Заключение

Наличие циклически замкнутого отношения использования на диаграмме пакетов означает возможность ошибки проектирования. Предложенный в работе алгоритм устраняет такие циклы, однако практическая реализация алгоритма должна оставлять решение о каждом преобразовании диаграммы пакетов за проектировщиком, который должен иметь возможность принять или запретить то или иное преобразование, осуществляемое алгоритмом.

Список литературы:

1. Борисенков Д.С. Диаграмма алгебраических типов данных расширенной нотации UML // Информационные технологии, энергетика и экономика. Сб. трудов 10-ой Межрегиональной (международной) научно-технической конференции студентов и аспирантов. Том 1. – Смоленск, 2013. – С. 99-103.

2. Фаулер М., Бек К., Брант Дж., Апдайк У., Робертс Д., Гамма Э. Рефакторинг. Улучшение существующего кода. – М.: Символ-Плюс, 2008. – 432 с.

3. Буч Г., Рамбо Г., Якобсон И. UML. Руководство пользователя. – М.: ДМК, 2000. – 358 с.

4. Кормен Т.Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы. Построение и анализ. – М.: Вильямс, 2013. – 1328 с.

РЕИНЖИНИРИНГ ДЛЯ ФУНКЦИОНАЛЬНЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

© Борисенков Д.С.[♣]

Филиал Национального исследовательского университета «МЭИ», г. Смоленск

В работе предлагается способ преобразования программ на функциональных языках программирования в диаграммы проектирования.

Ключевые слова: функциональное программирование, проектирование, реинжиниринг, UML.

[♣] Аспирант кафедры Вычислительной техники.

Обобщенный алгоритм реверс-инжиниринга

Обобщенный алгоритм реверс-инжиниринга можно представить следующим образом:

1. Исходными данными является текст программы на функциональном языке программирования.

2. Производится лексический анализ текста и генерируется последовательность токенов – программных представлений лексем языка – промежуточного языка представления программы, позволяющего описывать структуру программы абстрагируясь от конкретного языка программирования. Этот этап может быть разбит на два подэтапа:

   a. запуск лексера исходного языка программирования;
   b. преобразование последовательности токенов исходного языка программирования в последовательность токенов промежуточного языка.

Однако такое разбиение предпочтительно, только если исходный язык программирования позволяет осуществлять запуск лексера отдельно от остальных составляющих компилятора. В противном случае необходима разработка лексера промежуточного языка, который по исходным текстам программ, написанных на одном из поддерживаемых языков программирования, генерировал бы последовательность токенов промежуточного языка.

3. Последовательность токенов промежуточного языка передается на вход автомату, строящему диаграммы, рассмотренные в работах [1, 2].

Преобразование последовательности токенов промежуточного языка представления программы в структурные диаграммы расширенной нотации UML

Подмножество промежуточного языка представления программы, используемое для реверс-инжиниринга модульной структуры системы представлено в таблице 1.

Таблица 1

Токены «верхнего уровня» промежуточного языка представления программы и их семантика

Токен	Описание
Module	Переводит автомат в состояние чтения имени пакета (UML-аналог модуля)
Use	Переводит автомат в состояние чтения экспортируемого текущим пакетом модуля
ID(x)	Идентификатор x; переводит автомат в состояние по умолчанию.

Промежуточный язык представления является контекстно-зависимым, в частности, пакет не может импортировать данные несуществующего пакета. Кроме того, составные типы данных из одного пакета могут использовать в качестве своих элементов типы данных, описанных в других пакетах, при этом типы данных из импортированных пакетов добавляются в пространство имен рассматриваемого пакета и являются «видимыми» для него. Анализ последовательности токенов можно разбить на 3 прохода:

1. Выявление модулей, описываемых последовательностью токенов, генерация пустых пакетов диаграммы.
2. Выявление зависимостей между модулями, модификация пакетов диаграммы с учетом обнаруженных зависимостей. Выявление описаний типов данных, генерация диаграмм-заглушек для их представления.
3. Генерация полноценных диаграмм АТД для каждого модуля.

На рисунке 1 представлен автомат, распознающий модули в последовательности токенов. Каждая стрелка означает переход между состояниями автомата. Запись рядом со стрелкой показывает токен, по которому осуществляется переход и дополнительное действие, которое совершает автомат помимо перехода в новое состояние. На вход автомату подается последовательность токенов, на выходе – последовательность пакетов.

