$$M(f) \;=\; M(a;\; (s_1,\; \ldots s_k)) = \sum_{i=1}^{k} M(a; s_i),$$

$$M(a;\; s_i) \;=\; \begin{cases} \alpha(n(a) \;+\; n(s_i)),\; \text{если } s_i \text{ — выражение,} \\ M(a,\; a_i;\; (s_{i1},\; \ldots,\; s_{il})) \;= \sum\limits_{j=1}^{l} M(a,\; a_i;\; s_{ij}), \\ \text{если } s_i = (a_i;\; (s_{i1},\; \ldots,\; s_{il})) \text{ — НУО.} \end{cases}$$

Введённая мера СИ была применена к существующим программным системам — к версиям 0.01 и 2.6.24 ядра операционной системы Linux, являющимся соответственно самой первой и одной из самых последних версий ядра Linux. В качестве главной подсистемы была выбрана подсистема работы с файлами с помощью системных вызовов sys_open(), sys_mkdir(), sys_rmdir(), sys_link(), sys_unlink(), а в качестве подчинённой — реализация дискреционной политики безопасности.

Для Linux 0.01 получено значение СИ, равное $\alpha(1) \;+\; 5\alpha(2) + 5\alpha(3)$, а в случае Linux 2.6.24 СИ равна $3\alpha(1) \;+\; 2\alpha(2) \;+\; 3\alpha(3)$. Как было отмечено, функция α — монотонно неубывающая. Если принять $\alpha(n)$ за константу, то оценка для Linux 0.01 станет равной 26α, а для Linux 2.6.24 — равной 16α. Это уменьшение значения СИ является следствием уменьшения как общего количества идентификаторов подчинённой подсистемы в главной, так и тех из них, которые связаны (входят в один оператор).

ЛИТЕРАТУРА

1. *Landwehr C. E.* Formal models for computer security // ACM Comput. Surv. 1981. September. V. 13. No. 3. P. 247–278.
2. Security-Enhanced Linux / USA National Security Agency., Электрон. дан., 2009. Режим доступа: `http://www.nsa.gov/research/selinux/index.shtml`, свободный.
3. Grand Research Challenges in Information Systems / Computing Research Association., Электрон. дан., 2002. Режим доступа: `http://www.cra.org/reports/gc.systems.pdf`, свободный.
4. *Mills E. E.* Metrics in the software engineering curriculum // Ann. Softw. Eng. 1999. V. 6. No. 1–4. P. 181–200.

УДК 681.3.06:62-507

ТРАНСЛЯЦИЯ ОПИСАНИЙ АВТОМАТОВ, ПРЕДСТАВЛЕННЫХ В ФОРМАТЕ MICROSOFT VISIO, В ИСХОДНЫЙ КОД НА ЯЗЫКЕ C

Л. В. Столяров, И. Р. Дединский, А. А. Шалыто

Существует подход к созданию программ для систем логического управления с использованием конечных автоматов, названный в [1] автоматным программированием (АП). В последнее время этот подход быстро развивается [1, 2]. С его помощью удобно реализуются различные системы логического и событийного управления [3]. АП основывается на использовании конечных автоматов (КА) для описания логики работы программ. КА — это конечное множество состояний, в которых он может находиться, и переходов между этими состояниями. Подход основан на создании графов переходов КА (автоматных схем) [4] с последующей их трансляцией в исходный код программы. Автоматная схема является графическим отражением алгоритма в терминах состояний и переходов. Поэтому при разработке программ на основе АП автоматная схема

документирует алгоритм, что чрезвычайно важно для дальнейшего сопровождения программы. Главная особенность АП состоит в том, что программы, построенные на его основе, в отличие от программ, написанных традиционным путем, могут быть достаточно просто формально верифицированы на основе метода Model Checking [5]. Визуализация логики программы позволяет ускорить ее разработку и отладку, а также отделить алгоритмическую (логическую, управляющую) часть от остального кода. В АП, как и в теории автоматов, используются три понятия: входное воздействие, состояние и выходное воздействие. При этом состояния выделяются на этапе проектирования в явном виде. В АП, как и в других подходах к программированию, важна организация взаимодействия средства трансляции с разработчиком автоматной программы и интеграция в системы автоматической сборки (настройка режимов трансляции, способ запуска, сообщения об ошибках трансляции).

Граф переходов КА состоит из нескольких основных элементов: состояния, групповые состояния, описание входных и выходных воздействий, дуги переходов. *Состояние* имеет имя, список других автоматов для запуска, а также список выходных воздействий для этого состояния. *Событие* — один из видов входных воздействий, которые обеспечивают вызов автомата. Оно чаще всего используется в условиях переходов. *Дуга перехода* соединяет два состояния или группы состояний. Дуга помечается *условием*, представленным в виде произвольного логического выражения, в котором, в частности, используются *входные переменные*, а также списком *выходных воздействий*, которые выполняются при переходе. *Входная переменная* — это входное воздействие, представленное переменной, значение которой может быть использовано в условии перехода.

