УДК 519.683

Классификация программ, ориентированная на технологию программирования

В.И. Шелехов

Институт систем информатики им. А.П. Ершова СО РАН
630090, Новосибирск, пр. Лаврентьева, 6, Россия
vshel@iis.nsk.su

Рассматривается классификация программ по их внутренней организации, определяющей интерфейс с внешним окружением, форму спецификации программы и другие особенности. Классификация ориентирована на разработку адекватной технологии для каждого класса программ. Определены три класса программ: программ-функций, реактивных систем и языковых процессоров.

Ключевые слова: *программа, технология программирования, определение требований, спецификация программы, реактивная система, формальная семантика.*

Введение

История развития программной инженерии демонстрирует калейдоскопическую смену большого числа моделей, методов и инструментов. Разрабатываемые технологии программирования в основном ориентированы на вполне определенные ограниченные классы программ. Общеизвестны такие классы программ как, например, реактивные системы, системы реального времени, вероятностные автоматы.

Даже универсальная технология программирования не является одинаково успешной для всех программ. Поэтому полезно определить класс программ, соответствующих каждой технологии. Актуальной задачей является построение полной системы классов программ, где каждый класс определяется своей особенной внутренней организацией программ, предопределяющей технологию программирования.

В мировой литературе не обнаружено попыток построить полную систему классов программ. В работах встречаются лишь общие поверхностные определения классов, как правило, без устоявшихся общепринятых математических моделей. Известны другие классификации**:** классификации программ по их назначению, классификация языков программирования и классификация парадигм программирования. Все они имеют мало общего с требуемой классификацией программ, учитывающей внутреннюю организацию программ.

Попытка создания общего теоретического базиса для разных классов и парадигм предпринята Хоаром и другими учеными в рамках серии работ по *унифицированным теориям программирования* (*Unifying theories of programming*, UTP) [1]. Для каждого класса программ определяется минимальное ядро языка программирования; далее разрабатываются формальная семантика ядра и методы формальной верификации. Эти работы ограничены созданием общей математической теории программирования и пока не получили продолжения в создании новых инструментов и технологий программирования.

В данном контексте следует также отметить инициативу SEMAT (Software Engineering Method and Theory) [2], объявленную в 2009г. и получившую поддержку большого числа ученых, научных организаций и софтверных фирм. Цель инициативы – создание единой теории в программной инженерии, которая стала бы направляющей основой ее дальнейшего развития. Отмечался разрыв между научными исследованиями и практикой программирования. При отсутствии общего теоретического базиса современное развитие программной инженерии оказалось в зависимости от рекламного продавливания своих продуктов крупными софтверными корпорациями. Были сформулированы основные положения и цели инициати-

вы, которые послужили основой для ее обсуждения на нескольких научных конференциях, в т.ч. с применением мозгового штурма. Через четыре года работы в рамках инициативы были приостановлены.

Предикатное программирование [3-6] возникло как отрицание императивного программирования. Класс предикатных программ определен как класс *программ-функций*, соответствующий моделям «программа есть функция» или «программа – это предикат», аналогично, как и в UTP [1]. Класс реактивных систем реализован в другой парадигме – в автоматном программировании [7]. Здесь используется автоматная модель программы в виде гиперграфа, являющаяся продолжением аппарата гиперфункций [3, 6] в предикатном программировании.

Интересен опыт распространения функционального программирования на класс реактивных систем, в частности, с использованием языков Haskell и Erlang, на базе акторской модели [8], допускающей взаимодействие между параллельными подпрограммами реактивной системы только через прием и посылку сообщений. Отметим, что акторская модель покрывает лишь часть класса реактивных систем – не допускаются разделяемые переменные, которые обязательны, например, в серии протоколов взаимного исключения.

Работы по классификации программ начались с попытки найти точную границу между классом реактивных систем и классом программ-функций. Обнаружилось, что эти два класса не покрывают всего множества программ, в частности, программ трансляторов с языков программирования.

