Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Системного Программирования

Семин Алексей Васильевич

Разработка DSL для технического анализа

Курсовая работа

Научный руководитель:
ст. преп. Кириленко Я. А.

Технический консультант:
Елизаров Р. А.,
координатор проектов,
Devexperts

Санкт-Петербург
2015

Оглавление

Введение

Компьютерные технологии, постепенно проникая в различные сферы жизни людей и общества, не обошли стороной и сферу финансов. В частности, компьютеризация фондовых рынков началась ещё в 1970-х годах [2]. С тех пор ЭВМ стали вытеснять привычные методы ведения торгов и игры на бирже в целом, в основном за счёт скорости и надёжности.

Основным инструментом изучения текущего состояния финансового рынка и прогнозирования его поведения в ближайшем будущем является технический анализ. Технический анализ во многом опирается на методы численного анализа применённые к финансовым данным. В основе этого анализа лежат индикаторы. Экономические индикаторы – это статистические метрики, используемые для измерения подъёма или упадка экономики в целом, либо в локальных её областях. Технические индикаторы, в свою очередь, обозначают любой вид метрики, значение которых тем или иным образов зависит от поведения цен или активов на рынке. Они создаются, чтобы попытаться предсказать будущие уровни цен, либо просто общее направление их движения, опираясь на шаблоны и закономерности выявленные в прошлом.

Естественно, когда речь идёт о большом количестве вычислений и обработке огромного количества данных, человек уже не может соперничать с машиной. Со временем, графики, отражающие показания индикаторов, построенные на миллиметровой бумаге стали все чаще появляться на экранах мониторов. По началу люди, занимающиеся техническим анализом, использовали общедоступные математические пакеты, чтобы производить необходимые вычисления. Однако там, где есть спрос, рождается и предложение: уже долгое время ведётся активная разработка и совершенствование специализированных инструментов анализа фондовых бирж.

Долгое время популярным инструментом для проведения расчётов

были табличные редакторы, такие как Microsoft Excel. Преимущество их было в том, что они позволяли в наглядной форме располагать входные данные, а также производить обработку не только конкретных значений, но сразу наборов таких однообразных данных. Это особенно важно в связи со спецификой представления данных рынка как такового. Весь поток информации приходящий с рынка является дискретным набором значений, каждое из которых привязано к определённому моменту времени. В целом, такое описание полностью подходит под определение временных рядов [4]. Действительно, когда речь идёт о цене на какую-либо акцию, имеется ввиду не только её конкретное текущее значение, но и все предыдущие. Тогда можно обозначить основную цель технического анализа, как выявление трендов или закономерностей по имеющимся последовательностям данных.

Около 10 лет назад начал набирать популярность новый вид инструментов технического анализа, основанный на применении языковых технологий. На смену математическим формулам пришли алгоритмы. Во многом направление развития отрасли в эту сторону обязано не только вычислительной производительности компьютерных систем, но и возможности их мгновенно реагировать на полученные данные посредством обратной связи. Примером применения такого рода технологии являются торговые роботы или торговые стратегии. Получая данные с рынка (во многих случаях, в режиме реального времени), компьютер исполняет заложенный в него алгоритм, формируя ответную реакцию, которая может включать в себя непосредственное взаимодействие с рынком, как, например, покупка или продажа ценных бумаг.

Таким образом сформировались основные задачи, которые обязаны решать языки программирования, используемые в этой области. Кроме двух указанных выше (индикаторы и стратегии), достаточно распространённым является ещё одно предназначение данной технологии - алерты (англ. alerts). Данный вид программного обеспечения используется для оповещения пользователя, когда состояние рынка (цены, активы и т.д.) удовлетворяют заданным условиям.

Стоит отметить, что возможным способом решения перечисленных задач является использование существующих языков программирования общего назначения. Предоставление унифицированного интерфейса доступа к данным и вычислительные возможности выбранного языка могут обеспечить пользователя всем необходимым. Однако, стоит учитывать факторы специфичные для данной области во многом касающиеся самих пользователей.

