Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»**

Институт естественных наук и математики

Кафедра вычислительной математики и компьютерных наук

РАЗРАБОТКА ПРОТОТИПА СРЕДЫ ДЛЯ НАПИСАНИЯ И ВИЗУАЛЬНОЙ ОТЛАДКИ ПРОГРАММ НА ЯЗЫКЕ PROLOG

Допустить к защите:

Заведующий кафедрой, Профессор
Доктор физико-математических наук
ПИМЕНОВ ВЛАДИМИР
ГЕРМАНОВИЧ

Магистерская диссертация
студента VI курса
Смажилюка Игоря Павловича

Научный руководитель:
кандидат технических наук
старший преподаватель
каф. вычислительной математик и
компьютерных наук
АВЕРБУХ ВЛАДИМИР
ЛАЗАРЕВИЧ

Екатеринбург
2017 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день существует большое количество разнообразных языков программирования. Все языки можно разделить на две большие группы: императивные языки и декларативные языки. Для императивного подхода характерны следующие особенности:

- В исходном коде программы записываются инструкции (команды)
- Инструкции должны выполняться последовательно
- При выполнении инструкции данные, полученные при выполнении предыдущей инструкции, могут читаться из памяти
- Данные, полученные при выполнении инструкции, могут записываться в память

В целом императивный подход базируется на переменных, операторе присваивания и на последовательности команд для компьютера. Очевидно, первым императивным языком были машинные инструкции, которые содержали в себе готовые к исполнению команды процессора и не нуждались в каких либо преобразованиях. В дальнейшем были созданы более удобные языки программирования, которые позволили упростить процесс разработки и сократить время написания программ. К императивным языкам относятся следующие современные языки программирования: Ассемблер, Fortran, C#, Java и другие.

Декларативный подход к программированию сильно отличается от императивного программирования. При декларативном подходе мы не описываем четкую последовательность команд процессору, мы лишь описываем способ решение задачи, а компилятор или интерпретатор уже преобразовывает это в набор инструкций. Примером декларативных языков является язык SQL, в котором мы описываем, что мы хотим сделать с данными. Другим хорошим примером является язык HTML, в котором мы

описываем, как должна выглядеть страница в интернете. В обоих случаях мы не пишем точную последовательность команд для процессора.

Декларативные языки принято делить на три основных группы:

- Чистые декларативные языки (HTML)
- Функциональные языки (Haskell, Lisp)
- Логические (Prolog)

Стоит отметить, что деление на императивные и декларативные языки имеет условный характер, так на одном и том же языке программирования можно писать в разных стилях. Например, вы, вполне, можете написать программу в функциональном стиле на языке C#. Для этого вам нужно использовать только вызовы методов и не использовать переменные и оператор присваивания.

Императивные языки гораздо чаще используются в коммерческой разработке и для этого есть свои причины. Во-первых, на языках этой группы проще описывать популярные алгоритмы, такие как алгоритмы на графах и алгоритмы сортировки. Во-вторых, для этих языков разработаны отличные средства отладки. Любые современные среды разработки для императивных языков содержат мощный набор для отладки программ.

Несмотря на свои преимущества, функциональные и логические языки не часто используются в промышленной разработке программного обеспечения. На наш взгляд это связанно, отчасти, с отсутствием хороших сред разработки, которые поддерживают подсветку синтаксиса, механизм автодополнения и удобные средства визуальной отладки.

В данной работе описывается прототип среды разработки для языка Prolog. Которая поддерживает основные аспекты современных средств разработки, таких как: текстовый редактор кода, компилятор или интерпретатор и визуальный отладчик. На наш взгляд, описанный прототип,

позволит решить проблему, упомянутую выше, и увеличит лояльность программистов к языку Prolog.

1 ОСОБЕННОСТИ ЯЗЫКА PROLOG КАК ЛОГИЧЕСКОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Prolog – язык и система логического программирования, основанные на языке предикатов математической логики дизъюнктов Хорна, представляющей собой подмножество логики предикатов первого порядка. Язык сосредоточен вокруг небольшого набора основных механизмов, включая сопоставление с образцом, древовидное представление структур данных и автоматического перебора с возвратом. Prolog, благодаря своим особенностям, используется в области искусственного интеллекта и компьютерной лингвистики.

1. Основные синтаксические конструкции

Основными понятиями в языке являются факты, правила логического вывода и запросы, позволяющие описывать базы знаний, процедуры логического вывода и принятия решений. Рассмотрим эти конструкции языка подробнее.

