На правах рукописи

Бреслав Андрей Андреевич

МЕХАНИЗМЫ КОМПОЗИЦИИ В ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКАХ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург — 2010

Работа выполнена в Санкт-Петербургском государственном университете информационных технологий, механики и оптики (СПбГУ ИТМО).

Научный руководитель:	доктор физико-математических наук, профессор Попов Игорь Юрьевич
Официальные оппоненты:	доктор технических наук, профессор Гатчин Юрий Арменакович
	?
Ведущая организация:	?

Защита диссертации состоится “ ” 2010 г. в час. на заседании диссертационного совета Д 212.227.06 при Санкт-Петербургском государственном университете информационных технологий, механики и оптики . по адресу: 197101, Санкт-Петербург, Кронверкский пр., д. 49.

С диссертацией можно ознакомиться в библиотеке Санкт-Петербургского государственного университета информационных технологий, механики и оптики.

Автореферат разослан “ ” 2010 г.

Ученый секретарь диссертационного совета, Тарлыков В. А.
доктор технических наук, профессор

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы. Развитие технологии программирования исторически идет по пути повышения уровня абстракции поддерживаемого инструментами, используемыми при разработке программного обеспечения (ПО). Решающим шагом на этом пути являлось создание языков программирования высокого уровня, позволяющих лишь в небольшой степени заботиться об особенностях конкретной аппаратной архитектуры. Повышение уровня абстракции является ключевым фактором, определяющим затраты на разработку, поскольку высокоуровневые инструменты позволяют избегать определенных типов ошибок, повторно использовать разработанные решения и облегчают командную разработку.

С развитием языков высокого уровня неразрывно связан процесс развития автоматизированных инструментов разработки трансляторов, базирующийся на достижениях теории формальных языков и грамматик, в частности алгоритмах преобразования конекстно-свободных грамматик в магазинные автоматы и формализация семантики с помощью атрибутных грамматик.

Дальнейшее повышение уровня абстракции привело к возникновению идеи *предметно-ориентированных языков* (ПОЯ), предназначенных для решения задач в относительно узкой предметной области и нередко непригодных для решения задач за ее пределами. ПОЯ противопоставляются языкам общего назначения, являющимся вычислительно универсальными и позволяющим решать любые задачи. Основной мотивацией к разработке и использованию ПОЯ является тот факт, что моделирование предметной области в языках общего назначения часто бывает недостаточно явным, что приводит к большим объемам кода и затруднениям при чтении программ. При использовании ПОЯ эта проблема снимается, поскольку такие языки оперируют непосредственно понятиями предметной области и могут даже позволить специалистам в этой области, не имеющих квалификации разработчиков ПО, принимать участие в написании программ. В настоящее время ПОЯ применяются во множестве областей, начиная с систем управления базами данных и заканчивая системами моделирования бизнес-процессов.

Использование ПОЯ снижает затраты на разработку ПО в данной предметной области, но разработка самих ПОЯ также требует затрат. Если эти затраты высоки, то использование ПОЯ может быть нецелесообразным, поэтому возникает задача автоматизации разработки таких языков с целью минимизировать затраты на их создание и поддержку. Традиционные

средства разработки трансляторов не обеспечивают необходимый уровень автоматизации, поэтому разрабатываются новые подходы и инструменты, позволяющие быстро разрабатывать небольшие языки с поддержкой все более сложных механизмов. В частности, существенный интерес представляют механизмы композиции, обеспечивающие повторное использование кода, написанного на ПОЯ.

Предметом исследования являются механизмы композиции, пригодные для использования в предметно-ориентированных языках.

Целью работы является исследование и обоснование подходов и методов, позволяющих автоматически расширять предметно-ориентированные языки механизмами композиции, поддерживающими повторное использование.

Задачи исследования. Достижение поставленной цели подразумевает решение следующих задач:

- Сравнительный анализ механизмов композиции, используемых в современных предметно-ориентированных языках, с целью обоснования требований к средствам автоматизации.
- Проектирование и реализация предметно-ориентированного языка для хорошо изученной области (описание текстового синтаксиса искусственных языков), поддерживающего все основные механизмы композиции в полном объеме, с целью выявления связей между этими механизмами и их характерных особенностей, влияющих на подходы к автоматизации.
- Демонстрация адекватности разработанного языка нуждам конечных пользователей на примере описания синтаксиса сложных языков.
- Обобщение рассмотренных механизмов композиции в виде формализованных языковых конструкций. Описание их семантики и соответствующих систем типов.
- Разработка алгоритмов автоматического расширения языка механизмами композиции, обоснование их корректности.
- Демонстрация работоспособности предложенного подхода.

