Список использованной литературы

1. Эдвардс Р. Функциональный анализ.Теория и приложения. – М.:Мир,1969.– 1072с.
2. Волевич Л.Р., Гиндикин С.Г. Обобщенные функции и уравнения в свертках. – М.: Наука, 1994. – 336 с.
3. Орлов И.В. Теорема Хана–Банаха в индуктивных шкалах пространств //Доповіді НАН України. 1997. – Вып.9. С.32–36.
4. Orlov I.V. Hahn–Banach theorem in linear and nonlinear scales of the topological vector spaces //Spectral and Evolutionary Problems. – V.7. – 1997. – P.27–31.
5. Orlov I.V. Banach–Grothendieck theorem for the scales of spaces //Spectral and Evolutionary Problems. – V.10. – 2000. P.35–39.
6. Шефер Х. Топологические векторные пространства. – М.: Мир, 1971. – 359 с.
7. Orlov I.V. A termwise differentiation in the inductive scales of the locally convex spaces //Operator Theory: Advances & Appl. – V.118. – 2000. – P.321–333.
8. Orlov I.V. The space of measurable functions with almost everywhere convergence is a nonlinear scale of the locally convex spaces //Spectral and Evolutionary Problems. – V.8. – 1998. – P. 37–42.
9. Орлов И.В. Сходимость почти всюду как сходимость в нелинейной индуктивной шкале локально выпуклых пространств //Ученые записки ТНУ. Математика. Физика. – 2001– (в печати).

Поступила в редколлегию 17.09.2000

УДК 681.3

В. Б. ПОПОВ, канд. физ.-мат. наук, А. С. АНАФИЕВ, асп. Таврический нац. ун-т

ПРИМЕНЕНИЕ МОДАЛЬНЫХ ЛОГИК К СПЕЦИФИКАЦИИ И ВЕРИФИКАЦИИ ПРОГРАММ

Представлены результаты использования методов и алгоритмов временной логики для верификации программ. Предлагается метод временных семантических таблиц для исследования свойств динамических процессов.

Изучение того, что обычно называют ***"верификацией программ"*** (доказательство правильности программ), беспрерывно продолжающееся с момента написания первой программы, смогло привести к некоторым результатам только благодаря привлечению методов математической логики и оснований математики для анализа и синтеза программ. Правильность программы предполагает ее надежность. С тех пор, как программы стали неотъемлемым звеном любого научного, технического или экономического приложения, всегда вставал вопрос о надежности Software.

Самым традиционным методом обеспечения надежности является простое тестирование, которое заключается в следующем: опреде-

ляется множество значений данных, которое называется входом; для некоторых (для всех в большинстве случаев этого сделать нельзя) значений из этого множества проверяем работу программы, и затем оцениваем результаты работы программы на принадлежность множеству ожидаемых результатов программы. Во многих случаях для простых программ удается повысить устойчивость программы по отношению вход/выход. Однако этот метод уже (давно) не удовлетворяет растущим потребностям практики.

Использование языков формальных спецификаций и методов автоматического доказательства теорем и оснований математики в процессе разработки программ позволяет достигнуть качественно нового уровня в повышении надежности программного обеспечения вычислительных систем. Методы математической логики позволяют строить точные абстрактные структурированные описания программ на всех этапах разработки, тестировать их, доказывать соответствие их описаний на различных уровнях спецификации и эксплуатации.

Современные программы представляют собой очень сложные объекты, обладающие динамическими свойствами. Программные технологии представляют программы в виде множества параллельных процессов, взаимодействующих и конкурирующих друг с другом. Программы взаимодействуют и с внешней средой посредством своих процессов, с другими программами и их процессами. Верификация таких программ очень сложна и требует разработки новых средств формальных представлений и доказательств.

