

**Ордена Ленина  
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ  
имени М.В. Келдыша  
Российской академии наук**

**Ю.А. Климов**

**Специализатор SILPE:  
анализ времен связывания**

**Москва  
2009**

Ю.А. Климов

Специализатор CILPE: анализ времен связывания

## **Аннотация**

Анализ времен связывания (Binding Time Analysis, ВТА, ВТ-анализ) — основная часть метода частичных вычислений (Partial Evaluation, PE). В данной работе описана ВТ-разметка (Binding Time Annotation) программ на стековом объектно-ориентированном языке SOOL (Stack Object-Oriented Language) и правила, которым должна удовлетворять корректная разметка. Эта разметка строится анализом времен связывания и используется для генерации остаточной программы в специализаторе CILPE.

Работа поддержана проектами РФФИ № 08-07-00280-а и № 09-01-00834-а.

Yu.A. Klimov

Specializer CILPE: binding time analysis

## **Abstract**

Binding Time Analysis (BTA, BT-analysis) is an essential part of the Partial Evaluation (PE) technique of program specialization. The results of BT-analysis are used for decorating a program to be specialized with annotations, which are used later by the residual program generator. The specializer CILPE deals with programs written in SOOL, a simple Stack Object-Oriented Language. The paper present the approach to BT-analysis taken in CILPE. The annotations added to a SOOL program must satisfy certain requirements, which are formulated as a set of inference rules. The BT-analyzer in CILPE complies with these rules.

## Содержание

1. Введение.....	4
2. Анализ времен связывания.....	5
3. VT-разметка .....	7
3.1. VT-программа .....	7
3.2. Корректность VT-программы.....	8
3.3. VT-куча .....	10
3.4. Корректность VT-кучи.....	11
3.5. Корректность VT-разметки .....	11
4. Правила VT-разметки .....	12
4.1. VT-состояние .....	12
4.2. VT-разметка программы.....	12
4.3. VT-разметка метода.....	13
4.4. VT-разметка последовательности инструкций .....	14
4.5. VT-разметка инструкций .....	15
5. Выполнение размеченной программы .....	25
6. Заключение .....	26
7. Литература .....	27

## 1. Введение

Оптимизация программ на основе использования априорной информации о значении части переменных называется *специализацией*. Одним из широко используемых методов специализации является метод *частичных вычислений* (Partial Evaluation, PE) [Jones93].

Метод частичных вычислений основан на разделении операций и других программных конструкций на *статические* (S) и *динамические* (D). При этом понятие «статические операции и конструкции» не следует путать с понятием «статические методы и классы» (static), которое используется в объектно-ориентированных языках программирования, например, Ява и С#.

В процессе частичных вычислений операции над известными данными исполняются, а над неизвестными — переносятся в остаточную программу. Остаточная программа зависит только от неизвестной (на стадии специализации) части аргументов.

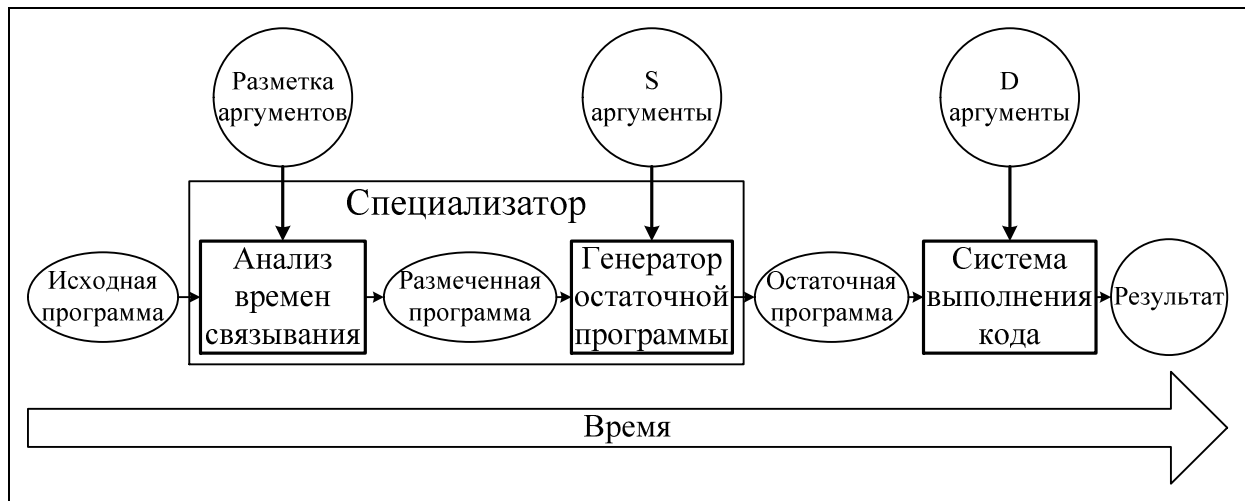


Рис. 1. Общепринятая структура специализатора, основанного на методе частичных вычислений.

Часть метода специализации, отвечающая за разделение операций и данных, называется *анализом времен связывания* (Binding Time Analysis, BTA, BT-анализ) (рис. 1). Вторая часть метода специализации, отвечающая за вычисление статической части программы и выделение динамической части в отдельную программу, называется *генератором остаточной программы* (Residual Program Generator, RPG). В этой части, собственно, и происходят час-

*точные вычисления.*

В данной работе формально описывается поливариантный анализ времен связывания [Klim05a,Klim05b] для стекового объектно-ориентированного языка SOOL (Stack-based Object-oriented Language) [Klim08c]. Описана структура ВТ-разметки и правила, которым должна удовлетворять корректно построенная разметка.

Данный анализ времен связывания используется в специализаторе CILPE [Чер03,Klim08a,Klim08b], созданном на основе метода частичных вычислений, для платформы Microsoft .NET [CLI,MS.NET].

## **2. Анализ времен связывания**

Цель анализа времен связывания (Binding Time Analysis, ВТ-анализ) — построить ВТ-разметку: приписать каждой инструкции ее ВТ-вид, а аргументам и результатам методов, элементам стека и переменным — ВТ-значение.

Для разметки инструкций используются ВТ-виды S, D и X. Разметка S (Static) показывает, что эта инструкция может быть выполнена генератором остаточной программы. Такие инструкции будем называть S-инструкциями. Разметка D (Dynamic) показывает, что инструкция не может быть обработана генератором остаточной программы и должна перейти в остаточную программу без изменения. Такие инструкции будем называть D-инструкциями. Разметка X (eXclusive) используется для разметки особых инструкций, которые переходят в остаточную программу в преобразованном виде. Такие инструкции будем называть X-инструкциями.

Разметка аргументов и результатов методов, элементов стека и переменных размечаются ВТ-значениями с ВТ-видами S или D и, в зависимости от типа элемента, дополнительной разметкой, о которой будет сказано ниже. Аргументы, результаты, элементы стека и переменные, размеченные S, будем называть S-аргументами, S-результатами, S-элементами стека и S-переменными соответственно. Данные, хранящиеся в них, будут вычислены генератором остаточной программы. Такие данные будем называть S-данными. Сущности, размеченные как D, будем называть D-аргументами, D-результатами, D-

элементами стека и D-переменными. Данные, записанные в таких местах, будем называть D-данными.

Разметка должна быть корректной, внутренне согласованной: S-инструкции должны потреблять и вырабатывать только S-данные, а D-инструкции — D-данные. Только в этом случае генератор остаточной программы сможет выполнить S-инструкции и вычислить все S-данные. Точные правила VT-разметки даны ниже.

Примитивное S-данное может стать аргументом D-инструкции, если оно будет предварительно «поднято» в D, путем вставки операции «поднятия» Lift. Обратный переход из D в S невозможен.