Методы исследования включают методы инженерии программного обеспечения, анализа алгоритмов и программ, аппарат теории типов, теории графов и теории формальных грамматик.

Научная новизна результатов работы состоит в том, что:

- Спроектирован и реализован предметно-ориентированный язык для описания текстового синтаксиса, поддерживающий композицию спецификаций с помощью модулей, шаблонов (типизированных макроопределений) и аспектов.
- На основе указанного языка разработан генератор трансляторов, поддерживающий проверку типов в семантических действиях и гарантирующий отсутствие ошибок типизации в сгенерированном коде сразу для многих языков реализации.
- Предложена формализация механизмов композиции на основе шаблонов и аспектов, и доказаны свойства данной формализации, гарантирующие раннее обнаружение ошибок программиста при использовании этих механизмов.
- Разработаны и апробированы алгоритмы, позволяющие автоматизировать расширение предметно-ориентированных языков механизмами композиции, основанными на шаблонах и аспектах.

Практическую ценность работы составляют:

- Разработанная библиотека, обеспечивающая трансляцию предложенного языка описания текстового синтаксиса.
- Программные генераторы, использующие данную библиотеку, позволяющие автоматически получать трансляторы и компоненты интегрированной среды разработки.
- Методика и алгоритмы расширения предметно-ориентированных языков механизмами композиции.

На защиту выносятся следующие положения:

- Предметно-ориентированный язык для описания текстового синтаксиса, поддерживающий композицию спецификаций с помощью модулей, шаблонов и аспектов.
- Подход к описанию и генерации трансляторов, позволяющий порождать код на нескольких языках программирования по одной спецификации, и гарантирующий отсутствие ошибок типизации в сгенерированном коде.

- Метод, позволяющий автоматически расширить имеющееся описание синтаксиса и семантики предметно-ориентированного языка таким образом, что результирующий язык поддерживает композицию с помощью шаблонов и аспектов.

Достоверность научных результатов и выводов обеспечивается формальной строгостью описания процесса композиции языков, обоснованностью применения математического аппарата, результатами тестирования алгоритмов и программного обеспечения.

Внедрение результатов работы. Результаты, полученные в ходе диссертационной работы, были использованы в компании OpenWay (Санкт-Петербург) при разработке предметно-ориентированного языка для писания отчетов, а также при выполнении НИОКР "Технология разработки предметно-ориентированных языков"по программе "У.М.Н.И.К." Фонда содействия развитию малых форм предприятий в научно-технической сфере.

Апробация работы. Изложенные в диссертации результаты обсуждались на 11 российских и международных научных конференциях и семинарах и школах, включая V, VI и VII всероссийские межвузовские научные конференции молодых ученых (2008, 2009 и 2010 гг., Санкт-Петербург), международную научную конференцию "Компьютерные науки и информационный технологии" (2009 г., Саратов), международные научные школы "Generative and Transformational Techniques in Software Engineering" (2009 г., Брага, Португалия), "Aspect-Oriented Software Development" (2009 г., Нант, Франция) и "15^{th} Estonian Winter School in Computer Science" (2010 г., Палмсе, Эстония), а так же международные семинары "Teooriapäevad" (2009 и 2010 гг., Эстония), семинар Лаборатории математической логики и семантики языков программирования Научно-исследовательского института кибернетики Эстонской Академии наук (2009 г., Таллинн, Эстония) и научном семинаре "Computer Science Клуба" при ПОМИ РАН (2009 г., Санкт-Петербург).

Публикации. По теме диссертации опубликовано пять печатных работ (из них две статьи — в изданиях, соответствующих требованиям ВАК РФ к кандидатским диссертациям).

Структура и объем работы. Диссертация состоит из введения, четырех глав, списка литературы (? наименований) и 3 приложений. Содержит ? с. текста (из них ? основного текста и ? — приложений), включая ? рис. и табл.

СОДЕРЖАНИЕ РАБОТЫ

Во введении обосновывается актуальность проблемы, формулируются цель и задачи исследования, отмечаются научная новизна и практическая значимость результатов, перечисляются основные положения, выносимые на защиту.