Во-первых, многие люди, занимающиеся техническим анализом, плохо знакомы с программированием. Поэтому в их интересах, как пользователей, чтобы язык был как можно проще в использовании. В частности, желательно, чтобы описываемые программы были короткими. Такое требование обусловлено тем, что в этой сфере считается нормальным в прямом смысле делиться кодом созданных индикаторов, и в том числе публиковать их на соответствующих интернет ресурсах или в журналах[1].

Во-вторых, язык должен учитывать специфику контекста, в котором исполняются программы. Это само по себе необходимо для реализации первого требования, однако включает в себя более обширный спектр возможностей по расширению. Например, использование особых языковых конструкций для быстрого доступа к специфичным для области данным. Такие данные включают в себя, например, показатели open, close, low и high для конкретного символа, т.е. для конкретных акций. В большинстве языков для технического анализа эти показатели доступны из любой части программы по умолчанию [1].

[1] `https://www.tradingview.com/script/`

1. Обзор предметной области

Существует немало программного обеспечения для технического анализа [3]. Каждый продукт в большинстве случаев привязан к конкретной биржевой платформе, для которой он разрабатывается. Также разработчики стараются создать как можно больше индикаторов, чтобы у пользователя сразу были в распоряжении все стандартные средства технического анализа. Некоторые из этих программных продуктов предоставляют возможность настраивать существующие индикаторы, снабжая их изменяемыми параметрами. Другие позволяют задавать эти параметры или определённые части индикаторов с помощью математических выражений. Продолжая цепочку расширений, наиболее масштабные проекты дают пользователям практически полноценный язык программирования для создания собственных алгоритмов вычислений индикаторов на основе существующих или полностью новых.

1.1. Иструменты для технического анализа

Рассмотрим один из таких языков - MQL4. Это язык программирования разработанный специально для создания торговых роботов, технических индикаторов и библиотек функций для платформы MetaTraider4. Он имеет C-подобный синтаксис и наиболее часто используется именно для проектирования торговых стратегий, получающих данные в реальном времени. Средний размер кода для описания индикаторов на этом языке составляет не менее ста строк[2]. К тому объём кода увеличивается из-за необходимости явно описывать как все известные показатели рынка (такие как open и close), так и типы всех переменных.

Другой такой инструмент разработанный компанией Wealth Lab - WealthScript. WealthScript - это набор библиотек для платформы .NET. С одной стороны это удобное тем, что в распоряжении пользователя находятся все средства платформы. С другой, программы на языке

[2] `http://www.mql5.com/en/code/mt4/indicators`

программирования общего назначения используемые для прикладных задач не отличаются краткостью. Также стандартные средства компиляции не имеют дополнительной информации определяемой спецификой отрасли, чтобы предупредить пользователя об распространённых ошибках ещё до непосредственного запуска программы.

1.2. DxEL

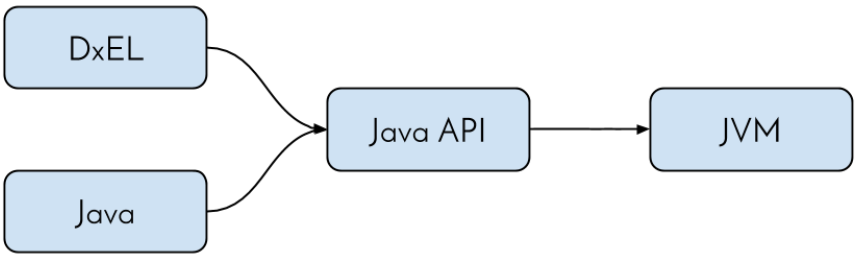
DxEL[3] - инструмент для технического анализа, разрабатываемый компанией Devexperts. Этот программный комплекс нацелен в основном на создание индикаторов, но также будет удобен для описания собственных алертов и даже торговых стратегий. Он предоставляет возможность пользователю, в зависимости от его предпочтений, использовать два разных способа определения собственных технических индикаторов. Первый - с помощью языка программирования Java и высокоуровнего API, второй - посредством специального языка, имя которого носит проект, - DxEL.

