Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

И. С. Ануреев

ПРИМЕНЕНИЕ СИСТЕМ ПЕРЕПИСЫВАНИЯ ФОРМУЛ В АВТОМАТИЧЕСКОЙ ВЕРИФИКАЦИИ ПРОГРАММ

Препринт
55

Новосибирск 1998

Предложена методология применения систем переписывания формул в автоматической верификации программ. Показана возможность автоматической верификации ряда программ редактирования текстов, сортировки массивов и сортировки файлов. Описан модуль системы верификации СПЕКТР, комбинирующий упрощающие процедуры, основанные на системах переписывания формул с другими разрешающими и упрощающими процедурами.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

I. S. Anureev

FORMULA REWRITING SYSTEMS APPLICATION TO AUTOMATED PROGRAM VERIFICATION

Preprint
55

Novosibirsk 1998

The methodology of applying formula rewriting systems to automated program verification is suggested. The possibility of automated verification of a number of programs of text editing, array sorting and file sorting is demonstrated. The verification system SPECTRUM module combining simplification procedures based on formula rewriting systems with other decision and simplification procedures is described.

ВВЕДЕНИЕ

В создании надежного программного обеспечения — важнейшей задачи технологии программирования — значительное место занимает классическая верификация программ [1, 10, 11, 14, 19, 27, 29, 39], основы которой были заложены в работах Флойда и Хоара. При этом подходе доказательство корректности программы (ее соответствие спецификации) сводится к проверке истинности ряда формул на языке спецификаций — условий корректности. В реальных программах условия корректности могут быть формулами значительный длины, поэтому процесс доказательства их истинности ввиду трудоемкости целесообразно автоматизировать.

В настоящее время существенный прогресс в этой области достигнут для верификации оборудования, использующей достаточно бедные языки спецификаций и специализированные методы доказательства условий корректности. Даже для таких формализованных классов программ, как программы сортировки, существующих методов автоматического доказательства оказывается недостаточно, чтобы полностью автоматизировать этап доказательства условий корректности. Известные примеры автоматической верификации отдельных программ, проведенные в автоматизированных системах верификации [13, 26], как правило, используют комбинацию разнородных средств доказательства и не позволяют перенести полученный алгоритм доказательства на другие программы этого класса. Поэтому разработка методов доказательства, ориентированных на автоматическую верификацию программ, является актуальной задачей теории верификации.

В данной работе предложена методология использования в качестве одного из таких методов недавно разработанного формализма систем переписывания формул (СПФ) [5, 7]. СПФ соединяют в себе элементы систем переписывания термов и сужения, в результате чего получается мощное средство построения упрощающих процедур, сохраняющих выполнимость (выполнимая формула переписывается в выполнимую, невыполнимая — в невыполнимую). Кроме того, работа содержит описание модуля переписывания формул, основанного на СПФ и реализованного в рамках системы верификации СПЕКТР, и результаты экспериментов в области автоматической верификации программ.

В п.1 изложены основы классической теории верификации программ и дано краткое введение в систему верификации СПЕКТР, на базе которой проводился эксперимент по применению СПФ в верификации про-

грамм. Знание теории верификации позволяет определить роль, которую методы и техники автоматического доказательства играют в автоматической верификации программ, а сведения по системе СПЕКТР необходимы для понимания связи разработанного модуля переписывания формул, основанного на СПФ, с другими частями этой системы. Кроме того, в этом же пункте описан модельный язык аннотированных программ, на котором изложены примеры верифицируемых программ.

В п.2 приведены разрешающие процедуры для арифметики Пресбургера и ее расширения на неинтерпретированные функциональные символы, для теорий равенства, частичного порядка и равенства мультимножеств, причем разрешающая процедура для теории равенства мультимножеств приводится впервые. Использование только этих процедур совместно с системами переписывания формул позволяет автоматически верифицировать ряд программ редактирования текстов, сортировки массивов и файлов.

В п.3 даются основы теории СПФ, излагаются некоторые методы доказательства завершимости СПФ и сформулирован базирующийся на СПФ метод сведения массивов и файлов к кортежам и целым числам.

Следующий пункт посвящен реализации модуля переписывания формул. Описана общая логическая структура модуля, предложен простой язык комбинирования разрешающих процедур (тактик) и изложены средства построения, хранения и использования тактик. В п.4.2 описана функция (тактикал), позволяющая по СПФ строить соответствующую тактику.

В п.5 на ряде примеров верификации программ изложена методология применения СПФ в автоматической верификации. В качестве примеров верифицируемых программ рассмотрены программы копирования файла, линейного поиска, сортировки массивов простой вставкой, быстрой сортировки массивов и сортировки файлов естественным слиянием.

В заключении приведены основные результаты и намечены перспективы дальнейшего совершенствования модуля переписывания формул.

1. ПРЕДВАРИТЕЛЬНЫЕ СВЕДЕНИЯ О ВЕРИФИКАЦИИ

В этом пункте рассмотрены ключевые моменты, связанные с верификацией программ с помощью метода Флойда — Хоара, система верификации СПЕКТР, основанная на этом методе, и модельный язык аннотированных программ. В п.1.1 излагается общая схема проведения

верификации на основе метода Флойда — Хоара, в п.1.2 дается краткое введение в систему верификации СПЕКТР, п.3 посвящен модельному языку аннотированных программ.

1.1. Метод Хоара

Задача верификации состоит в математически точном доказательстве соответствия программы спецификациям. Опишем схему проведения такого доказательства, основанную на широко применяемом в практической верификации методе индуктивных утверждений Флойда — Хоара [3, 9, 21, 23], детальное изложение которого может быть найдено в [14, 28].

На первом этапе выбирается и фиксируется язык спецификаций [41, 22, 2], предназначенный для записи входного и выходного условий программы и инвариантов циклов. Выбор языка спецификаций определяется классом программ (проблемной областью), который нужно верифицировать. В случае автоматической верификации дополнительным требованием к языку спецификаций является аксиоматизация понятий проблемной области.

На втором этапе проводится аннотирование программы, при котором программе приписывается входное и выходное условие, а также некоторые промежуточные утверждения, называемые инвариантами циклов. Как правило, процесс аннотирования программы осуществляется вручную. Программа называется частично корректной относительно заданных спецификаций, если, начав свою работу в состоянии, удовлетворяющем входному условию, она завершает ее в состоянии, удовлетворяющем выходному условию, в том случае, если программа вообще завершает свою работу для данного входного состояния. Программа называется (тотально) корректной, если она частично корректна и завершается для любого входного состояния, удовлетворяющего входному условию.