Первая глава посвящена обзору существующих подходов к автоматизации разработки предметно-ориентированных языков. Долгое время в этой области доминировали инструменты, предназначенные для разработки трансляторов текстовых языков, основанные на принципах синтаксически управляемой атрибутной трансляции. Наиболее популярными инструментами этого семейства являются Yacc, Bison и ANTLR. Такие инструменты подразумевают описание синтаксиса языка с помощью контекстно-свободной грамматики, с продукциями которой ассоциированы правила вычисления атрибутов, описывающих семантические свойства предложений языка. С помощью атрибутов могут быть реализованы все стадии семантического анализа от разрешения имен до преобразования в машинный код.

Основным недостатком традиционных средств разработки трансляторов применительно к созданию предметно-ориентированных языков является трудоемкость разработки. Это вызвано в первую очередь тем, что такие инструменты создавались как универсальные средства разработки языков программирования и не предназначены для быстрого решения типичных задач, возникающих при создании небольших языков.

Распространенным способом решения проблемы быстрой разработки ПОЯ является встраивание таких языков в языки общего назначения. При этом используются встроенные в язык общего назначения средства метапрограммирования или гибкие синтаксические конструкции. Чаще всего ПОЯ встраиваются в динамические языки, такие как Ruby или Groovy, и в языки с гибкой системой типов, такие как Haskell или Scala. Основным недостатком данного подхода является привязка к тому или иному языку общего назначения: ПОЯ, легко встраиваемый в один язык, может быть практически невозможно встроить в другой. Это затрудняет повторное использование ПОЯ и приводит к смешиванию уровней абстракции: аспекты реализации сильно влияют на высокоуровневое описание предметной области.

Другое направление в развитии средств разработки языков основано на использовании графического синтаксиса вместо текстового. Толчок к

развитию этого направление дал UML, унифицированный язык моделирования. Чаще всего нотации таких языков используют диаграммы, представляющие собой графы с различными типами вершин и ребер, уложенные на плоскости. Такие языки показали высокую эффективность при решении определенного круга задач, но они не заменяют текстовых языков, в частности, потому, что требуют серьезной инструментальной поддержки, что создает трудности с совместимостью и переносимостью. Следует отметить также, что в области графического синтаксиса существует подход, реализованный в проекте MPS, объединившим сильные стороны текстового и графического синтаксиса. Однако в настоящее время этот подход также требует серьезной инструментальной поддержки, что создает препятствия к его использованию в индустрии.

С развитием идеи предметно-ориентированных языков стали появляться инструменты, ориентированные на быструю разработку автономного транслятора вместе с интегрированной средой разработки. Преимущества этих инструментов состоят в том, что они, во-первых, позволяют создавать простые трансляторы с минимальными затратами времени, а во-вторых, автоматически генерируют расширения для той или иной интегрированной среды, облегчающие разработчику создание и поддержку программ на новом языке. Как правило, такие инструменты ориентированы на решение типичных проблем, и реализация поддержки более сложных механизмов с их помощью затруднительна, однако на практике оказывается, что простота и скорость разработки являются очень весомым фактором, и именно этим инструментам отдают предпочтение разработчики. Возникает необходимость расширять возможности подобных средств, не увеличивая нагрузки на программиста, то есть идет о полной или практически полной автоматизации разработки тех или иных механизмов в языке. Одним из таких механизмов является композиция, обеспечивающая повторное использование спецификаций, основными примерами универсальных механизмов такого рода являются шаблоны (статически проверяемые макроопределения) и аспекты. Существующие подходы к автоматизации разработки таких механизмов не позволяют автоматически получать языки, поддерживающие их в полном объеме.

Вторая глава описывает предметно-ориентированный язык Grammatic, предназначенный для описания текстового синтаксиса, поддерживающий композицию с помощью модулей, шаблонов и аспектов. Этот язык используется как обобщенная , которую можно применять для при разработке различных инструментов, основанных на контекстно-

свободных грамматиках. Реализация данного языка представляет собой библиотеку, позволяющую транслировать текстовые описания грамматик во внутреннее объектно-ориентированное представление, которое в дальнейшем используется различными генераторами для построения тех или иных компонент ПО, таких как синтаксические анализаторы и компоненты интегрированных сред разработки.