Термы

Программа на языке Prolog описывает отношения, определяемые с помощью предложений. Как и в любом другом языке, ориентированном на символьные вычисления, предложения выстраиваются из термов, которые в свою очередь подразделяются на атомы, числа, переменные и структуры. Атом записывается со строчной буквы или заключается в кавычки, когда требуется запись с прописной буквы.

```
atom
'Atom'
```

Переменные, записывающиеся с прописной буквы, отличаются от переменных в процедурных языках программирования, они не связаны с конкретной ячейкой памяти, а скорее ближе к математической переменной.

```
X is 2 + 2.
```

Структуры представляют собой совокупности термов, заключенные в круглые скобки, в том числе и другие структуры. Структура обозначается именем (функтором), которое располагается перед круглыми скобками.

```
book( 'Название','2009','Спб', authors( 'Первый автор', 'Второй автор'
)).
```

Ещё одной конструкцией являются списки, элементы которых заключаются в квадратные скобки. В основе списков в языке Prolog лежат связные списки.

```
List = [ a, b, [ c, d ], e ].
```

Правила

Правила в Прологе записываются в форме правил логического вывода с логическими заключениями и списком логических условий. В чистом Прологе предложения ограничиваются дизъюнктами Хорна:

```
Вывод :- Условие.
```

и читаются так: «Заголовок ИСТИНА, если тело ИСТИНА». Тело правила содержит ссылки на предикаты, которые называются целями правила.

Встроенные предикаты

`,/2` - оператор с двумя аргументами. Определяет конъюнкцию целей.

`;/2` - оператор с двумя аргументами. Определяет дизъюнкцию целей.

Факты

Факты в языке Prolog описываются логическими предикатами с конкретными значениями. Факты в базах знаний на языке Prolog представляют конкретные сведения (знания). Обобщённые сведения и знания в языке Prolog задаются правилами логического вывода (определениями) и наборами таких правил вывода (определений) над конкретными фактами и обобщёнными сведениями. Предложения с пустым телом называются фактами. Пример факта:

```
Кот( Иван ).
```

Этот факт эквивалентен правилу:

```
Кот( Иван ) :- ИСТИНА.
```

2. Подходы к отладке программ на языке Prolog

На сегодняшний день существует большой список различных сред разработки на языке Prolog. Стоит отметить, что большинство из этих сред сильно отличаются друг от друга. Различая начинаются в платформах, для которых написана среда и заканчиваются стандартом языка Prolog, который используется. Многие среды разработки поддерживают только свой стандарт языка, который может значительно отличаться от стандарта ISO-Prolog. Кроме того, разные среды используют разные инструменты для отладки программ.

Условно среды разработки на языке Prolog можно разделить на две группы по тому, какие средства отладки они используют. В первую группу стоит отнести программы, которые не используют визуальные средства отладки. Такие среды разработки предоставляют интерфейс терминала, в котором можно задавать вопросы к текущей базе знаний. Популярным представителем этой группы является GNU Prolog. Ко второй группе относятся программы, которые предоставляют визуальные инструменты отладки. Наиболее мощная среда разработки из этой группы это Visual Prolog. Рассмотрим упомянутые программы подробнее.

GNU Prolog

GNU Prolog – компилятор языка программирования Prolog, с встроенным интерактивным отладчиком. Доступен для Unix, Windows и Mac OS X. Поддерживает расширения языка: программирование в ограничениях над конечными множествами, работу с файлами при помощи грамматик, построенных на определённых предложениях (DC-грамматик). Также предоставляет интерфейс к операционной системе. Внешний вид программы выглядит следующим образом:

Как видно интерфейс данной среды разработки достаточно прост. Он позволяет добавлять новые правила в базу знаний как непосредственно из главного окна, так и загружать их из файла. Кроме того можно выполнять запросы в обычном режиме и в режиме отладки. В режиме отладки можно пошагово выполнять программу и следить за промежуточными результатами.

На наш взгляд, отсутствие удобного окна редактирования кода является минусом этой системы. Кроме того трассировка вызовов, хоть и является распространённым подходом, при отладке программ на логических языках,

не дает достаточной информативности. К плюсам этой среды относится то, что она доступна под все современные операционные системы, а так же поддерживает стандарт ISO-Prolog.