Итак, задача VT-анализа — построить корректную VT-разметку программы и программных элементов, основываясь на разметке аргументов и, может быть, каких-нибудь других элементов программы.

Программу можно корректным образом разметить многими способами. Но для более эффективной работы остаточной программы нужно, чтобы как можно больше инструкций были классифицированы как S и выполнены во время генерации остаточной программы. Поэтому при построении разметки необходимо разметить S как можно больше инструкций.

При построении разметки каждому исходному методу может соответствовать ровно один размеченный метод. В этом случае VT-анализ называется моновариантным по методам. Это влечет одинаковость разметки аргументов метода во всех точках вызова этого метода. Поэтому удобней и эффективней рассматривать поливариантный по методам VT-анализ [Klim05a,Klim05b]. В этом случае VT-анализ может построить несколько разметок метода в зависимости от разметок аргументов метода.

Также во время построения разметки метода VT-анализ может выполнять эквивалентные преобразования программы (например, разворачивать циклы или дублировать код после условия) для достижения более качественной разметки. Такой VT-анализ называют поливариантным по инструкциям [Klim05a,Klim05b]. В противоположность этому, моновариантный по инструкциям VT-анализ размечает методы без преобразования (за исключением

добавления инструкции Lift, о которой сказано ниже).

Опишем допустимую разметку и требования, предъявляемые к разметке. Данные требования подходят как для моновариантного по методам и инструкциям ВТ-анализа, так и для поливариантного ВТ-анализа.

### 3. ВТ-разметка

Размеченная программа состоит из двух частей: ВТ-программы и ВТ-кучи. В ВТ-программе к каждой инструкции приписана разметка S, D или X, а к каждому типу аргумента и результата приписана разметка, описываемая объектами в ВТ-куче.

#### 3.1. ВТ-программа

Разметка определяется для всей программы, поэтому дадим определение размеченной ВТ-программе (рис. 2).

<pre> ВТProgram = [ВТClassDef]  ВТClassDef = (ClassName, [ClassName], [FieldDec], [ВТMethodDef]) ВТMethodDef = (MethodName, IsInline, [ВТType], [ВТType], ВТMethodBody) ВТMethodSig = (MethodName, IsInline, [ВТType], [ВТType])  IsInline = INLINE   NOINLINE ВТType = (Type, ВТValue) </pre>
--

Рис. 2. ВТ-программа.

ВТ-программа состоит из определений ВТ-классов. Каждый ВТ-класс, как и обычный класс, описывается именем класса, списком имен классов (от которых данный класс наследуется), объявлением полей и определением ВТ-методов.

Каждое определение ВТ-метода состоит из имени метода, флага IsInline, о котором сказано ниже, двух списков ВТ-типов, обозначающих разметку аргументов и результатов, и ВТ-тело метода. ВТ-тип — это тип и ВТ-значение, о котором сказано ниже.

В ВТ-программе к каждому имени метода может быть приписан флаг INLINE или NOINLINE. Это означает, что вызов этого метода будет раскрыт при генерации остаточной программы: вместо вызова метода будет подстав-

лено тело метода.

BT-тело метода состоит из объявлений переменных и BT-инструкций, состоящих из инструкции и разметки S, D или X (рис. 3). По сравнению с описанием обычного языка SOOL добавлена новая инструкция Lift, которая описана ниже.

```

BTMethodBody = ([VarDec], [BTInstruction])
BTInstruction = (Instruction', BTInstructionAnnotation) = InstructionBTInstructionKind
BTInstructionAnnotation = BTInstructionKind | BTNewObjectAnnotation
BTNewObjectAnnotation = (BTInstructionKind, BTRefValue)
BTInstructionKind = S | D | X
Instruction' = Instruction | Lift

```

Рис. 3. BT-метод.

BT-виды, записанные в BT-инструкциях, будем называть BT-разметкой или просто разметкой BT-инструкций.

Для компактного описания правил разметки BT-инструкции будем записывать не в виде пары инструкция и BT-вид, а в виде инструкции с верхним индексом BT-вида: например, DuplicateStackTop<sup>S</sup>.

Для инструкции NewObject class используется расширенная разметка BTNewObjectAnnotation, описывающая как разметку инструкции, так и разметку создаваемого объекта. В таком случае разметку будем записывать в виде NewObject<sup>BTInstructionKind</sup> ClassName<sup>BTRefValue</sup>.

Если сравнивать программу и размеченную BT-программу, то разница заключается в следующих аспектах: во-первых, к каждому методу приписан флаг, показывающий нужно ли раскрывать метод (INLINE) во время генерации остаточной программы или нет (NOINLINE); во-вторых, к каждому аргументу и результату метода в BT-программе приписана разметка BT-Value; в-третьих, к каждой инструкции метода в BT-программе приписана разметка BTInstructionKind.

### 3.2. Корректность BT-программы

Как и для корректных программ на языке SOOL, для корректных размеченных программ требуется, чтобы все имена локальных переменных, классов, полей и методов (за исключением переопределенных методов) были уни-



кальными. Этого можно добиться переименованием имен. Также требуется, чтобы в теле метода в инструкциях использовались только определенные в программе локальные переменные, классы, поля и методы.

Все вызовы в языке SOOL виртуальные — т.е. тело метода будет выбрано по типу первого аргумента. Поэтому при определении ВТ-программы требуется, чтобы все методы с именем `mthd` имели одинаковую ВТ-сигнатуру: чтобы значения флага и ВТ-значения соответствующих аргументов и результатов совпадали.

Будем считать, что для каждой размеченной программы определена функция `MethodBTSignaturebtProg`, которая по имени метода возвращает разметку аргументов и результатов (рис. 4).

$$\boxed{\text{MethodBTSignature}_{\text{btProg}} : \text{MethodName} \rightarrow ([\text{BTValue}], [\text{BTValue}])}$$

Рис. 4. Вспомогательная функция.

Как и для обычных программ, метод `Main` класса `MAIN` должен существовать и его аргументы должны иметь примитивные типы. Этот метод должен иметь флаг `NOINLINE`.

Программа должна быть типизируемой. О выполнении и типизации новой инструкции `Lift` сказано ниже.

В зависимости от значения флага метода `IsInline` на разметку аргументов и результатов метода накладываются различные условия.

Если метод с флагом `INLINE`, то ВТ-вид первого аргумента обязан быть `S`. В этом случае генератор остаточной программы сможет произвести раскрытие виртуального вызова: по типу первого аргумента выбрать нужный метод.

Метод может возвращать значение как через результаты, так и изменяя значения полей объектов или элементов массивов, переданных как аргументы метода. Если указан флаг `NOINLINE`, то генератор остаточной программы не раскрывает метод, а генерирует вызов специализированной версии метода. И если бы возвращаемые значения (как результаты, так и поля объектов или элементы массивов в аргументах) имели разметку `S`, то возможна ситуация,

что в зависимости от неизвестных данных в одном случае метод должен вернуть одно S-значение, а в другом — другое. И если метод не раскрывается, то нельзя выбрать подходящее S-значение для передачи в вызывающий метод. Поэтому если метод с флагом NOINLINE, то BT-виды результатов и BT-виды полей объектов или элементов массивов в аргументах должны быть D.

Если вышеприведенные требования выполнены, то такую BT-программу будем называть корректной (рис. 5). Ниже рассматриваются только корректные BT-программы.

- Все локальные переменные, классы, поля имеют различные уникальные названия.
  - Все методы имеют различные уникальные названия за исключением переопределенных методов.
  - btProg — типизируема
  - Все методы с одним именем имеют одинаковую BT-сигнатуру.
  - Метод Main класса MAIN определен и имеет только примитивные аргументы и флаг NOINLINE.
  - Если метод INLINE, то BT-вид первого аргумента S.
  - Если метод NOINLINE, то BT-вид всех результатов и полей объектов и элементов массивов D.
- 
- btProg — корректна

Рис. 5. Корректность BT-программы.