Подробнее рассмотрим, как устроена схема работы данного продукта 1. Как уже было упомянуто, у пользователя будет два основных пути написания программы для технического анализа или стратегию для торгового робота. В случае с Java используется специальный API, который позволяет получать доступ ко всем необходимым данным, а также набор аннотаций, упрощающий для пользователя процесс кодирования. Как уже указывалось ранее, применения языка общего назначения привносит как все его преимущества, так и недостатки.

Однако более интересен второй путь, а именно, использование языка DxEL. Каким в таком случае должен быть язык, чтобы избежать большинства описанных проблем? Чтобы ответить на этот вопрос, необходимо более формально подойти к требованиям со стороны пользователя к такого рода продукту.

[3]Devexperts Expression Language

Рис. 1: Схема использования DxEL



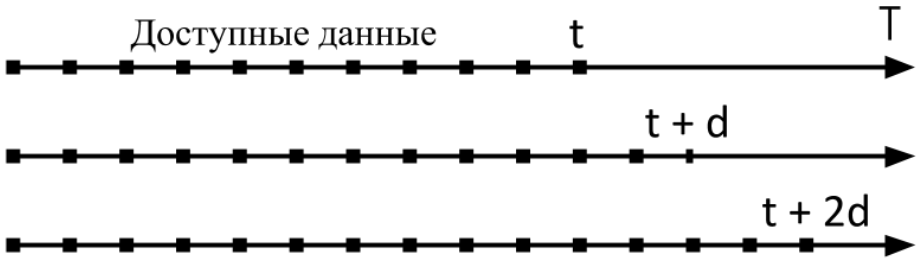
Жизненный цикл программы DxEL

Способ исполнения программ описанных с помощью продукта DxEL (как на Java, так и на самом языке) несколько отличается от традиционного. После того, как программа подготовлена к запуску (например, скомпилирована), наиболее распространённый способ её использования - запуск по инициативе пользователя, в течение которого программа исполняется один раз. В некоторых случаях прикладные программы выполняются автоматически, например, по событию в системе или по таймеру. Программы на DxEL же, перед запуском получают входные параметры, после чего в среде её исполнения формируется необходимый контекст, и далее происходит множественное последовательное её исполнение.

Как уже упоминалось ранее, все данные на рынке представлены последовательностями, где каждый элемент последовательности обозначает величину в определённый дискретный момент времени (Рис. 2). Перед каждым запуском программы для неё устанавливается точка отсчёта в единицах времени. Потенциально, она имеет доступ как к текущим данным (точке отсчёта), так и ко всей информации ”из прошлого”.

Итеративность запуска обусловлена тем, что данная точка отсчёта постоянно сдвигается по этой временной шкале, предоставляя программе доступ к новым данным.

Рис. 2: Последовательные запуски программы



2. Постановка задачи

Целью данной курсовой работы является задание синтаксиса и семантики языка DxEL, а также реализация программного обеспечения для подготовки исходного кода программ на данном языке к трансляции в Java посредством Java API.

По многим причинам описанным ранее, синтаксис языка DxEL должен быть как можно выразительнее. Под этим понимается возможность описания всех необходимых математических и алгоритмических средств небольшим (по отношению к языкам общего назначения) объёмом исходного кода. Во многих современных языках программирования с пользователя снимается обязанность описания типов используемых объектов. Это достигается с помощью использования механизмов вывода и проверки типов и приближает к достижению желаемой краткости программ.

Другой фактор противопоставляющийся простоте кода - его надёжность. Для того, чтобы уберечь пользователя от возможных ошибок, их нужно научиться определять и сообщать о них как можно раньше. Соотношение между количеством запусков программы, включая ещё большее количество итераций на каждый запуск, и временем (вычислительной сложностью) затрачиваемом на подготовку программы к исполнению – очень велико. В связи с этим, наилучшим решением будет сообщать пользователю об ошибках прежде, чем произойдёт запуск программы, то есть на стадии трансляции. Также, в большинстве случаев дополнительные проверки и оптимизации можно совершить, когда

становятся известны входные параметры программы, тем не менее ещё не приступая к её исполнению.