Настоящая работа по классификации программ – это одна из попыток построить единый фундамент в программной инженерии. Определены три класса программ: программ-функций, реактивных систем и языковых процессоров. Данная работа также может быть использована для обучения студентов и аспирантов, специализирующихся в области программной инженерии. Описанию трех классов программ предшествуют определения базисных понятий программы, требований и спецификации программы.

1. Общее понятие программы

В теоретическом и общеметодологическом плане понятие *программы* является предметом *информатики* (Computer science), в производственном – *программной инженерии* (Software engineering) – научно-инженерной дисциплины, определяющей методы и инструменты разработки программ. Технология программирования часто используется как эквивалент программной инженерии, хотя содержание этих понятий не совпадает.

Понятие программы появилось вместе с первыми компьютерами в 1940-х [1]. Развитие средств вычислений сопровождалось все большей степенью *автоматизации вычислений*. Потребность в проведении сложных и длительных расчетов привела человека к необходимости создания и совершенствования механических устройств и машин для облегчения процесса вычисления. С древних времен известно устройство, которое в России называется «счеты». Счетные машины и арифмометры, создаваемые в XVII–XIX вв., автоматизировали выполнение четырех арифметических действий. В середине XX столетия актуальные потребности науки и практики привели к созданию новой отрасли техники – конструированию и производству счетно-решающих устройств, т. е. приборов и машин для решения математических задач [9]. Применение счетно-решающих устройств становится массовым: они используются для быстрых расчетов в боевой обстановке, в навигации, в автоматизированном управлении сложными агрегатами и т. д. Появление универсальных электронно-вычислительных машин (ЭВМ), умеющих автоматически выполнять программу, построенную для произвольного алгоритма, стало неизбежным.

Промежуточное состояние автоматизации вычислений алгоритма реализуется современными калькуляторами, сравнимыми по своим возможностям со счетно-решающими уст-

[1] Первые программы писала математик Ада Лавлейс, дочь поэта лорда Байрона, для спроектированной, но не реализованной вычислительной машины Чарльза Беббиджа в 1830-х гг.

ройствами 1940-х гг. На очередном шаге своей работы калькулятор либо принимает данные от пользователя, либо в автоматическом режиме вычисляет одну из операций. Более развитые калькуляторы позволяют сохранять промежуточные значения вычислений, именовать отдельные значения и массивы значений и использовать эти имена в дальнейших вычислениях. Таким образом, эти калькуляторы могут хранить в своей памяти значения нескольких переменных. Подчеркнем, что в сравнении с ручным счетом калькулятор автоматизирует вычисление только одной операции на каждом шаге, а между шагами требуются действия человека. С помощью калькулятора можно провести расчеты в принципе по любому алгоритму. Если мы захотим исключить действия человека между шагами и ограничить его действия лишь вводом начальных данных, то потребуется каким-то способом задать всю совокупность операций алгоритма и указать порядок вычисления операций. Необходимым инструментом для этого является программа.

Программа есть алгоритм, реализованный таким способом и в такой форме, что вычисление алгоритма проводится автоматически. По способу реализации различаются аппаратно реализованные программы (например, в виде интегральной схемы) и программы на языках программирования. *Автоматическая вычислимость* является неотъемлемым свойством программы. Приведенное определение уточняет краткое определение в энциклопедиях: «программа есть описание алгоритма решения задачи, заданное на языке программирования» [10].

Понятия алгоритма и программы не тождественны. Понятие *алгоритма* известно давно и считается хорошо изученным. В математике существуют следующие формализации понятия алгоритма: машина Тьюринга [11], частично рекурсивные функции, нормальный алгорифм Маркова [12] и каноническая система Поста [13]. Используется также другая трактовка понятия алгоритма как набора алгоритмических предписаний, применяемых в различных сферах человеческой деятельности.

Окружение программы – программные и аппаратные компоненты, непосредственно взаимодействующие с программой при ее исполнении и реализующие входные и выходные информационные потоки программы.

2. Требования

Требование — утверждение относительно одного из свойств, которым должна обладать создаваемая *система*. В качестве системы может быть все, что угодно: токарный станок, лекарственный препарат, робот, космический аппарат, программная система и многое другое. Определение требований и методы работы с ними являются предметом *инженерии требований,* примыкающей к другой научно-инженерной дисциплине – *системной инженерии* [14], определяющей методы проектирования сложных систем.