Одним из методов доказательств в формальной логике является метод поиск цели - подцели в последовательном исчислении, который называется методом семантических таблиц[1]. Отправным пунктом здесь является проблема выяснения того, является ли данная формула $\mathfrak{R}$ логическим следствием данных формул $Q_1,Q_2,\ldots$. Эта задача рассматривается двумя эквивалентными способами: первый способ предполагает построение подходящего контрпримера для доказательства того, что $\mathfrak{R}$ не является логическим следствием формул $Q_1,Q_2,\ldots$, но если этот подход терпит неудачу, то формула $\mathfrak{R}$ логическим следствием данных формул $Q_1,Q_2,\ldots$; другим способом пытаются получить непосредственный вывод $\mathfrak{R}$ из $Q_1,Q_2,\ldots$, применяя правила вывода некоторой системы натуральной дедукции. В [1] показана эквивалентность этих двух походов. Если семантическая таблица показывает, что метод построения таблицы терпит неудачу, то можно перестроить таблицу так, чтобы получить непосредственный вывод, и, наоборот, всякий вывод в подходящей системе натуральной дедукции дает таблицу, которая показывает, что нельзя построить конрпример. Правила подходящей системы натурального вывода по-

лучаются из правил построения семантической таблицы. В определенных случаях эти правила требуют расщепления таблицы в подтаблицы. Серьезной проблемой может оказаться, что нельзя завершить построение подходящего контрпримера за конечное число шагов. Имеются определенные трудности применения метода на практике. С теоретической точки зрения метод остается актуальным и в настоящее время.

Целью исследований являются общие схемы перевода стратегий последовательного исчисления в области стратегий вывода в модальных логиках, в частности, во временных логиках. Нас интересуют также приложения затронутых логических методов к спецификациям и верификации сложных параллельных программ. Вопросы применения модальных логик рассматривались, например, в работах [3–5,6].

Рассматриваемый в статье подход к формальной спецификации динамических свойств программ связан с применением временной логики. Временные логики используют модальные операторы, примененные к формуле $\mathfrak{R}$. Вводятся модальности “иногда” и “всегда” (“в будущем” и “прошлом”) вместе с их отрицаниями “часто” и “никогда”. Интерпретируются модальные операторы следующим образом:

- “$\mathfrak{R}$ будет истинна всегда” – $G\mathfrak{R}$;
- “$\mathfrak{R}$ будет истинна в следующий момент” – $X\mathfrak{R}$;
- “$\mathfrak{R}$ будет истинна когда-нибудь” – $F\mathfrak{R}$;
- “$H\mathfrak{R}$ – истинно, если $\mathfrak{R}$ всегда было истинным”;
- “$P\mathfrak{R}$ – иногда в прошлом, $P\mathfrak{R}$ – истинно, если $\mathfrak{R}$ иногда оказывалось истинным”; $\mathfrak{R}\,U\,\mathfrak{K}$ – до тех пор пока, $\mathfrak{R}\,U\,\mathfrak{K}$ истинно, если $\mathfrak{R}$ истинно (начиная с текущего момента) до тех пор, пока $\mathfrak{K}$ не станет истинным в некоторый момент в будущем”.

G, F и H, P – двойственные операторы в смысле $F\mathfrak{K} = \neg G \neg \mathfrak{K}$, $P\mathfrak{K} = \neg H \neg \mathfrak{K}$. $F\mathfrak{K}$ эквивалентна формуле $\mathrm{true}\,U\,\mathfrak{K}$. $X\mathfrak{K}$ эквивалентна формуле $\mathrm{false}\,U\,\mathfrak{K}$.

В качестве иллюстрации рассмотрим следующий пример алгоритма реализации критической секции. Алгоритмы взаимоисключения для асинхронных параллельных процессов описаны, например, в классической монографии [2]. Пусть программа p составлена из двух параллельно работающих программных процессов, где $\mathfrak{I}$ – выбор и переход на соответствующую метку:

L0: $\mathfrak{I}$(либо S(L0), либо S(L1));
L1: y1:=True; t:=1; $\mathfrak{I}$(S(L2));
L2: if (¬y2) v (t=2) then $\mathfrak{I}$(S(L3))
else $\mathfrak{I}$(S(L2));