Visual Prolog

Visual Prolog – это система состоящая из объектно-ориентированного расширение языка Prolog и системы визуального программирования. Язык программирования, реализованный в Visual Prolog отличается от классического пролога тем, что он основан на строгой статической типизации. В него также добавлены средства объектно-ориентированного программирования, анонимные предикаты (лямбда-предикаты), императивные конструкции (foreach, if...then...else) и пр.

Среда разработки приложений системы Visual Prolog включает текстовый редактор, различные редакторы ресурсов, средства разработки справочных систем в гипертекстовом представлении, систему отслеживания изменений, которая обеспечивает перекомпиляцию и перегенерацию только измененных ресурсов и модулей, ряд экспертов Кода, оптимизирующий компилятор, набор средств просмотра различных типов информации о проекте и отладчик. Полная интеграция всех средств обеспечивает повышение скорости разработки приложений. Полученные приложения являются исполняемыми .EXE программами. Внешне система выглядит следующим образом:

Как видно Visual Prolog это мощная среда разработки с удобным редактором кода и визуальными средствами отладки. Способ отладки здесь позаимствован из императивных языков и содержит стандартные для этого подхода инструменты: точки остановки, шаг далее, шаг с заходом и шаг с выходом. Если учитывать, что Visual Prolog основывается на нескольких парадигмах программирования, то такой подход к отладке кажется вполне обоснованным.

Не смотря на то, что Visual Prolog обладает богатым набором инструментов, эта среда не лишена недостатков. Во-первых, в среде используется своя спецификация языка пролог, что сильно затрудняет переносимость кода. Во-вторых, данная система реализована только для семейства операционных систем Windows. В-третьих, из-за расширения языка подходами из императивного программирования, среда слабо подходит для обучения первоначальным навыкам программирования на языке Prolog.

2 ОПИСАНИЕ ПРОТОТИПА

Так как задача этой работы исследовать подходы к отладке программ на языке Prolog, было принято решение реализовать прототип среды разработки. При разработке прототипа преследовались следующие цели:

1) Среда должна быть легкой в использовании.
2) Инструменты, для написания прототипа, должны иметь широкое распространение.
3) Среда должна иметь удобный редактор кода на языке Prolog.
4) Должна быть возможность выполнять запросы.
5) Должна быть возможность исследовать подходы к отладке программ с использованием прототипа.

2.1 Инструменты разработки

Перед созданием прототипа рассматривались разные возможные варианты реализации. Сначала была идея реализовать прототип как обычное оконное приложение Windows или веб приложение. Преимущество этих вариантов, очевидно, мы пишем приложение с нуля и можем реализовать абсолютно любой функционал, нас не стесняют ограничения сторонних программ и компонент. С другой стороны, так как мы не используем внешние компоненты, нам бы пришлось реализовывать очень много вспомогательных модулей с нуля. Например, окно редактора кода.

Исходя из выше сказанного, было принято решение, реализовывать прототип как расширение для современной и хорошо распространённой среди разработки программного обеспечения. А качестве таковых рассматривались Notepad++ и Visual Studio 2015. У программы Notepad++ есть некоторые ограничения на создание окон с инструментами, поэтому

выбор пал в сторону Visual Studio. Тем более что эта среда разработки очень распространена на нашем факультете.

Visual Studio – это целый набор средств и инструментов для разработки программного обеспечения от компании Microsoft. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. В целом это очень мощный и удобный инструмент разработки. Кроме того Visual Studio содержит в себе большой набор средств для написания расширений. Например, целый ряд шаблонов элементов становится доступно разработчику после установки Visual Studio Sdk, среди которых:

- Custom Tool Window. Позволяет создавать окна инструментов не связанные с файлами исходного кода. Примерами таких окон могут служить Обозреватель решения, Окно свойств и так далее.
- Custom command. Позволяет добавлять произвольные кнопки с действиями в вертикальные и горизонтальные панель инструментов, которых находятся в стандартных окнах.
- Editor Classifier. Позволяет расширить большинство функций редактора кода Visual Studio или написать свой редактор для нового языка программирования.

Для написания программ в среде Visual Studio наиболее часто используется язык C#. Это очень мощный и постоянно развивающийся императивный язык программирования. Таким образом, прототип среды отладки для языка Prolog был написан в среде Visual Studio 2015 на языке C#.

