Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б09-мм

Ghidra: сворачивание исходного кода

Клопов Григорий Сергеевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преподаватель кафедры ИАС Смирнов К. К.

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. Реализация в других IDE 5

2.2. Ghidra и решение сообщества 6

2.3. Реализация в фреймворке, используемом Ghidra 7

2.4. Выводы 8

3. Реализация **9**

3.1. Исправление решения сообщества 9

3.2. Доработка решения 10

Заключение **13**

Список литературы **14**

Введение

Реверс-инжиниринг — это процесс анализа и изучения программного обеспечения с целью понять его структуру, функциональность и принципы работы. Основной целью реверс-инжиниринга является восстановление исходного кода, схемы работы или логики продукта, чтобы можно было воспроизвести, улучшить, или адаптировать его. Этот метод находит широкое применение в самых разных областях. Например, в сфере кибербезопасности он позволяет специалистам анализировать вредоносное программное обеспечение, обнаруживать уязвимости и разрабатывать методы защиты. В научно-исследовательских лабораториях реверс-инжиниринг используется для изучения устаревших или незадокументированных систем, что позволяет сохранить работоспособность оборудования, потерявшего техническую поддержку.

Ghidra — это инструмент для реверс-инжиниринга. Ghidra поддерживает декомпиляцию машинного кода для различных процессорных архитектур, таких как x86, ARM, RISC-V, что значительно расширяет сценарии её применение. Ключевая функция Ghidra — декомпиляция машинного кода в высокоуровневый язык программирования C, что упрощает анализ сложных систем.

Однако инструменту не хватает ряда функций, влияющих на удобство использования. Например, отсутствует возможность сворачивания кода, что создаёт трудности при работе с длинными функциями с множеством вложенных операторов, затрудняя определение текущей области выполнения. Добавление данной функции достаточно востребовано, что подтверждается трекером ошибок [8]. Также присутствует временное решение [9], которое необходимо проанализировать и доработать.

Цель данной работы — разработать и внедрить функцию сворачивания кода в Ghidra. Это позволит улучшить удобство работы с инструментом, особенно при анализе сложных программных систем.

1. Постановка задачи

Целью данной работы является добавление функции сворачивания кода в Ghidra. Для её выполнения были поставлены следующие задачи:

1. Проанализировать текущую архитектуру Ghidra и временное решение;
2. Реализовать функцию сворачивания кода;
3. Внедрить новую функцию в Ghidra.

2. Обзор

В данной главе будет представлен обзор существующих технологий, которые позволяют реализовать задачу сворачивания кода. Для этого были выбраны следующие источники:

1. Реализация в других IDE;
2. Реализация в фреймворке, используемом Ghidra;
3. Ghidra и решение сообщества

2.1. Реализация в других IDE

Для анализа подхода в IDE была выбрана популярная Eclipse[7]. Она, как и Ghidra, написана на языке программирования Java и обладает открытым исходным кодом, что делает её подход к реализации функций сворачивания потенциально применимым в Ghidra.

В Eclipse функциональность сворачивания кода реализована с использованием фреймворка JFace Text [1], который позволяет отделять отображаемый текст от полного содержимого файла. Центральной частью архитектуры (схематично изображена на рис. 1) является модель взаимодействия между полным текстом документа **Master Document** и его проекционным представлением **Projection Document**, предназначенным для отображения только несвёрнутых участков текста (рис. 1).

В реализации проекционной модели участвуют 3 основных класса: **ProjectionDocument**, **ProjectionDocumentEvent**, **ProjectionDocumentManager**[2]. **ProjectionDocument**[4] представляет проекцию основного документа, где отображаются только выбранные фрагменты текста, **ProjectionDocumentEvent**[5] вызывается при изменении в содержании основного документа или в проекции, а **ProjectionDocumentManager**[6] управляет созданием и обновлением проекционных документов. Он обеспечивает связь