M0: $\mathfrak{I}$(либо S(M0), либо S(M1));
M1: y2:=True; t:=2; $\mathfrak{I}$(S(M2));
M2: if (¬y1) v (t=1) then $\mathfrak{I}$(S(M3))
else $\mathfrak{I}$(S(M2));

L3: y1:=False; $\Im$(S(L0)); M3: y2:=False; $\Im$(S(M0));
с критическими секциями L3 и M3, соответственно. Вход в критическую секцию разрешен одновременно только одной из программ. Обмен информацией между процессами производится с помощью разделяемых булевских переменных y1, y2, t. Формула S(L0) $\supset$ X(S(L0) v S(L1)) описывает состояние процесса. Если состояние процесса определяется меткой L0, то в следующий момент времени процесс перейдет в одно из состояний либо L0, либо L1. Последовательность формул, определяющую состояния процессов запишем в следующем виде:

S(M0)$\supset$X(S(M0) v S(M1)); S(L0)$\supset$X(S(L0) v S(L1));
S(M1) $\supset$ X((y1&t&S(M2)); S(L1) $\supset$ X((y1&t&S(L2));
S(M2)&($\neg$ y1 v t) $\supset$X(S(M3)); S(L2)&($\neg$ y2 v $\neg$t) $\supset$X(S(L3));
S(M2)& $\neg$ ($\neg$ y1 v t) $\supset$X(S(M2)); S(L2)& $\neg$ ($\neg$ y2 v $\neg$t) $\supset$X(S(L2));
S(M3)$\supset$X(y1&S(M0)); S(L3)$\supset$X(y1&S(L0));

Нам требуется показать, что программа обладает свойством исключения критических секций, т. е. что программа не окажется одновременно в двух своих критических секциях L3 и M3, соответственно, т.е. **GØ(S(L3)&S(M3))**.

Доказательство будем проводить методом от противного; предположим, что программа окажется одновременно в двух своих критических секциях – процесс P1 находится на метке L3, а процесс P2 на метке M3, в один и тот же момент времени.

Дадим определение охраняющего условия для метки программы. Условие ***a*** называется ***охраняющим*** для метки ***L***, если, **a**É**S(L)**. Таблица, состоящая из двух столбцов, в левом из которых находятся выражения вида ***S(Li)***, где ***Li*** – метки программы (i пробегает по всем меткам), а в правом – соответствующие им охраняющие условия, называется ***охраняющей***. Для программы p она будет выглядеть так

	Процесс 1 -**P1**-
S(L0)	S(L0) v S(L3)
S(L1)	S(L0)
S(L2)	S(L1) v S(L2)&y2&t
S(L3)	S(L2) & ($\neg$y2 v $\neg$t)
	Процесс 2 -**P2**-
S(M0)	S(M0) v S(M3)
S(M1)	S(M0)
S(M2)	S(M1) v S(M2)&y1&$\neg$t
S(M3)	S(M2) & ($\neg$ y1 v t)

Аналогично строится таблица ***следствий*** – таблица, в которой имеются также два столбца, в левом находятся выражения вида ***S(Li),*** где ***Li*** – метки программы (i пробегает по всем меткам), а в правом – условия вида ***S(L) É X(a),*** где *a* не включает выражение вида ***S(L)*)**:

	Процесс 1 **-P1-**
S(L0)	
S(L1)	X(y1)&X(t)
S(L2)	
S(L3)	X (¬y1)
	Процесс 2 **-P2-**
S(M0)	
S(M1)	X(y2)&X(¬t)
S(M2)	
S(M3)	X(¬y2)

Пустая строка в правом столбце таблицы означает, что программа, находящаяся на метке, соответствующей этой строке, осуществляет лишь переход на другую метку, либо завершает выполнение процесса, но не изменяет другие переменные или условия.

Далее переходим к построению **временной семантической таблицы**. Эта таблица учитывает семантику формул, входящих в нее, аналогично ***семантическим таблицам Бета***[1]. Таблица последовательно строится в каждый момент времени и содержит информацию о прошедших моментах времени, которая используется для построения окончательной таблицы.