Основная часть данных, с которыми работает программа, представлена в виде массивов, где каждый элемент массива – это значение с меткой на шкале времени. Таким образом, об этих данных можно думать, как о последовательности значений некой величины, упорядоченных по времени. Для повышения производительности и оптимизации работы с памятью во время исполнения программы было бы полезно иметь информацию относительно того, какой объем данных запрашивается для каждой отдельной такой последовательности в коде программы. Безусловно, не всегда такую проверку можно выполнить, не имея конкретных параметров. В этом смысле можно пойти на некоторые ограничения пользователя со стороны языка и предупреждать его, если невозможно определить количество данных требуемых для исполнения программы или определить всегда ли одна итерация программы завершит свою работу.

Такие меры во многом оправдывают себя, высоко поднимая планку надёжности запускаемых программ. Во всех остальных случаях, когда пользователю не будет хватать возможностей непосредственно DxEL, всегда можно использовать его свободный от ограничений, но менее безопасный в этом отношении, способ описания программ с помощью Java.

Таким образом можно выявить ряд задач, необходимых для достижения цели и учитывающих описанные особенности предметной области.

- Задать синтаксис языка в формальной нотации.
- Определить и реализовать промежуточное представление структур языка.
- Реализовать синтаксический анализатор, обеспечивающий перевод исходного кода в промежуточное представление.
- Реализовать анализ, проверяющий корректность объявления, ини-

циализации и использования переменных.

- Реализовать вывод и проверку типов.
- Реализовать вычисление глубины запрашиваемых программной данных до ее запуска.

Рис. 3: Общая структура скрипта DxEL.

<тип скрипта> <имя скрипта>;
<объявления пользовательских типов>
<объявление входных параметров>
<остальной код>

3. Реализация

3.1. Синтаксический анализ

Для реализации синтаксического анализа был выбран ANTLR[4]. Это пакет для генерации парсеров с открытым исходным кодом. Для генерации синтаксического анализатора ему требуется формальное описание синтаксиса языка в форме грамматики. Также, он предоставляет средства интеграции с Java. Это позволяет, во многом, контролировать ошибки допускаемые пользователем при наборе исходного кода, облекая их описание в удобную для чтения человеком форму.

3.1.1. Структура программ на языке DxEL

Исходный код программ хранится в текстовых файлах с расширением ".dxel". Каждый файл может содержать несколько программ или скриптов. Общая структура отдельного скрипта выглядит как показано на схеме 3.

Для каждого типа скрипта в языке существует отдельное ключевое слово. От типа указанного в заголовке зависти семантика программы.

study – тип скрипта для создания индикаторов. Один такой скрипт может независимо производить вычисление нескольких индикаторов, т.е. результатом его работы является несколько значений. Это используется, когда их вычисление тесно связано между со-

[4] `http://www.antlr.org/`

бой, либо для совместного отображения их на графике для визуального сравнения.

function используется для создания пользовательских библиотек функций. Семантически ничем не отличается от **study**, за один исключением: результат работы у функции должен быть единственным.

alert используется для создания алертов, т.е. создан для того, чтобы оповещать пользователя о том, что текущее данные (состояние рынка) удовлетворяет заданному условию. Поэтому результат работы таких скриптов - единственное значение, которое имеет логический тип.

strategy задают торговые стратегии. Также как и **study** не имеют дополнительных ограничений. Более того, только в таком типе скриптов пользователю доступен интерфейс обратной связи с рынком. Например, команды покупки или продажи активов или ценных бумаг.

В языке существует несколько способов объявлений пользовательских типов и данных.

enum в текущей спецификации, единственный тип, который может объявить пользователь. Его семантика полностью соответствует перечислимым типам в языках общего назначения.

in – ключевое слово используемое для объявления входных параметров. Входные параметры могут экспортироваться в приложение с графическим интерфейсом пользователя, чтобы задать конфигурацию скрипта перед запуском.

def декларирует внутренние для скрипта промежуточные значения.

out используется для описание результата работы скрипта. Как уже было отмечено ранее в **study** и **strategy** таких объявлений может быть несколько, и каждое обязано иметь уникальное имя. В случае с **function** и **alert** имя опускается, обозначая уникальность

результата. Также может использоваться в конце блоков кода, для возвращения результатов локальных вычислений.

