ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 7 за 2009 г.

Климов Ю. А.

Специализатор CILPE: анализ времен связывания

Рекомендуемая форма библиографической ссылки: Климов Ю. А. Специализатор CILPE: анализ времен связывания // Препринты ИПМ им. М.В.Келдыша. 2009. № 7. 28 с. URL: http://library.keldysh.ru/preprint.asp?id=2009-7

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

Ю.А. Климов

Специализатор CILPE:
анализ времен связывания

Москва
2009

Ю.А. Климов

Специализатор CILPE: анализ времен связывания

Аннотация

Анализ времен связывания (Binding Time Analysis, BTA, BT-анализ) — основная часть метода частичных вычислений (Partial Evaluation, PE). В данной работе описана BT-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

Оптимизация программ на основе использования априорной информации о значении части переменных называется *специализацией*. Одним из широко используемых методов специализации является метод ч*астичных вычислений* (Partial Evaluation, PE) [Jone93].

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 C#.

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.

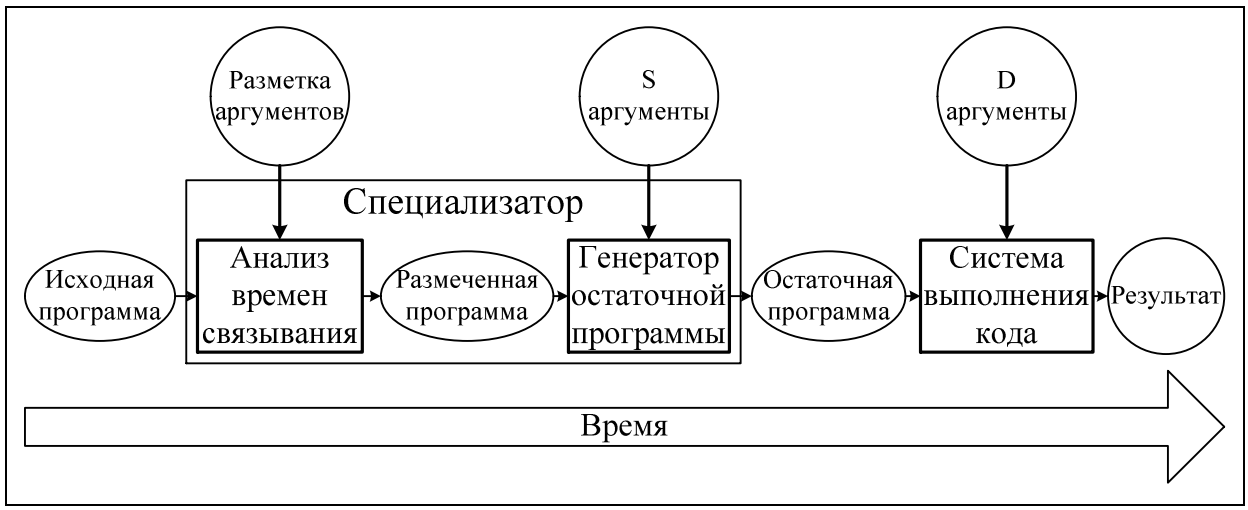

Рис. 1. Общепринятая структура специализатора,
основанного на методе частичных вычислений.

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-*

тичные 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 BT-разметки и правила, которым должна удовлетворять корректно построенная разметка.

Данный анализ времен связывания используется в специализаторе CILPE [Chep03,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, BT-анализ) — построить BT-разметку: приписать каждой инструкции ее BT-вид, а аргументам и результатам методов, элементам стека и переменным — BT-значение.

Для разметки инструкций используются BT-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 BT-значениями с BT-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 BT-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 BT-анализа — построить корректную BT-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 BT-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 BT-анализ [Klim05a,Klim05b]. В этом случае BT-анализ может построить несколько разметок метода в зависимости от разметок аргументов метода.

Также во время построения разметки метода BT-анализ может выполнять эквивалентные преобразования программы (например, разворачивать циклы или дублировать код после условия) для достижения более качественной разметки. Такой BT-анализ называют поливариантным по инструкциям [Klim05a,Klim05b]. В противоположность этому, моновариантный по инструкциям BT-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 BT-анализа, так и для поливариантного BT-анализа.

3. BT-разметка

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

3.1. BT-программа

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

```
BTProgram = [BTClassDef]

BTClassDef = (ClassName, [ClassName], [FieldDec], [BTMethodDef])
BTMethodDef = (MethodName, IsInline, [BTType], [BTType], BTMethodBody)
BTMethodSig = (MethodName, IsInline, [BTType], [BTType])

IsInline = INLINE | NOINLINE
BTType = (Type, BTValue)
```

Рис. 2. BT-программа.