Инженерия требований для программных систем рассматривается отдельно, примыкая к программной инженерии. Эффективные методы построения требований зафиксированы стандартом IEEE 830-1998 [15, 16]. В соответствии со стандартом требования должны быть: корректными, однозначными, полными, согласованными, ранжированными по значимости и обязательности, проверяемыми, модифицируемыми, хорошо организованными для анализа.

Виды требований. Требования делятся на два класса: функциональные и нефункциональные. *Функциональные требования* определяют поведение программной системы. Есть разные подходы к описанию функциональных требований. Наиболее популярной формой являются *сценарии использования* (*use case*): на каждое событие в окружении программы определяется ее реакция с указанием всех вариантов. Виды *нефункциональных требований* следующие: требования к окружению программы, требования к оборудованию, ограничения реализации, требования баз данных, стандарты. К нефункциональным требованиям также относятся характеристики программы: надежность, доступность, защищенность, сопровождаемость, переносимость и другие.

Языками спецификации требований являются: естественный язык, английский (80% случаев), формализованное подмножество естественного языка с использованием аппарата онтологий (15%) и формальный язык (5%) [17]. Формальными языками являются: Statechart [18] SDL[19], UML и др. Большинство из них являются графическими. Формальными языками спецификации требований являются также общеизвестные универсальные языки спецификаций: VDM, Z, Event-B [20] и др. Семейство темпоральных языков спецификаций также используется для спецификации требований программных систем; наиболее популярным является язык LTL.

В мировой практике инженерия требований применяется преимущественно для больших информационных и телекоммуникационных систем. Разработку требований осуществляют специалисты – *инженеры требований,* составляющих иногда половину персонала.

3. Спецификация программы

Спецификация программы – это описание программы, в принципе любое, достаточно полное, чтобы на его основе можно было построить алгоритм программы. При таком определении код программы является ее спецификацией, что, разумеется, неверно. В действительности, программа и ее спецификация противопоставляются. Спецификация декларативна: она определяет, что вычисляет программа, а не как реализуется процесс исполнения. Более точное определение следующее. *Спецификация программы* – точное, полное и однозначное описание преобразования информации, реализуемого программой, т.е. описание зависимости результатов исполнения программы от исходных данных.

Спецификация определяется как результат намерения создать программу. Поэтому спецификация первична по отношению к программе. Однако она не всегда фиксируется в документальном виде, а если фиксируется, то может быть неполной, неточной, устаревшей.

Требование корректности программы. *Программа должна соответствовать спецификации*: набор утверждений, составляющих спецификацию программы, должен быть истинным для значений результатов исполнения программы и входных данных при любом исполнении программы. *Верификации программы* – проверка корректности программы относительно спецификации. Используются различные методы верификации программ: тестирование, экспертиза кода, статический анализ, формальные методы верификации [21].

Спецификации часто записываются на естественном языке, расширенном набором специальных обозначений предметной области. Спецификация, записанная на строгом формальном языке спецификации, является *формальной*. Она определяет математическое описание реализуемой программы.

Формальные методы (Formal methods) базируются на формальной спецификации программ. Формальные методы включают:

- тестирование на базе формальной спецификации;
- статическую верификацию (software model checking);
- проверку на моделях (model checking);
- дедуктивную верификацию;
- программный синтез.

Спецификация программы конструируется с учетом требований к программе и может содержать описание некоторых требований.

4. Задача классификации программ

Методы программной инженерии, доказавшие свою эффективность для многих программ, не всегда успешно применимы для всех программ. Причина здесь в различиях архитектур программ. Подобная ситуация часто возникает в контексте применения формальных методов. Это ставит задачу *классификации программ*, т.е. построения системы классов программ, и разработку адекватных методов для каждого класса.