### 3.3. BT-куча

Для определения разметки аргументов и результатов будем использовать BT-значение и BT-кучу (рис. 6).

```

BTHeap = Address → (BTKind, [Type], BTOBJECT)

BTValue = BTPrimValue | BTRefValue
BTPrimValue = (PrimType, BTKind) = PrimTypeBTKind
BTKind = S | D

BTRefValue = Address
BTOBJECT = (FieldName | ELEMENT) → BTValue

BTKindOfbtHeap : BTValue → BTKind
BTKindOfbtHeap (primType, btKind) = btKind
BTKindOfbtHeap ref = btKind where (btKind, _, _) = btHeap(ref)

```

Рис. 6. BT-куча.

BT-значение — это либо примитивное BT-значение, состоящее из примитивного типа и BT-вида  $S$  или  $D$ , либо ссылочное BT-значение, являющееся адресом.

BT-куча — это отображение адреса в тройку: BT-вид  $S$  или  $D$ , список типов и BT-объект. В отличие от обычной кучи, в BT-куче каждый BT-объект определяется списком типов. BT-объект это отображение имен полей всех классов, определенных в этом списке типов, или особого имени ELEMENT в BT-значение. Это отображение используется для определения разметки полей BT-объекта и разметки элементов массива (по ключу ELEMENT).

Для удобства описания примитивные типы будем записывать не в виде пары (тип, BT-вид), а приписывать BT-вид в виде верхнего индекса к типу, например,  $INT^S$ ,  $FLOAT^D$ .

Для определения BT-вида ссылочных и примитивных BT-значений будем использовать функцию  $BTKindOf_{btHeap}$ , которая по BT-значению возвращает BT-вид. Этот BT-вид будем называть BT-разметкой или просто разметкой BT-значений.

### 3.4. Корректность BT-кучи

BT-куча корректна, если у объектов BT-вида  $D$  все поля имеют BT-вид  $D$  (рис. 7).

$\begin{aligned} &\forall btRefValue \in \text{Domain}(btHead), \quad (\_, \_, btObject) = btHead(btRefValue), \\ &\quad \forall fldName \in \text{Domain}(btObject), \quad btVal = btObject(fldName) : \\ &\quad BTKindOf_{btHeap}(btRefValue) = D \Rightarrow BTKindOf_{btHeap}(btVal) = D \end{aligned}$ <hr style="width: 80%; margin: 0 auto;"/> $btHeap \text{ — корректна}$
--

Рис. 7. Корректность BT-кучи.

### 3.5. Корректность BT-разметки

Для проверки, что BT-разметка построена правильно, как и в случае с типизацией, необходимо построить размеченное состояние в каждой точке метода, которое определяет разметку элементов стека и переменных.

BT-разметка зависит от BT-разметки аргументов программы. Напомним, что аргументами и результатами программы могут быть только данные примитивных типов. Поэтому для задания BT-разметки аргументов достаточ-

но задать ВТ-виды аргументов.

#### 4. Правила ВТ-разметки

Правила ВТ-разметки описывают ограничения на разметку до и после выполнения инструкции. Если построена ВТ-программа, ВТ-куча и разметка в каждой точке программы, и они согласованы по ниже приведенным правилам, то такая разметка называется корректной.

По корректной разметке и начальным данным генератор остаточной программы может построить специализированную версию программы, выполнив все S-инструкции.

Правила ВТ-разметки похожи на правила, описываемые в стиле операционной семантики [Plot83, Kahn87]. Однако эти правила следует рассматривать, не как правила вывода ВТ-состояний, а как ограничение на состояние до и после выполнения инструкции (аналогично правилам для типизации программы). И для построения корректной разметки необходимо решить систему уравнений, рассматривая каждое правило как уравнение.

##### 4.1. ВТ-состояние

ВТ-состояние похоже на обычное состояния, только в нем нет кучи. ВТ-состояние состоит из ВТ-стека и ВТ-окружения (рис. 8). ВТ-стек — это список ВТ-значений, а ВТ-окружение — это отображение переменных в ВТ-значения.

$\text{BTState} = (\text{BTStack}, \text{BTEnv})$ $\text{BTStack} = [\text{BTValue}]$ $\text{BTEnv} = \text{Var} \rightarrow \text{BTValue}$
--

Рис. 8. ВТ-состояние.

##### 4.2. ВТ-разметка программы

Чтобы проверить, что ВТ-программа `btProg` и ВТ-куча `btHeap` построены правильно, нужно проверить следующее (рис. 9):

1. Правила ВТ-разметки корректны только для корректной ВТ-кучи `btHeap`.
2. Правила ВТ-разметки корректны только для корректной и типизируе-

мой программы, поэтому необходимо проверить, что программа  $btProg$  корректна и типизируема.

3. Для начального метода  $[NewObject^D MAIN, CallMethod^X Main, Leave^X]$  необходимо проверить существование отображения  $\lambda_0 : Integer \rightarrow VTState$ , проверяющего VT-разметку этого метода. Причем
  - а. VT-аргументы программы  $\lambda_0(0)$  должны быть примитивного типа, т.к. аргументы программы и метода  $Main$  должны быть примитивного типа.
  - б. VT-виды разметки результатов  $\lambda_0(2)$  должны быть  $D$ .
4. Для каждого определения VT-метода  $btMthdDef$  необходимо проверить существование отображения  $\lambda_{btMthdDec} : Integer \rightarrow VTState$ , проверяющего VT-разметку этого метода.

<ol style="list-style-type: none"> <li>1. <math>btHeap</math> — корректна</li> <li>2. <math>btProg</math> — корректна и типизируема</li> <li>3. <math>\exists \lambda_0 : Integer \rightarrow VTState :</math>  <math>instrs = [NewObject^D MAIN, CallMethod^X Main, Leave^X],</math>  <math>btArg = fst \lambda_0(0), \quad btRes = fst \lambda_0(2),</math>  <math>\forall btVal \in btArg : (primType, \_) = btVal,</math>  <math>\forall btVal \in btRes : BTKind(btVal) = D,</math>  <math>\forall n \ 0 \leq n &lt; 3 : btProg, btHeap, \lambda_0, btRes \vdash_{bt} (instrs, n)</math></li> <li>4. <math>\forall btClass \in btProg, (\_, \_, \_, btMthdDefs) = btClasss,</math>  <math>\forall btMthdDef \in btMthdDefs, \exists \lambda_{btMthdDec} : Integer \rightarrow VTState :</math>  <math>btProg, btHeap, \lambda_{btMthdDec} \vdash_{bt} btMthdDef</math></li> </ol> <hr style="border: 0.5px solid black; margin: 5px 0;"/> $btHeap \vdash_{bt} btProg$
--

Рис. 9. Правило VT-разметки программы.

### 4.3. VT-разметка метода

Размеченная VT-программа  $btProg$ , VT-куча  $btHeap$  и отображение  $\lambda_{btMthdDec}$  корректно указывают VT-типы для VT-метода  $btMthdDef$ , если (рис. 10):

1. Для всех VT-типов аргументов и результатов метода необходимо, чтобы их тип и тип VT-значения совпадали в случае примитивного типа, или их тип был элементом списка типов VT-значения в случае ссылочного типа.

2. ВТ-значения, находящиеся на стеке перед выполнением первой инструкции ( $\text{fst } \lambda_{\text{btMthdDec}}(0)$ ), совпадают с ВТ-значениями, приписанными к типам в ВТ-типах.
3. Для каждого номера инструкции  $n$  в контексте ВТ-программы  $\text{btProg}$ , ВТ-кучи  $\text{btHeap}$ , отображения  $\lambda_{\text{btMthdDec}}$  и ВТ-типов результата  $\text{btRes}$  разметка  $n$ -ой инструкции из списка  $\text{instrs}$  корректна.

