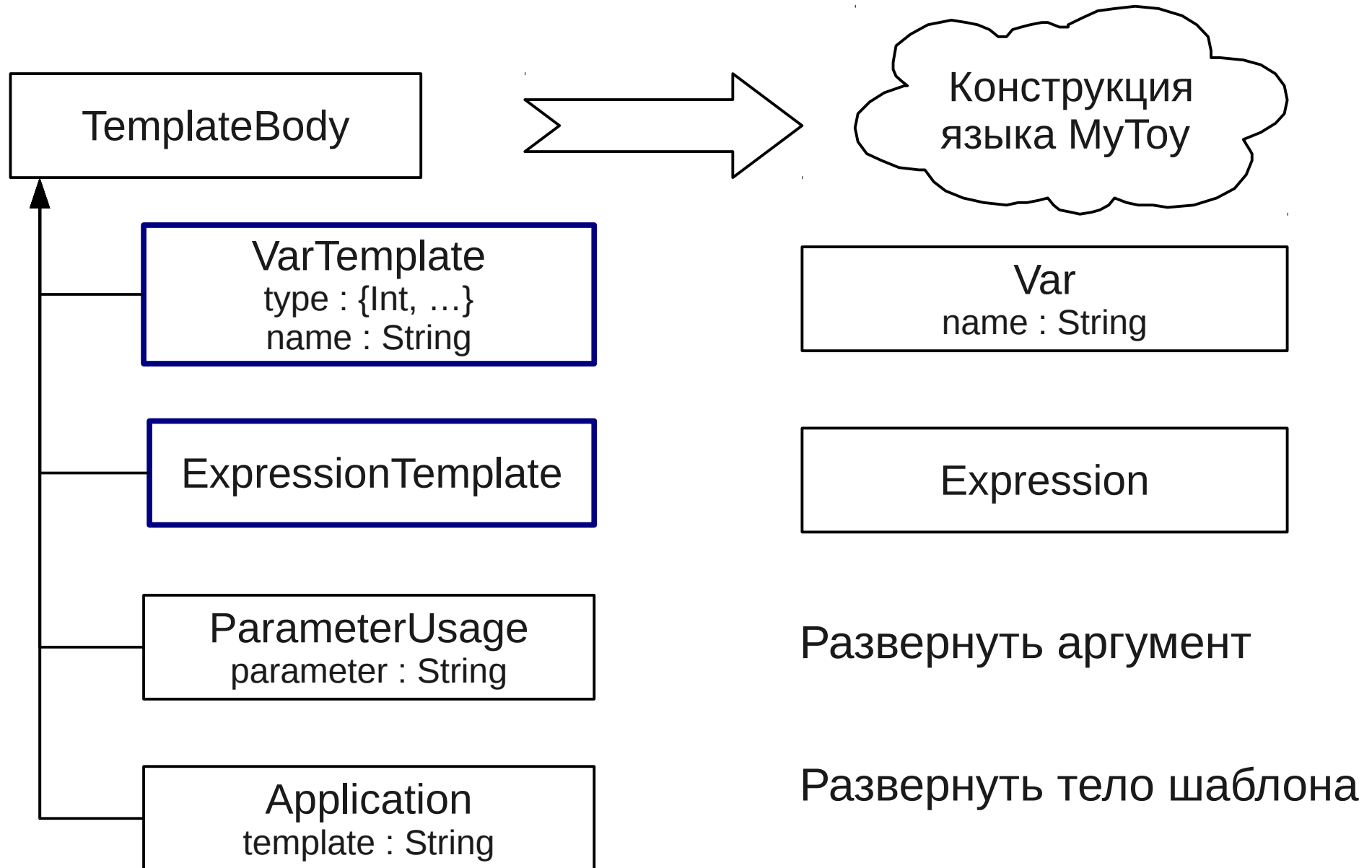


Шаблонное AST





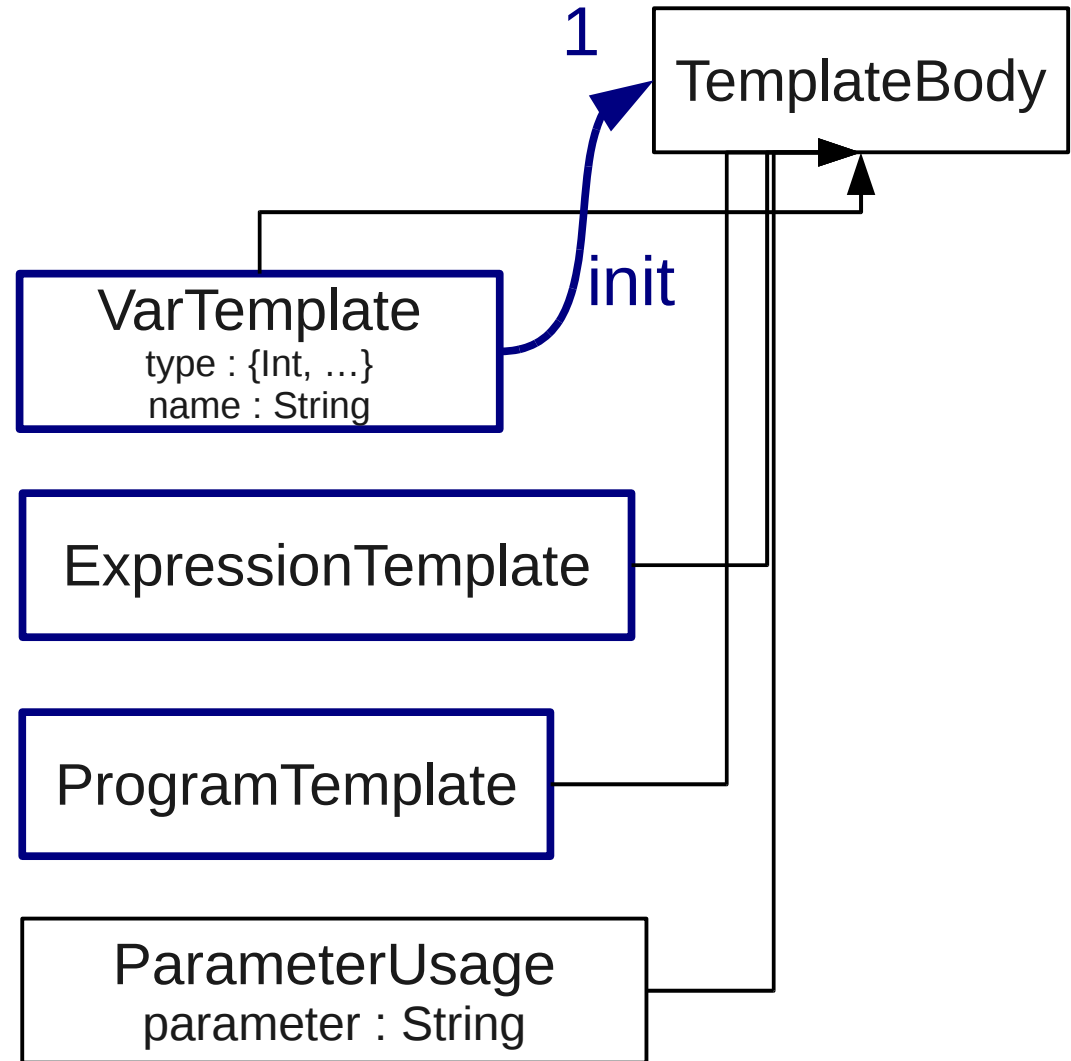
# Разворачивание



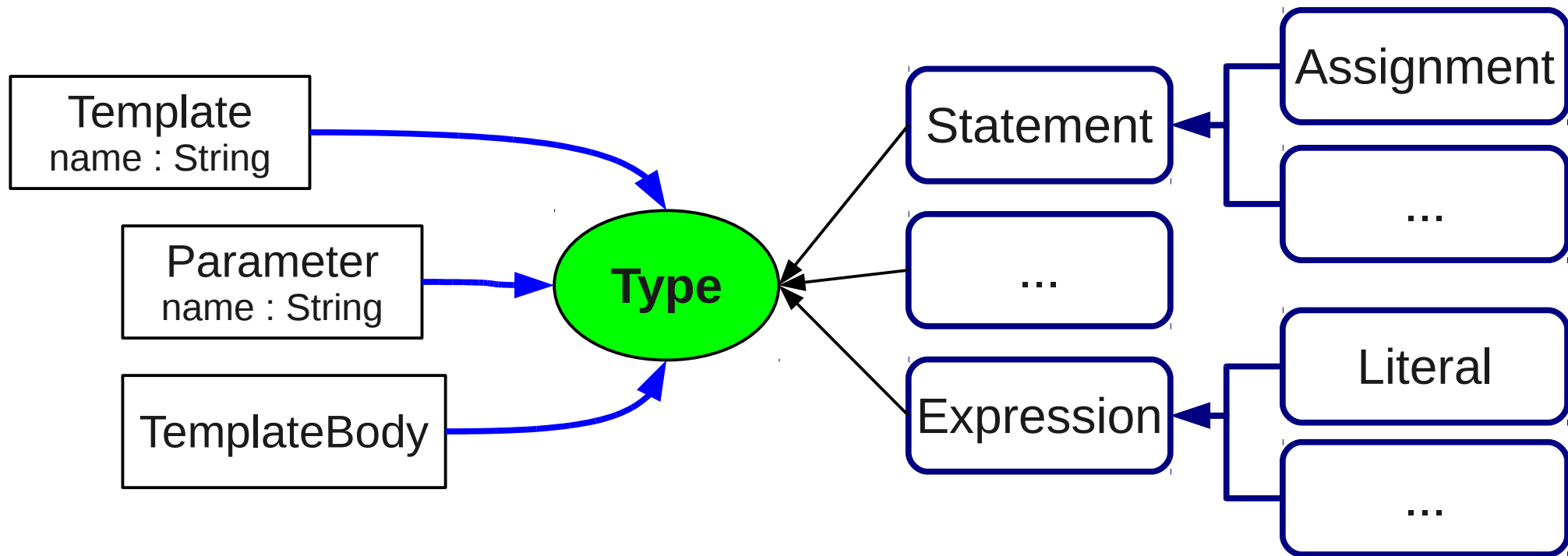
# Структурная корректность

## Ограничения:

- Типы вершин
- Атрибуты в вершинах
- Типы ребер
- Атрибуты на ребрах
- Типизация концов ребер



# Типы в шаблонах



```