На третьем этапе утверждение о корректности программы сводится к некоторому набору утверждений об истинности логических формул (условий корректности) на языке спецификаций. Такая редукция осуществляется автоматически с помощью специальных алгоритмов генерации условий корректности по аннотированным программам [28, 32].

На заключительном этапе осуществляется проверка истинности сгенерированных условий корректности с помощью тех или иных методов автоматического доказательства. В случае истинности всех условий

корректности исходная программа корректна. Если же некоторое условие оказывается ложным, то либо программа некорректна, либо неверно написаны инварианты циклов. Особенностью задачи верификации является то, что для ложного условия корректности желательно выдавать в качестве результата контрпример, на основе которого осуществляется модификация программы или ее аннотаций. При доказательстве некоторого условия корректности может оказаться, что используемые техники доказательства не могут ни доказать, ни опровергнуть это условие корректности. В этом случае в модификации нуждаются сами техники доказательства и/или язык спецификаций, так как проведение эффективного доказательства требует компромисса между выразительностью языка спецификаций и существующими техниками доказательства.

1.2. Система верификации СПЕКТР

Основная мотивация для разработки новой техники автоматического доказательства, основанной на системах переписывания формул, была связана с неприменимостью традиционных методов и техник для проведения автоматического доказательства условий корректности, которые возникали при верификации программ в проблемно-ориентированной системе верификации СПЕКТР [13, 15, 36, 37]. Предложенная техника позволяет проводить автоматическое доказательство таких условий корректности, и одним из применений модуля переписывания формул, основанного на этой технике, является использование его в качестве одного из компонентов блока доказательства системы СПЕКТР. Опишем кратко структуру этой системы (более подробно см. в [16, 36]).

Система состоит из трех основных частей: генератора условий корректности, конструктора аксиом-функций и доказателя условий корректности.

Генератор условий корректности порождает множество условий корректности по данной аннотированной программе. В качестве входного языка используется Паскаль, дополненный аннотациями.

Конструктор преобразует аксиомы, записанные в естественной нотации, в программные функции на языке реализации. Кроме самих аксиом передается также информация о том, какие стратегии следует использовать при применении этих аксиом. Файл с функциями компилируется, и полученный объектный модуль участвует в сборке доказателя. Эти модули называются модулями аксиом-функций, а сами функции

— аксиомами-функциями. Использование программно-реализованных аксиом-функций вместо аксиом позволяет увеличить эффективность доказательства.

Доказатель пытается установить истинность условий корректности. Он состоит из четырех основных компонентов: упростителя, управляющего модуля, нормализатора и модулей аксиом-функций. Первые три компонента составляют постоянную часть доказателя, общую для всех проблемных областей. Модули аксиом-функций создаются конструктором по наборам аксиом для каждой проблемной области.

На вход доказателю подается условие корректности. Упроститель преобразует условие к совокупности формул вида

$$p_1 \wedge ... \wedge p_n \rightarrow p_0,$$

где p_i — литералы (атомарные формулы или их отрицания), проводя при этом некоторые логические упрощения (проверку на отсутствие противоречивых посылок, проверку совпадения посылки и заключения и т. п.).

Недоказанные упростителем формулы передаются управляющему модулю, который определяет, какая аксиома-функция и стратегия должна быть применена, и передает им формулу для дальнейшего доказательства, после проведения которого полученный результат возвращается упростителю.

Функцией нормализатора является нормализация определенных классов формул, например арифметических выражений.

Доказатель включает специальный модуль, который по недоказанному условию корректности порождает путь в аннотированной программе, соответствующий этому условию.

1.3. Модельный язык аннотированных программ

Для того чтобы проиллюстрировать применение техники переписывания формул, основанной на СПФ, на примерах верификации программ, требуется ввести язык, на котором записываются аннотированные программы. В качестве такого языка возьмем структурированный паскалеподобный язык. Синтаксическая категория `<анн.прогр>` определяется следующими правилами:

```
<анн.прогр> := <функция>
            := <функция> ; <программа>
```

```
<функция>   := {<вход.усл.>} function <имя функции>:
                              (<параметры>) -> (<параметры>)
                                       <тело> {<выход.усл.>}
<параметры> := <переменная>,
            := <переменная>, <список_параметров>
<тело>      := if <формула> then <тело> else <тело> fi;
            := {<инвариант>} while <формула> do <тело> od;
            := {<инвариант>} for <переч.перем.> := <терм>
                                                to <терм>
                              do <тело> od;
            := repeat <тело> until <формула>; {<инвариант>}
            := <переменная> := <терм>;    (*присваивание*)
            := <лог.перем.> := <формула>; (*присваивание*)
```

Элементами синтаксической категории `<имя функции>` являются произвольные буквенно-цифровые идентификаторы. Элементами категорий `<переменная>`, `<лог.перем.>`, `<терм>`, `<формула>` являются переменные, логические переменные, термы, формулы языка спецификации соответственно. Элементы синтаксической категории `<переч.перем.>` — переменные перечислимых типов. Элементы синтаксических категорий `<вход.усл.>`, `<выход.усл.>`, `<инвариант>` — формулы. Все сложные структуры данных, такие как массивы, файлы и т. п., описываются как типы языка спецификации.

2. БАЗОВЫЕ РАЗРЕШАЮЩИЕ ПРОЦЕДУРЫ

Системы переписывания формул предназначены прежде всего для использования в качестве упрощающих процедур, сохраняющих выполнимость. Тогда, чтобы получить на их основе процедуры проверки выполнимости формул некоторых теорий, требуется комбинирование СПФ с разрешающими процедурами для теорий, формулы которых являются нормальными формами относительно применяемых СПФ. Назовем такие разрешающие процедуры базовыми. В этом разделе изложены разрешающие процедуры для бесквантворных фрагментов четырех теорий: арифметики Пресбургера и ее расширений, теории равенства, теории частичного порядка и теории равенства на мультимножествах. Выбор этих разрешающих процедур в качестве базовых обусловлен тем, что они реализованы в блоке доказательства системы верификации СПЕКТР и в комбинации с упрощающими процедурами, основанными на системах переписывания формул, их достаточно для

автоматической верификации ряда программ редактирования текстов и сортировки последовательных файлов и массивов. Довольно полный перечень разрешимых теорий можно найти в обзоре [17]. Отдельные разрешающие процедуры также изложены в работах [12, 18, 24, 25, 31, 33–35, 38, 40, 42].