$ \begin{aligned} & (\_, \text{btTypeArg}, \text{btTypeRes}, (\_, \text{btInstrs})) = \text{btMthdDef} \\ & \forall \text{btType} \in \text{btTypeArg} : \\ & \quad \bullet \text{ либо } (\text{btType} \text{ — ссылочный ВТ-тип}) \\ & \quad \quad (\text{type}, \text{btOref}) = \text{btType}, (\_, \text{types}, \_) = \text{btHeap}(\text{btOref}), \text{type} \in \text{types} \\ & \quad \bullet \text{ либо } (\text{btType} \text{ — примитивный ВТ-тип}) \\ & \quad \quad (\text{type}_1, (\text{type}_2, \text{btKind})) = \text{btType}, \text{type}_1 = \text{type}_2 \\ & \forall \text{btType} \in \text{btTypeRes} : \\ & \quad \bullet \text{ либо } (\text{btType} \text{ — ссылочный ВТ-тип}) \\ & \quad \quad (\text{type}, \text{btOref}) = \text{btType}, (\_, \text{types}, \_) = \text{btHeap}(\text{btOref}), \text{type} \in \text{types} \\ & \quad \bullet \text{ либо } (\text{btType} \text{ — примитивный ВТ-тип}) \\ & \quad \quad (\text{type}_1, (\text{type}_2, \text{btKind})) = \text{btType}, \text{type}_1 = \text{type}_2 \\ & \text{fst } \lambda_{\text{btMthdDec}}(0) = \text{btArg} = \text{map snd btTypeArg} \quad \text{btRes} = \text{map snd btTypeRes} \\ & \forall n \ 0 \leq n < \text{length}(\text{instrs}) : \text{btProg}, \text{btHeap}, \lambda_{\text{btMthdDec}}, \text{btRes} \vdash_{\text{bt}} (\text{btInstrs}, n) \\ \hline & \text{btProg}, \text{btHeap}, \lambda_{\text{btMthdDec}} \vdash_{\text{bt}} \text{btMthdDef} \end{aligned} $
--

Рис. 10. Правило ВТ-разметки метода.

#### 4.4. ВТ-разметка последовательности инструкций

Корректность разметки  $n$ -ой инструкции в списке инструкций  $\text{instrs}$  проверяется в зависимости от этой инструкции:

1. Если  $n$ -ая инструкция — это инструкция  $\text{Leave}^X$ , то ВТ-значения на стеке ( $\text{fst } \lambda_{\text{btMthdDec}}(n)$ ) должны совпадать со ВТ-значениями результата (рис. 11).

$ \frac{\text{Leave}^X = \text{btInstrs}[n] \quad \text{fst } \lambda_{\text{btMthdDec}}(n) = \text{btRes}}{\text{btProg}, \text{btHeap}, \lambda_{\text{btMthdDec}}, \text{btRes} \vdash_{\text{bt}} (\text{btInstrs}, n)} $
---

Рис. 11. Правило ВТ-разметки последовательности инструкций, инструкция  $\text{Leave}^X$ .

2. Если  $n$ -ая инструкция — это инструкция  $\text{Goto}^S \ m$ , то ВТ-значения на стеке и в переменных перед выполнением этой инструкции  $\lambda_{\text{btMthdDec}}(n)$  должны совпадать с ВТ-значениями перед  $m$ -ой инструкцией

$\lambda_{\text{btMthdDec}}(m)$  (рис. 12).

$$\frac{\text{Goto}^S m = \text{btInstrs}[n] \quad \lambda_{\text{btMthdDec}}(n) = \lambda_{\text{btMthdDec}}(m)}{\text{btProg}, \text{btHeap}, \lambda_{\text{btMthdDec}}, \text{btRes} \vdash_{\text{bt}} (\text{btInstrs}, n)}$$

Рис. 12. Правило ВТ-разметки последовательности инструкций,  
инструкция  $\text{Goto}^S m$ .

3. Если  $n$ -ая инструкция — это инструкция  $\text{Branch}^x m$ , то разметка  $x$  инструкции — либо  $S$ , либо  $D$ . ВТ-окружения до выполнения этой  $n$ -ой инструкции, до выполнения  $(n+1)$ -ой инструкции и до выполнения  $m$ -ой инструкции должны совпадать. ВТ-стеки до выполнения этой  $n$ -ой инструкции без головного элемента и стеки до выполнения  $(n+1)$ -ой инструкции и до выполнения  $m$ -ой инструкции должны совпадать. ВТ-значение, находящееся на вершине стека до выполнения этой  $n$ -ой инструкции, должно быть  $\text{INT}^x$ , где  $x$  — ВТ-вид инструкции  $\text{Branch}^x m$  (рис. 13).

$$\frac{\text{Branch}^x m = \text{btInstrs}[n] \quad (\text{INT}^x::\text{btStack}, \text{btEnv}) = \lambda_{\text{btMthdDec}}(n) \quad (\text{btStack}, \text{btEnv}) = \lambda_{\text{btMthdDec}}(n+1) = \lambda_{\text{btMthdDec}}(m) \quad x = S \mid x = D}{\text{btProg}, \text{btHeap}, \lambda_{\text{btMthdDec}}, \text{btRes} \vdash_{\text{bt}} (\text{btInstrs}, n)}$$

Рис. 13. Правило ВТ-разметки последовательности инструкций,  
инструкция  $\text{Branch}^x m$ .

4. Для остальных инструкций необходимо проверить правила, записанные в виде правил связи состояния до и после выполнения инструкции (рис. 14):

$$\frac{\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{btInstrs}[n] : \lambda_{\text{btMthdDec}}(n) \rightarrow \lambda_{\text{btMthdDec}}(n+1)}{\text{btProg}, \text{btHeap}, \lambda_{\text{btMthdDec}}, \text{btRes} \vdash_{\text{bt}} (\text{btInstrs}, n)}$$

Рис. 14. Правило ВТ-разметки последовательности инструкций.

#### 4.5. ВТ-разметка инструкций

##### **DuplicateStackTop**

Инструкция `DuplicateStackTop` копирует элемент на вершине стека.

Правило ВТ-разметки требует, чтобы два ВТ-значения на вершине стека после выполнения инструкции совпадали с ВТ-значением на вершине стека до выполнения инструкции (рис. 15). Остальные элементы стека и окружения

до и после выполнения инструкции также должны совпадать.

$$\frac{x = \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \mid D \quad \text{btStack}' = \text{btVal}::\text{btVal}::\text{btStack}}{\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{DuplicateStackTop}^x : (\text{btVal}::\text{btStack}, \text{btEnv}) \rightarrow (\text{btStack}', \text{btEnv})}$$

Рис. 15. Правило ВТ-разметки инструкции  $\text{DuplicateStackTop}^x$ .

Также правило требует, чтобы разметка ВТ-значения на вершине стека до выполнения инструкции и разметка инструкции совпадали и равнялись либо S, либо D.

### **RemoveStackTop**

Инструкция  $\text{RemoveStackTop}$  удаляет элемент на вершине стека.

Правило ВТ-разметки требует, чтобы стек до выполнения инструкции без верхнего элемента совпадал со стеком после выполнения инструкции (рис. 16). Окружение не изменялось.

$$\frac{x = \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \mid D}{\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{RemoveStackTop}^x : (\text{btVal}::\text{btStack}, \text{btEnv}) \rightarrow (\text{btStack}, \text{btEnv})}$$

Рис. 16. Правило ВТ-разметки инструкции  $\text{RemoveStackTop}^x$ .