Краткий список конструкций доступных в языке: математические и логические операторы, if, switch, fold.

Примечание. ***fold*** *– единственный в языке способ описать цикл (итерированные вычисления) внутри программы. Также в целях возможности проверки, что программа всегда завершится, запрещена само-рекурсия и взаимная рекурсия..*

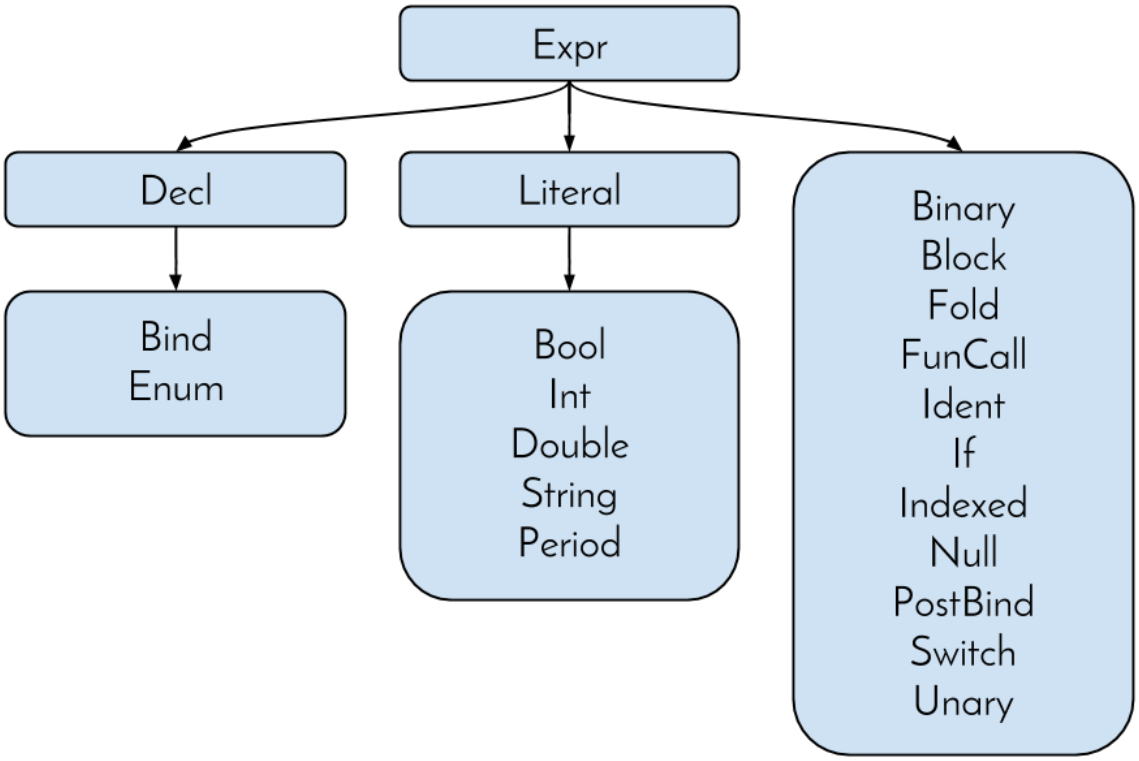
В языке присутствуют все распространённые примитивные типы и, соответственно, литералы: bool, int, double, string. Помимо этого, часто используемыми данными в предметной области являются периоды времени. В связи с этим было предложено добавить в язык новый тип литералов period для удобного их описания: 1000s (1000 секунд), 7d (7 дней), 3y (3 года).

3.1.2. Промежуточное представление

Промежуточное представление строится с помощью обхода абстрактного синтаксического дерева, которое генерируется парсером. Во многом оно напрямую соответствует синтаксическим конструкциям языка и имеет древовидную структуру. Также во время построения промежуточного представления раскрывается весь синтаксический сахар.