2.2 Реализация

Как уже отмечалось ранее, для того чтобы начать создавать расширение для Visual Studio нужно установить пакет Visual Studio Sdk. После этого вы будете иметь доступ ко всем инструментам для разработки расширений. Для создания расширения нам потребуется шаблон проекта VSIX Project, который находит в Templates -> Visual C# -> Extensibility -> VSIX Project. Этот шаблон проекта позволяет создавать новые типы шаблонов или проектов или оборачивать уже реализованные расширения в один пакет. Дальнейшая разработка была разбита на три этапа: создание шаблона проекта для языка Prolog, создания редактора кода и создания окна инструментов для выполнения программы. Давайте рассмотрим каждый из этих шагов подробнее.

2.2.1 Шаблон проекта

Visual Studio дает возможность легко создавать новые типы проектов на базе уже имеющего шаблона. При таком подходе файл проекта будет содержать много ненужной информации и настроек, которые не имеют никакого отношения к языку Prolog (версия платформы .Net, формат бинарного файла результата и другие). Чтобы не тащить эти настройки, было принято создать новый тип проекта с расширением .plproj. К сожалению, Visual Studio 2015 не имеет шаблона проекта для создания нового типа проекта. Такой шаблон появился в Visual Studio 2017 и называется Project Type. По этой причине, пришлось вручную создавать классы, которые описывают новый тип проекта. Основной класс, описывающий тип проекта, является наследником класса Microsoft.VisualStudio.Shell.Package и выглядит он следующим образом:

```
[PackageRegistration(UseManagedResourcesOnly = true)]
[InstalledProductRegistration("#110", "#112", "1.0")]
[ProvideProjectFactory(typeof(PrologProjectFactory), "Prolog",
"Prolog Project Files (*.plproj);*.plproj", "plproj", "plproj",
@"ProjectTemplate\Template", LanguageVsTemplate = "PrologProject")]
[ProvideMenuResource("Menus.ctmenu", 1)]
[ProvideToolWindow(typeof(ToolWindow))]
[Guid(PackageGuids.GuidPkgString)]
[SuppressMessage("StyleCop.CSharp.DocumentationRules",
                 "SA1650:ElementDocumentationMustBeSpelledCorrectly",
                 Justification = "pkgdef, VS and vsixmanifest are valid VS terms")]
public sealed class PrologPackage : Package
{
    protected override void Initialize()
    {
        base.Initialize();
        RegisterProjectFactory(new PrologProjectFactory(this));
        ToolWindowCommand.Initialize(this);
    }
}
```

На данном этапе стоит обратить внимание на атрибут ProvideProjectFactory. С помощью этого атрибута описываются параметры нового типа проекта такие как: фабрика для создания проекта, расширение файла проекта, расположение шаблона проекта в дереве окна New Project.

Следующий класс – это класс наследник от Microsoft.VisualStudio.Project.ProjectFactory. Как не трудно догадаться из названия в нем описывается фабрика, которая используется при создании нового проекта.