Разметка инструкции и разметка ВТ-значения на вершине стека до выполнения инструкции должны совпадать и равняться либо S, либо D.

### **LoadConst const**

Инструкция  $\text{LoadConst const}$  загружает константу  $\text{const}$ .

Правило ВТ-разметки требует, чтобы стек после выполнения инструкции без верхнего элемента совпадал со стеком до выполнения инструкции (рис. 17). Окружение не изменялось. ВТ-значение, находящееся на вершине стека после выполнения инструкции может быть следующим:

$$\frac{\begin{array}{l} x = \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \mid D \\ \bullet \text{ либо } \text{btVal} = \text{INT}^x, \text{ если } \text{const} \text{ — целое число} \\ \bullet \text{ либо } \text{btVal} = \text{FLOAT}^x, \text{ если } \text{const} \text{ — число с плавающей точкой} \\ \bullet \text{ либо } \text{btVal} : \text{BTRefValue}, \text{ если } \text{const} = \text{NULL} \end{array}}{\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{LoadConst}^x \text{ const} : (\text{btStack}, \text{btEnv}) \rightarrow (\text{btVal}::\text{btStack}, \text{btEnv})}$$

Рис. 17. Правило ВТ-разметки инструкции  $\text{LoadConst}^x \text{ const}$ .

- если  $\text{const}$  — целое число, то ВТ-значение — это  $\text{INT}^x$ ;
- если  $\text{const}$  — число с плавающей точкой, то ВТ-значение — это



FLOAT<sup>x</sup>;

- если `const` — `NULL`, то ВТ-значение — адрес какого-либо ВТ-объекта в ВТ-куче `btHeap`.

Разметка инструкции и разметка этого ВТ-значения должны совпадать и могут быть либо `S`, либо `D`.

### UnaryOp op

Инструкция `UnaryOp op` выполняет унарную операцию `op`.

Правило ВТ-разметки требует, чтобы совпадали окружения и стеки до и после выполнения за исключением вершин стеков (рис. 18). ВТ-значения, расположенные на вершинах стеков могут быть следующими:

- либо оба типа `INTx`, если операция `op` — это `NOT` или `NEG`;
- либо оба типа `FLOATx`, если операция `op` — это `NEG`;
- либо ВТ-значение до выполнения — `INTx`, а после выполнения — `FLOATx`, если операция `op` — это `INT2FLOAT`;
- либо ВТ-значение до выполнения — `FLOATx`, а после выполнения — `INTx`, если операция `op` — это `FLOAT2INT`.

$x = \text{BTKindOf}_{\text{btHeap}}(\text{btVal}_1) = \text{BTKindOf}_{\text{btHeap}}(\text{btVal}_2) = S \mid D$ <ul style="list-style-type: none"> <li>• либо <math>\text{btVal}_1 = \text{btVal}_2 = \text{INT}^x</math>, если <math>op = \text{NOT}</math> или <math>op = \text{NEG}</math></li> <li>• либо <math>\text{btVal}_1 = \text{btVal}_2 = \text{FLOAT}^x</math>, если <math>op = \text{NEG}</math></li> <li>• либо <math>\text{btVal}_1 = \text{INT}^x</math>, <math>\text{btVal}_2 = \text{FLOAT}^x</math>, если <math>op = \text{INT2FLOAT}</math></li> <li>• либо <math>\text{btVal}_1 = \text{FLOAT}^x</math>, <math>\text{btVal}_2 = \text{INT}^x</math>, если <math>op = \text{FLOAT2INT}</math></li> </ul>
$\text{prog, btHeap} \vdash_{\text{bt}} \text{UnaryOp}^x \text{ op} : (\text{btVal}_1::\text{btStack}, \text{btEnv}) \rightarrow (\text{btVal}_2::\text{btStack}, \text{btEnv})$

Рис. 18. Правило ВТ-разметки инструкции `UnaryOpx op`

Разметка инструкции и разметка этих ВТ-значений должны совпадать и могут быть либо `S`, либо `D`.

### BinaryOp op

Инструкция `BinaryOp op` выполняет бинарную операцию `op`.

Правило ВТ-разметки требует, чтобы совпадали окружения и стеки до и после выполнения за исключением двух ВТ-значений, расположенных на вершине стека до выполнения инструкции, и одного ВТ-значения, расположенного на вершине стека после выполнения инструкции (рис. 19). Эти ВТ-

значения могут быть следующими:

- все три ВТ-значения —  $\text{INT}^x$ , если операция  $op$  — любая операция: арифметическая (ADD, DIV, MUL, REM, SUB), побитовая (AND, OR, XOR, SHL, SHR) или операция сравнения (CEQ, CGT, CLT);
- все три ВТ-значения —  $\text{FLOAT}^x$ , если операция  $op$  — арифметическая операция (ADD, DIV, MUL, REM, SUB);
- два ВТ-значения на вершине стека до выполнения инструкции —  $\text{FLOAT}^x$ , а ВТ-значение на вершине стека после выполнения инструкции —  $\text{INT}^x$ , если операция  $op$  — операция сравнения (CEQ, CGT, CLT);
- два ВТ-значения на вершине стека — это адреса ВТ-объектов в ВТ-куче  $btHeap$ , а ВТ-значение на вершине стека после выполнения инструкции —  $\text{INT}^x$ , если операция  $op$  — операция сравнения на равенство (CEQ).

$x = \text{BTKindOf}_{btHeap}(btVal_1) = \text{BTKindOf}_{btHeap}(btVal_2) =$ $= \text{BTKindOf}_{btHeap}(btVal_3) = S \mid D$ <ul style="list-style-type: none"> <li>• либо <math>btVal_1 = btVal_2 = btVal_3 = \text{INT}^x</math>, если <math>op = \text{ADD, DIV, MUL, REM, SUB, AND, OR, XOR, SHL, SHR, CEQ, CGT, CLT}</math></li> <li>• либо <math>btVal_1 = btVal_2 = btVal_3 = \text{FLOAT}^x</math>, если <math>op = \text{ADD, DIV, MUL, REM, SUB}</math></li> <li>• либо <math>btVal_1 = btVal_2 = \text{FLOAT}^x, btVal_3 = \text{INT}^x</math>, если <math>op = \text{CEQ, CGT, CLT}</math></li> <li>• либо <math>btVal_1 : \text{BTRefValue}, btVal_2 : \text{BTRefValue}, btVal_3 = \text{INT}^x</math>, если <math>op = \text{CEQ}</math></li> </ul> <hr style="border: 0.5px solid black;"/> $\text{prog}, btHeap \vdash_{bt} \text{UnaryOp}^x op : (btVal_1::btVal_2::btStack, btEnv) \rightarrow$ $(btVal_3::btStack, btEnv)$
--

Рис. 19. Правило ВТ-разметки инструкции  $\text{BinaryOp}^x op$ .

Разметка инструкции и разметка этих ВТ-значений должны совпадать и могут быть либо S, либо D.

### **LoadVar var**

Инструкция `LoadVar var` читает значение переменной `var`.

Правило ВТ-разметки требует, чтобы стек после выполнения инструкции состоял из ВТ-значения переменной `var`, записанного в окружении, и сте-

ка до выполнения инструкции (рис. 20). Окружения до и после выполнения инструкции должны совпадать.

$$\begin{array}{c}
 \text{btVal} = \text{btEnv}(\text{var}) \\
 \bullet \text{ либо } x = S, \text{ BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \\
 \bullet \text{ либо } x = X, \text{ BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \\
 \hline
 \text{prog, btHeap} \vdash_{\text{bt}} \text{LoadVar}^x \text{ var} : (\text{btStack}, \text{btEnv}) \rightarrow (\text{btVal}::\text{btStack}, \text{btEnv})
 \end{array}$$

Рис. 20. Правило ВТ-разметки инструкции  $\text{LoadVar}^x \text{ var}$ .