Обычно рассматриваются классификации по назначению программ. Здесь же определяется классификация программ по их внутренней организации. Цель классификации – разработка адекватной технологии программирования для каждого класса программ. Теория программ каждого класса должна определять методы спецификации, верификации (в широком смысле), моделирования и эффективной реализации программ. Базисом классификации являются:

- внешняя форма программы (интерфейс с окружением);
- базисные конструкции языка программирования;
- формы определения спецификации программы;
- виды условий корректности программы.

Генеральная классификация определяет два класса программ: невзаимодействующие программы (или *программы-функции*) и реактивные системы (или *программы-процессы*). Данные два класса составляют более 90% всех программ. Имеются другие, более сложные классы, например, языковые процессоры и операционные системы; они находятся на метауровне по отношению к первым двум классам. Языковые процессоры – это интерпретаторы программ, компиляторы, оптимизаторы, трансформаторы и т.д. Приведенная классификация не покрывает всего спектра программ, в частности, определяемых разнообразного вида фреймворками, и различных особенностей, например такой, как тесная интеграция программы с данными.

5. Класс программ-функций

Программа принадлежит этому классу, если она не взаимодействует с внешним окружением; точнее, если возможно перестроить программу таким образом, чтобы все операторы ввода данных находились в начале программы, а весь вывод собран в конце программы. Если подобная перестройка программы принципиально невозможна, ее следует пытаться определять в виде реактивной системы. Программа обязана всегда нормально завершаться с получением результата, поскольку бесконечно работающая и невзаимодействующая программа бесполезна. Программа определяет функцию, вычисляющую по набору входных данных (аргументов) некоторый набор результатов. Класс программ-функций, по меньшей мере, содержит программы для задач дискретной и вычислительной математики.

Таким образом, окружение программы-функции имеет простую структуру и состоит из средств ввода аргументов и вывода результатов программы.

Программу-функцию U с набором аргументов x и набором результатов y будем записывать в виде $U(x\mathbf{:}\ y)$. *Спецификацией* программы-функции являются два предиката: *предусловие* $P(x)$ и *постусловие* $Q(x, y)$. Предусловие ограничивает допустимый набор аргументов, а постусловие определяет связь между значениями аргументов и результатов. Спецификацию программы будем записывать в виде $[P(x), Q(x, y)]$.

Однозначность и тотальность спецификации $[P(x), Q(x, y)]$ определяются, соответственно, формулами:

$$P(x)\ \&\ Q(x, y_1)\ \&\ Q(x, y_2) \Rightarrow y_1 = y_2.$$
$$P(x) \Rightarrow \exists y.\ Q(x, y);$$

Программа должна соответствовать спецификации. Это требование формулируется в виде условия *частичной корректности*: если перед исполнением программы $U(x\mathbf{:}\ y)$ истинно предусловие $P(x)$, то в случае завершения программы должно быть истинно постусловие $Q(x, y)$ в момент ее завершения. Обязательным является также *условие завершения программы*: если перед исполнением программы $U(x\mathbf{:}\ y)$ истинно предусловие $P(x)$, то программа обязана завершаться. Объединение условий частичной корректности и завершения программы определяет условие *тотальной корректности*.

Адекватная формализация условий корректности программы возможна лишь при использовании формальной операционной семантики языка программирования. *Операционную семантику* программы U(x: y) определим в виде предиката:

$\mathcal{R}(U)(x, y) \cong$ для значения набора x исполнение программы U всегда завершается и существует исполнение программы, при котором результатом вычисления является значение набора y.

Данное определение исключает ситуацию, когда для некоторого значения набора x существуют два разных исполнения программы, одно из которых завершается, а другое – не завершается. В этом случае $\mathcal{R}(U)(x, y)$ будет ложным для любых y.