Каждый скрипт содержит набор объявленных пользователей структур Enum, набор входных параметров, и ровно один блок, который в свою очередь представляет собой последовательность выражений. Выражения могут быть вложены друг в друга, поэтому каждое выражение также представляется деревом. Элементы являющиеся узлами дерева промежуточного представления показаны на рисунке 4.

Рис. 4: Элементы промежуточного представление



3.2. Корректность объявления и инициализации переменных

Реализованный алгоритм анализа проверяет промежуточное представление и выявляет типичный набор ошибок: дублированное объявление переменной, использование необъявленных переменных и другие.

Особое требование к семантике языка затрагивает инициализацию: *каждая объявленная переменная должна быть явно инициализирована в любой ветке потока управления, при том только один раз.*

В приведённом примере 1 переменная **x**, объявленная в первой строке скрипта инициализируется корректно. Соответственно, в зависимости от условий в операторе ветвления, по окончании исполнения этого оператора переменная может иметь различные значения, но заведомо известно, что во всех случаях это значение будет задано пользователем.

Listing 1: Пример отложенной инициализации.

```
study Study2;
def x;
if (false) {
    x <- 10;
    def x;
    x <- 20;
} else {
    if (true) {
        x <- 30;
    } else {
        x <- 40;
    }
}
out z = 2 * x;
```

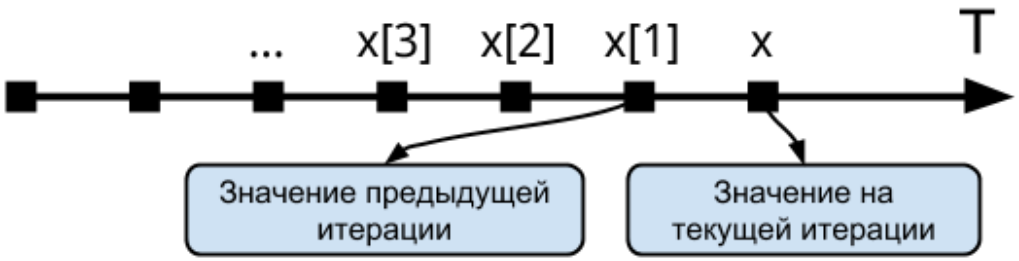
Семантика языка такова, что нет необходимости строить граф потока управления отдельно, чтобы провести эту проверку. Его обход можно произвести неявно, используя для этого имеющееся промежуточное представление. Это возможно, благодаря отсутствию в языке переходов по метке и механизма исключений.

3.3. Вывод типов

В данной предметной области библиотечные функции, предоставляемые разработчиками языков, имеют множество параметров, большинство из которых редко меняются пользователем. Для того, чтобы позволить выборочно изменять параметры вызова таких функций, в языке DxEL каждый входной параметр имеет значение по умолчанию. Так как это значение может зависеть только от ранее определённых (литералов языка или значений доступных вне скрипта), для каждого входного параметра можно сразу вывести его тип.

Далее, после семантического анализа, если не было обнаружено ошибок, известно, что использование переменной встречается строго после её инициализации. Следовательно строго до её использования имеется выражение, тип которого соответствует типу данной переменной. Так-

Рис. 5: Последовательные запуски программы



же, отсутствие само-рекурсии и взаимной рекурсии обеспечивает знание типов всех функций, вызываемых внутри скрипта, поскольку они были описаны ранее. Литералы языка имеют постоянный тип, который не зависит ни от чего.

Только перечисленные выше конструкции и никакие другие не могут являться листьями дерева внутреннего представления. Таким образом можно заключить, что вывести типы всех выражений скрипта можно, поднимаясь от листьев к корню дерева. То есть вывод и проверку типов за один проход по дереву можно реализовать с помощью метода рекурсивного спуска, что и было выполнено в ходе данной работы.

