• опора на гиперэрланговское семейство функций распределения, обладающее хорошими в смысле различных вероятностных метрик аппроксимирующими свойствами;

• простота выбора и параметризации избранных функций распределения;

• возможность получения аналитического решения при исследовании моделей, включающих указанные функции распределения.

Иллюстрацией этих положений служит предлагаемый в докладе метод оценки надежности систем по структурным схемам надежности, когда для каждого элемента схемы априори известны лишь среднее и дисперсия наработки.

В основе метода помимо описанного подхода к аппроксимации функций распределения наработки элементов лежит идея последовательного "укрупнения" структурной схемы надежности системы путем "свертывания" пар последовательных или параллельных элементов в псевдоэлементы с оценкой надежности новообразований. Нетрудно видеть, что в случае предлагаемого подхода к аппроксимации функций распределения наработки необходимо располагать решением задачи оценки всего для 10 сочетаний распределений наработки ES-ES, ES-E, ES-M, ES-HS, ..., HS-HS. Оценки среднего и дисперсии псевдоэлемента для всех этих сочетаний (а также учетом последовательного или параллельного соединения исходных элементов схемы надежности) получены в виде замкнутых аналитических выражений.

В докладе сообщается о программной реализации рассмотренного метода оценки надежности. Речь идет о практически ориентированной системе с развитыми функциями, способной, в частности, работать с произвольными (т.е. не строго последовательно-параллельными) структурными схемами надежности.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ГРАФО-СИМВОЛИЧЕСКОГО ПРОГРАММИРОВАНИЯ

О.П. Солдатова, А.Н. Коварцев, А.В. Баландин

Бурное внедрение средств вычислительной техники во все сферы деятельности человека стимулировало всплеск спроса на разработку новых программных продуктов, что в свою очередь вызвало рост числа фирм, занимающихся произ-

водством и распространением программ. При разработке конкурентоспособных программ немаловажными факторами являются время разработки продукта и надежность его функционирования. Следует отметить, что при всем многообразии автоматизированных систем, облегчающих работу на компьютере пользователей-непрофессионалов, круг средств автоматизации для профессионала-программиста не слишком велик. Мечтой программиста является такая система , которая по некоторому, достаточно неформальному, описанию автоматически генерирует текст синтаксически и семантически корректной программы. Однако, все усилия в этом направлении поначалу сводились к этапу программирования, а именно к разработке новых языков программирования, что позволяло решить ряд проблем, связанных с ограничением функциональных возможностей имеющихся языков. На этом этапе следует выделить бурное развитие специализированных языков, а также непроцедурных языков [1]. Непроцедурные языки это функциональные или логические языки (LISP, PROLOG), имеющие в своей основе вполне определенную математическую модель, что позволяет получить более качественные программные продукты. Диапазон применения подобных языков ограничен, а эффективность разработанного программного обеспечения не слишком велика.

Внимание разработчиков систем автоматизации привлек этап отладки программ. Создано обширное семейство систем автоматизации отладки программ, разработанных на разных языках, как правило, процедурного типа [2,3]. Кроме того, фирмы-разработчики трансляторов языков программирования стали поставлять их в составе интерактивных оболочек, содержащих отладчики программ и имеющих удобный интерфейс пользователя. Подобные системы во многом облегчают работу программиста, однако они бессильны чем-либо помочь при наличии ошибок проектирования программного продукта, то есть ошибок, заложенных при создании проекта или спецификации программы. Автоматизация именно этого этапа разработки программ стала основной идеей современных систем.

Одним из возможных решений этой проблемы явилась разработка объектно-ориентированных языков программирования, которые позволяют во многом унифицировать процесс создания программного обеспечения. К достоинствам этого метода относится то, что более полно реализуется технология структурного программирования, облегчается процесс создания сложных иерархических систем, появляется удобная возможность для создания пользовательских библио-

тек объектов для конкретных областей применения. Но, с другой стороны, растут затраты времени на этапе предварительной обработки информации.

Другой подход выразился в разработке CASE-технологий. Интенсивное развитие интерактивных оболочек программиста ведет к появлению большого количества стандартных библиотек, которые содержат в себе наиболее проверенные и оптимальные коды основных базисных функций для различных предметных областей. Увеличение числа таких библиотек влечет за собой необходимость стандартизации процесса организации межмодульных связей, что делает возможным автоматизацию процесса сборки программных комплексов из заранее разработанных модулей.

Автоматизация построения межмодульных программных связей ведет к сокращению сроков разработки сложных программных продуктов и к улучшению их качества. Авторами разработана технология графического программирования (ГП-технология), в рамках которой идеи объектно-ориентированного и логического программирования сочетаются с CASE-технологией и технологией образного программирования. ГП-технология реализована в виде средства автоматизации программирования GRAF, в котором процесс разработки программ сводится к конструированию ее из набора подготовленных программных модулей.