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

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

В BT-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) = Instruction^BTInstructionKind
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, $\text{DuplicateStackTop}^{S}$.

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 $\text{NewObject}^{\text{BTInstructionKind}}\ \text{ClassName}^{\text{BTRefValue}}$.

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 BT-программы требуется, чтобы все методы с именем mthd имели одинаковую BT-сигнатуру: чтобы значения флага и BT-значения соответствующих аргументов и результатов совпадали.

Будем считать, что для каждой размеченной программы определена функция $\text{MethodBTSignature}_{\text{btProg}}$, которая по имени метода возвращает разметку аргументов и результатов (рис. 4).

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

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 BT-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) = $PrimType^{BTKind}$
BTKind = S | D

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

$BTKindOf_{btHeap}$: BTValue→ BTKind
$BTKindOf_{btHeap}$ (primType, btKind) = btKind
$BTKindOf_{btHeap}$ 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).

$$\frac{\begin{array}{c}\forall btRefValue \in Domain(btHead),\quad (\_, \_, btObject) = btHead(btRefValue),\\ \forall fldName \in Domain(btObject),\quad btVal = btObject(fldName) :\\ BTKindOf_{btHeap}(btRefValue) = D \quad \Rightarrow \quad BTKindOf_{btHeap}(btVal) = D\end{array}}{btHeap \text{ — корректна}}$$

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 BT-виды аргументов.

4. Правила BT-разметки

Правила BT-разметки описывают ограничения на разметку до и после выполнения инструкции. Если построена BT-программа, BT-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 BT-разметки похожи на правила, описываемые в стиле операционной семантики [Plot83,Kahn87]. Однако эти правила следует рассматривать, не как правила вывода BT-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 правила как уравнение.

4.1. BT-состояние

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

BTState = (BTStack, BTEnv)

BTStack = [BTValue]
BTEnv = Var → BTValue

Рис. 8. BT-состояние.

4.2. BT-разметка программы

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

1. Правила BT-разметки корректны только для корректной BT-кучи btHeap.
2. Правила BT-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 λ_0 : Integer → BTState, проверяющего BT-разметку этого метода. Причем
   a. BT-аргументы программы $\lambda_0(0)$ должны быть примитивного типа, т.к. аргументы программы и метода Main должны быть примитивного типа.
   b. BT-виды разметки результатов $\lambda_0(2)$ должны быть D.
4. Для каждого определения BT-метода btMthdDef необходимо проверить существование отображения $\lambda_{btMthdDec}$: Integer → BTState, проверяющего BT-разметку этого метода.

1. btHeap — корректна
2. btProg — корректна и типизируема
3. $\exists\lambda_0$: Integer → BTState :
   instrs = [NewObject^D MAIN, CallMethod^X Main, Leave^X],
   btArg = fst $\lambda_0(0)$, btRes = fst $\lambda_0(2)$,
   $\forall$btVal ∈ btArg : (primType, _) = btVal,
   $\forall$btVal ∈ btRes : BTKind(btVal) = D,
   $\forall n\ 0 \le n < 3$: btProg, btHeap, λ_0, btRes $\vdash_{bt}$ (instrs, n)
4. $\forall$btClass ∈ btProg, (_, _, _, btMthdDefs) = btClasss,
   $\forall$btMthdDef ∈ btMthdDefs, $\exists\lambda_{btMthdDec}$: Integer → BTState :
   btProg, btHeap, $\lambda_{btMthdDec}$ $\vdash_{bt}$ btMthdDef

---

btHeap $\vdash_{bt}$ btProg

Рис. 9. Правило BT-разметки программы.

4.3. BT-разметка метода

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

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

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

(_, btTypeArg, btTypeRes, (_, btInstrs)) = btMthdDef
∀btType ∈ btTypeArg :
- либо (btType — ссылочный BT-тип)
  (type, btOref) = btType, (_, types, _) = btHeap(btOref), type ∈ types
- либо (btType — примитивный BT-тип)
  ($type_1$, ($type_2$, btKind)) = btType, $type_1 = type_2$

∀btType ∈ btTypeRes :
- либо (btType — ссылочный BT-тип)
  (type, btOref) = btType, (_, types, _) = btHeap(btOref), type ∈ types
- либо (btType — примитивный BT-тип)
  ($type_1$, ($type_2$, btKind)) = btType, $type_1 = type_2$

fst $\lambda_{btMthdDec}(0)$ = btArg = map snd btTypeArg btRes = map snd btTypeRes
∀n 0≤n<length(instrs) : btProg, btHeap, $\lambda_{btMthdDec}$, btRes $\vdash_{bt}$ (btInstrs, n)

---

btProg, btHeap, $\lambda_{btMthdDec} \vdash_{bt}$ btMthdDef

Рис. 10. Правило BT-разметки метода.