True		**False**	
1. **S(L3)**	**S(M3)**		
3. ¬y2 v ¬t 3.1. ¬y2 3.2. ¬t **7.** y1	**4.** ¬y1 v t **4.1.** ¬y1 **4.2.** t **4.** t CLOSE(3) **8.** y2	**3.1.** y2 CLOSE(8) **3.2.** t **3.** t CLOSE(4)	**4.1.**y1 CLOSE(7)
5. S(L2)	**6. S(M2)**		
7.1 X(y1) **X**(t) **7.2** y2 t	**8.1 X**(y2) **X**(¬t) **8.2** y1 ¬t		**8.2** t
7.1 S(L1) **7.2 S(L2)**	**8.1 S(M1)** **8.2 S(M2)**		

Отсутствие разделительной линии между столбцами (между двумя параллельными процессами) в соответствующей строке с четным номером, означает, что условия в первом и втором процессах не должны противоречить друг другу, а в соответствующей нечетной строке означает, что состояние процесса определяется этими метками.

Правила для построения и закрытия временных семантических таблиц:

1. Нечетным строкам соответствует условная метка, показывающая в какой точке в данный момент находится программа. Четным строкам соответствуют охраняющее условие следующей метки (меток) (в таблице выше) и условие следствия предыдущей метки (меток). Каждой строке соответствует момент времени.
2. Правила для ", $, &, Ú, É, Ø такие же, как и правила построения семантических таблиц.
3. Построение таблицы обращено по времени назад, при появлении выражения вида **X**(α), нужно в следующий момент времени (в таблице он находится выше) дописать условие α по правилу **5** (см. ниже).
4. Если встречается выражение вида **S(L),** соответствующее строке **t1**, то выполняют следующее: **а)** в строку **t1-1** вносят выражение из таблицы следствий, соответствующей этой строке и **б)** в строки соответствующие **t1+1** и **t1+2** записывают логические выражение из охраняющей таблицы. В **t1+2** выражения вида S(Li). В **t1**+1 соответствующие выражения для **y1, y2** и **t.**
5. Если встречается в таблице **X**(α), то: **а)** если α в таблице следствий находится в той части таблицы, которая относится только к одному процессу, то α записывается во все периоды времени этого процесса, следующие за ним, т.е. стоящие в таблице выше и только в четные строки, до момента, когда встречаются **X**α или **X**¬α; **б)** если условие α встречается как в первом, так и во втором процессе и противоположное по значению, т.е. **X**α и **X**¬α (подразумевается, что в первом процессе только имеется условия вида **X**α, а во втором **X**¬α или наоборот), то ничего не пишется; **в)** если α встречается не в единственном экземпляре, то предпочтение отдается вышестоящим во *временной семантической таблице*, а дальше по правилам **5. а)** и **5. б)**.
6. Если в текущий момент времени одному процессу соответствует условие вида **X**α, а другому процессу – а, то это приводит к противоречию и соответствующие строки закрываются.
7. Процесс построения таблицы прекращается, если закрылась какая-либо формула, либо достигнуто условие конца.

Пошаговый процесс построения временной семантической таблицы.

Шаг № 1

True	**False**
1.S(L3) **S(M3)**	

Шаг № 2

True		**False**	
1.S(L3) **S(M3)**			
3. Øy2 v Øt **3.1. Øy2** **3.2. Øt**		**3.1. y2** **3.2. t**	
5. S(L2)			

Шаг № 3

True		**False**	
1.S(L3) **S(M3)**			
3. ¬y2 v ¬t **3.1.** ¬y2 **3.2.** ¬t	**4. Øy1 v t** **4.1. Øy1** **4.2. t**	**3.1.** y2 **3.2.** t	**4.1. y1**
5. S(L2)	**6. S(M2)**		

Шаг № 4

True		**False**	
1.S(L3) **S(M3)**			
3. ¬y2 v ¬t **3.1.** ¬y2 **3.2.** ¬t	**4.** ¬y1 v t **4.1.** ¬y1 **4.2.** t	**3.1.** y2 **3.2.** t	**4.1.** y1
5. S(L2)	**6. S(M2)**		
7.1 S(L1) **7.2 S(L2)**			