Однозначность и тотальность программы U(x: y) для некоторого значения x определяются, соответственно, формулами:

$$\mathcal{R}(U)(x, y_1) \,\&\, \mathcal{R}(U)(x, y_2) \Rightarrow y_1 = y_2$$
$$\exists y.\ \mathcal{R}(U)(x, y)\ .$$

Отметим, что тотальность программы для некоторого значения x есть в точности условие завершения программы для этого значения x. Программа называется *однозначной*, если она однозначна для всех значений аргументов

Операционная семантика $\mathcal{R}(U)$ является эквивалентом программы U. Доказательство некоторого свойства программы W(x, y) реализуется доказательством истинности формулы: $\mathcal{R}(U)(x, y) \Rightarrow W(x, y)$. Кроме того, эту формулу достаточно доказать при истинном предусловии. С учетом этого, условие частичной корректности программы U(x: y) записывается в виде формулы:

$$\forall x,y.\ P(x) \,\&\, \mathcal{R}(U)(x, y) \Rightarrow Q(x, y)\ .$$

Условие завершения программы U(x: y) при истинном предусловии представляется формулой:

$$\forall x.\ P(x) \Rightarrow \exists y.\ \mathcal{R}(U)(x, y)\ .$$

Формула для условия тотальной корректности программы получается объединением двух формул:

$$\forall x.\ P(x) \Rightarrow [\forall y.\ \mathcal{R}(U)(x, y) \Rightarrow Q(x, y)\ ] \,\&\, \exists y.\ \mathcal{R}(U)(x, y)\ .$$

Лемма 1. Если программа U(x: y) тотально корректна относительно спецификации [P(x), Q(x, y)], то спецификация тотальна.

Для конструирования программ-функций допустим любой язык императивного или функционального программирования. Минимальный язык P_0, из которого можно построить полный язык предикатного программирования, определен в работе [22]. Операторами языка P_0 являются: *оператор суперпозиции* B(x: z); C(z: y), *параллельный оператор* B(x: y) || C(x: z), *условный оператор* **if (**e) B(x: y) **else** C(x: y), *вызов программы* и *оператор каррирования* D(y: z) {B(x, y: z)}.

6. Класс программ-процессов

Программа-процесс является *реактивной системой*, реагирующей на определенный набор событий (сообщений) во внешнем окружении программы. Взаимодействие программы с окружением реализуется через прием / посылку сообщений и разделяемые переменные, доступные в программе и окружении.

Сообщение – объект вида $m(x_1,x_2,...,x_n)$, где m – *имя сообщения*, $x_1,x_2,...,x_n$ ($n\geq 0$) – переменные, являющиеся *параметрами сообщения*. Сообщение, пришедшее из окружения программы-процесса, может быть получено *оператором приема сообщения*, которому становятся доступными значения переменных $x_1,x_2,...,x_n$. Оператор **send** $m(e_1,e_2,...,e_n)$ посылает сообщение m с параметрами – значениями выражений $e_1,e_2,...,e_n$.

Разделяемая переменная является глобальной по отношению к программе-процессу. Она доступна по чтению и/или записи внутри программы, а также может быть модифицирована в окружении программы.

Программа-процесс является либо автоматной программой, либо она определяется в виде композиции нескольких автоматных программ, исполняемых параллельно и взаимодействующих между собой через сообщения и разделяемые переменные.

Автоматная программа состоит из одного или нескольких сегментов. *Сегмент* имеет один *вход*, помеченный меткой – *управляющим состоянием*. Сегмент имеет один или несколько выходов. *Выход* реализуется оператором перехода на начало другого сегмента (или того же самого), либо как выход из автоматной программы. Автоматная программа определяет конечный автомат в виде *гиперграфа* с набором управляющих состояний в качестве вершин и набором сегментов в качестве ориентированных гипердуг. Имеется одно *начальное управляющее состояние*, с которого начинается исполнение автоматной программы. Автоматная программа может иметь одно или несколько *выходных управляющих состояний*, которыми завершается исполнение автоматной программы. Отметим, что гиперграфовая модель является наиболее общей среди других используемых моделей автоматных программ.

Состояние автоматной программы определяется значениями набора модифицируемых переменных, локальных по отношению к автоматной программе и глобальных по отношению к каждому ее сегменту.

Структура класса реактивных систем. Частью задачи классификации программ является определение подклассов класса программ–процессов, т.е. класса реактивных систем.