2.1. Арифметика Пресбургера и ее расширения

Арифметика Пресбургера над целыми числами является фрагментом целочисленной арифметики, из которой удалена операция умножения. Сигнатура арифметики Пресбургера включает обычные арифметические операции: меньше $<$, меньше или равно $\leq$, равно $=$, операции сложения $+$ и вычитания $-$, а также встроенные целочисленные константы $c \in \mathcal{N}$ и семейство операций умножения на целочисленные константы $c*$, где $c \in \mathcal{N}$.

2.1.1. Метод Шостака

Опишем метод Блэдсоэ — Шостака (или sup-inf-метод) разрешения бескванторных формул арифметики Пресбургера. Этот метод определяет процедуру проверки выполнимости конъюнкций литералов, к которым сводима проверка выполнимости формул бескванторной арифметики Пресбургера.

На первом этапе из формул с помощью замен

$$\begin{array}{lll} x = y & \text{на} & x \leq y \wedge y \leq x, \\ \neg(x = y) & \text{на} & x < y \vee y < x, \\ x < y & \text{на} & x + 1 \leq y, \\ \neg(x < y) & \text{на} & y \leq x, \\ \neg(x \leq y) & \text{на} & y + 1 \leq x \end{array}$$

исключаются операции $=$, $<$ и термы вида $t \leq s$.

На втором этапе после нормализации арифметических выражений формулы принимают вид

$$t_1 \leq s_1 \wedge ... \wedge t_n \leq s_n,$$

где термы s_i, t_j, в свою очередь, имеют вид

$$c_0 + c_1 * x_1 + ... + c_n * x_n$$

нормализованных арифметических выражений. Назовем эти формулы задачами линейного целочисленного программирования.

Пусть $\mathcal{L}$ — множество всех таких задач.

Главную роль в алгоритме играют две рекурсивные функции sup и inf вида $\mathcal{V} \times \mathcal{L} \to \mathcal{Z} \cup \{-\infty, \infty\}$. Пусть $s \in \mathcal{L}$. Тогда для каждой переменной $x \in \mathcal{V}ar(s)$ функция $sup(x, s)$ вычисляет максимальное значение, а функция $inf(x, s)$ — минимальное значение, которое принимает переменная x на множестве решений формулы s. Если максимального значения x не существует, то $sup(x, s) = \infty$, а если минимального значения x не существует, то $inf(x, s) = -\infty$. Алгоритмы вычисления функций sup и inf могут быть найдены в работе Шостака [40].

Идея алгоритма заключается в последовательном исключении переменных. Назовем формулу s точкой возврата, если она ложна и не содержит переменных или если для некоторой переменной $x \in \mathcal{V}ar(s)$ интервал $I = [inf(x, s), sup(x, s)]$ не содержит целых чисел. Назовем формулу s примером, если она истинна и не содержит переменных. Шаг алгоритма состоит в выборе очередной переменной x из формулы s, которая не является ни точкой возврата, ни примером, вычислении интервала $I = [inf(x, s), sup(x, s)]$, выборе произвольного целого числа x_0 из этого интервала и подстановки его в формулу s. Пусть s' — полученная формула. Если s' — точка возврата, то повторяем предыдущий шаг при других значениях переменной. Если s' — пример, то исходная формула выполнима. В противном случае применяем шаг для формулы s'. Если всевозможные значения переменых перебраны, то исходная формула невыполнима.

Таким образом, sup-inf-метод способен установить выполнимость любой выполнимой в целых числах формулы из класса $\mathcal{L}$, но на некоторых невыполнимых формулах он может зацикливаться. Несмотря на неполноту, этот метод в практических задачах предпочтительнее полных методов, например метода Купера.

2.1.2. Арифметика Пресбургера с неинтерпретированными функциональными символами

В ряде работ предложены различные расширения арифметики Пресбургера, одно из которых — арифметика Пресбургера с неинтерпретированными функциональными символами. В этом расширении к арифметики Пресбургера добавлены неинтерпретированные функциональные символы всевозможных местностей, удовлетворяющие лишь аксиомам

подстановочности равенства:

$$x = y \Rightarrow f(..., x...) = f(..., y, ...).$$

Идея алгоритма проверки выполнимости для формул арифметики Пресбургера с неинтерпретированными функциональными символами заключается в следующем. Вначале из аргументов неинтерпретированных функциональных символов устраняются символы операций $+$, c и $c*$ посредством добавления новых переменных. Например, формула

$$g(1) = f(y + z),$$

где f, g — неинтерпретированные функциональные символы, преобразуется в формулу

$$u = 1 \wedge w = y + z \wedge g(u) = f(w),$$

где u, w — новые переменные.

Затем применяется разрешающая процедура для арифметики Пресбургера, где каждый максимальный терм с неинтерпретированным символом в корне считается новой переменной с "причудливым"именем. Если формула невыполнима как формула арифметики Пресбургера, то она невыполнима и как формула с неинтерпретированными функциональными символами. Если же находится некоторый пример для формулы, то проверяется, удовлетворяют ли неинтерпретированные функциональные символы, входящие в этот пример, аксиомам подстановочности равенства. Если удовлетворяют, то найденный пример — пример и для формулы с неинтерпретированными функциональными символами. В противном случае требуется искать другой пример. Если все примеры перебраны и ни один из них не удовлетворяет аксиомам подстановочности равенства, то исходная формула невыполнима.

2.2. Теории равенства и частичного порядка

Рассмотрим сначала разрешающую процедуру для равенства в теориях, сигнатура которых, кроме равенства, включает только набор констант

$$c_1, ..., c_k,$$

определяемых аксиомами: $\neg(c_i = c_j)$ при $i \neq j$ и $1 \leq i, j \leq k$.

Пусть $x \in \mathcal{V}$, t — терм теории. Процедура проверки выполнимости формул этой теории заключается в применении до тех пор, пока возможно применение следующих двух замен:

$$A \wedge x = t \wedge B \quad \text{на} \quad (A \wedge B)(x \to t),$$
$$A \wedge t = x \wedge B \quad \text{на} \quad (A \wedge B)(x \to t).$$

В результате конъюнкция атомарных формул преобразуется в формулу, которая не содержит переменных и проверяется непосредственным образом. Если запоминать замены вида $x \to t$, возникающие в приведенных выше правилах, то рассмотренная процедура позволяет не только проверить формулу на выполнимость, но и дает пример в случае выполнимой формулы.

Заметим, что рассмотренная теория равенства является частным случаем теории равенства с неинтерпретированными функциональными символами.