В системе GRAF, как и в объектно-ориентированном программировании, реализован механизм наследования свойств модулей (объектов) и полиморфизм данных. Система GRAF, подобно CASE-технологии, позволяет автоматизировать процессы проектирования, сопровождения и документирования программного обеспечения. Однако, по мнению авторов, наиболее важным аспектом системы GRAF является возможность хотя бы частичной автоматизации процесса программирования сложных программных продуктов. Разумеется разработчики систем понимают, что полная автоматизация, исключающая необходимость программирования на алгоритмических языках, практически невозможна, поскольку это связано с решением проблемы автоматизации процесса разработки алгоритмов, что несомненно в настоящее время является привилегией человека. Однако, автоматизировать некоторые рутинные операции процесса перевода алгоритма в программу можно и нужно.

Главным фактором повышения эффективности программирования авторы считают придание образности формам двух основных компонентов любой про

граммы-спецификации данных и описанию алгоритма. Причем таким формам, которые бы наиболее полно соответствовали способу образного мышления человека. Основным недостатком существующих технологий программирования авторы считают текстуальный способ описания перечисленных выше компонентов программы на выбранном алгоритмическом языке. Для того, чтобы восстановить алгоритм программы (например, для ее модификации) необходимо "вычитывать" практически весь текст программы, как правило, с многочисленными межмодульными связями. Еще более сложно в современных языках обстоит дело со спецификацией данных, которые бывают "разбросаны" в разных местах макета программы, что делает их совокупное представление чрезвычайно невыразительным. То, что операторы, спецификации данных и описание алгоритма находятся в одном и том же тексте программы, делает ее еще более невыразительной и сложной для восприятия человека. В последних версиях систем программирования, таких как C++, PASCAL в целях придания большей выразительности текста программы различные компоненты языка "подсвечиваются" различными цветами, что улучшает восприятие программы, но не снимает проблемы в целом.

В системе GRAF такие важные этапы процесса программирования как спецификация данных и описание алгоритма "пространственно" отделены друг от друга, а для выражения соответствующих компонентов программы выбраны по возможности образные формы их представления. Спецификациям данных придана табличная форма представления, а алгоритм программы выражается в виде графа, напоминающего блок-схему алгоритма. Тип данных в системе GRAF подразумевает традиционную трактовку типа данных T сигнатуры Σ, как пары: спецификации типа данных сигнатуры Σ и соответствующая ей реализация типа данных [4]. Здесь под сигнатурой понимается пара $\Sigma = <S, \Omega>$, где S-множество имен - основ (имена базовых типов, используемого алгоритмического языка или производных от них типов данных), а Ω есть $(S^* \times S)$ - индексированное семейство множества имен операций, S^* - множество всех цепочек элементов множества S. В таком определении типа данных отражаются два аспекта: пользовательский, когда программист, составляя свою программу, видит тип как определенную спецификацию; и машинный, связанный со способом реализации в ЭВМ обозначенного типа данных. Чтобы создать такой тип данных, необходимо построить спе-

цификацию и сделать в ней необходимую реализацию. Для каждого типа данных имеется свой набор операций, гарантирующий "невыпадение" значения данных описанного типа из представленной спецификации. В системе GRAF понятие типа данного расширено понятием интерпретации данного. Дело в том, что во многих предметных областях чисто "языковое" представление типа данного оказывается недостаточным. Например, в физике в формуле F=a*m все три компоненты формулы F, a и m имеют естественный языковой тип double (т.е.вещественные числа) с определенными на этом типе операциями (+,-,*,/), однако каждое из перечисленных данных имеет самостоятельную (смысловую) интерпретацию: F-сила [Н], a-ускорение [м/сек],m-масса[кг]. Поэтому передача в программу в качестве данного v (тип double) вместо a (тип double), имеющий одинаковый языковой тип должно привести к очевидной ошибке.

В качестве основы введения типа систематизации в системе GRAF используется теория размерности. Определим множество основ типа интерпретации данных Sln как множество образующих типов и их производных. Например, для физических задач то множество будет иметь вид:

Sln = {[м], [кг], [сек], [м/сек] , ... }.

Множество операций в простейшем случае может быть представлено естественными операциями над типами интерпретаций $\Omega m = \{+, -, *, /\}$. При описании типа интерпретации целесообразно ввести понятие множества аксиом Eln, действующих над типами интерпретаций. Множество E состоит из замкнутых Σ формул. В представленном выше примере в качестве одной из формул может быть рассмотрена формула [Н] = [кг]*[м]/[сек*сек]. В этом случае тип интерпретации данного можно определить как $T = < \Sigma ln , Eln >$, где $\Sigma ln = < Sln , \Omega in >$.

Таким образом, под типом данного в системе GRAF понимается пара

$T = < \Sigma , \Sigma ln >$.

В системе GRAF рассматриваются две категории модулей: абстрактные типы модулей (базовые модули) и объекты.

Базовые типы модулей, разработанные на базовом алгоритмическом языке, реализуют отображение A типов данных из области определения на область их значений: A: Ti1, Ti2, ... ,Tin -> Tj1 ,Tj2 ,...,Tjm .