Подклассом реактивных систем являются *гибридные системы*, соединяющие дискретное и непрерывное поведение. Часть переменных состояния гибридной системы соответствует непрерывным параметрам (типа **real**), изменение которых реализуется независимо от программы гибридной системы (вне ее) по определенным законам, обычно формулируемым в виде дифференциальных уравнений. Важнейшими подклассами гибридных систем являются контроллеры систем управления и временные автоматы.

Система управления реализует взаимодействие с *объектом управления* для поддержания его функционирования в соответствии с поставленной целью. Системы управления используются в аэрокосмической отрасли, энергетике, медицине, робототехнике, массовом транспорте и др. отраслях. На каждом шаге вычислительного цикла *контроллер системы управления* получает входную информацию из окружения и обрабатывает ее. Результаты вычисления используются для передачи управляющего сигнала для воздействия на объект управления.

Временной автомат реализует функционирование процесса, используя показания времени. Пересчет времени проводится вне программы-процесса (временного автомата). Имеются различные модели автоматов с дискретным и непрерывным временем [23]. Временной автомат является *системой реального времени*, если взаимодействие с окружением должно удовлетворять временным ограничениям, что характерно для встроенных систем. В системах с жестким реальным временем непредоставление результатов вычислений к определенному сроку является фатальной ошибкой. Большинство систем управления являются встроенными системами.

Автоматная программа является *детерминированной*, если каждое управляющее состояние метит не более одного сегмента. Для *недетерминированного автомата* одно управляющее состояние может метить несколько сегментов. При исполнении программы из данного управляющего состояния недетерминировано выбирается один из сегментов. Отметим, что недетерминизм неявно реализуется для параллельной композиции автоматных программ.

Автомат является *вероятностным*, если для каждого сегмента определена вероятность его выбора, причем сумма вероятностей сегментов, исходящих из одного управляющего состояния, равна единице.

Программа может быть составлена из частей, принадлежащим разным подклассам реактивных систем. Например, возможно сочетание вероятностных и недетерминированных автоматов в рамках одной программы. Системы реального времени в большинстве случаев являются системами управления. В дополнении к этому автоматная программа может быть частью *распределенной системы*. Отметим также возможность интеграции с объектно-ориентированной технологией, когда состояние автоматной программы реализовано как объект класса.

Валидация и верификация реактивной системы. Разработка реактивной системы начинается с определения ее требований. *Валидация требований* заключается в проверке требований на соответствие потребностям пользователей. Обычно здесь применяется моделирование. Результаты валидации оцениваются совместно разработчиком и заказчиком.

Спецификация реактивной системы включает: инварианты управляющих состояний, свойства, формулируемые на языке темпоральной логики, и описание части требований.

Инвариант управляющего состояния – предикат, который должен быть истинным в начале сегмента, ассоциированного с данным управляющим состоянием. Инварианты реактивных систем принципиально отличаются от инвариантов циклов и инвариантов классов императивных программ.

Объектами верификации могут быть также *свойства* автоматной программы, обычно формулируемые на языке темпоральной логики. Они могут быть верифицированы с помощью инструментов проверки на модели (model checking).

Программа-функция – это автоматная программа с двумя управляющими состояниями: одно входное и одно выходное. Предусловие и постусловие полностью определяют функциональные требования к программе. Иногда формулируются отдельные нефункциональные требования, однако обычно эта часть требований отсутствует.

7. Класс языковых процессоров

Языковые процессоры – это интерпретаторы программ, трансляторы, оптимизаторы, трансформаторы, смешанные вычислители, конверторы и другие операции с программами.

Язык программирования – формальный язык, определяющий правила записи программы в виде текста в конечном *алфавите* символов. *Лексические правила* определяют правила кодирования символов алфавита (*лексем*) посредством алфавита компьютера, а также программы в целом в конкретной файловой системе.

Описание языка программирования определяет: типы данных, структуру памяти исполняемой программы, виды языковых конструкций программы, правила исполнения конструкций каждого вида и программы в целом.

Языковая конструкция – независимая часть программы с определенным для нее синтаксисом и семантикой. Язык программирования характеризуется набором *видов языковых конструкций*. Например, вид Оператор присваивания, синтаксически изображаемый композицией: <Переменная> := <Выражение>, с двумя *позициями* для подконструкций. Виды конструкций объединяются в *группы*. Примеры групп: оператор, выражение, переменная, операция.