Сигнатура теории частичного порядка включает отношение (нестрогого) частичного порядка $\leq$ и равенство $=$, определяемое через $\leq$ с помощью аксиомы

$$x \leq y \wedge y \leq x \Leftrightarrow x = y.$$

Процедура проверки выполнимости формул этой теории основана на построение транзитивного и рефлексивного замыкания для частичного порядка $\leq$ после предварительного устранения равенства согласно описанной выше аксиоме. Пусть формула A состоит из конъюнкции элементов множества формул $U \cup U'$, где U — множество формул вида $t \leq s$, U' — множество формул вида $\neg(t' \leq s')$ и U^* обозначает транзитивное и рефлексивное замыкание множества формул U. Тогда формула A выполнима тогда и только тогда, когда $U^* \cap U' = \emptyset$.

2.3. Теория равенства мультимножеств

Сигнатура рассматриваемой теории включает сорта $elems$ и $msets$ и операции объединения мультимножеств $\cup$, конструктора одноэлементного мультимножества $\{.\}$, пустого мультимножества $\{\}$ и равенства мультимножеств $=$. Сорт $msets$ интерпретируется как множество всех конечных мультимножеств из элементов сорта $elems$, $elems$ — как произвольное множество, на котором определена операция равенства $=$. Пусть буквы x, y, z обозначают кортежи, u, v — элементы кортежей.

Если для теорий, описанных в предыдущих пунктах, алгоритмы проверки выполнимости формул хорошо известны, то для теории равенства

мультимножеств алгоритм такого рода не рассматривался. Родственной теорией является теория перестановочности массивов [42]. Однако разрешающая процедура для нее использует индексирование массивов и поэтому не может быть применена в данном случае. Мы предлагаем следующий алгоритм проверки выполнимости формул теории равенства мультимножеств. Опишем сначала представление данных в алгоритме . Атомарная формула

$$t_1 \cup ... \cup t_k = s_1 \cup ... \cup s_m,$$

где t_i, s_j не содержат операции объединения, представляется в виде пары $(=, U)$, где

$$U = \{\{t_1, ..., t_k\}, \{s_1, ..., s_m\}\}$$

— мультимножество из двух мультимножеств. В случае отрицания такой атомарной формулы соответствующая пара имеет вид $(\neq, U)$. Атомарные формулы вида $u = v$ преобразуются к виду $\{u\} = \{v\}$. Поэтому они и их отрицания также представляются согласно описанной выше схеме. Конъюнкция литералов представляется как мультимножество пар, представляющих литералы. Будем использовать буквы U, V, W для обозначения введенных представлений формул. Пусть s — функция, сопоставляющая переменным типа $msets$ целочисленные переменные, причем разным переменным сопоставляются разные переменные. Распространим эту функцию на представления формул следующим образом:

$$\begin{aligned} &s(\{\{u\}\} \cup W) = 1 + s(W),\\ &s(\{x\} \cup W) = s(x) + s(W),\\ &s(\{\emptyset\} \cup W) = 0 + s(W). \end{aligned}$$

Алгоритм проверки выполнимости формул теории равенства мультимножеств состоит из следующих шагов:

0. Пусть V — представление формулы A, V' — пустое мультимножество.

1. Провести нормализацию выражений в представлении V, состоящую в недетерминированном применении замен

$$\begin{array}{lll} \{(=, \{\{w\} \cup W, \{w\} \cup W'\})\} \cup U & \text{на} & \{(=, \{W, W'\})\} \cup U,\\ \{(=, \{\{\emptyset\} \cup W, W'\})\} \cup U & \text{на} & \{(=, \{W, W'\})\} \cup U,\\ \{(=, \{\emptyset, \emptyset\})\} \cup U & \text{на} & U \end{array}$$

до тех пор, пока они выполняются.

2. Если система формул арифметики Пресбургера

$$\{s(W) = s(W')|(=, \{W, W'\}) \in V \cup V'\}$$

невыполнима при неотрицательных значениях переменных, то закончить алгоритм с результатом: "формула A невыполнима". Если $\{(\neq, \{\emptyset, \emptyset\})\} \in V$, то алгоритм завершается с результатом "формула A невыполнима". Если V не содержит равенств, то алгоритм завершается с результатом "формула A выполнима". В противном случае перейти к следующему шагу.

3. Если V имеет вид

$$\{(=, \{\{\underbrace{x, ..., x}_{n \text{ раз}}\} \cup W, W'\})\} \cup V'',$$

то заменить V на

$$\{(*, \{\underbrace{W' \cup ... \cup W'}_{n' \text{ раз}} \cup \underbrace{U \cup ... \cup U}_{m' \text{ раз}}, \underbrace{U' \cup ... \cup U'}_{m' \text{ раз}} \cup \underbrace{W \cup ... \cup W}_{n' \text{ раз}}\})| \\ (*, \{\{\underbrace{x, ..., x}_{m \text{ раз}}\} \cup U, U'\}) \in V''\}$$

и V' на

$$\{(=, \{\{x\} \cup W, W'\})\} \\ \cup \\ \{(*, \{\underbrace{W' \cup ... \cup W'}_{n \text{ раз}} \cup \underbrace{U \cup ... \cup U}_{m' \text{ раз}}, \underbrace{U' \cup ... \cup U'}_{m' \text{ раз}} \cup \underbrace{W \cup ... \cup W}_{n' \text{ раз}}\})| \\ (*, \{\{\underbrace{x, ..., x}_{m \text{ раз}}\} \cup U, U'\}) \in V'\},$$

где $* \in \{=, \neq\}$, $x \notin U$, $nn' = p$, $mm' = p$, p — наименьшее общее кратное m и n, и возвратиться к шагу 1. В противном случае перейти к следующему шагу.

4. Пусть V имеет вид

$$\{(=, \{\underbrace{w, ..., w}_{m \text{ раз}}\} \cup W, \{\underbrace{w', ..., w'}_{n \text{ раз}}\} \cup W')\} \cup V'',$$

где $m \geq n > 0$, $w \notin W$, $w' \notin W'$. Тогда для каждого такого w' применить рекурсивно алгоритм к паре (V_n, V'_n), где V_n имеет вид

$$\{(=, \{\underbrace{w, ..., w}_{m-n \text{ раз}}\} \cup W, W')\}$$

$$\cup$$

$$\{(*, \{\underbrace{w, ..., w}_{k \text{ раз}}\} \cup U, U') | (*, \{\{\underbrace{w', ..., w'}_{k \text{ раз}}\} \cup U, U'\}) \in V''\},$$

а V'_n имеет вид

$$\{(*, \{\underbrace{w, ..., w}_{k \text{ раз}}\} \cup U, U') | (*, \{\{\underbrace{w', ..., w'}_{k \text{ раз}}\} \cup U, U'\}) \in V'\}$$

для $* \in \{=, \neq\}$ и $w' \notin U$. Если для всех случаев результат — невыполнимость, то алгоритм завершается с результатом "формула A невыполнима". В противном случае алгоритм завершается с результатом "формула A выполнима".