В настоящее время известны инструментальные средства для поддержки АП, такие, как Visio2Switch [6], MetaAuto [7], UniMod [8]. В этих средствах есть как свои достоинства, так и недостатки — диагностика ошибок в графе переходов недостаточна, надежность и качество сгенерированного кода низки (MetaAuto), интеграция в автоматизированные процессы сборки приложений невозможна или чрезвычайно сложна (Visio2Switch).

Целью работы являлось создание инструментального средства для трансляции описаний графов переходов автоматов, представленных в формате MS Visio, в исходный код на языке C и другие формы представления данных (язык XML). Основная задача работы — устранение недостатков, перечисленных выше. В качестве языка реализации выбран язык Microsoft C# для платформы .NET, так как он обеспечивает хорошую поддержку технологии COM, используемой при взаимодействии с MS Visio. Интерфейсы COM-объекта MS Visio позволяют открывать любые файлы формата MS Visio и получать список всех элементов автоматной схемы, описанной в файле. Эта схема состоит из описаний свойств автомата, его элементов и графа переходов. Специализированных шаблонов для описания схемы в стандартной библиотеке MS Visio нет, поэтому одной из задач работы являлась разработка файла с такими шаблонами.

В процессе работы транслятора схема, содержащаяся в исходном документе MS Visio, преобразуется в исходный код для дальнейшего использования в автоматном проекте, а также сохраняется в формате XML. Первым этапом процесса трансляции является чтение данных из исходного документа MS Visio. Для реализации этого этапа был применен подход, основанный на непосредственном взаимодействии с редактором MS Visio при помощи COM-технологии. Следующая стадия — генерация выходных данных (программного кода на языке C). Она происходит с использованием файлов-шаблонов кода из выбираемой пользователем директории. В этих файлах содержатся

специальные метки, которые в процессе трансляции заменяются на соответствующие участки сгенерированного кода. Такой подход позволяет любому пользователю настроить стиль генерируемых файлов на свой вкус, не прикладывая особых усилий.

В работе также было уделено внимание пользовательскому интерфейсу транслятора, так как он является инструментом, который обязан быть удобным. Программа может быть запущена в двух режимах. В первом из них трансляция производится неинтерактивно, и вся информация считывается из командной строки. Этот режим удобен при использовании в средствах автоматической сборки. Во втором режиме интерфейсом программы является диалоговое окно, в котором пользователь может задавать параметры, используя элементы графического интерфейса .NET. Если программа запущена без аргументов командной строки, то она открывается в режиме графического интерфейса.

При разработке проекта также было уделено внимание системе обработки ошибок проектирования автоматной схемы, обнаруженных транслятором в графе переходов. Ошибкой считается некий найденный дефект, без исправления которого разработчиком нельзя продолжить формальную трансляцию. По остальным дефектам выводятся предупреждения, так как они скорее всего являются следствиями высокоуровневых логических ошибок в исходной схеме. Во время трансляции все ошибки и предупреждения выводятся в консоль транслятора.

В результате работы создано инструментальное средство для трансляции графов переходов автоматов, представленных в формате Microsoft Visio, в исходный код на языке C. Средство обеспечивает сохранение исходных данных в XML-файл, диагностику ошибок в исходных данных, интеграцию в автоматические процессы сборки приложений (например, make-файлы, MS Visual Studio build system).

ЛИТЕРАТУРА

1. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
2. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации. СПб.: Наука, 2000. 780 с.
3. *Шалыто А. А., Туккель Н. И.* Реализация автоматов при программировании событийных систем // Программист. 2004. № 2. С. 74–80.
4. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009. 176 с.
5. *Гуров В. С., Шалыто А. А., Яминов Б. Р.* Технология верификации автоматных программ без их трансформации во входной язык верификатора // Материалы Междунар. науч.-технич. конф. «Многопроцессорные вычислительные и управляющие системы (МВУС-2007)». Таганрог: НИИ МВС, 2007. Т. 1. С. 198–203.
6. `http://is.ifmo.ru/progeny/visio2switch` — *Головешин А.* Конвертор Visio2Switch. 2002.
7. `http://is.ifmo.ru/projects/metaauto` — *Канжелев С. Ю., Шалыто А. А.* Преобразование графов переходов, представленных в формате MS Visio, в исходные коды программ для различных языков программирования (инструментальное средство MetaAuto). 2005.
8. *Гуров В. С.* Технология проектирования и разработки объектно-ориентированных программ с явным выделением состояний (метод, инструментальное средство, верификация): Дис. ... канд. техн. наук. СПбГУ ИТМО, 2008.