Для всякого вида языковых конструкций определены синтаксис и семантика. *Синтаксис* определяет правила представления конструкции в виде последовательности символов алфавита. *Семантика* определяет правила *исполнения* произвольной конструкции данного вида. Частью семантики является *статическая семантика*: правила идентификации переменных и других объектов программы, совокупность ограничений на типы (и другие атрибуты) конструкций и система умолчаний. Вторая часть семантики является собствен*но семантикой исполнения* программы.

Точная спецификация языка программирования, проводимая преимущественно для целей формальной верификации программ, реализуется в виде *формальной семантики*, представляющей математическое описание семантики языка программирования. Формальная семантика есть описание семантики исполнения, абстрагированное от синтаксиса и статической семантики. Различают следующие виды формальной семантики: операционную, денотационную и аксиоматическую [24, 25]; алгебраическая семантика [26] является разновидностью денотационной. *Операционная семантика* для всякого вида K определяет предикат R(K), истинный при нормальном завершении исполнения конструкции вида K. Описание формальной семантики реальных языков программирования оказывается сложным и громоздким.

Интерпретатор – программа, реализующая автоматическое исполнение произвольной программы на языке программирования. В содержательном пользовательском описании семантики исполнения языка программирования интерпретатор подразумевается неявно. Чтобы определить *спецификацию интерпретатора*, рассмотрим отображение S формальной операционной семантики на пользовательское представление языка программирования: конструкции формальной семантики кодируются в синтаксических структурах исходного языка. Необходимо будет явно определить интерпретатор исходного языка, в частности, формализовать структуру памяти исполняемой программы. Исполнение конструкции вида K реализуется независимой подпрограммой интерпретатора. Результаты ее исполнения должны удовлетворять предикату S(R(K)).

Интерпретатор программы на исходном языке крайне неэффективен. Необходимым инструментом программирования является *транслятор*, преобразующий программу на другой язык, называемый *объектным*, для которого обеспечено эффективное исполнение программы. При построении транслятора необходимо определить объектное представление программы. Формально это можно представить как отображение O формальной операционной семантики на объектный язык. Тогда *спецификацию транслятора* можно было бы представить в виде функции, отображающей S(R(K)) в O(R(K)). Такая спецификация должна гарантировать, что любое исполнение исходной программы будет совпадать по всем промежуточным результатам с соответствующим исполнением объектной программы.

Однако любой транслятор, иногда даже простой конвертор, дополнительно проводит эквивалентные оптимизирующие преобразования программы, например, константные вычисления. Поэтому в действительности спецификация транслятора значительно сложнее: это перевод программы на объектный язык с выполнением набора эквивалентных оптимизирующих преобразований, гарантирующих совпадение конечных результатов вычисления.

Класс языковых процессоров существенно сложнее класса программ-функций. Спецификация языкового процессора базируется на формальной семантике, оперирующей множеством исполнений произвольной программы.

Заключение

В настоящей работе рассматривается классификация программ по их внутренней организации, определяющей интерфейс с внешним окружением, форму спецификации программы, базисные языковые конструкции и другие особенности. Классификация ориентирована на разработку адекватной технологии для каждого класса программ. На базе общей системы понятий, используемых в программной инженерии, определены три класса программ: программ-функций, реактивных систем и языковых процессоров. В дальнейшем предстоит идентифицировать и определить класс операционных систем, являющийся метауровневым по отношению к первым двум классам. Приведенная система классов заведомо неполна. Для продолжения классификации программ необходимо будет проанализировать самые разные виды приложений, в частности, связанные с базами данных и фреймворками.

Одной из задач классификации программ является дальнейшая детализация подклассов внутри класса реактивных систем. Задача следующего уровня – построение моделей подклассов с ориентацией на технологию программирования. На примере программы управления беспилотным летальным аппаратом описана типовая модель систем управления с включением трех дополнительных слоев: интеграции автоматического и ручного управления, мониторинга и защиты от несанкционированного доступа [27]. Дальнейшей задачей является определение модели движения объекта в трехмерном пространстве с ориентацией на системы робототехники.