Шаг № 5

True		**False**	
1.S(L3)	**S(M3)**		
3. ¬y2 v ¬t **3.1.** ¬y2 **3.2.** ¬t	**4.** ¬y1 v t **4.1.** ¬y1 **4.2.** t	**3.1.** y2 **3.2.** t	**4.1.** y1
5. S(L2)	**6. S(M2)**		
7.1 S(L1) **7.2 S(L2)**	**8.1 S(M1)** **8.2 S(M2)**		

Шаг № 6

True		**False**	
1.S(L3)	**S(M3)**		
3. ¬y2 v ¬t **3.1.** ¬y2 **3.2.** ¬t **7. y1**	**4.** ¬y1 v t **4.1.** ¬y1 **4.2.** t **4. t**	**3.1. y2** **3.2. t CLOSE(4)** **3. y2**	**4.1. y1 CLOSE(7)**
5. S(L2)	**6. S(M2)**		
7.1 X(y1) **X(t)** **7.2 y2** **t**			
7.1 S(L1) **7.2 S(L2)**	**8.1 S(M1)** **8.2 S(M2)**		

Шаг № 7

True		**False**	
1.S(L3)	**S(M3)**		
3. ¬y2 v ¬t **3.1.** ¬y2 **3.2.** ¬t **7.** y1	**4.** ¬y1 v t **4.1.** ¬y1 **4.2.** t **4.** t **8. y2**	**3.1. y2** **3.2.** t CLOSE(4) **3. y2 CLOSE(8)**	**4.1.** y1 CLOSE(7)
5. S(L2)	**6. S(M2)**		
7.1 X(y1) **X**(t) **7.2** y2 t	**8.1 X(y2)** **X(Øt)** **8.2 y1** **Øt**		**8.2 t**

7.1 S(L1) **7.2 S(L2)**	**8.1 S(M1)** **8.2 S(M2)**		

Шаг № 8

True		**False**	
1.S(L3)	**S(M3)**		
3. ¬y2 v ¬t **3.1.** ¬y2 **3.2.** ¬t **7.** y1	**4.** ¬y1 v t **4.1.** ¬y1 **4.2.** t **4.** t CLOSE(3) **8.** y2	**3.1.** y2 CLOSE(8) **3.** t CLOSE(4)	**4.1.** y1 CLOSE(7)
5. S(L2)	**6. S(M2)**		
7.1 X(y1) **X**(t) **7.2** y2 t	**8.1 X**(y2) **X**(¬t) **8.2** y1 ¬t		**8.2** t
7.1 S(L1) **7.2 S(L2)**	**8.1 S(M1)** **8.2 S(M2)**		

Результатом исследований является также компьютерный эксперимент. Программное обеспечение было создано в системах программирования Delphi 5.0 и Builder 3.0 с использованием библиотеки VCL (IBM PC – Intel). Реализованы алгоритмы построения для параллельных программ таблиц (“охраняющей” и “следствий”), а также алгоритм, последовательно строящий семантическую временную таблицу.

Список использованной литературы

1. Э.В. Бет. Метод семантических таблиц.//Математическая теория логического вывода. –М.: Наука, 1967.– С. 191–199.
2. Г. Дейтел. Введение в операционные системы. –М.:Мир, 1987.– 359 с.
3. A. Pnueli. The temporal semantic of concurrent programs // Theoretical Computer Science –1981.– V 13. –P. 44–60.
4. Manna Z., Wolper P. Synthesis of communicating processes from temporal logic specifications // ACM Trans. Progr. Lang. Systems.– 1984.– V.6, N1.– P. 69–92.
5. Manna Z., A. Pnueli. Proving precedence properties: the temporal way. Proc. Intern. Coll. On Automata, Languages, Programming // LNCS.– 1983.– V.154– P.490–512.
6. Verification de Logiciels. Techniques et outils du model-checking. Ouvrage collectiv – coordination Philippe Schnoebelen. Vuibert Informatique. Paris, 1999.– 197 p.

Поступила в редколлегию 20.05.2000