Основными конструкциями данного языка являются правила контекстно-свободной грамматики. Каждое правило включает одну или несколько продукций, описывающих один нетерминальный символ. При этом используются операции расширенной нотации BNF (см. Таб. 1). С помощью

Нотация	**Значение**
`#empty`	Пустое выражение
`a`	Ссылка на символ a
`a b`	Последовательность
`a | b`	Альтернатива
`a*`	Итерация от 0 до бесконечности
`a+`	Итерация от 1 до бесконечности
`a?`	Итерация от 0 до 1
`['a'-'z']`	Множество символов от 'a' до 'z'
`'abc'`	Строка символов 'abc'
`(a | b) c`	Круглые скобки для группировки выражений

Рис. 1: Выражения Grammatic

этих операций можно эффективно описать не только синтаксическую, но и лексическую структуру языка, поэтому терминальным алфавитом в спецификациях Grammatic всегда считается Unicode, а лексические правила задаются в виде продукций для соответствующих нетерминальных символов. Пример использования Grammatic для описания простого языка арифметических выражений приведен в Лист. 1.

Большинство инструментов, основанных на контекстно-свободных грамматиках, используют дополнительные средства (*метаданные*) для описания семантики или других свойств языков. В языке Grammatic этот подход обобщается за счет введения произвольных аннотаций, которые могут быть добавлены к любому элементу грамматики: символу, продукции, выражению и т.д. Каждая аннотация состоит из *атрибутов*, записываемых в форме "имя = значение". Предопределенные типы значений приведены в Таб. 2; разработчик может определять собственные типы значе-

```
INT : ['0'--'9']+;
VAR : ['a'--'z''A'--'Z''_']['a'--'z''A'--'Z''_''0'--'9']*;
expr
    : expr '+' expr
    : expr '*' expr
    : '(' expr ')'
    : INT
    : VAR
    ;
```

Листинг 1: Синтаксис языка арифметических выражений

ний. Семантика аннотаций в GRAMMATIC не зафиксирована, что позволяет

Нотация	**Значение**
`'abc'`	Строка
`10`	Целое число
`abc`	Идентификатор
`{ a = b; c = 10}`	Аннотация
`{{ a, b, c ; }}`	Последовательность
`<< s | (a b)* >>`	Грамматическое выражение

Рис. 2: Предопределенные типы значений атрибутов

различным генераторам использовать их для различных целей, таких как описание атрибутной трансляции, классификация лексем для подсветки синтаксиса или спецификация правил автоматического форматирования кода.

Для повторного использования грамматик и их фрагментов применяются механизмы модулей, шаблонов и аспектов. Модули обеспечивают разделение спецификации на несколько физических файлов, имеющих собственные пространства имен символов. Включение одного модуля другим происходит с помощью директивы цитирования `import`, синтаксис которой иллюстрируется следующим примером:

```
import 'a/b/c/d.grammar' {A, B as C};

B : A C*;
```

Директива `import` принимает два аргумента: идентификатор импортируемого файла в одинарных кавычках и список импортируемых символов в фигурных скобках, который может быть заменен символом “*”, если требуется импортировать все символы из данного модуля. Идентификатором

файла является его имя в *виртуальной файловой системе*, конфигурация которой подается на вход транслятору GRAMMATIC вместе с файлом основной грамматики. Ключевое слово `as` в списке импортируемых символов для использования импортированного символа под другим именем.

Шаблоны (типизированные макроопределения) позволяют повторно использовать фрагменты грамматики, внося в них некоторые изменения, посредством подстановки на место параметров шаблона реальных значений. Проиллюстрируем описание шаблона следующим примером:

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

Данный шаблон описывает грамматические выражения, задающие списки элементов, чередующихся с разделителем, например "`INT ('*' INT)*`". Здесь `List` — имя шаблона, после него в угловых скобках следует список параметров с типами, позволяющими контролировать корректность результата разворачивания шаблона. В теле шаблона параметры можно использовать на вместо тех или иных фрагментов спецификации, указывая имена в угловых скобках со знаком вопроса (например, "`<?sep>`"). Шаблон можно использовать, указав в угловых скобках его имя и аргументы, занимающие места формальных параметров, например, выражение `<List INT, '*'>` будет развернуто в `INT ('*' INT)*`. Результатом применения шаблона может быть не только грамматическое выражение, но и любая другая конструкция языка GRAMMATIC, например, одно или несколько правил. Это позволяет реализовать с помощью шаблонов концепцию *параметрических модулей*.