Работа выполнена при поддержке РФФИ, грант № 16-01-00498.

Список литературы

1. Hoare C.A.R. and Jifeng He. Unifying Theories of Programming. – Prentice Hall Series in Computer Science. Prentice Hall Europe. −1998.

2. Software Engineering Method and Theory. URL:http://www.semat.org/

3. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования Р // Препринт №153. – Новосибирск: ИСИ СО РАН, 2010. – 42с.

4. Shelekhov V. I. Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // // Automatic Control and Computer Sciences. – 2011. Vol. 45, No. 7. – P. 421–427.

5. Шелехов В.И. Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования // Программная инженерия, 2011, № 2. — С. 14-21.

6. Шелехов В.И. Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования // Препринт №164. – Новосибирск: ИСИ СО РАН, 2012. – 30с.

7. Шелехов В.И. Язык и технология автоматного программирования // Программная инженерия» , 2014, №4. – С. 3-15.
URL: http://persons.iis.nsk.su/files/persons/pages/automatProg.pdf

8. Hewitt C., Bisshop P. and Steiger R. A Universal Modular Actor Formalism for Artificial Intelligence // 3rd International Joint Conference on Artificial Intelligence, Stanford, CA, 1973. – P. 235-245.

9. Ершов А .П., Шура-Бура М. Р. Пути развития программирования в СССР // Препринт №12. – Новосибирск: ВЦ СО АН СССР, 1976.

10. Программа // Краткая российская энциклопедия. М.: Большая российская энциклопедия ОНИКС 21 век, 2003.

11. Клини С. К. Введение в метаматематику: Пер. с англ., М., 1957.

12. Марков А. А., Нагорный Н.М. Теория алгорифмов. М.: Наука, Физматлит, 1984. – 432 с.

13. Минский М. Вычисления и автоматы: Пер. с англ. М.: Мир, 1971. – 364 с.

14. Левенчук А.В. Системноинженерное мышление. – М.: МФТИ, 2015. – 305 с.

15. IEEE Recommended Practice for Software Requirements Specifications. Revision: 29/Dec/11.

16. Методика составления спецификаций требований к программному обеспечению (IEEE-830-1998). URL: http://www.webisgroup.ru/services/programming/srs/ieee-830-1998/

17. Mich L, Franch M, Novi Inverardi P. Market research for requirements analysis using linguistic tools // Requirements Engineering 9(1), 2004. – P. 40–56.

18. Zhang W., Beaubouef T., and Ye H. Statechart: A Visual Language for Software Requirement Specification // International Journal of Machine Learning and Computing, 2012. – P. 52-61.

19. Specification and description language (SDL). ITU-T Recommendation Z.100 (03/93). – URL: http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf

20. Abrial J.-R. Modelling in Event-B: System and Software Engineering / Cambridge Univ. Press, 2010.

21. Кулямин В. В. Методы верификации программного обеспечения. – М.: Институт Системного Программирования РАН, 2008. URL:http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf

22. Шелехов В.И. Семантика языка предикатного программирования // ЗОНТ-15. — Новосибирск, 2015. — 13с. URL:http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf

23. Alur R., Dill D.L. A theory of timed automata // Theor. Comput. Sci. − 1994. − P. 183-235.

24. Лавров С.С. Программирование. Математические основы, средства, теория. – СПб: БХВ-Петербург, 3001. – 320 с.

25. Meyer B. Introduction to the Theory of Programming Languages. Prentice Hall, 1990. – 448 p.

26. Замулин А. В. Алгебраическая семантика императивного языка программирования // Программирование. 2003, № 6. – С. 1–14.

27. Тумуров Э.Г., Шелехов В.И. Требования к системе управления квадрокоптером // Системная информатика. №5, 2015 — ИСИ СО РАН, Новосибирск. — С. 39-54. URL: http://persons.iis.nsk.su/files/persons/pages/QuadReq.pdf