В зависимости от разметки ВТ-значения переменной  $\text{var}$  S или D, инструкция должна быть размечена либо S, либо X, соответственно.

### **StoreVar var**

Инструкция  $\text{StoreVar var}$  записывает значение переменной  $\text{var}$ .

Правило ВТ-разметки требует, чтобы стек до выполнения инструкции состоял из ВТ-значения и стека после выполнения инструкции (рис. 21). Окружения после выполнения инструкции должны совпадать с окружением до выполнения инструкции за исключением значения на переменной  $\text{var}$ , которое должно быть равным ВТ-значению со стека.

$$\begin{array}{c}
 \bullet \text{ либо } x = S, \text{ BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \\
 \bullet \text{ либо } x = X, \text{ BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \\
 \text{btEnv}' = \text{btEnv}[\text{var} \rightarrow \text{btVal}] \\
 \hline
 \text{prog, btHeap} \vdash_{\text{bt}} \text{StoreVar}^x \text{ var} : (\text{btVal}::\text{btStack}, \text{btEnv}) \rightarrow (\text{btStack}, \text{btEnv}')
 \end{array}$$

Рис. 21. Правило ВТ-разметки инструкции  $\text{StoreVar}^x \text{ var}$ .

В зависимости от разметки ВТ-значения на вершине стека S или D, инструкция должна быть размечена либо S, либо X, соответственно.

### **NewObject class**

Инструкция  $\text{NewObject class}$  создает новый объект класса  $\text{class}$ .

Правило ВТ-разметки требует, чтобы стек после выполнения инструкции состоял из ВТ-значения, адреса ВТ-объекта в ВТ-куче  $\text{btHeap}$ , и стека до выполнения инструкции (рис. 22). Окружения до и после выполнения инструкции должны совпадать. Также необходимо, чтобы в списке типов этого ВТ-объекта содержался класс  $\text{class}$ .

В зависимости от разметки ВТ-объекта на вершине стека после выполнения инструкции S или D, инструкция должна быть размечена либо X, либо

D, соответственно.

$$\begin{array}{c}
 (\_, \text{types}, \text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{class} \in \text{types} \\
 \bullet \text{ либо } x = X, \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S \\
 \bullet \text{ либо } x = D, \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = D \\
 \hline
 \text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{NewObject}^x \text{ class}^{\text{btOref}} : (\text{btStack}, \text{btEnv}) \rightarrow \\
 (\text{btOref}::\text{btStack}, \text{btEnv})
 \end{array}$$

Рис. 22. Правило ВТ-разметки инструкции  $\text{NewObject}^x \text{ class}$ .

### LoadField fld

Инструкция  $\text{LoadField fld}$  читает у объекта поле  $\text{fld}$ .

Правило ВТ-разметки инструкции  $\text{LoadField fld}$  требует, чтобы ВТ-значение на вершине стека до выполнения инструкции было адресом ВТ-объекта в ВТ-куче  $\text{btHeap}$  (рис. 23). ВТ-значение на вершине стека после выполнения инструкции должно совпадать с ВТ-значением, соответствующим полю  $\text{fld}$  в этом ВТ-объекте. Остальные элементы стеков и окружения до и после выполнения инструкции должны совпадать.

$$\begin{array}{c}
 (\_, \_, \text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{btVal} = \text{btObj}(\text{fld}) \\
 \bullet \text{ либо } x = S, \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S, \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \\
 \bullet \text{ либо } x = X, \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S, \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \\
 \bullet \text{ либо } x = D, \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = D, \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \\
 \hline
 \text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{LoadField}^x \text{ fld} : (\text{btOref}::\text{btStack}, \text{btEnv}) \rightarrow (\text{btVal}::\text{btStack}, \text{btEnv})
 \end{array}$$

Рис. 23. Правило ВТ-разметки инструкции  $\text{LoadField}^x \text{ fld}$ .

В зависимости от разметки ВТ-объекта на вершине стека до выполнения инструкции и разметки ВТ-значения на вершине стека после выполнения инструкции, инструкция должна быть размечена:

- либо S, если ВТ-объект и ВТ-значение размечены S;
- либо X, если ВТ-объект размечен S, а ВТ-значение размечено D;
- либо D, если ВТ-объект и ВТ-значение размечены D.

### StoreField fld

Инструкция  $\text{StoreField fld}$  записывает у объекта поле  $\text{fld}$ .

Правило ВТ-разметки инструкции  $\text{StoreField fld}$  требует, чтобы на стеке до выполнения инструкции находилось ВТ-значение и адрес ВТ-объекта в ВТ-куче  $\text{btHeap}$ , причем ВТ-значение должно совпадать с ВТ-значением, со-

ответствующим полю fld в этом ВТ-объекте (рис. 24). Остальные элементы стеков и окружения до и после выполнения инструкции должны совпадать.

$$\begin{array}{l}
 (\_, \_, \text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{btVal} = \text{btObj}(\text{fld}) \\
 \bullet \text{ либо } x = S, \text{ ВТKindOf}_{\text{btHeap}}(\text{btOref}) = S, \text{ ВТKindOf}_{\text{btHeap}}(\text{btVal}) = S \\
 \bullet \text{ либо } x = X, \text{ ВТKindOf}_{\text{btHeap}}(\text{btOref}) = S, \text{ ВТKindOf}_{\text{btHeap}}(\text{btVal}) = D \\
 \bullet \text{ либо } x = D, \text{ ВТKindOf}_{\text{btHeap}}(\text{btOref}) = D, \text{ ВТKindOf}_{\text{btHeap}}(\text{btVal}) = D \\
 \hline
 \text{prog, btHeap} \vdash_{\text{bt}} \text{StoreField}^x \text{ fld: } (\text{btVal}::\text{btOref}::\text{btStack}, \text{btEnv}) \rightarrow (\text{btStack}, \text{btEnv})
 \end{array}$$

Рис. 24. Правило ВТ-разметки инструкции StoreField<sup>x</sup> fld.

В зависимости от разметки ВТ-значения и ВТ-объекта на вершине стека до выполнения инструкции, инструкция должна быть размечена:

- либо S, если ВТ-объект и ВТ-значение размечены S;
- либо X, если ВТ-объект размечен S, а ВТ-значение размечено D;
- либо D, если ВТ-объект и ВТ-значение размечены D.

### CallMethod mthd

Инструкция CallMethod mthd вызывает у объекта виртуальный метод mthd.

Правило ВТ-разметки инструкции CallMethod mthd требует, чтобы на вершине стека до выполнения инструкции находились ВТ-значения, совпадающие с ВТ-значениями аргументов метода mthd, заданные в сигнатуре этого метода (рис. 24). ВТ-значения на вершине стека после выполнения инструкции должны совпадать с ВТ-значениями результатов метода mthd, заданных в сигнатуре этого метода. Остальные элементы стеков и окружения до и после выполнения инструкции должны совпадать. Разметка инструкции всегда X.

$$\begin{array}{l}
 (\text{btArg}, \text{btRes}) = \text{MethodBTSignature}_{\text{btProg}}(\text{mthd}) \\
 \hline
 \text{prog, btHeap} \vdash_{\text{bt}} \text{CallMethod}^x \text{ mthd : } (\text{btArg}++\text{btStack}, \text{btEnv}) \rightarrow \\
 (\text{btRes}++\text{btStack}, \text{btEnv})
 \end{array}$$

Рис. 24. Правило ВТ-разметки инструкции CallMethod<sup>x</sup> mthd.