Докажем полноту алгоритма.

Теорема 2ю1. Алгоритм проверки выполнимости формул теории равенств мультимножеств является полным.

Доказательство. Достаточно показать, что все шаги алгоритма сохраняют выполнимость формул и алгоритм завершается.

Алгоритм может не завершаться, если цикл из шагов 1-3-1 бесконечен или имеется бесконечное число рекурсивных вызовов алгоритма. Конечность цикла следует из того факта, что при каждой итерации уменьшается число различных переменных (элиминируется переменная x на шаге 3). Конечность числа рекурсивных вызовов обеспечивается тем, что при каждом таком вызове уменьшается число различных переменных типа $elems$ (на шаге 4 элиминируется терм w', имеющий вид $\{u\}$ для некоторой переменной u).

Шаг 1 сохраняет выполнимость, так как первая замена только удаляет одинаковые элементы из обеих частей равенства, вторая замена удаляет пустые мультимножества, третья — исключает из формулы выполнимые конъюнктивные члены вида $\emptyset = \emptyset$.

Докажем, что шаг 2 сохраняет выполнимость. Невыполнимость системы формул арифметики Пресбургера означает невозможность подобрать значения переменных в равенствах так, чтобы мощности мультимножеств, образующих левую и правую части равенств, совпадали во всех равенствах одновременно. Если формула содержит конъюнктивный член вида $\emptyset \neq \emptyset$, то она невыполнима. Пусть V содержит только неравенства. Тогда можно подобрать такие значения переменных, входящих в V, что все неравенства будут истинны. Докажем, что при этих

значениях переменных формула, соответствующая представлению V', выполнима. Представление V' имеет вид

$$\{(=, W_1 \cup \{\underbrace{x_1, ..., x_1}_{n_1 \text{ раз}}\}, U_1), ..., (=, W_k \cup \{\underbrace{x_k, ..., x_k}_{n_k \text{ раз}}\}, U_k)\},$$

где W_i, U_j не содержат переменных $x_1, ..., x_k$ и эти переменные попарно различны и не входят в V. Очевидно, что в случае выполнимости системы формул арифметики Пресбургера, образованной по представлению V', всегда можно подобрать значения переменных $x_1, ..., x_k$ такие, что формула, соответствующая V', будет истинна.

Шаги 3 и 4 представляют собой обычные замены переменных с сохранением равенства с заменяемой переменной в V' и поэтому сохраняют выполнимость. □

3. СИСТЕМЫ ПЕРЕПИСЫВАНИЯ ФОРМУЛ

Системы переписывания формул [5, 7, 20] представляют собой формализм, позволяющий переписывать формулы в формулы. Они объединяют механизм сопоставления с образцом, используемый в системах переписывания термов, с сужением с помощью наиболее общего унификатора. Здесь мы ограничимся изложением основных определений теории СПФ и свойствами, достаточными для целей данной работы (полное изложение см. в [7, 20]). В п.3.1 вводится формальный аппарат систем переписывания формул, в п.4.2 изложены некоторые методы доказательства завершимости СПФ (подробно см. в [7]), в п.4.3 описано использование СПФ для элиминации операций над массивами и файлами.

3.1. Формальный аппарат систем переписывания формул

Здесь изложены основные определения теории СПФ и сформулирована теорема о сохранении выполнимости — ключевая в теории СПФ, так как в ней даются достаточные условия, позволяющие при переписываниях с помощью СПФ сохранять выполнимость, т. е. выполнимая формула переписывается в выполнимую, а невыполнимая в невыполнимую.

3.1.1. Обозначения

Считая, что читатель знаком с основными понятиями логики первого порядка и теорией унификации, приведем только используемую нотацию.

Пусть S — множество, M — мультимножество, $m \in M$. Тогда $|S|$ обозначает мощность S, $\mathcal{FS}(\mathcal{S})$ — множество всех конечных подмножеств S, $\mathcal{FM}(\mathcal{S})$ — множество всех конечных мультимножеств, состоящих из элементов S, и $\mathcal{O}(m, M)$ — число вхождений элемента m в M. Пусть $\Sigma = (\mathcal{F}, \mathcal{P}, \mathcal{V})$ — сигнатура с множеством функциональных символов $\mathcal{F}$, множеством предикатных символов $\mathcal{P}$ и множеством переменных $\mathcal{V}$. Тогда $\mathcal{T}$ обозначает множество всех Σ-термов, $\mathcal{UF}$ — множество всех бесквантворных Σ-формул с равенством $=$, $\mathcal{E} = \mathcal{T} \cup \mathcal{UF}$ — множество всех Σ-выражений и $\mathcal{S}$ — множество всех подстановок над $\mathcal{E}$.

Используем буквы p, A, B для обозначения формул, u, v, l, r, s — для обозначения выражений, x, y, z — для обозначения переменных и σ, θ, ϕ, μ, τ — для обозначения подстановок. Пусть $\mathcal{V}ar(u)$, $\mathcal{MV}ar(u)$ обозначают соответственно множество и мультимножество (с учетом числа вхождений) переменных, входящих в u, $|u|$ — число функциональных, предикатных символов и переменных, входящих в u, $root(u)$ — корень u, $\mathcal{P}(u)$ — множество позиций в u с Λ как самой верхней позицией, $\mathcal{D}om(\sigma), \mathcal{VR}ange(\sigma)$ — область определения и область значения σ соответственно.

Пусть S — множество, $\rightarrow$ — бинарное отношение на S, $\succ$ — частичный порядок на S. Отношение $\rightarrow$ называется нетеровым, если не существует бесконечной цепи вида $s_0 \rightarrow s_1 \rightarrow \ldots$. Бинарное отношение $\succ_m$, определенное на $\mathcal{FM}(S)$, называется расширением порядка $\succ$ на мультимножества из $\mathcal{FM}(S)$, если для любых мультимножеств $M, M' \in \mathcal{FM}(S)$ $M \succ_m M'$ тогда и только тогда, когда существуют такие мультимножества $X, Y \in FM(S)$, что $\emptyset \neq X \subseteq M$, $M' = (M \setminus X) \cup Y$ и для любого $y \in Y$ найдется $x \in X$ такой, что $x \succ y$.