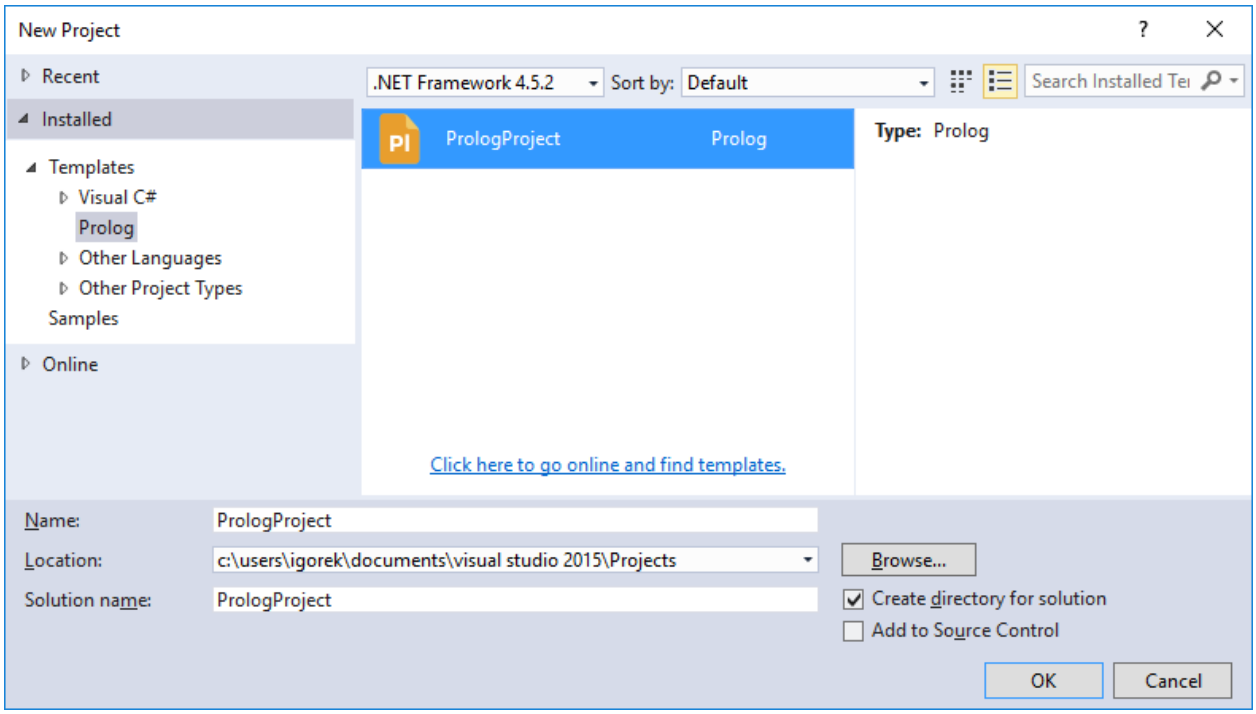
```
[Guid(PackageGuids.GuidFactoryString)]
public class PrologProjectFactory : ProjectFactory
{
    private readonly PrologPackage _package;

    public PrologProjectFactory(PrologPackage package) : base(package)
    {
        _package = package;
    }

    protected override ProjectNode CreateProject()
    {
        var project = new PrologProjectNode();

        project.SetSite((IOleServiceProvider)((IServiceProvider)_package)
```

```
                .GetService(typeof(IOleServiceProvider)));
            return project;
        }
    }
```

Как видно из примера кода, для создания фабрики проекта нужно лишь перегрузить метод CreateProject, который возвращает объект класса наследника от Microsoft.VisualStudio.Project.ProjectNode. Класс наследник ProjectNode отвечает за копирование файлов проекта при создании проекта, за переименования файла проекта (используя имя, которое ввел пользователь при создании проекта), за иконку проекта, которая будет отображаться в окне обозревателя решения и за некоторые другие аспекты.

Стоит отметить, что создание нового типа проекта достаточно кропотливая задача. Вероятность совершить ошибку очень велика, так как к проблемам может привести опечатка в одном параметре атрибута ProvideProjectFactory. Разобраться же в причинах проблемы достаточно сложно, так как среда Visual Studio либо просто не показывает новый тип проекта в окне New Project, либо просто завершает свою работу с критической ошибкой. Ситуация усугубляется тем, что информации в сети интернет по созданию нового типа проекта весьма мало.

В целом, после ряда неудачных попыток автору удалось создать новый тип проекта с расширением .plproj, который отображается в окне New Project в узле Templates -> Prolog -> PrologProject. Внешне это выглядит следующим образом:



При создании нового проекта PrologProject, в окно обозревателя решения добавляется новое решение, которое содержит проект для языка Prolog. В этом проекте находится файл Program.pl с кодом на языке Prolog. И выглядит это следующим образом:

2.2.2 Редактор кода

Как было отмечено ранее удобный редактор кода упрощает написания программ. Как правило, современные редакторы поддерживают два основных инструмента: подсветка синтаксиса и механизм автодополнения. На данном этапе исследования, в прототипе среды разработки, было решено реализовать подсветку синтаксиса.

Для реализации редактора был использован шаблон элемента для создания расширения Editor Classifier, который был описан выше. Этот шаблон расположен в Visual C# Items -> Extensibility -> Editor Classifier. При

использовании этого шаблона в проект добавляются четыре класса, которые описывают работу механизма подсветки синтаксиса. Рассмотрим эти классы подробнее.

```
[Export(typeof(IClassifierProvider))]
[ContentType("Prolog")]
internal class PrologEditorClassifierProvider : IClassifierProvider
{
    [Export]
    [Name("Prolog")]
    [BaseDefinition("code")]
    internal static ContentTypeDefinition PrologContentType = null;

    [Export]
    [FileExtension(".pl")]
    [ContentType("Prolog")]
    internal static FileExtensionToContentTypeDefinition PrologPlFileType = null;

    [Import]
    private IClassificationTypeRegistryService classificationRegistry;

    public IClassifier GetClassifier(ITextBuffer buffer)
    {
        return buffer.Properties
            .GetOrCreateSingletonProperty<PrologEditorClassifier>
            (creator: () => new PrologEditorClassifier(this.classificationRegistry));
    }
}
```