template $UNTIL :: Statement[1..*]  
  <condition :: Assignment[1]>  
  <body :: Statement[0..*]> {  
    VAR trigger = 0 :: Literal[1];  
    comefrom trigger = 1;  
    <body>  
    Trigger = 1; :: Assignment[1]  
    comefrom <condition>;  
  }
```

Типы  
отражают  
кратность

# Резюме

- Язык описывается тройкой (Graph,  $\leftrightarrow$ , AST)
- Язык расширяется поддержкой шаблонов
- Генерируется алгоритм проверки типов
  - (аннотации не нужны)
  - Так обеспечивается диагностика ошибок
- Генерируется процедура разворачивания
- Все остальное переиспользуется из исходной реализации

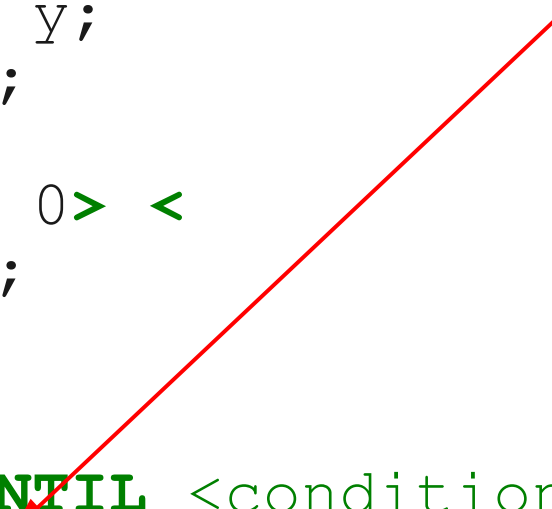
# Спасибо за внимание!