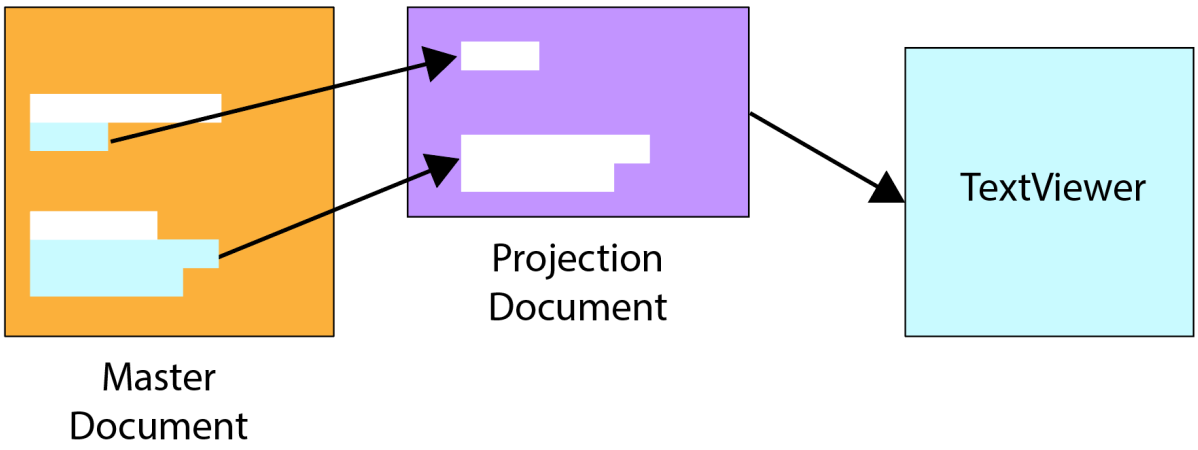

Рис. 1: Схема взаимодействия компонентов

между основным текстом и проекцией, задаёт правила отображения и обработку изменений в режиме реального времени.

Также стоит упомянуть о **ProjectionAnnotation**[3]. Этот класс используется для представления сворачиваемых или развёрнутых областей текста в виде аннотаций. Каждая хранит информацию о границах соответствующей области текста и её текущем состоянии — скрыта она или отображается. Кроме того, аннотация используется для отображения графических индикаторов в редакторе.

Архитектуру сворачивания кода, реализованную в Eclipse, можно адаптировать для использования в Ghidra.

2.2. Ghidra и решение сообщества

В рамках Ghidra нас интересует модификация окна декомпилятора и связанной с ним инфраструктуры. Рассмотрим основные классы, с которыми будем работать:

1. **DecompilerPanel**: Отвечает за отображение окна декомпиляции, обработку пользовательского ввода и взаимодействие с декомпилированным кодом, позволяя пользователю видеть, выделять и работать с текстом декомпиляции.

2. **LineNumberDecompilerMarginProvider**: Отвечает за отобра-

жение номеров строк в боковой панели окна декомпиляции.

3. **ClangToken**: Представляет лексические элементы декомпилированного кода, такие как ключевые слова и идентификаторы.

4. **ClangLayoutController**: Управляет размещением и визуальным оформлением токенов на экране, отвечает за корректное отображение текста и выделение элементов.

5. **DecompilerUtils**: Содержит утилитарные функции для работы с декомпилятором, такие, как обработка токенов, преобразование текста и вспомогательные операции для взаимодействия с декомпилированным кодом.

Сообщество предложило решение[9], позволяющее реализовать сворачивание кода с помощью клавиатурных команд. Основная идея заключается в следующем: классу **ClangToken** добавляется свойство **collapsedToken**. Если это свойство имеет значение true, блок кода свёрнут, иначе он развёрнут. Для определения части кода, которую необходимо свернуть, используются встроенные функции Ghidra из класса **DecompilerUtils**. Они позволяют обнаружить ближайшие открывающую и закрывающую скобки. При нажатии соответствующей клавиши блок кода с **collapsedToken**, установленным в false, скрывается из отображения.

Однако после проверки выяснилось, что предложенное решение не работает в текущем виде и требует дальнейшей доработки.

2.3. Реализация в фреймворке, используемом Ghidra

Ghidra использует для отображения интерфейса фреймворк Swing. Хотя в рамках Swing существуют технологии, которые могли бы быть полезны для реализации сворачивания кода, такие как класс **JTree**, позволяющий отображать структуры в виде иерархии с возможностью разворачивания и сворачивания, их использование в данном случае проблематично. Интеграция подобных крупных структур в уже существующую архитектуру Ghidra достаточно сложна, поэтому для реализации

сворачивания кода лучше использовать более простые и адаптивные подходы.

2.4. Выводы

Предложенное сообществом решение требует доработки и тестирования для полноценного функционирования, а также дальнейших улучшений.