В классе PrologEditorClassifier описывается к файлу, с каким расширением может быть применен данный классификатор текста. Кроме того указывается какой классификатор будет использоваться.

```
public class PrologEditorClassifier : IClassifier
{
    private readonly IClassificationTypeRegistryService _registry;
    private readonly IClassificationType classificationType;

    internal PrologEditorClassifier(IClassificationTypeRegistryService registry)
    {
        _registry = registry;
        this.classificationType = registry
                                  .GetClassificationType("PrologEditorClassifier");
    }

    public event EventHandler<ClassificationChangedEventArgs> ClassificationChanged;

    public IList<ClassificationSpan> GetClassificationSpans(SnapshotSpan span)
    {
        return TokenParser.Parse(span, _registry);
    }

}
```

Класс PrologEditorClassifier имеет единственный метод, который должен преобразовать входную строчку кода в массив объектов, которые описывают стиль для отображения отдельных кусочков строки. Как видно из кода данная ответственность полностью делегирована классу TokenParser. Класс TokenParser содержит достаточно большое количество строк кода и не приводится в данной работе. Однако общая идея разбора строчки кода на языке Prolog состоит в следующем. Мы разбиваем строчку на токены учитывая комментарии, оператор вывода ":-", ключевое слово is и так далее.

Для каждого токена, если он относится к какой либо группе, мы возвращаем индекс начала токена, его длину и группу, к которой он относится. Если токен не относится ни к какой группе, то его можно просто пропустить. В конечном итоге мы получаем список объектов ClassificationSpan, которые описывают, какой кусок текста к какой группе относится. Описание групп будет приведено ниже.

Стоит отметить, что при разборе текста не использовалась синтаксическая грамматика распознающая язык Prolog. Так как написание такой грамматики выходит за рамки данного исследования. В целом, структура кода на языке Prolog достаточно простая и анализатор разбивающий текст на токены был реализован с использованием стандартных методов работы со строками и регулярных выражений.

```
internal static class PrologEditorClassifierClassificationDefinition
{
    [Export(typeof(ClassificationTypeDefinition))]
    [Name("PrologEditorClassifier")]
    private static ClassificationTypeDefinition typeDefinition;

    [Export(typeof(ClassificationTypeDefinition))]
    [Name(PrologTokens.PrologTokenNames.Comment)]
    internal static ClassificationTypeDefinition comment = null;

    [Export(typeof(ClassificationTypeDefinition))]
    [Name(PrologTokens.PrologTokenNames.RuleName)]
    internal static ClassificationTypeDefinition ruleName = null;

    [Export(typeof(ClassificationTypeDefinition))]
    [Name(PrologTokens.PrologTokenNames.Clause)]
    internal static ClassificationTypeDefinition clause = null;
```

```
        [Export(typeof(ClassificationTypeDefinition))]
        [Name(PrologTokens.PrologTokenNames.Keyword)]
        internal static ClassificationTypeDefinition keyword = null;

        [Export(typeof(ClassificationTypeDefinition))]
        [Name(PrologTokens.PrologTokenNames.Text)]
        internal static ClassificationTypeDefinition text = null;
    }
```

Класс PrologEditorClassifierClassificationDefinition нужен исключительно, чтобы объявить набор групп, которые упоминались выше. Сами же группы описываются в следующем классе:

```
    internal sealed class PrologEditorClassifierFormat : ClassificationFormatDefinition
    {
        public PrologEditorClassifierFormat()
        {
            this.DisplayName = "PrologEditorClassifier";
            this.BackgroundColor = Colors.BlueViolet;
            this.TextDecorations = System.Windows.TextDecorations.Underline;
        }

        [Export(typeof(EditorFormatDefinition))]
        [ClassificationType(ClassificationTypeNames =
PrologTokens.PrologTokenNames.Comment)]
        [Name("CommentFormat")]
        [UserVisible(false)]
        [Order(Before = Priority.Default)]
        internal sealed class CommentFormat : ClassificationFormatDefinition
        {
            public CommentFormat()
            {
                this.DisplayName = "This is a comment";
                this.ForegroundColor = Colors.Green;
            }
        }

...
```

Класс PrologEditorClassifierFormat представлен не полностью. Однако по этой части можно понять, как описываются группы. То есть, для каждой группы мы создаем класс наследник от ClassificationFormatDefinition в конструкторе, которого описываем все свойства (цвет фона, цвет и стиль текста и так далее) для визуального отображения куска текста.