3.4. Вычисление глубины запрашиваемых данных

В языке DxEL определена специальная конструкция для обращения к значениям переменных, вычисленных на предыдущих итерациях исполнения скрипта 5:

<идентификатор>[<индекс>]

Задача состоит в том, чтобы для каждого идентификатора в программе (до её запуска) найти максимальное значение индекса, т.е. *глубину*, либо вывести ошибку о том, что это невозможно.

Синтаксически и семантически в качестве индекса можно использовать не все языковые конструкции. Тем не менее, описанные выше проверки не могут гарантировать выполнение необходимых условий.

Таким образом необходимо явно задать выражения, которые разрешено использовать (3.4).

Из всех выражений в скрипте, можно выделить те, которые имеют целочисленный тип и вычислимые до запуска. Безусловно такие выражения могут зависеть только от целочисленных литералов и других выражений, удовлетворяющих таким свойствам. Для краткости, в рамках данной проверки, обозначим такие выражения *корректными*.

Точно также, как в семантическом анализе можно провести обход неявного графа потока управления и произвести абстрактную интерпретацию всех таких выражений.

Стоит отметить важную особенность. Не смотря на то, что все переменные инициализируются ровно один раз, они могут принимают множество значений в зависимости от условий известных только во время выполнения, то есть после запуска. Тогда комбинирование переменных с другими такими переменными в более сложных конструкциях может привести к экспоненциальному росту количества различных значений этих конструкций. Однако задача не состоит в том, чтобы найти их все, но только в поиске максимума. Оказывается, что для её решения достаточно знать лишь минимальное и максимальной значение всех подвыражений, а не все возможные. В таблице 3.4 приведены способы вычисления новых границ выражений, по известным границам подвыражений.

Таким образом, с помощью метода рекурсивного спуска, используя не более двух чисел на каждый узел дерева, можно вычислить все искомые глубины для данной последовательности и выбрать из них наибольшую.

Выражение	Условие корректности	Вычисление границ
Literal Int		min=значение литерала max=значение литерала
Ident (a)	Значение идентификатор корректно	min=min(a) max=max(a)
Unary (-a)	Подвыражение корректно	max=min(a) min=max(a)
Binary (a+b)	Подвыражения корректны	max=max(a)+max(b) min=min(a)+min(b)
Binary (a-b)	Подвыражения корректны	max=max(a)-min(b) min=min(a)-max(b)
Binary (a%b)	Подвыражения корректны	max=min{max(a), min(b)-1} min=0
Binary (a*b)	Подвыражения корректны	maxmin=maxmin{max(a)*max(b), max(a)*min(b), min(a)*max(b), min(a)*min(b)}
If | Switch	Каждая ветка - корректное выражение	max=max{max(каждой ветки)} min=min{min(каждой ветки)}

Таблица 1: Выражения, используемые в качестве индекса

Заключение

В результате проделанной работы были выполнены все поставленные задачи.

- Задан синтаксис языка с помощью формальной грамматики.
- Определено и закодировано промежуточное представление структур языка.
- С помощью генератора парсеров был реализован синтаксический анализ и перевод исходного кода в промежуточное представление.
- Реализованы проверки корректности объявления, инициализации и использования переменных в программе.
- Реализован вывод и проверка типов.
- Разработан и реализован эффективный алгоритм вычисления глубины запрашиваемых программной данных до непосредственного её запуска.

Список литературы

[1] TradeStation Securities Inc. EasyLanguage Essentials. — TradeStation Securities, Inc., 2007. — URL: http://goo.gl/FSzFk2.

[2] Wikipedia. Algorithmic trading // Wikipedia, The Free Encyclopedia. — 2015. — URL: https://en.wikipedia.org/wiki/Algorithmic_trading (online; accessed: 28.05.2015).

[3] Wikipedia. Technical analysis software // Wikipedia, The Free Encyclopedia. — 2015. — URL: https://en.wikipedia.org/wiki/Technical_analysis_software (online; accessed: 28.05.2015).

[4] Wikipedia. Time series // Wikipedia, The Free Encyclopedia. — 2015. — URL: https://en.wikipedia.org/wiki/Time_series (online; accessed: 28.05.2015).