4.4. BT-разметка последовательности инструкций

Корректность разметки n-ой инструкции в списке инструкций instrs проверяется в зависимости от этой инструкции:

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

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

Рис. 11. Правило BT-разметки последовательности инструкций, инструкция $Leave^X$.

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

$\lambda_{btMthdDec}(m)$ (рис. 12).

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

Рис. 12. Правило BT-разметки последовательности инструкций, инструкция $Goto^S$ m.

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

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

Рис. 13. Правило BT-разметки последовательности инструкций, инструкция $Branch^x$ m.

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

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

Рис. 14. Правило BT-разметки последовательности инструкций.

4.5. BT-разметка инструкций

DuplicateStackTop

Инструкция DuplicateStackTop копирует элемент на вершине стека.

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

до и после выполнения инструкции также должны совпадать.

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

Рис. 15. Правило BT-разметки инструкции $DuplicateStackTop^{x}$.

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

RemoveStackTop

Инструкция RemoveStackTop удаляет элемент на вершине стека.

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

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

Рис. 16. Правило BT-разметки инструкции $RemoveStackTop^{x}$.

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

LoadConst const

Инструкция LoadConst const загружает константу const.

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

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

Рис. 17. Правило BT-разметки инструкции $LoadConst^{x}$ const.

- если const — целое число, то BT-значение — это INT^{x};
- если const — число с плавающей точкой, то BT-значение — это

$FLOAT^x$;

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

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

UnaryOp op

Инструкция UnaryOp op выполняет унарную операцию op.

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

- либо оба типа INT^x, если операция op — это NOT или NEG;
- либо оба типа $FLOAT^x$, если операция op — это NEG;
- либо BT-значение до выполнения — INT^x, а после выполнения — $FLOAT^x$, если операция op — это INT2FLOAT;
- либо BT-значение до выполнения — $FLOAT^x$, а после выполнения — INT^x, если операция op — это FLOAT2INT.

$$\frac{\begin{array}{c} x = BTKindOf_{btHeap}(btVal_1) = BTKindOf_{btHeap}(btVal_2) = S \mid D \\ \bullet\ \text{либо } btVal_1 = btVal_2 = INT^x, \text{ если op = NOT или op = NEG} \\ \bullet\ \text{либо } btVal_1 = btVal_2 = FLOAT^x, \text{ если op = NEG} \\ \bullet\ \text{либо } btVal_1 = INT^x,\ btVal_2 = FLOAT^x, \text{ если op = INT2FLOAT} \\ \bullet\ \text{либо } btVal_1 = FLOAT^x,\ btVal_2 = INT^x, \text{ если op = FLOAT2INT} \end{array}}{prog,btHeap \vdash_{bt} UnaryOp^x\ op : (btVal_1::btStack,btEnv) \rightarrow (btVal_2::btStack,btEnv)}$$

Рис. 18. Правило BT-разметки инструкции $UnaryOp^x$ op

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

BinaryOp op

Инструкция BinaryOp op выполняет бинарную операцию op.

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

значения могут быть следующими:

- все три BT-значения — 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 BT-значения — $FLOAT^x$, если операция op — арифметическая операция (ADD, DIV, MUL, REM, SUB);
- два BT-значения на вершине стека до выполнения инструкции — $FLOAT^x$, а BT-значение на вершине стека после выполнения инструкции — INT^x, если операция op — операция сравнения (CEQ, CGT, CLT);
- два BT-значения на вершине стека — это адреса BT-объектов в BT-куче btHeap, а BT-значение на вершине стека после выполнения инструкции — INT^x, если операция op — операция сравнения на равенство (CEQ).

$x = BTKindOf_{btHeap}(btVal_1) = BTKindOf_{btHeap}(btVal_2) = BTKindOf_{btHeap}(btVal_3) = S \mid D$

- либо $btVal_1 = btVal_2 = btVal_3 = INT^x$, если op = ADD, DIV, MUL, REM, SUB, ADD, DIV, MUL, REM, SUB, AND, OR, XOR, SHL, SHR, CEQ, CGT, CLT
- либо $btVal_1 = btVal_2 = btVal_3 = FLOAT^x$, если op = ADD, DIV, MUL, REM, SUB
- либо $btVal_1 = btVal_2 = FLOAT^x$, $btVal_3 = INT^x$, если op = CEQ, CGT, CLT
- либо $btVal_1$: BTRefValue, $btVal_2$: BTRefValue, $btVal_3 = INT^x$, если op = CEQ

$prog, btHeap \vdash_{bt} UnaryOp^x\ op : (btVal_1::btVal_2::btStack, btEnv) \rightarrow (btVal_3::btStack, btEnv)$

Рис. 19. Правило BT-разметки инструкции $BinaryOp^x$ op.