- Ваши вопросы

# One more type of errors

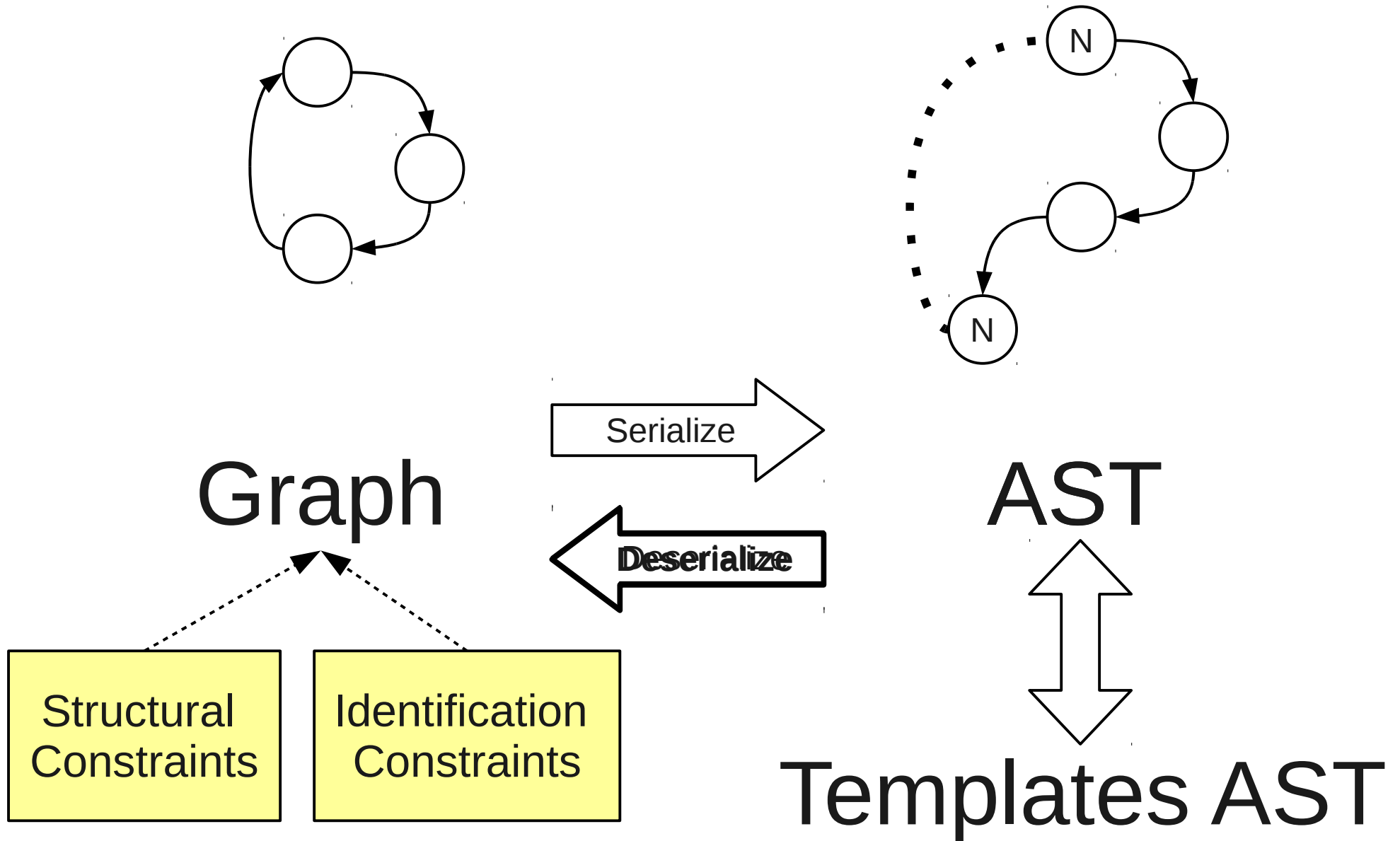
```
VAR x = 2; VAR y = 5;  
VAR xy = 0;  
$UNTIL <x = 0> <  
    xy = xy + y;  
    x = x - 1;  
>;  
$UNTIL <y = 0> <  
    y = y - 1;  
>;
```

**Error: variable  
'trigger' is already  
defined!**



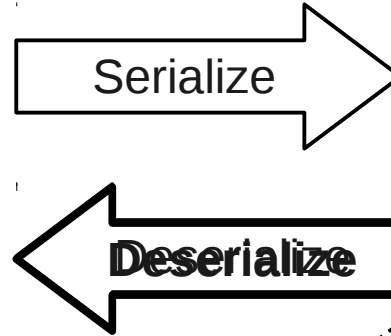
```
template $UNTIL <condition> <body> {  
    VAR trigger = 0;  
    comefrom trigger = 1;  
    <body>  
    trigger = 1;  
    comefrom <condition>;  
}
```