Еще одним механизмом композиции, реализованным в GRAMMATIC являются аспекты. Основополагающие понятия аспектно-ориентированного программирования реализуются в GRAMMATIC следующим образом:

- *Точками встраивания* (join points) являются все элементы языка GRAMMATIC, такие как символы, различные выражения, продукции и аннотации.

- *Срезы* (point-cuts), обеспечивающие квантификацию (quantification), описываются в виде образцов (patterns), использующих как конструкции из Таб. 1, так и переменные, с которыми связываются конкретные фрагменты сопоставляемых выражений. Кроме того, используются подстановочные*подстановочные знаки*, соответствующие произволь-

ным конструкциям одного определенного типа (например, произвольным выражениям или произвольным символам).

- *Советы* (advice), описывающие изменения, привносимые в точках встраивания, позволяют встраивать результаты разворачивания данных шаблонных выражений до, после или вместо фрагментов точек встраивания.

Приведем пример *аспектного правила*, использующего данные понятия:

```
example : .. e=example  .. ;
    instead ?e : '(' <?e> ')' ;
```

Данный пример заменяет рекурсивное вхождение символа `example` в правой части на тот же символ, заключенный в скобки. Другой способ записать то же преобразование использует вставку элементов до и после символа:

```
example : .. e=example  .. ;
    before ?e : '(';
    after  ?e : ')';
```

С помощью аспектов достигается повторное использование и расширение спецификаций, в которых не предусмотрены такие возможности (например, эти спецификации не являются шаблонными). Кроме того, этот механизм позволяет разделять спецификацию на фрагменты с различным назначением, например, отделять метаданные для различных генераторов от грамматических правил.

Целесообразность использования описанных механизмов композиции продемонстрирована на примере описания синтаксиса языка SQL92 и нескольких его диалектов, а также самого языка Grammatic. В обоих случаях удается сократить объем спецификации и ослабить зависимости между модулями по сравнению с другими инструментами.

Третья глава посвящена описанию генератора трансляторов Grammatic^{SDT}, построенного на базе языка Grammatic. Целью создания этого генератора является не только демонстрация возможностей Grammatic, но и решение задачи генерации кода на разных языках программирования по одной спецификации транслятора. Эта задача важна для разработки предметно-ориентированных языков, поскольку эти языки должны быть легко совместимыми с различными языками реализации, чтобы не ограничивать разработчиков в выборе платформы.

Grammatic^{SDT} принимает на вход грамматику, аннотированную семантическими действиями, задающими схему L-атрибутной трансляции, и

генерирует транслятор. Семантические действия могут использовать следующие операции: (а) чтение и запись атрибута, (б) вызов внешней функции, написанной на языке реализации, (в) передача параметров функциям, осуществляющим разбор. Отличительной особенностью GRAMMATIC^{SDT} является наличие статической проверки типов внутри семантических действий, гарантирующей отсутствие ошибок типизации в сгенерированном коде при условии корректной конфигурации генератора. Этот механизм обеспечивает раннее обнаружение ошибок, допущенных разработчиком транслятора, ускоряя таким образом процесс разработки. Для генерации кода, непосредственно осуществляющего синтаксический анализ, могут быть использованы различные существующие генераторы, которые можно подключать к GRAMMATIC^{SDT}. Как правило бывает целесообразным использование различных существующих генераторов для различных языков реализации.

Механизм работы с типами, используемый в GRAMMATIC^{SDT}, имеет два уровня: *абстрактный уровень* выражает свойства общие для всех языков реализации; с помощью типов этого уровня проверяются семантические действия. *Конкретный уровень* описывает процесс генерации из типов и функций абстрактного уровня соответствующих элементов языка реализации. Таким образом, для данной спецификации существует ровно одна система типов абстрактного уровня и, возможно, несколько систем конкретного уровня — как минимум, по одной на каждый язык реализации.

Системы типов абстрактного уровня поддерживают параметрический полиморфизм типов и функций, аналогично JAVA (generics) или HASKELL (parametric polymorphism), причем поддерживается локальный вывод типов. Спецификация системы типов конкретного уровня позволяет отображать параметрически полиморфные функции сообразно специфике языка реализации: если параметрический полиморфизм не поддерживается, его можно заменить *ad hoc* полиморфизмом в той или иной форме, например, сгенерировать множество мономорфных функций — по одной для каждого значения типовой переменной.