### CastObject type

Инструкция CastObject type приводит объект или массив к типу type.

Правило ВТ-разметки инструкции CastObject type требует, чтобы стеки и окружения до и после выполнения инструкции совпадали. Но также требует,

чтобы на вершине стека находился адрес ВТ-объекта. Причем в списке типов этого ВТ-объекта должен быть хотя бы один тип являющийся наследником типа  $type$ , заданного в параметре инструкции.

$$\frac{\text{btOref}::\_ = \text{btStack} \quad (\_, \text{types}, \_) = \text{btHeap}(\text{btOref}) \quad \exists \text{type}' \in \text{types} : \text{type}' <_{\text{prog}} \text{type} \quad x = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S \mid D}{\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{CastObject}^x \text{type} : (\text{btStack}, \text{btEnv}) \rightarrow (\text{btStack}, \text{btEnv})}$$

Рис. 25. Правило ВТ-разметки инструкции  $\text{CastObject}^x \text{type}$ .

Разметка инструкции и разметка ВТ-объекта на вершине стека должны совпадать и могут быть либо  $S$ , либо  $D$ .

### **NewArray type**

Инструкция  $\text{NewArray type}$  создает новый массив из элементов типа  $type$ .

Правило ВТ-разметки инструкции  $\text{NewArray type}$  требует, чтобы на вершине стека до выполнения инструкции находилось ВТ-значение  $\text{INT}^{x_1}$ , а после выполнения инструкции — ссылочное ВТ-значение  $\text{btOref}$  (рис. 26). Остальные элементы стека и окружения до и после выполнения инструкции должны совпадать. Ссылочное ВТ-значение  $\text{btOref}$  должно указывать в ВТ-куче на ВТ-объект, содержащий особое поле  $\text{ELEMENT}$ . Также требуется, чтобы список типов ВТ-объекта содержал тип массива  $type[]$ , где тип  $type$  задается в параметре инструкции.

$$\frac{(\_, \text{types}, \text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{type}[] \in \text{types} \quad \text{btVal} = \text{btObj}(\text{ELEMENT}) \quad \begin{array}{l} \bullet \text{ либо } x = S, x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S, \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \\ \bullet \text{ либо } x = X, x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S, \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \\ \bullet \text{ либо } x = D, x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = D, \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \end{array}}{\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{NewArray}^x \text{type} : (\text{INT}^{x_1}::\text{btStack}, \text{btEnv}) \rightarrow (\text{btOref}::\text{btStack}, \text{btEnv})}$$

Рис. 26. Правило ВТ-разметки инструкции  $\text{NewArray}^x \text{type}$ .

Разметки инструкции, ВТ-значения  $\text{INT}^{x_1}$ , ссылочного ВТ-значения  $\text{btOref}$  и ВТ-значения  $\text{btVal}$ , соответствующего элементам массива, связаны следующими правилами. Во-первых, разметки ВТ-значения  $\text{INT}^{x_1}$  и ссылочного ВТ-значения  $\text{btOref}$  должны совпадать. Во-вторых, в зависимости от разметки ВТ-значения  $\text{INT}^{x_1}$  и ВТ-значения  $\text{btVal}$  инструкция должна быть раз-

мечена:

- либо S, если BT- значения размечены S;
- либо X, если BT-значение  $INT^{x1}$  размечено S, а BT-значение btVal — D;
- либо D, если BT- значения размечены D.

### LoadLength

Инструкция LoadLength загружает длину массива.

Правило BT-разметки инструкции LoadLength требует, чтобы на вершине стека до выполнения инструкции находилось ссылочное BT-значение btOref, которое указывает в BT-куче btHeap на BT-объект btObj (рис. 27). Это отображение btObj должно быть определено на особом значении ELEMENT. На вершине стека после выполнения инструкции должно находиться примитивное BT-значение  $INT^x$ . Все остальные элементы стека и окружения до и после выполнения инструкции должны совпадать.

$(\_, \_, btObj) = btHeap(btOref) \quad btObj(ELEMENT) \text{ — определено}$ $x = BTKindOf_{btHeap}(btOref) = S \mid D$ <hr/> $prog, btHeap \vdash_{bt} LoadLength^x : (btOref::btStack, btEnv) \rightarrow (INT^x::btStack, btEnv)$
--

Рис. 27. Правило BT-разметки инструкции LoadLength<sup>x</sup>.

Разметки инструкции, ссылочного BT-значения на вершине стека до выполнения и примитивного BT-значения на вершине стека после выполнения инструкции должны совпадать и могут быть либо S, либо D.

### LoadElement

Инструкция LoadElement читает элемент массива.

Правило BT-разметки инструкции LoadElement требует, чтобы на вершине стека до выполнения инструкции находились примитивное BT-значение  $INT^{x1}$  и ссылочное BT-значение btOref, которое указывает в BT-куче btHeap на BT-объект btObj (рис. 28). Значение этого отображения btObj на особом значении ELEMENT (соответствующего элементам массива) должно совпадать с BT-значением btVal. На вершине стека после выполнения инструкции должно находиться BT-значение btVal, соответствующее элементам массива. Остальные элементы стека и окружения до и после выполнения инструкции должны

совпадать.

$(\_, \_, \text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{btVal} = \text{btObj}(\text{ELEMENT})$
<ul style="list-style-type: none"> <li>• либо <math>x = S</math>, <math>x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S</math>, <math>\text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S</math></li> <li>• либо <math>x = X</math>, <math>x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S</math>, <math>\text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D</math></li> <li>• либо <math>x = D</math>, <math>x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = D</math>, <math>\text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D</math></li> </ul>
<hr/> $\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{LoadElement}^x : (\text{INT}^{x_1}::\text{btOref}::\text{btStack}, \text{btEnv}) \rightarrow$ $(\text{btVal}::\text{btStack}, \text{btEnv})$

Рис. 28. Правило ВТ-разметки инструкции  $\text{LoadElement}^x$ .

Разметки инструкции, ВТ-значения  $\text{INT}^{x_1}$ , ссылочного ВТ-значения  $\text{btOref}$  и ВТ-значения  $\text{btVal}$ , соответствующего элементам массива, связаны следующими правилами. Во-первых, разметки ВТ-значения  $\text{INT}^{x_1}$  и ссылочного ВТ-значения  $\text{btOref}$  должны совпадать. Во-вторых, в зависимости от разметок ВТ-значения  $\text{INT}^{x_1}$  и ВТ-значения  $\text{btVal}$  инструкция должна быть размечена:

- либо  $S$ , если ВТ- значения размечены  $S$ ;
- либо  $X$ , если ВТ-значение  $\text{INT}^{x_1}$  размечено  $S$ , а ВТ-значение  $\text{btVal}$  —  $D$ ;
- либо  $D$ , если ВТ- значения размечены  $D$ .

### StoreElement

Инструкция  $\text{StoreElement}$  записывает элемент массива.

Правило ВТ-разметки инструкции  $\text{LoadElement}$  требует, чтобы на вершине стека до выполнения инструкции находились ВТ-значение  $\text{btVal}$ , примитивное ВТ-значение  $\text{INT}^{x_1}$  и ссылочное ВТ-значение  $\text{btOref}$ , которое указывает в ВТ-куче  $\text{btHeap}$  на ВТ-объект  $\text{btObj}$  (рис. 29). Значение этого отображения  $\text{btObj}$  на особом значении  $\text{ELEMENT}$  (соответствующего элементам массива) должно совпадать с ВТ-значению  $\text{btVal}$ . Остальные элементы стека и окружения до и после выполнения инструкции должны совпадать.