# Who said that was an error?!

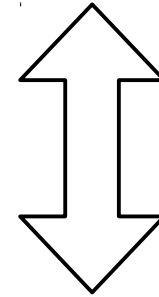


# Our plan

Graph



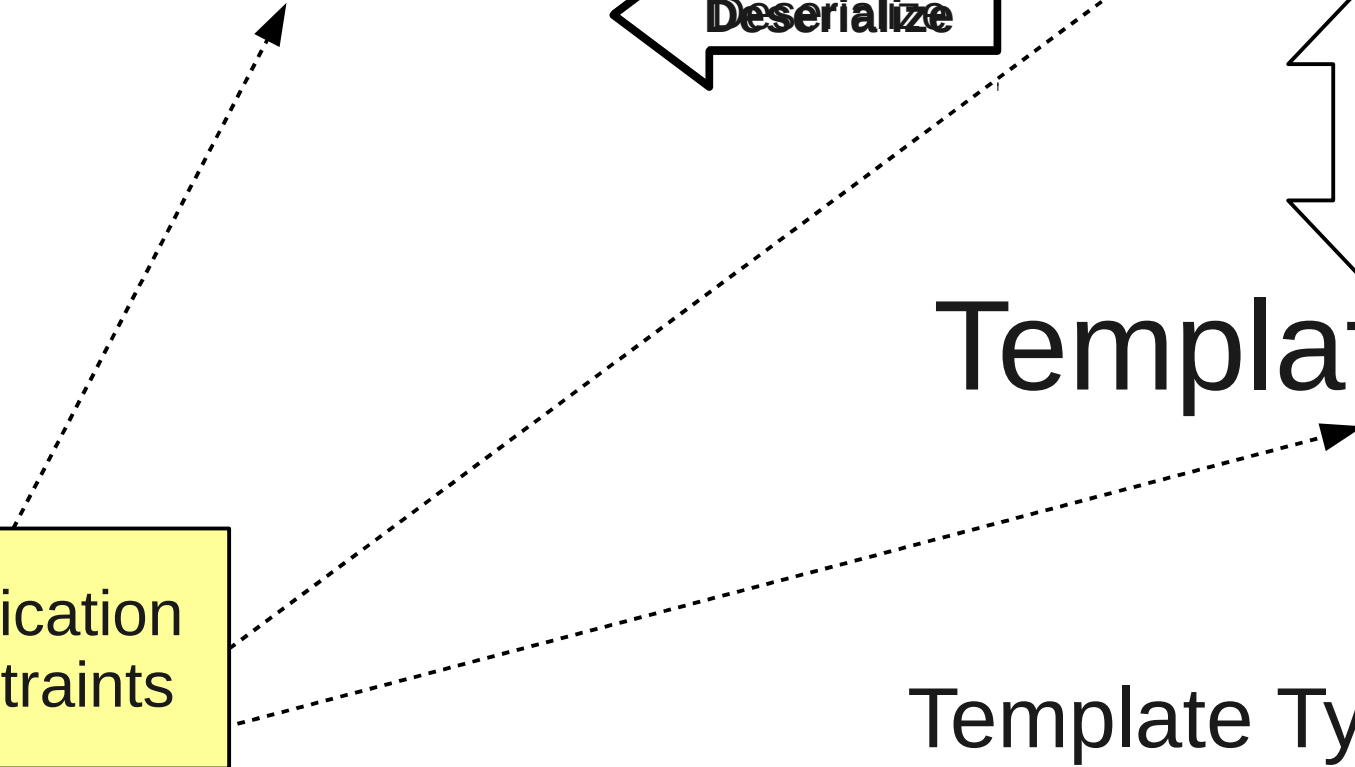
AST



Templates AST

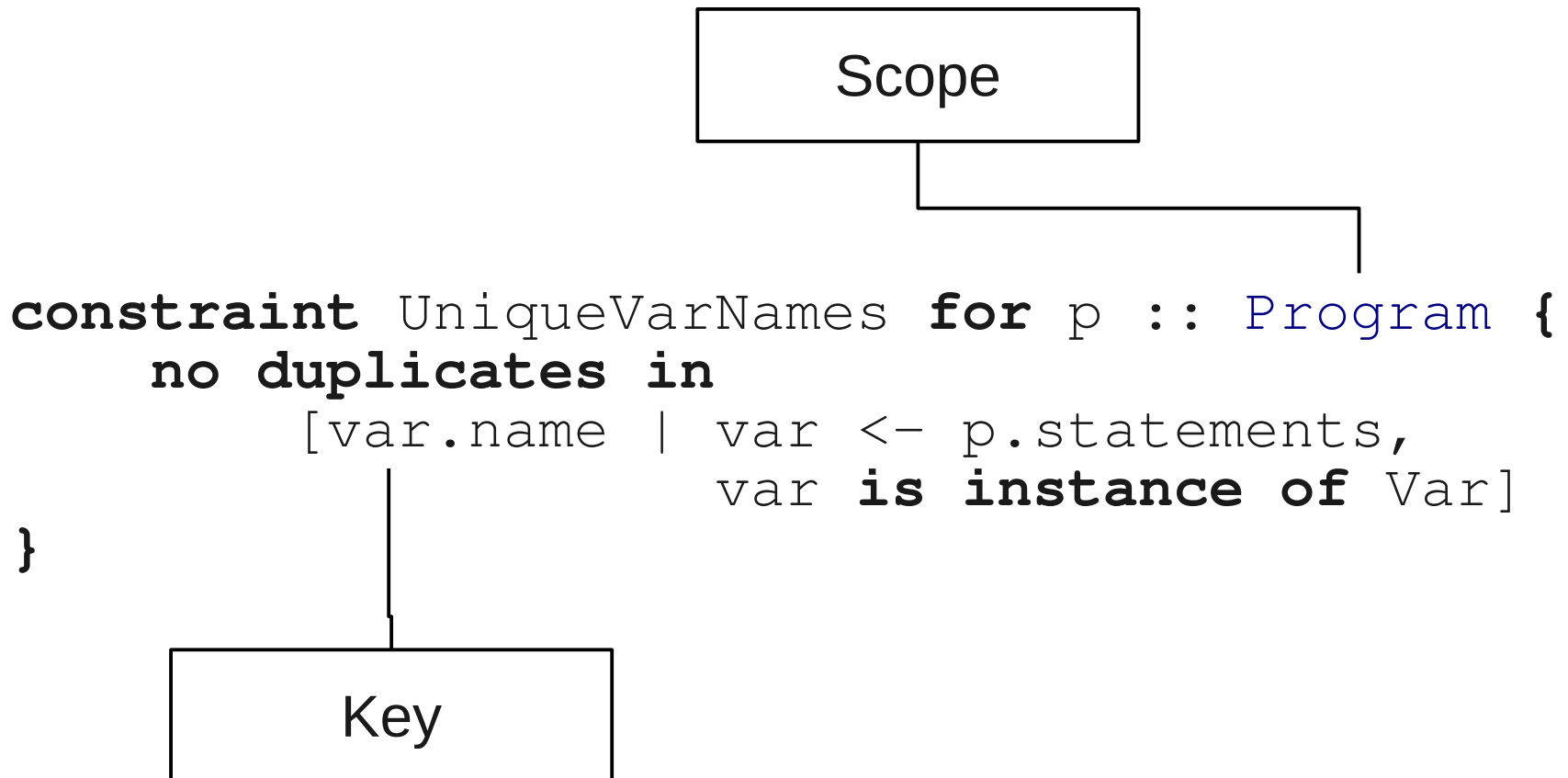
Identification  
Constraints

Template Type System  
must be extended



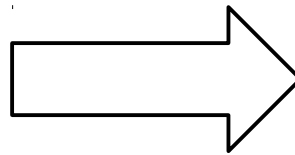


# Identification Constraints



# Name Checking for MyToy

```
VAR x = 2; VAR y = 5;  
VAR xy = 0;  
...  
VAR trigger = 0;  
...  
VAR trigger = 0;  
...
```



```
{x, y, xy,  
  trigger,  
  trigger}
```

- for every scope
  - compute the collection of keys
  - check it for duplicates

# Is it sufficient?

- **Yes!**
  - MOF/Ecore
  - EBNF
  - DDL
  - DTD
  - Other language which describe static structures
- **No**
  - Programming languages
  - ...