GRAMMATIC^{SDT} предоставляет декларативные предметно-ориентированные языки для описания систем типов обоих уровней и программный интерфейс, позволяющий добавлять другие способы описания систем типов. С помощью последнего интерфейса реализована тесная интеграция с системой типов языка JAVA.

Для демонстрации возможностей GRAMMATIC^{SDT} приведены приме-

ры спецификаций, для которых определены системы типов конкретного уровня для двух языков реализации: Java и C. Из таких спецификаций можно генерировать код на соответствующих языках, используя один и тот же файл с описанием транслятора. Транслятор языка Grammatic был реализован с помощью Grammatic^{SDT} (основным языком реализации является Java, а для генерации синтаксического анализатора используется ANTLR).

В четвертой главе описан метод, позволяющий автоматически расширять предметно-ориентированные языки механизмами композиции, основанными на шаблонах и аспектах аналогичных использованным в языке Grammatic.

Механизм шаблонов Grammatic обобщается для произвольного языка следующим образом. Выделяется набор *базовых понятий* (синтаксических категорий или классов в целевой мета-модели), характеризующих такой механизм в любом языке: *шаблон*, *параметр*, *шаблонное выражение*, *применение шаблона*, *использование шаблонного параметра*. Для каждого конкретного языка этот набор дополняется конкретными типами шаблонных выражений, соответствующих конструкциям данного языка. Шаблонные версии конструкций допускают использование шаблонных параметров и применений шаблонов вместо тех или иных своих частей.

Семантика шаблонов определяется алгоритмом разворачивания, который обозначается $\mathcal{I}_\gamma\,[\bullet]$, где *среда* γ сопоставляет шаблонным параметрам фактические значения. Для применений базовых понятий этот алгоритм работает одинаково вне зависимости от конкретного языка, а для специфичных шаблонных выражений создаются экземпляры соответствующих конструкций языка и алгоритм вызывается рекурсивно на подвыражениях. Конечность такой процедуры обеспечивается тем, что шаблонам запрещено вызывать себя рекурсивно, и результирующие объекты, таким образом, всегда конечны. Время работы алгоритма разворачивания линейно от размера входных данных.

Для того, чтобы диагностика ошибок происходила в терминах исходной спецификации, содержащей шаблоны, а не в терминах результата разворачивания, необходимо иметь возможность убедиться в корректности результата, не осуществляя разворачивания. Эта задача решается с помощью системы типов, которая строится следующим образом: правила типизации для базовых понятий фиксированы; наиболее важные правила приведены на Рис. 3. Для специфичных шаблонных выражений генерируются новые правила системы типов, отражающие структурные ограни-

$$\frac{}{\Gamma \cup \{v : \tau\} \vdash <?v> : \tau}\ \text{var}$$

$$\frac{\Gamma \cup \bigcup_{i=1}^{n} \{p_i : \tau_i\} \vdash b : \sigma}{\Gamma \vdash \textbf{template}\,(T<p_1 : \tau_1, \ldots, p_n : \tau_n> : \sigma\,\{b\})}\ \text{abstr}$$

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

Рис. 3: Основные правила системы типов языка шаблонов

чения, накладываемые целевой мета-моделью исходного языка (включая кратность ссылок).

Для построенных таким образом алгоритма $\mathcal{I}_\gamma[\bullet]$ и системы типов доказаны следующие теоремы, показывающие, что система типов гарантирует корректность результата разворачивания.

Теорема 4.1 (О сохранении типов)**.** *Если среда γ согласована с контекстом Γ и $\Gamma \vdash e : \tau$, то $\Gamma \vdash \mathcal{I}_\gamma[e] : \tau$. Другими словами, преобразование $\mathcal{I}[\bullet]$ сохраняет типы.*

Теорема 4.2 (О нормализации)**.** *Если среда γ согласована с контекстом Γ и $\Gamma \vdash e : \tau$, то вычисление $\mathcal{I}_\gamma[e]$ заканчивается за конечное количество шагов, и результат не содержит применений шаблонов и шаблонных параметров.*

Аналогично названному выше строится алгоритм реконструкции типов, позволяющий пользователю не декларировать типы шаблонов и их параметров явно.