В целом были выделены следующие группы: комментарии, ключевые слова, символ вывода, название предиката в заголовке правила вывода. Для каждой группы был выбран свой цвет текста. В результате программа для

нахождения наибольшего общего делителя на языке Prolog выглядит следующим образом:

2.2.3 Окно выполнения

Заключительным шагом в создании прототипа была разработка окна выполнения программы. Естественно, для решения этой задачи требуется интерпретатор языка Prolog. Как правило, такие интерпретаторы преобразовывают программу в промежуточный язык WAM (Абстрактная машина Уорена). Трансляция программы выходит за рамки этого исследования, поэтому было принято решение использовать стороннюю библиотеку для выполнения программы. В сети интернет было обнаружено две подходящих библиотеки с открытым исходным кодом это - XProlog и Prolog.Net. Первая библиотека отличается простотой реализации, однако у автора возникли вопросы о полноценности и актуальности этой библиотеки. Последний коммит в репозиторий был сделан больше четырех лет назад. Более того, не удалось выполнить программу на языке Prolog с использованием данной библиотеки.

Проект Prolog.Net, напротив, находится на стадии активного развития. Эта библиотека содержит интерпретатор и компилятор, которые позволяют выполнять и отлаживать программы с использованием стандартных методов: шаг с заходом и шаг с выходом. Так же в проекте есть достаточный набор примеров использования классов и методов. Кроме того, библиотека доступна для скачивания через систему управления пакетов Nuget. Исходя из выше сказанного, было решено использовать в реализации прототипа среды эту библиотеку.

При наличии библиотеки для выполнения кода, для завершения третьей части работы, нужно было реализовать окно инструментов, которое предоставляло бы интерфейс для выполнения запросов. Чтобы создать окно инструментов был использован шаблон элемента Custom Tool Window, который был описан выше. Этот шаблон расположен в Visual C# Items -> Extensibility -> Custom Tool Window. При использовании этого шаблона в проект добавляются три класса и файл PrologPackage.vsct.

Первый класс ToolWindow наследник ToolWindowPane, в котором описаны заголовок окна инструментов и класс его порождающий. Второй класс ToolWindowCommand нужен для обработки события создания окна. В файле PrologPackage.vsct описывается название и расположение кнопки создания окна инструментов в пункте меню View –> Other Windows. Последний и наиболее важный класс – это ToolWindowControl наследник UserControl. Этот класс совместно с файлом разметки ToolWindowControl.xaml описывает внешний вид окна инструментов и его поведение.

В целом можно заметить, что, не считая нескольких вспомогательных классов, новое окно инструментов создается с использованием популярной технологии xaml. Внешний вид окна описан в файле ToolWindowControl.xaml, а обработчики событий окна в классе ToolWindowControl.

Xaml разметка, определяющая интерфейс окна инструментов, достаточно проста и не стоит упоминания в тексте этой работы. Однако в данном месте стоит показать пример использования библиотеки Prolog.Net. Обращение к этой библиотеке происходит в методе ExecutePrologProgram класса ToolWindowControl и выглядит следующим образом:

```
private void ExecutePrologProgram(string programStr)
{
    var lines = programStr.Split(Environment.NewLine.ToCharArray());
    var codeSentences = new List<CodeSentence>();
    foreach (var line in lines)
    {
        codeSentences.AddRange(Prolog.Parser.Parse(line));
    }
    Prolog.Program program = new Prolog.Program();
    foreach (var codeSentence in codeSentences)
    {
        program.Add(codeSentence);
    }
    var qw = Prolog.Parser.Parse($":-{QueryTextBox.Text}");
    Query query = new Query(qw[0]);
    PrologMachine machine = PrologMachine.Create(program, query);
    StringBuilder sb = new StringBuilder();
    sb.AppendLine(machine.RunToSuccess().ToString());
    if (machine.QueryResults != null)
    {
        foreach (var v in machine.QueryResults.Variables)
        {
            sb.AppendLine($"{v.Name} = {v.Text}");
        }
    }

    ResultTextBox.Text = sb.ToString();
}
```

Как видно из кода, строки программы на языке Prolog преобразовываются с помощью метода Prolog.Parser.Parse и собираются в список объектов CodeSentence. Далее с использованием этого же метода создается запрос. Текст запроса берется из текстового поля формы QueryTextBox. Далее создается Prolog машина, которой передается программа и запрос. Результат выполнения запроса сохраняется в текстовое поле ResultTextBox.