3.1.2. Определения

Дадим определение системы переписывания формул и отношения редукции, порождаемого ее.

Правило переписывания формул ρ определяется как пара $(\mathcal{B}, s)$, где $\mathcal{B} \in FS(\mathcal{UF} \times \mathcal{E} \times \mathcal{E})$ и $s \in \mathcal{E}$ такая, что для любой тройки $(p, l, r) \in \mathcal{B}$ выражения l и s унифицируемы. $\mathcal{B}$ называется основой ρ, (p, l, r) —

ветвью ρ, s — образцом ρ. Множество правил переписывания формул образует систему переписывания формул (СПФ). Будем писать $p|l \to r$ вместо (p, l, r). Пусть $\pi = p|l \to r$ — ветвь правила ρ. Назовем p посылкой, l — левой частью и p — правой частью ветви π. В случае, когда ветвь имеет вид $true|l \to r$, посылка $true$ будет опускаться.

Считаем, что все дальнейшие понятия определяются для некоторой фиксированной СПФ R. Пусть $\rho \in R$, $\pi = p|l \to r$ — ветвь ρ, $q \in \mathcal{P}(A)$, θ — НОУ l и s.

Так же, как и в случае систем переписывания термов, СПФ сопоставляется некоторое отношение переписывания (редукции), порождаемое ею. Это отношение использует понятие тривиального унификатора — частный случай наиболее общего унификатора (НОУ). Пусть

$$\bar{x} = \mathcal{D}om(\sigma) \cup \mathcal{D}om(\sigma'),\ \bar{y} = \mathcal{V}ar(\bar{x}\sigma = \bar{x}\sigma'),\ \bar{z} = \mathcal{VR}ange(\theta).$$

НОУ θ подстановок σ и σ' называется тривиальным унификатором σ и σ', если формула

$$\forall\bar{y}\exists\bar{z}(\bar{x}\sigma = \bar{x}\sigma' \Rightarrow \bar{y} = \bar{y}\theta)$$

тождественно-истинна. Подстановки σ и σ' тривиально унифицируемы, если существует тривиальный унификатор этих подстановок. Алгоритм поиска тривиального унификатора представляет собой вырожденный случай алгоритма унификации [30] и получается из него удалением правила редукции термов.

Пусть $q \in \mathcal{P}(A)$, $l\sigma = A_q$, ϕ — тривиальный унификатор σ и θ. Чтобы избежать конфликта переменных, считаем, что A и R не имеют общих переменных, множества $\mathcal{VR}ange(\theta), \mathcal{VR}ange(\phi)$ образованы новыми переменными,

$$\mathcal{D}om(\theta) = \mathcal{V}ar(l) \cup \mathcal{V}ar(s)$$

и

$$\mathcal{D}om(\phi) = \mathcal{V}ar(\{x\sigma = x\theta | x \in \mathcal{D}om(\sigma) \cup \mathcal{D}om(\theta)\}).$$

Отношение редукции $\to_R$, порождаемое R, определяется следующим образом:

$$\to_R = \{(A, \{(p\sigma \wedge A[r\sigma]_q)\phi|\pi\}) | \text{по всем} A, q, \rho, \theta, \phi\}.$$

Для $p = true$ формула $p\sigma \wedge A[r\sigma]_q$ сокращается до $A[r\sigma]_q$. Это отношение обычным образом распространяется на конечные мультимножества формул: для любых $v \in \mathcal{UF}$, $U, V \in FM(\mathcal{UF})$, если $v \to_R V$, то $U \cup \{v\} \to_R U \cup V$. В случае системы из единственного правила ρ будем писать $\to_\rho$ вместо $\to_R$.

3.1.3. Свойства

Как и для обычных СПТ, для СПФ можно определить понятия нетеровости, нормальной формы, редекса и согласования с частичным порядком.

Определение 3ю1. Пусть $U, V, W \in \mathcal{FM}(\mathcal{UF})$. V называется нормальной формой U относительно R, если $U \to_R^* V$ и для любого W неверно, что $U \to_R W$. Выражение t называется редексом R, если найдется правило $\rho \in R$ такое, что $t = s\sigma$ для некоторой подстановки σ и для любой ветви π этого правила подстановки θ и σ тривиально унифицируемы. СПФ R называется нетеровой, если $\to_R$ — нетерово отношение, и согласована с частичным порядком $\succ$, если $\to_R \subseteq \succ$.

Заметим, что СПФ, согласованная с нетеровым порядком, — нетерова.

Считаем, что под выполнимостью и истинностью формул понимается их выполнимость и истинность в некоторой фиксированной алгебраической системе $\mathcal{A}$. Использование систем переписывания формул обусловлено тем, что для них сформулированы условия, которые позволяют сохранять выполнимость формул при переписываниях. Мультимножество формул выполнимо, если найдется выполнимая формула, входящая в это мультимножество. Отношение $\to$, определенное на мультимножествах формул, сохраняет выполнимость, если для любых $U, V \in FM(\mathcal{UF})$ при $U \to V$ выполнимо U тогда и только тогда, когда V выполнимо. Достаточные условия сохранения выполнимости для СПФ имеют следующий вид. Пусть $B_\rho(\bar{x}, \bar{y}, \sigma)$ обозначает формулу

$$\forall \bar{x}(\vee_\pi \exists \bar{y}(p\sigma \wedge \bar{x} = \bar{x}\sigma)),$$

где $\bar{x}, \bar{y} \subseteq \mathcal{V}$.

Теорема 3ю2. Пусть

$$\bar{x}^s = \mathcal{V}ar(s),\ \bar{x}^l = \mathcal{V}ar(l),\ \bar{x} = (\mathcal{V}ar(p) \cup \mathcal{V}ar(r)) \setminus \bar{x}^l\ \bar{x}^\theta = \mathcal{VR}ange(\theta).$$

Тогда, если для любых ρ и π формула $p \Rightarrow l = r$ истинна и для любого ρ формула

$$B_\rho(\bar{x}^s \cup \bar{x}^l, \bar{x} \cup \bar{x}^\theta, \theta)$$

истинна, то отношение $\to_R$ сохраняет выполнимость.

Доказательство теоремы можно найти в [7]. Более простой случай рассмотрен в [5].