Несмотря на то, что в языках программирования при разработке программ используется понятие формальных параметров процедуры, которые обозначаются посредством их имен, на самом деле в программе важен лишь тип формальных параметров и порядок их использования. В данном аспекте имена формальных параметров не несут никакой смысловой нагрузки и "осмысленные" значения параметры приобретают только после их аппликации к фактическим параметрам

Определим тип базового модуля как список имен основ, образующих область определения и список имен основ, образующих область значений данных модуля, то есть $T_{БМ} = < T_{i1}, T_{i2}, \ldots, T_{in} ; T_{j1}, T_{j2}, \ldots, T_{jm} ; E >$.

Каждый базовый модуль содержит формулу или систему формул E, определяющих возможные интерпретации типов данных.

Например, базовый модуль, вычисляющий формулу F = a*m образует языковая конструкция вида:

```
A (double *F,double a,double m)
{ *F = a*m; }
/* [H] = [м/(сек*сек)]*[кг]*/.
```

Модуль имеет тип

A = <<double,[H]>,<double,[м/(сек*сек)]>;<double,[кг]>>.

Формула над типами интерпретации в представленном примере откомментирована, так как в существующих языках программирования не поддерживаются типы интерпретации данных. В системе GRAF этот недостаток устранен и описание типов интерпретации данных, а также формулы их преобразования размещаются в специальных информационных структурах на этапе регистрации базовых модулей системы GRAF. Точнее, базовый модуль A следует рассматривать в следующей нотации.

```
A(<double, [H]>, <double, [м/(сек*сек)]>,<double, [кг]>)
{<double, [H]> = <double, [м/(сек*сек)]>*<double, [кг]>;}
/*[H] = [м/(сек*сек)]*[кг]*/
```

Объекты системы GRAF порождаются из базовых модулей в результате выполнения операции конкретизации типов данных базового модуля однотипными данными предметной области разрабатываемого программного продукта. В системе GRAF такая операция получила название паспортизации модуля, после

выполнения которой абстрактный базовый модуль преобразуется в объект имеющий конкретный смысл.

Тип объекта по аналогии можно определить как список имен данных, образующих область определения объекта, и список имен данных, образующих области значений объекта.

Спецификация данных в системе GRAF реализована в виде совокупности таблиц. В отдельную таблицу, называемую словарем типов данных, сведены описания всех необходимых атрибутов типов данных, начиная с имени типа, описания родового языкового типа или его редукций, описания типа интерпретации и заканчивая описанием области возможных значений (домена). Вся совокупность данных, используемых в предметной области, описана в словаре данных, который в табличной форме хранит информацию об именах данных, типах данных, начальных значениях данных, области возможных значений. Паспорта объектов (определяющих содержание операции конкретизации), также представлены в табличной форме, которая выражает бинарное отношение между типами данных базового модуля и данными из словаря предметной области.

В предложенной табличной схеме описания спецификаций данных сочетается наглядность представления информации и возможность реализации операций поиска, сортировки и модификации средствами реляционных баз данных. Более того, в предложенной схеме каждое данное или тип данного рассматриваются независимо друг от друга. Упорядочивание и порождение соответствующих языковых конструкций для описания данных реализуется транслятором системы GRAF на основе представленной табличной информации

Для представления алгоритма в системе GRAF используются выразительные возможности формализма теории графов. Каждый новый объект предметной области, описывающий программный модуль, порождается либо путем конкретизации типов данных базовых модулей, либо агрегацией типов объектов более низкого иерархического уровня в ориентированный помеченный граф, который будем называть граф-объектом. В граф-объекте вершины помечены типами объектов (простыми объектами или граф-объектами), а дугами являются предикативные функции, которые разрешают или запрещают переход от одной вершины графа к другой

Предикативные функции (предикаты) представляют собой частный случай понятия объекта, поскольку реализуют функциональное отображение с множества данных предметной области на множество значений { 1,0 } (истина, ложь).

В предложенной схеме описания алгоритма реализовано разделение операторов следования и операторов управления. Аналогом оператора следования в граф-представлении алгоритма являются вершины графа. Управление реализуется за счет ориентированных дуг графа (предикатов). Опыт эксплуатации системы GRAF показал, что вершины графа (объекты) часто выполняют узкоспециализированные функции, в то время как предикаты сохраняют свои универсальные возможности для всей предметной области на протяжении всего времени развития программного продукта. В этом смысле множество предикатов предметной области образуют базу знаний (систему аксиом), которой подчиняется вся совокупность программных продуктов предметной области.

Первая версия системы GRAF разработана на языках Турбо Си и Турбо ПРОЛОГ в среде MS DOS. В качестве языка программирования для разработки базовых модулей выбран язык Си.

ЛИТЕРАТУРА

1. Ин Ц.,Соломон Д. Использование ТУРБО-ПРОЛОГА , пер. с англ. - М.: Мир, 1993, 606с.

2. Липаев В.В. Надежность программного обеспечения АСУ.-М.:Энергоиздат, 1981, 240с.

3. Липаев В.В. Тестирование программ.-М.:Радио и связь,1986, 292 с.

4. Замулин А.В. Системы программирования баз данных и знаний. Новосибирск ,"Наука", 1990, 352 с.