В целом, если правильно реализовать все описанные выше классы, то в меню View –> Other Windows появиться новый пункт. В данном случае он называется Prolog Executer. Выглядит это следующим образом:

Если кликнуть на новый пункт меню, то откроется окно выполнения запросов на языке Prolog. Стоит отметить, что это окно никак не связанно с окном редактора кода. В качестве рабочей директории окна инструментов берется текущий plproj проект. Внешне окно имеет следующий вид:

Как видим, окно имеет поле для ввода запроса, кнопку выполнить и поле для вывода результата. В качестве примера, приведено выполнение предиката gcd(6, 3, A). Видно, что Prolog машина вернула следующий результат: запрос является истинным для A равного трем.

3 РАЗВИТИЕ ПРОТОТИПА

Безусловно, одним из важных инструментов полноценной среды разработки является отладчик кода. Следующим шагом развития прототипа среды разработки будет реализация одной или нескольких концепции отладки. На наш взгляд к логическим языкам программирования применимы разные подходы к отладке. Стоит выделить три основных из них.

Первый подход широко распространён в императивных языках программирования, и связан он с концепцией точек остановки, шагом с заходом и шагом с выходом. При этом можно настроить параметры точки остановки. Окно настройки точки остановки может выглядеть следующим образом:

Конечно, переменные в логических языках программирования имеют смысл, отличающийся от переменных в императивных языках. Однако, такой подход может быть применен, о чем мы наглядно убедились на примере среды Visual Prolog.

Второй подход связан с построением деревьев поиска. В качестве примера здесь стоит рассмотреть другой декларативный язык SQL. При выполнении запроса некоторые среды позволяют построить план выполнения запроса.

Используя граф выполнения, программист может вносит коррективы в Sql запрос. Нам кажется, что такой подход применим к языку Prolog так же как и к языку SQL. Так как между ними прослеживается некоторая связь: оба языка декларативных, и там и там есть запросы к базе данных/знаний.

Третий подход применим к любым языкам программирования, и заключается в статическом анализе кода. Современные статические анализаторы позволяют выявить самые разные ошибки. Самые простые из

них это неиспользуемые переменные, недостижимые участки кода, опечатки и ошибки copy-paste. Несомненно, подомные инструменты могут быть реализованы и для логических языков, в том числе для языка Prolog.

ЗАКЛЮЧЕНИЕ

В результате исследовательской работы достигнуты следующие результаты:

- Изучена парадигма логического программирования на примере языка Prolog.
- Изучены популярные среды для разработки программ на языке Prolog, среди которых GNU-Prolog и Visual Prolog.
- Изучены базовые инструменты создания расширения для интегрированной среды разработки Visual Studio 2015
- Реализован прототип среды разработки для написания и выполнения программ на языке Prolog. Прототип включается в себя новый тип проекта для среды Visual Studio 2015. Редактор кода на языке Prolog. Окно инструментов для выполнения программ. Исходный код прототипа на сайт GitHub.
- Предложены три концепции для отладки программ на логических языках программирования, которые в дальнейшем будут внедрены в прототип среды отладки.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Адаменко Анатолий, Кучуков Андрей, Логическое программирование и Visual Prolog. БХВ-Петербург, 2003. 992с.

Братко Иван, Язык PROLOG: алгоритмы искусственного интеллекта. Вильямс, 2004. 640с.

Джеси Рассел, Отладка программ, VSD. 2013, 103с.

Роберт У. Себеста, Основные концепции языков программирования. Вильямс, 2001. 672 с.

Описание расширений для Visual Studio. Режим доступа: http://www.visualstudioextensibility.com, свободный.

Документация по созданий расширений для Visual Studio. Режим доступа: https://msdn.microsoft.com/en-us/library/dn919654.aspx, свободный.

Среда разработки GNU Prolog. Режим доступа: http://www.gprolog.org, свободный.

Среда разработки Visual Prolog. Режим доступа: http://www.visual-prolog.com, свободный.

Интерпретатор и компилятор программ на языке Prolog. Режим доступа: https://prolog.codeplex.com, свободный.