Будем говорить, что система R сохраняет выполнимость, если порождаемое ею отношение редукции $\to_R$ сохраняет выполнимость.

3.2. Классы завершимых систем переписывания формул

Все рассматриваемые в этой работе СПФ являются конструктивными. Конструктивная СПФ R есть СПФ, в которой множество функциональных и предикатных символов может быть разделено на множество анализаторов и множество конструкторов таких, что для каждого правила $\rho \in R$ его образец имеет вид $f(t_1, ..., t_n)$, где f — анализатор и термы $t_1, ..., t_n$ состоят только из конструкторов и переменных. В общем случае проблема нетеровости СПФ неразрешима [7]. Для доказательства завершимости процедур (тактик), основанных на СПФ, выделим классы завершимых конструктивных СПФ.

3.2.1. Системы элиминации анализаторов

Одним из классов конструктивных СПФ, на основе которого можно создавать завершающиеся тактики, является класс систем элиминации анализаторов. Хотя этот класс систем включает и ненетеровы системы, в работе [7] было показано, что любая система из этого класса завершается относительно стратегии "любой самый внутренний", что позволяет строить завершающиеся процедуры, основанные на таких системах.

Пусть $\mathcal{C} \subseteq \mathcal{F} \cup \mathcal{P}$; $\mathcal{E}(\mathcal{C})$ ($\mathcal{T}(\mathcal{C})$) обозначает множество всех выражений (термов), построенных только из элементов множества $\mathcal{C} \cup \mathcal{V}$.

В дальнейшем будем считать R конструктивной СПФ с множеством конструкторов $\mathcal{C}$ и множеством анализаторов $\mathcal{D}$. Для определения систем элиминации анализаторов потребуются некоторые вспомогательные понятия, позволяющие анализировать структуру выражений относительно множеств $\mathcal{C}$ и $\mathcal{D}$.

Выражение u называется конструктивным, если $u \in \mathcal{E}(\mathcal{C})$. Подстановка σ называется конструктивной, если для любой переменной z имеет место $z\sigma \in \mathcal{T}(\mathcal{C})$. Выражение u называется линейным, если для любой переменной z

$$|\mathcal{O}(z, \mathcal{M}\mathcal{V}ar(u))| \leq 1.$$

Подстановка σ называется линейной на $X \subseteq \mathcal{V}$, если для любых $x, y \in X$ терм $x\sigma$ — линейный и

$$x \neq y \Rightarrow \mathcal{V}ar(x\sigma) \cap \mathcal{V}ar(y\sigma) = \emptyset.$$

Пусть $q_1, q_2 \in \mathcal{P}(u)$, $q_2 \neq q_1$, q_2 — префикс q_1. Выражение u называется:

– вложенным, если найдутся q_1, q_2 такие, что

$$root(u_{q_1}), root(u_{q_2}) \in \mathcal{D};$$

– простым, если u не является вложенным;
– вызовом, если $root(u) \in \mathcal{D}$.

Пусть $C_m(u)$ обозначает мультимножество всех простых вызовов, входящих в u. Отображение μ_c называется мерой конструкторов, если

$$mu_c(u) = \{|t||t \in C_m(u)\}.$$

Отображение Dec_v называется декомпозицией переменных, если

$$Dec_v(u) = \cup_{t \in C_m(u)} \mathcal{MV}ar(t).$$

Определим теперь системы элиминации анализаторов. Ветвь π называется:

– униформной, если l совпадает с образцом;
– просачиваемой, если π униформна, p, r — простые выражения,

$$Dec_v(l) \supseteq Dec_v(p) \cup Dec_v(r),$$
$$\mu_c(l) >_m \mu_c(r),$$
$$\mu_c(l) >_m \mu_c(p);$$

– элиминирующей, если любой НОУ θ выражений l и s является линейной на $\mathcal{V}ar(s)$ и конструктивной подстановкой, p, r — конструктивные выражения. Конструктивная СПФ R называется системой элиминации анализаторов, если любой простой вызов является редексом R и правила R состоят только из просачиваемых и элиминирующих ветвей.

Согласно стратегии “любой самый внутренний” правила системы R применяются к такому подвыражению формулы, которое является редексом R и не содержит редексов R в качестве собственных подвыражений. Связь между СЭА и этой стратегией устанавливает следующая

Теорема Зю3. Система элиминации анализаторов завершается относительно стратегии “любой самый внутренний”.

Доказательство теоремы приведено в [7].

3.2.2. Системы элиминации анализаторов со статусом

Еще одним классом систем переписывания формул, нетеровых относительно стратегии "любой самый внутренний", являются системы элиминации анализаторов со статусом, обобщающие обычные системы элиминации анализаторов. Идея обобщения состоит в том, что при анализе структуры выражений относительно конструкторов и анализаторов учитываются не все аргументы функциональных и предикатных символов, а лишь специальным образом выбранные. Такой выбор осуществляется с помощью специальной функции, сопоставляющей каждому функциональному и предикатному символу множество позиций выбираемых аргументов. Введем формальное

Определение 3ю4. Функция $\mathcal{A} : \mathcal{F} \cup \mathcal{P} \to \mathcal{FS}(\mathcal{N})$ называется аргументной функцией, если $\mathcal{A}(f) \subseteq \{1, ..., \mathcal{A}r(f)\}$ для каждого $f \in \mathcal{F} \cup \mathcal{P}$.

Зафиксируем некоторый аргументный статус $\mathcal{A}$ и рассмотрим, как изменятся понятия, введенные для СЭА, если учитывать статус $\mathcal{A}$. Пусть $q_1, q_2 \in \mathcal{P}(u)$, $q_2 \neq q_1$, q_2 — префикс q_1. Выражение u называется вложенным, если найдутся q_1, q_2 такие, что $root(u_{q_1}), root(u_{q_2}) \in \mathcal{D}$, и простым, если u не является вложенным. Пусть $C_m(u)$ обозначает мультимножество всех простых вызовов, входящих в u, $I(t)$ — мультимножество аргументов вызова t, находящихся в позициях $\mathcal{A}(root(t))$. Отображение μ_c называется мерой конструкторов, если

$$\mu_c(u) = \{ \sum_{t' \in I(t)} |t'| | t \in C_m(u)\}.$$

Отображение Dec_v называется декомпозицией переменных, если

$$Dec_v(u) = \cup_{t \in C_m(u)} \cup_{t' \in I(t)} \mathcal{MV}ar(t').$$

Определим теперь системы элиминации анализаторов со статусом. Ветвь π называется:

– просачиваемой, если π униформна, p, r — простые выражения,

$$Dec_v(l) \supseteq Dec_v(p) \cup Dec_v(r),$$
$$\mu_c(l) >_m \mu_c(r),$$
$$\mu_c(l) >_m \mu_c(p),$$

– элиминирующей, если любой НОУ θ выражений l и s является линейной на $\mathcal{V}ar(s)$ и конструктивной подстановкой, p, r — конструктивные выражения, и для любой $x \in \mathcal{V}ar(s)$

$$x\theta \notin \mathcal{V} \Rightarrow x \in \cup_{s' \in I(s)} \mathcal{V}ar(s').$$

Конструктивная СПФ R называется системой элиминации анализаторов, если любой простой вызов (не обязательно относительно $\mathcal{A}!$) является редексом R и правила R состоят только из просачиваемых и элиминирующих ветвей.