Поддержка аспектов строится на тех же принципах, что и поддержка шаблонов, причем многие конструкции, генерируемые для поддержки шаблонов используются также и для аспектов. Основные отличия лежат в области семантики, которая в случае аспектов описывается двумя алгоритмами: сопоставление выражения e исходного языка с образцом (срезом) p, обозначаемое e **match** p, и применение аспектного правила $\mathcal{R}$ в случае успешного сопоставления, обозначаемое $\mathcal{R}@e$.

Операция сопоставления является в некотором смысле обратной к разворачиванию шаблона: вместо того, чтобы заменять вхождения переменных их значениями, она должна найти значения переменных, удовлетворяющие данному образцу. Основной интерес представляет алгоритм сопоставления коллекций (списков и множеств) в случае использования под-

становочных знаков, допускающих сопоставление с частью коллекции. В случае списков для решения этой задачи применяется перебор с возвратами, а в случае множеств задача сводится к поиску максимальных паросочетаний в друдольных графах. В литературе показано, что задача сопоставления образцов является NP-полной при наличии переменных и неупорядоченных коллекций, и используемый алгоритм имеет экспоненциальную оценку времени работы в худшем случае, однако этот алгоритм показывает вполне приемлемые результаты на практике, что было установлено в процессе использования аспектов, реализованных в Grammatic.

Применение аспектного правила сводится к замене объектов, сопоставленных переменным, другими объектами, полученными при разворачивании соответствующих шаблонов. В общем случае смысл имеют только советы с ключевым словом `instead`, то есть заменяющие, а не добавляющие, поскольку понятия вставки до и после могут не иметь смысла в конкретном языке. Тем не менее, если эти понятия имеют смысл, они легко реализуются через замену.

Система типов, используемая для статического контроля корректности результатов применения аспектов базируется на системе типов для шаблонов с той лишь разницей, что в случае аспектов требуется соответствие между типом переменной и шаблонным выражением, которым значения этой переменной будут заменены, что выражается следующим правилом:

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

Здесь $\mathit{Allowed}(\mathcal{R})$ обозначает множество всех корректных пар “переменная-шаблонное выражение”.

В качестве примера применения описанного метода разработано расширение языка описания мета-моделей Emfatic, добавляющее в данный язык поддержку композиции с помощью шаблонов и аспектов.

Основные результаты диссертационной работы

- Разработан предметно-ориентированный язык описания текстового синтаксиса, реализующий механизмы композиции, основанный на использовании модулей, шаблонов и аспектов. Продемонстрирована эффективность механизмов композиции, использованных в данном языке.

- Разработан генератор трансляторов, поддерживающий проверку типов в семантических действиях и гарантирующий отсутствие ошибок

типизации в сгенерированном коде сразу для многих языков реализации.

- Предложен метод автоматизации разработки предметно-ориентированных языков с поддержкой механизмов композиции, основанных на шаблонах и аспектов.
- С использованием предложенного подхода созданы расширения языка Emfatic.

Публикации по теме диссертационной работы

1. *Breslav, A.* Creating textual language dialects using aspect-like techniques / A. Breslav // Pre-proceedings of 3rd Summer School on Generative and Transformational Techniques in Software Engineering (Braga, Portugal). — 2009. — Pp. 63–64.
2. *Breslav, A.* A type checker for multi-targeted parser specifications / A. Breslav // Proceedings of 15th Estonian Winter School in Computer Science (Palmse, Estonia). — 2010. — Pp. 12–13.
3. *Бреслав, А. А.* Создание диалектов языков программирования с использованием грамматических аспектов / А. А. Бреслав // Компьютерные науки и информационные технологии: Материалы Междунар. науч. конф. / Под ред. В. А. Твердохлебов, Ю. И. Митрованов, А. Г. Федорова и др. — Саратов: Издательство Саратовского университета, 2009. — С. 231–232.
4. *Бреслав, А. А.* Средства повторного использования формальных грамматик и их применение для создания диалектов / А. А. Бреслав // *Научно-технический вестник СПбГУ ИТМО.* — Выпуск №61. — 2009. — С. 75–81.
5. *Бреслав, А. А.* Применение принципов MDD и аспектно-ориентированного программирования к разработке ПО, связанного с формальными грамматиками / А. А. Бреслав, И. Ю. Попов // *Научно-технический вестник СПбГУ ИТМО.* — Выпуск №57. — 2008. — С. 87–96.