3. Реализация

3.1. Исправление решения сообщества

После выполнения всех указаний от сообщества изменений в собранной версии Ghidra не произошло. С помощью отладочных сообщений удалось определить источник проблемы: при нажатии клавиш "-" или "=" не срабатывал обработчик события **keyPressed**, из-за чего не запускалась дальнейшая цепочка действий, описанная в обзоре. Попытки смены отслеживаемых клавиш не привели к положительному результату. В процессе анализа кода Ghidra было установлено, что для корректной работы необходимо зарегистрировать интерфейс в конструкторе класса **DecompilerPanel** с использованием следующей команды:

```
fieldPanel.addFieldInputListener(this);
```

После внесения этого изменения функциональность была восстановлена. Теперь при нахождении курсора возле фигурных скобок ({ или }) нажатие клавиши **-** приводит к сворачиванию блока кода, начинающегося с данной скобки, а нажатие клавиши = — к его разворачиванию.

Однако такая форма взаимодействия недостаточно наглядна и удобна:

1. Пользователь не имеет представления о том, какие блоки кода могут быть свёрнуты;
2. При разворачивании блока автоматически разворачиваются все вложенные свёрнутые блоки;
3. Для сворачивания блока требуется точно навести курсор на фигурную скобку.

Для повышения удобства работы интерфейс требует доработки с учётом современных стандартов IDE.

3.2. Доработка решения

В качестве ориентира для улучшения внешнего вида был выбран интерфейс IDE JetBrains (рис. 2). Для реализации улучшений необходимо добавить следующие функции:

1. Отображение стрелки рядом с каждым сворачиваемым блоком кода;
2. Возможность сворачивания и разворачивания блока с помощью клика на стрелку;
3. Отображение состояния блока с помощью визуальных индикаторов.

```
250    public HighFunction getHighFunction() {  Dan
251        if (currentDecompileData != null) {
252            return currentDecompileData.getHighFunction();
253        }
254        return null;
255    }
```

Рис. 2: Интерфейс IDE JetBrains

3.2.1. Отображение стрелки рядом с каждым сворачиваемым блоком кода

Для внесения изменений был выбран класс **LineNumberDecompilerMarginProvider**, отвечающий за отображение номеров строк на панели. Класс был модифицирован для добавления визуальных индикаторов — стрелок, расположенных рядом с каждым блоком, который может быть свернут или развернут.

Для определения строк, в которые необходимо добавить стрелки, используется метод **DecompilerPanel.getLinesWithOpeningBraces**. Этот метод выполняет обход всех строк и ищет позиции, соответствующие открывающим фигурным скобкам {. В результате эти строки помечаются как начало сворачиваемых блоков кода.

На рис. 3 показано состояние окна декомпилятора до внесения изменений, а на рис. 4 — после добавления стрелок.

```
4  {
5    void *pvVar1;
6    long in_FS_OFFSET;
7    undefined8 *in_stack_00000018;
8    long in_stack_00000078;
9
10   pvVar1 = (void *)*in_stack_00000018;
11   if (pvVar1 != (void *)0x0) {
12     operator.delete(pvVar1,in_stack_00000018[2] - (long)pvVar1);
13   }
14   if (in_stack_00000078 == *(long *)(in_FS_OFFSET + 0x28)) {
15                     /* WARNING: Subroutine does not return */
16     _Unwind_Resume();
17   }
18                     /* WARNING: Subroutine does not return */
19   __stack_chk_fail();
20 }
21
```

Рис. 3: Окно декомпиляции Ghidra до модификаций

```
4  {
5    void *pvVar1;
6    long in_FS_OFFSET;
7    undefined8 *in_stack_00000018;
8    long in_stack_00000078;
9
10   pvVar1 = (void *)*in_stack_00000018;
11   if (pvVar1 != (void *)0x0) {
12     operator.delete(pvVar1,in_stack_00000018[2] - (long)pvVar1);
13   }
14   if (in_stack_00000078 == *(long *)(in_FS_OFFSET + 0x28)) {
15                     /* WARNING: Subroutine does not return */
16     _Unwind_Resume();
17   }
18                     /* WARNING: Subroutine does not return */
19   __stack_chk_fail();
20 }
21
```

Рис. 4: Окно декомпиляции Ghidra после модификаций

3.2.2. Отображение состояния блока с помощью визуальных индикаторов

Для реализации отображения состояния блока был добавлен обработчик событий кликов мыши в класс

LineNumberDecompilerMarginProvider. При клике на стрелку вызывается метод **DecompilerPanel.arrowClickAction**, который благодаря встроенным методам Ghidra определяет номер строки по ординате клика и инициирует выполнение метода **toggleCollapseToken** с соответствующими параметрами.

Для корректного сохранения состояния вложенных блоков была изменена переменная **boolean collapsedToken** в классе **ClangToken** (используемая в решении сообщества) на **int collapseLevel**. Также были обновлены соответствующие геттеры и сеттеры. Это изменение позволяет каждому токену сохранять информацию о текущем уровне свёрнутости блока.