$(\_, \_, \text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{btVal} = \text{btObj}(\text{ELEMENT})$
<ul style="list-style-type: none"> <li>• либо <math>x = S</math>, <math>x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S</math>, <math>\text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S</math></li> <li>• либо <math>x = X</math>, <math>x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S</math>, <math>\text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D</math></li> <li>• либо <math>x = D</math>, <math>x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = D</math>, <math>\text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D</math></li> </ul>
<hr/> $\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{StoreElement}^x : (\text{btVal}::\text{INT}^{x_1}::\text{btOref}::\text{btStack}, \text{btEnv}) \rightarrow$ $(\text{btStack}, \text{btEnv})$

Рис. 29. Правило ВТ-разметки инструкции  $\text{StoreElement}^x$ .



Разметки инструкции, BT-значения  $INT^{x1}$ , ссылочного BT-значения  $btOref$  и BT-значения  $btVal$ , соответствующего элементам массива, связаны следующими правилами. Во-первых, разметки BT-значения  $INT^{x1}$  и ссылочного BT-значения  $btOref$  должны совпадать. Во-вторых, в зависимости от разметок BT-значения  $INT^{x1}$  и BT-значения  $btVal$ , соответствующего элементам массива, инструкция должна быть размечена:

- либо  $S$ , если BT- значения размечены  $S$ ;
- либо  $X$ , если BT-значение  $INT^{x1}$  размечено  $S$ , а BT-значение  $btVal$  —  $D$ ;
- либо  $D$ , если BT- значения размечены  $D$ .

### Lift

Инструкция `Lift` вводится в язык на этапе BT-анализа. В размеченной программе она «преобразует»  $S$ -данные в  $D$ -данные, но разрешается преобразовывать только примитивные данные — данные типа `INT` или типа `FLOAT`.

Правило BT-разметки инструкции `Lift` требует, чтобы на вершине стека до выполнения инструкции должно находиться примитивное BT-значение с разметкой  $S$ , а на вершине стека после выполнения инструкции — примитивное BT-значение с разметкой  $D$  с тем же типом (рис. 30). Инструкция `Lift` всегда имеет разметку  $X$ .

$$\boxed{\text{prog, btHeap} \vdash_{\text{bt}} \text{Lift}^X : (\text{primType}^S :: \text{btStack}, \text{btEnv}) \rightarrow (\text{primType}^D :: \text{btStack}, \text{btEnv})}$$

Рис. 30. Правило BT-разметки инструкции `Lift` <sup>$X$</sup> .

## 5. Выполнение размеченной программы

Размеченную программу можно выполнять как любую другую программу на языке `SOOL`.

Отличие размеченной программы от программы на языке `SOOL` заключается, во-первых, в разметках, приписанных к инструкциям и типам аргументов и результатов метода, во-вторых, в дополнительной информации, заданной BT-кучей, и, в-третьих, в новой инструкции `Lift`, которая может встречаться в теле размеченной программы.

При выполнении размеченной программы разметка и BT-куча игнорируются, а инструкция `Lift` выполняется, как ничего не делающая инструкция

(рис. 31).

$$\boxed{\text{prog} \vdash_{\tau} \text{Lift} : (\text{val}::\text{stack}, \text{env}, \text{heap}) \rightarrow (\text{val}::\text{stack}, \text{env}, \text{heap})}$$

Рис. 31. Правило выполнения инструкции Lift.

Аналогично правило типизации инструкции Lift показывает, что типы до и после выполнения инструкции должны совпадать (рис.32).

$$\boxed{\text{prog} \vdash_{\tau} \text{Lift} : \text{primType}::\text{tStack} \rightarrow \text{primType}::\text{tStack}}$$

Рис. 32. Правило типизации инструкции Lift.

Приведенные правила показывают, что размеченную программу можно рассматривать как обычную программу на языке SOOL.

Приведенные выше правила корректности ВТ-программы показывают только внутреннюю согласованность разметки. Но при построении размеченной программы, также необходимо гарантировать, что исходная и размеченная программы эквивалентны: при любых входных данных результаты выполнения программ должны совпадать.

Одним из способов достижения эквивалентности является минимальное изменение программы при разметке: только добавление разметки и в необходимых местах добавление инструкции Lift. Но такой способ, как было отмечено выше, не всегда позволяет достаточно качественно разметить программу: почти все инструкции могут оказаться размечены D.

## 6. Заключение

В работе описаны ВТ-разметка программы и правила, которым должна удовлетворять эта разметка. Поливариантный анализ времен связывания [Klim05a,Klim05b] должен строить ВТ-разметку, удовлетворяющую описанным правилам. По этой разметке генератор остаточной программы [Klim06] может построить остаточную программу.

На основе этих правил построен поливариантный анализ времен связывания [Klim05a,Klim05b], использующийся в специализаторе CILPE [Cher03,Klim08a,Klim08b] для языка CIL платформы Microsoft .NET [CLI,MS.NET].

## 7. Литература

[Cher03] A.M.Chevovsky, An.V.Klimov, Ar.V.Klimov, Yu.A.Klimov, A.S.Mishchenko, S.A.Romanenko, S.Yu.Skorobogatov "Partial Evaluation for Common Intermediate Language" // Manfred Broy and Alexandre V. Zamulin (eds.), Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI'2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003. LNCS 2890, Springer-Verlag, December 2003, pp.171-177.

[CLI] Common Language Infrastructure (CLI) // <http://www.ecma-international.org/publications/standards/Ecma-335.htm>, <http://msdn2.microsoft.com/en-us/netframework/aa569283.aspx>.

[Jones93] N.D.Jones, C.K.Gomard, P.Sestoft "Partial Evaluation and Automatic Compiler Generation" // C.A.R. Hoare Series, Prentice-Hall, 1993.

[Kahn87] G.Kahn "Natural semantics" // In Proceedings of the Symposium on Theoretical Aspects of Computer Sciences. LNCS 247, Springer-Verlag, 1987, pp.22-39.

[Klim05a] Ю.А.Климов "Поливариантный анализ времен связывания в специализаторе CILPE для Common Intermediate Language платформы Microsoft.NET" // Труды Всероссийской конференции "Технологии Microsoft в теории и практике программирования", Москва, 17-18 февраля 2005 г., М: Изд-во МГТУ им. Н.Э. Баумана, 2005, с.128.

[Klim05b] Ю.А.Климов "О поливариантном анализе времен связывания в специализаторе объектно-ориентированного языка" // Труды Всероссийской научной конференции "Научный сервис в сети Интернет: технологии распределенных вычислений", Новороссийск, 19-24 сентября 2005 г., М: Изд-во МГУ, с.89-91.

[Klim06] Ю.А.Климов "Генератор остаточной программы и корректность специализатора объектно-ориентированного языка" // Труды Всероссийской научной конференции "Научный сервис в сети Интернет: технологии параллельного программирования", Новороссийск, 18-23 сентября 2006 г., М: Изд-во МГУ, с.137-140.

[Klim08a] Ю.А.Климов "Особенности применения метода частичных вычис-

лений к специализации программ на объектно-ориентированных" // Препринт Института прикладной математики им. М.В. Келдыша РАН, 2008 г., №12.

[Klim08b] Ю.А.Климов "Возможности специализатора CILPE и примеры его применения к программам на объектно-ориентированных языках" // Препринт Института прикладной математики им. М.В. Келдыша РАН, 2008 г., №30.

[Klim08c] Ю.А.Климов "SOOL: объектно-ориентированный стековый язык для формального описания и реализации методов специализации программ" // Препринт Института прикладной математики им. М.В. Келдыша РАН, 2008 г., №44.

[MS.NET] Microsoft .NET Framework // <http://www.microsoft.com/net/>.

[Plot83] G.D.Plotkin "An Operational Semantics for CSP" // D.Bjorner (ed.), Formal Description of Programming Concepts II. North-Holland, Amsterdam, 1983, pp.199-223.