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

LoadVar var

Инструкция LoadVar var читает значение переменной var.

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

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

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

Рис. 20. Правило BT-разметки инструкции $LoadVar^{x}$ var.

В зависимости от разметки BT-значения переменной var S или D, инструкция должна быть размечена либо S, либо X, соответственно.

StoreVar var

Инструкция StoreVar var записывает значение переменной var.

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

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

Рис. 21. Правило BT-разметки инструкции $StoreVar^{x}$ var.

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

NewObject class

Инструкция NewObject class создает новый объект класса class.

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

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

D, соответственно.

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

Рис. 22. Правило BT-разметки инструкции $NewObject^{x}$ class.

LoadField fld

Инструкция LoadField fld читает у объекта поле fld.

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

$$\frac{(\_, \_, btObj) = btHeap(btOref) \quad btVal = btObj(fld) \quad \bullet \text{ либо } x = S, BTKindOf_{btHeap}(btOref) = S, BTKindOf_{btHeap}(btVal) = S \quad \bullet \text{ либо } x = X, BTKindOf_{btHeap}(btOref) = S, BTKindOf_{btHeap}(btVal) = D \quad \bullet \text{ либо } x = D, BTKindOf_{btHeap}(btOref) = D, BTKindOf_{btHeap}(btVal) = D}{prog, btHeap \vdash_{bt} LoadField^{x}\ fld: (btOref::btStack, btEnv) \rightarrow (btVal::btStack, btEnv)}$$

Рис. 23. Правило BT-разметки инструкции $LoadField^{x}$ fld.

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

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

StoreField fld

Инструкция StoreField fld записывает у объекта поле fld.

Правило BT-разметки инструкции LoadField fld требует, чтобы на стеке до выполнения инструкции находилось BT-значение и адрес BT-объекта в BT-куче btHeap, причем BT-значение должно совпадать с BT-значением, со-

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

$$\frac{(\_, \_, btObj) = btHeap(btOref) \quad btVal = btObj(fld)}{prog,btHeap \vdash_{bt} StoreField^{x}\ fld: (btVal::btOref::btStack,btEnv) \rightarrow (btStack,btEnv)}$$

- либо x = S, $BTKindOf_{btHeap}(btOref) = S$, $BTKindOf_{btHeap}(btVal) = S$
- либо x = X, $BTKindOf_{btHeap}(btOref) = S$, $BTKindOf_{btHeap}(btVal) = D$
- либо x = D, $BTKindOf_{btHeap}(btOref) = D$, $BTKindOf_{btHeap}(btVal) = D$

Рис. 24. Правило BT-разметки инструкции $StoreField^{x}$ fld.

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

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

CallMethod mthd

Инструкция CallMethod mthd вызывает у объекта виртуальный метод mthd.

Правило BT-разметки инструкции CallMethod mthd требует, чтобы на вершине стека до выполнения инструкции находились BT-значения, совпадающие с BT-значениями аргументов метода mthd, заданные в сигнатуре этого метода (рис. 24). BT-значения на вершине стека после выполнения инструкции должны совпадать с BT-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.

$$\frac{(btArg, btRes) = MethodBTSignature_{btProg}(mthd)}{prog, btHeap \vdash_{bt} CallMethod^{X}\ mthd : (btArg{+}{+}btStack, btEnv) \rightarrow (btRes{+}{+}btStack, btEnv)}$$

Рис. 24. Правило BT-разметки инструкции $CallMethod^{x}$ mthd.

CastObject type

Инструкция CastObject type приводит объект или массив к типу type.

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

чтобы на вершине стека находился адрес BT-объекта. Причем в списке типов этого BT-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 BT-разметки инструкции CastObject^{x} type.

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

NewArray type

Инструкция NewArray type создает новый массив из элементов типа type.