Отношения импорта между пакетами (токены Use, ID(x)) определяются контекстно-зависимым подмножеством промежуточного языка: пакет может импортировать только объявленный пакет и сама операция импорта имеет смысл только в контексте некоторого пакета. Список объявленных пакетов был уже получен нами на предыдущем проходе. Для того чтобы учесть кон-

текст, в котором используется операция импорта, добавим к автомату ячейку памяти P, в которой будет храниться текущий рассматриваемый пакет. Так же нам понадобится функция Пакет, принимающая на вход строку и возвращающая пакет из списка, полученного в первом проходе с именем, равным переданному. Если такого пакета нет, функция возвращает пустое значение.

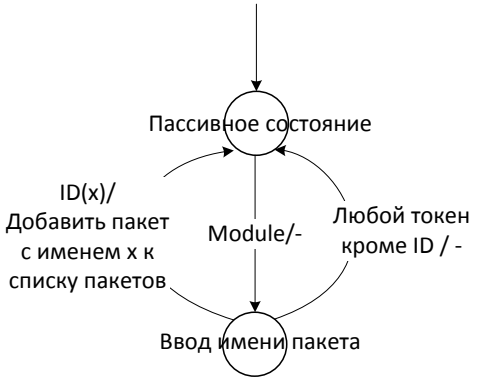

Рис. 1. Генерация последовательности пустых пакетов

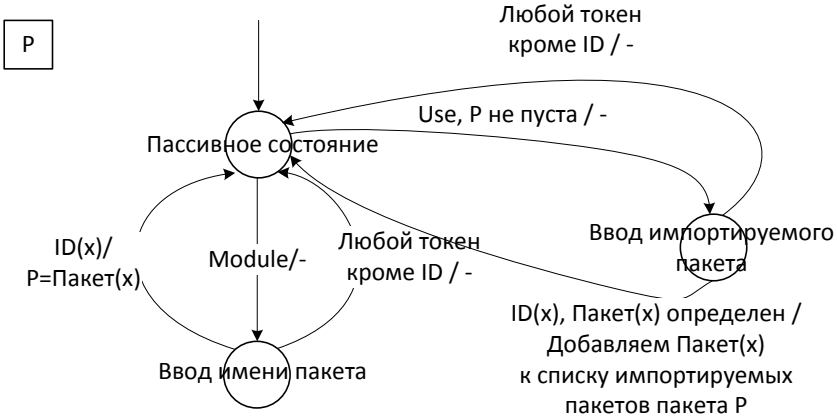

Рис. 2. Определение отношения импорта между пакетами

На рисунке 2 представлен автомат с памятью, распознающий зависимости между пакетами. Если в процессе обработки последовательности токе-

нов возникнет ситуация, когда невозможно выполнить переход в автомате, генерируется ошибка с указанием токена, котором она возникла.

Помимо определения зависимостей, во втором проходе собираются описания типов данных, определенных в модулях аналогично тому, как в первом проходе собирались описания самих модулей – отыскиваются токены Data/Type, ID(x), помещенные в контекст некоторого пакета и для этого пакета создается заглушка диаграммы АТД с именем x. Таким образом, на вход автомата второго прохода подается список пакетов, полученных на первом проходе и последовательность токенов. Результатом работы автомата является диаграмма пакетов с указанными отношениями импорта и диаграммы-заглушки для каждого из АТД пакетов.

Третий проход анализирует токены, предназначенные для описания типов данных функциональных языков программирования. Эти токены приведены в таблице 2.

Таблица 2

Токены, предназначенные для описания типов данных

Токен	Описание
Data	Объявление нового типа данных
Tuple	Объявление синонима типа данных (кортежа)
TupleSep	Разделитель между элементами кортежа
ADTSep	Разделитель между определением и объявлением алгебраического типа данных
Case	Разделитель между конструкторами АТД
ConsSep	Разделитель между конструктором данных и данными
TypeSep	Разделитель между данными и типом
ElemSep	Разделитель между элементами записи
RecordStart(ID)	Начало определения записи

Так как объявления типов данных находятся в контексте модулей и могут использовать в своих определениях другие типы данных из пространства имен модуля, для проверки корректности определений необходима дополнительная память для запоминания контекста и проверка наличия описания соответствующего значению идентификатора типа данных в пространстве имен соответствующего модулю пакета. Эти проверки во многом