3.2.3. Отображение состояния блока с использованием визуальных индикаторов стрелок

Для реализации отображения состояния блока через визуальные индикаторы стрелок (рис. 5) необходимо различать стрелки для свёрнутых и обычных блоков. Для этого функция **getLinesWithOpeningBraces** возвращает словарь, в котором каждой строке сопоставляется значение **true** или **false**. Эти данные затем используются в изменённой версии метода **LineNumberDecompilerMarginProvider.paint**, который отображает стрелки на соответствующих строках в разных позициях, в зависимости от состояния блока (свёрнутый или развёрнутый).

```
6  {
7    void *pvVar1;
8    void *unaff_RBP;
9    ulong unaff_R12;
10   long in_FS_OFFSET;
11
12   if (unaff_RBP != (void *)0x0) {
13     operator.delete(unaff_RBP,unaff_R12);
14   }
15   pvVar1 = (void *)*param_9;
16   if (pvVar1 != (void *)0x0) {}
17   if (param_11 == *(long *)(in_FS_OFFSET + 0x28)) {}
18                     /* WARNING: Subroutine does not return */
19   __stack_chk_fail();
20 }
21
```

Рис. 5: Свёрнутый блок в окне декомпилятора Ghidra

Заключение

В ходе выполнения работы были успешно решены все поставленные задачи, в результате чего получены следующие результаты:

1. Проведен анализ Ghidra и существующего решения сообщества;
2. Внесены исправления в решение сообщества;
3. Доработана и интегрирована функция сворачивания кода в Ghidra.

Весь исходный код реализации доступен на GitHub в виде Pull Request в репозиторий кафедры СП[1].

[1]https://github.com/spbu-se/ghidra/pull/1

Список литературы

[1] Deva Prashant. Folding in Eclipse Text Editors. — URL: https://www.eclipse.org/articles/Article-Folding-in-Eclipse-Text-Editors/folding.html (дата обращения: 21 ноября 2024 г.). Описание реализации сворачивания текста в редакторах Eclipse.

[2] Eclipse Contributors. Package org.eclipse.jface.text.projection. — URL: https://help.eclipse.org/latest/rtopic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/projection/package-summary.html (дата обращения: 21 ноября 2024 г.). Описание пакета org.eclipse.jface.text.projection в документации Eclipse.

[3] Eclipse Contributors. ProjectionAnnotation (Eclipse Platform API Specification). — URL: https://help.eclipse.org/latest/rtopic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/source/projection/ProjectionAnnotation.html (дата обращения: 21 ноября 2024 г.). Описание класса ProjectionAnnotation в документации Eclipse.

[4] Eclipse Contributors. ProjectionDocument (Eclipse Platform API Specification). — URL: https://help.eclipse.org/latest/rtopic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/projection/ProjectionDocument.html (дата обращения: 21 ноября 2024 г.). Описание класса ProjectionDocument в документации Eclipse.

[5] Eclipse Contributors. ProjectionDocumentEvent (Eclipse Platform API Specification). — URL: https://help.eclipse.org/latest/rtopic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/projection/ProjectionDocumentEvent.html (дата обращения: 21 ноября 2024 г.). Описание класса ProjectionDocumentEvent в документации Eclipse.

[6] Eclipse Contributors. ProjectionDocumentManager (Eclipse Platform API Specification). — URL: https://help.eclipse.org/latest/rtopic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/projection/ProjectionDocumentManager.html (дата обращения: 21 ноября 2024 г.). Описание класса ProjectionDocumentManager в документации Eclipse.

[7] Eclipse Foundation. Eclipse IDE. — URL: https://eclipseide.org/ (дата обращения: 21 ноября 2024 г.). Официальный сайт интегрированной среды разработки Eclipse IDE.

[8] Enigmatrix. Code folding/Indentation marking in DecompilerPanel. — 2019. — URL: https://github.com/NationalSecurityAgency/ghidra/issues/1294 (дата обращения: 21 ноября 2024 г.). Обсуждение добавления сворачивания кода и маркировки отступов в DecompilerPanel Ghidra.

[9] Wall-AF. Comment on "Code folding/Indentation marking in DecompilerPanel" Issue. — 2022. — URL: https://github.com/NationalSecurityAgency/ghidra/issues/1294#issuecomment-1133644447 (дата обращения: 21 ноября 2024 г.). Комментарий с предложением временного решения для сворачивания кода в DecompilerPanel Ghidra.