Правило BT-разметки инструкции NewArray type требует, чтобы на вершине стека до выполнения инструкции находилось BT-значение INT^{x1}, а после выполнения инструкции — ссылочное BT-значение btOref (рис. 26). Остальные элементы стека и окружения до и после выполнения инструкции должны совпадать. Ссылочное BT-значение btOref должно указывать в BT-куче на BT-объект, содержащий особое поле ELEMENT. Также требуется, чтобы список типов BT-объекта содержал тип массива type[], где тип type задается в параметре инструкции.

$$\frac{\begin{array}{l}(\_,\text{types},\text{btObj}) = \text{btHeap}(\text{btOref}) \quad \text{type}[] \in \text{types} \quad \text{btVal} = \text{btObj}(\text{ELEMENT}) \\ \bullet\ \text{либо } x = S,\ x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S,\ \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = S \\ \bullet\ \text{либо } x = X,\ x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = S,\ \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D \\ \bullet\ \text{либо } x = D,\ x_1 = \text{BTKindOf}_{\text{btHeap}}(\text{btOref}) = D,\ \text{BTKindOf}_{\text{btHeap}}(\text{btVal}) = D\end{array}}{\begin{array}{l}\text{prog}, \text{btHeap} \vdash_{\text{bt}} \text{NewArray}^{x}\ \text{type} : (\text{INT}^{x1}::\text{btStack}, \text{btEnv}) \rightarrow \\ \qquad\qquad (\text{btOref}::\text{btStack}, \text{btEnv})\end{array}}$$

Рис. 26. Правило BT-разметки инструкции NewArray^{x} type.

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 разметки BT-значения INT^{x1} и BT-значения 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.

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

Рис. 27. Правило BT-разметки инструкции $LoadLength^{x}$.

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.

$$\frac{(\_, \_, btObj) = btHeap(btOref) \quad btVal = btObj(ELEMENT)}{\begin{array}{l}\bullet\ \text{либо } x = S,\ x_1 = BTKindOf_{btHeap}(btOref) = S,\ BTKindOf_{btHeap}(btVal) = S \\ \bullet\ \text{либо } x = X,\ x_1 = BTKindOf_{btHeap}(btOref) = S,\ BTKindOf_{btHeap}(btVal) = D \\ \bullet\ \text{либо } x = D,\ x_1 = BTKindOf_{btHeap}(btOref) = D,\ BTKindOf_{btHeap}(btVal) = D\end{array}}$$
$$prog, btHeap \vdash_{bt} LoadElement^{x} : (INT^{x1}::btOref::btStack, btEnv) \rightarrow (btVal::btStack, btEnv)$$

Рис. 28. Правило BT-разметки инструкции $LoadElement^{x}$.

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 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.

StoreElement

Инструкция StoreElement записывает элемент массива.

Правило BT-разметки инструкции LoadElement требует, чтобы на вершине стека до выполнения инструкции находились BT-значение btVal, примитивное BT-значение INT^{x1} и ссылочное BT-значение btOref, которое указывает в BT-куче btHeap на BT-объект btObj (рис. 29). Значение этого отображения btObj на особом значении ELEMENT (соответствующего элементам массива) должно совпадать с BT-значению btVal. Остальные элементы стека и окружения до и после выполнения инструкции должны совпадать.

$$\frac{(\_, \_, btObj) = btHeap(btOref) \quad btVal = btObj(ELEMENT)}{\begin{array}{l}\bullet\ \text{либо } x = S,\ x_1 = BTKindOf_{btHeap}(btOref) = S,\ BTKindOf_{btHeap}(btVal) = S \\ \bullet\ \text{либо } x = X,\ x_1 = BTKindOf_{btHeap}(btOref) = S,\ BTKindOf_{btHeap}(btVal) = D \\ \bullet\ \text{либо } x = D,\ x_1 = BTKindOf_{btHeap}(btOref) = D,\ BTKindOf_{btHeap}(btVal) = D\end{array}}$$
$$prog, btHeap \vdash_{bt} StoreElement^{x} : (btVal::INT^{x1}::btOref::btStack, btEnv) \rightarrow (btStack, btEnv)$$

Рис. 29. Правило BT-разметки инструкции $StoreElement^{x}$.

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 находится 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.

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

Рис. 30. Правило BT-разметки инструкции $Lift^{X}$.

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).

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

Рис. 31. Правило выполнения инструкции Lift.

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

$$\boxed{prog \vdash_t Lift : primType::tStack \rightarrow primType::tStack}$$

Рис. 32. Правило типизации инструкции Lift.

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

Приведенные выше правила корректности BT-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 BT-разметка программы и правила, которым должна удовлетворять эта разметка. Поливариантный анализ времен связывания [Klim05a,Klim05b] должен строить BT-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 [Chep03,Klim08a,Klim08b] для языка CIL платформы Microsoft .NET [CLI,MS.NET].

7. Литература

[Chep03] A.M.Chepovsky, 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.

[Jone93] 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.