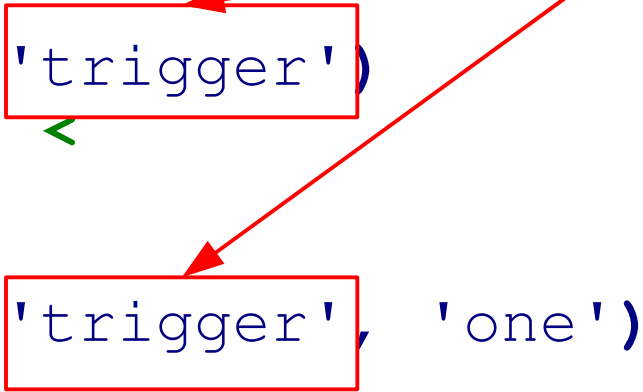
# Types for Name checking

**Forget about names of templates  
and their parameters!**

```
template $UNTIL :: defines('trigger', bodyDef)
  <condition> <body :: defines(bodyDef)> {
    VAR trigger = 0; :: defines('trigger')
    comefrom trigger = 1;
    <body> :: defines(bodyDef)
    trigger = 1;
    comefrom <condition>;
  }
```

# Name Checking Example

```
VAR x = 2; :: defines('x')
VAR y = 5; :: defines('y')
VAR xy = 0; :: defines('xy')
$UNTIL <x = 0> <
  xy = xy + y;
  x = x - 1;
>; :: defines('trigger')
$UNTIL <y = 0> <
  VAR one = 1;
  y = y - one;
>; :: defines('trigger', 'one')
```



**Error: templates  
'UNTIL' and  
'UNTIL' define the  
same variable:  
trigger!**

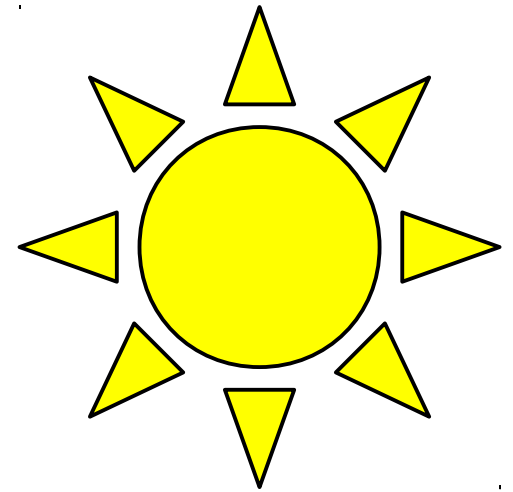
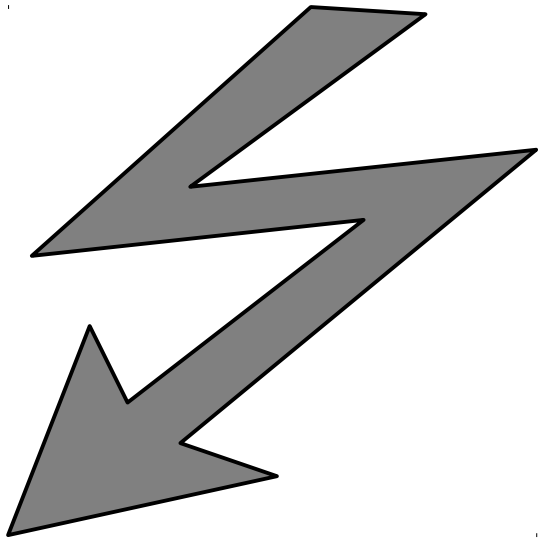
```
template $UNTIL :: defines('trigger', bodyDef)
```

# Naming Errors

- Duplicate definition
  - done
- “Hygienic” templates
  - achievable
- Usage of undefined name (only AST)
  - symmetric
- Illegal context
  - In the graph an object is used while it's name is not visible
- Collision
  - Two distinct objects with the same names are used

# Symmary

- If a language is described as (Graph  $\leftrightarrow$  AST)
  - Automatically extend it with template support
  - Structural correctness guarantees
  - Some naming correctness guarantees
- Ecore, EBNF, DTD, etc.
  - Complete support for templates
- More complicated languages
  - Subject to future work...



Thanks for your attention!





# Существующие подходы

- Генераторы трансляторов
  - Решают задачу частично
- Модельно-ориентированные технологии
  - Преимущественно графический синтаксис
  - Поддержка композиции моделей через трансформации общего назначения
- Специализированные инструменты (xText, TCS)
  - Удобны для решения наиболее распространенных задач
  - Пока позволяют быстро решать лишь очень простые задачи
    - Построение дерева
    - Простое разрешение имен
    - Примитивные модули