аналогичны приведенным во втором проходе проверкам наличия имени пакета в глобальном пространстве имен и запоминанию текущего пакета при обработке отношения импорта. Для упрощения графа автомата распознания типов данных опустим эти проверки – будем подразумевать, что все описания типов данных выполняются в контексте некоторого модуля и находятся в его пространстве имен. Если это не так, пользователю сообщают об ошибке разбора последовательности токенов с указанием номера токена, на котором возникла ошибка. Автомат, распознающий типы данных приведен на рисунке 3.

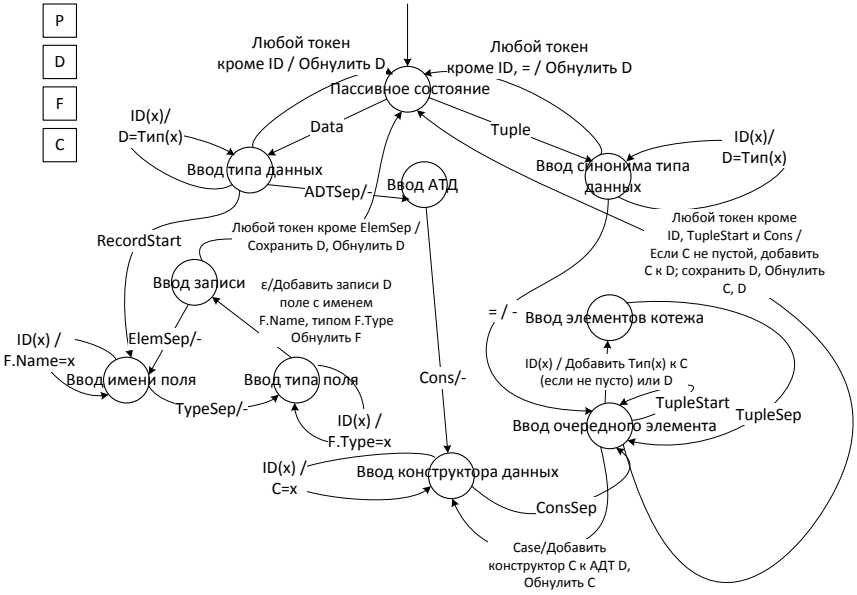

Рис. 3. Распознание типов данных

Примечание: все присвоения переменным осуществляются только если ячейка памяти пуста, иначе генерируется ошибка.

Входными данными третьего прохода являются диаграмма пакетов второго прохода и исходная последовательность токенов, а результатом является диаграмма пакетов с вложенными диаграммами АТД. Таким образом, исходная задача реверс-инжиниринга решена.

Заключение

В данной работе была решена задача автоматизации преобразования исходных текстов программ на функциональных языках программирования в структурные диаграммы расширенной нотации UML. Предложенный алгоритм может быть использован совместно с промежуточным языком представления программы и расширениями нотации UML может быть использован в CASE-средстве для функционального проектирования.

Список литературы:

1. Борисенков Д.С. Нотация и методология функционального проектирования // Информационные технологии, энергетика и экономика. Сб. трудов 9-ой Межрегиональной (международной) научно-технической конференции студентов и аспирантов. Том 1. – Смоленск, 2012. – С. 22-25.

2. Борисенков Д.С. Диаграмма алгебраических типов данных расширенной нотации UML // Информационные технологии, энергетика и экономика. Сб. трудов 10-ой Межрегиональной (международной) научно-технической конференции студентов и аспирантов. Том 1. – Смоленск, 2013. – С. 99-103.

ПРИМЕНЕНИЕ СИСТЕМНО-МОРФОЛОГИЧЕСКОГО ПОДХОДА ДЛЯ СИНТЕЗА СИСТЕМ С УВЕЛИЧЕННОЙ СТЕПЕНЬЮ ИДЕАЛЬНОСТИ

© Гонжал Е.И.♣

Волгоградский государственный технический университет, г. Волгоград

Предложен новый метод концептуального проектирования системы, основанный на системно-морфологическом подходе, представляющий из себя структурированную систему, правила в которой изменяются в зависимости от выбранного направления идеализации.

♣ Магистрант.