Теорема 3ю5. Система элиминации анализаторов с аргументным статусом завершается относительно стратегии "любой самый внутренний".

Доказательство теоремы см. в [7].

3.3. Техника доказательства свойств типов данных

Важным условием, которое накладывается на технику автоматического доказательства, используемую в верификации, является ее способность доказывать свойства структур данных языка программирования. Ниже напомним результаты, полученные в [6], касающиеся использования СПФ для сведения операций над массивами и файлами к операциям на кортежах. Такое преобразование позволяет свести доказательство свойств операций над массивами и файлами к доказательству формул над кортежами и целыми числами.

3.3.1. Массивы

Рассмотрим массивы из элементов типа $elems$ как тип $arrays[elems]$, определяемый конструктором $array$. Так как значение массива для некоторого индекса может быть не определено, расширим множество элементов массива константой ω_s, означающей неопределенное значение. Пусть массив $a = array(x, i, j)$. Тогда кортеж x называется основой массива a, i — левой границей a и j — правой границей a. Определим над типом $arrays$ операции взятия основы массива $tuple$, взятия левой границы массива $left$, взятия правой границы массива $right$, индексной выборки .[.] (значение этой операции не определено, если ω_s — выбираемый элемент), обновления массива upd, сужения границ массива nar. Считаем, что следующие переменные имеют фиксированные типы:

$$\begin{array}{l} a, b : arrays[elems], \\ x, y, z : tuples[elems_{\omega_s}], \\ i, j, k, l : ints, \\ u, v : elems_{\omega_s}. \end{array}$$

Пусть $cond$ — сокращение для формулы

$$i \leq j \wedge i + len(x) = j + 1.$$

Теория массивов $ARRAY$ в дополнение к аксиомам для конструктора array определяется следующими аксиомами для введенных операций:

– для $tuple$, $left$ и $right$:

$$\begin{array}{lcl} cond & \Rightarrow & tuple(array(x,i,j)) = x; \\ \neg cond & \Rightarrow & tuple(array(x,i,j)) = \omega; \\ cond & \Rightarrow & left(array(x,i,j)) = i; \\ \neg cond & \Rightarrow & left(array(x,i,j)) = \omega; \\ cond & \Rightarrow & right(array(x,i,j)) = j; \\ \neg cond & \Rightarrow & right(array(x,i,j)) = \omega; \end{array}$$

– для upd:

$$\begin{array}{lcl} left(a) \le i \le right(a) & \Rightarrow & upd(a,i,u) = array(\\ & & tuple(a)[1, i - left(a)] \cup \{u\} \cup \\ & & tuple(a)[i - left(a) + 2, \\ & & \qquad len(tuple(a))], \\ & & left(a), right(a)); \\ \neg(left(a) \le i \le right(a)) & \Rightarrow & upd(a,i,u) = \omega; \end{array}$$

– для .[.]:

$$\begin{array}{lcl} left(a) \le i \le right(a) \wedge & & \\ tuple(a)[i - left(a) + 1] \ne \omega_s & \Rightarrow & a[i] = tuple(a)[i - left(a) + 1]; \\ \neg(left(a) \le i \le right(a)) \vee & & \\ tuple(a)[i - left(a) + 1] = \omega_s & \Rightarrow & a[i] = \omega; \end{array}$$

– для nar:

$$\begin{array}{lcl} left(a) \le i \le j \le right(a) & \Rightarrow & nar(a,i,j) = array(\\ & & tuple(a)[i - left(a) + 1, \\ & & \qquad j - left(a) + 1], i, j); \\ \neg(left(a) \le i \le j \le right(a)) & \Rightarrow & nar(a,i,j) = \omega; \end{array}$$

– для $=$:

$$a = b \Leftrightarrow (tuple(a) = tuple(b) \wedge left(a) = left(b) \wedge right(a) = right(b)).$$

В [6] приведена СПФ, позволяющая элиминировать операции над массивами.

3.3.2. Файлы

С помощью СПФ можно также свести последовательные файлы к кортежам. Последовательные файлы из элементов типа $elems$ будем определять как тип $files[elems]$, определяемый конструктором $file$. Пусть файл $f = file(x, y, u)$. Тогда кортежи x и y называются левой (прочитанной) и правой (непрочитанной) частями файла f соответственно, а u — буфером файла f. Введем также специальную константу ω_s, которую будем использовать для описания неопределенного значения некоторого элемента структуры данных (например неопределенное значение некоторой компоненты массива или неопределенное значение буфера файла), и дополнительный тип

$$elems_{\omega_s} = elems \cup \{\omega_s\}.$$

Условимся везде в этом пункте использовать буквы f, g для обозначения файлов, x, y, z для обозначения кортежей и u, v, w для обозначения элементов типа $elems_{\omega_s}$. Определим над типом $files$ операции взятия левой части файла $left$, взятия правой части файла $right$, взятия значения буферной переменной $buffer$, проверки конца файла eof, чтения из файла get, записи в файл put, установки в начало файла res, очистки файла rew, обновления буфера buf и равенства файлов $=$. Теория файлов $FILE$ определяется следующими аксиомами для введенных операций:

– для $file$:

$$\exists x \exists y \exists u (f = file(x, y, u));$$
$$file(x, \{u\} \cup y, v) = file(x, \{v\} \cup y, \omega_s);$$

– для $left, right$:

$$left(file(x, y, u)) = x;$$
$$right(file(x, y, u)) = y;$$

– для $eof, =$:

$$\begin{array}{lcl} eof(f) & \Leftrightarrow & emp(right(f)); \\ f = g & \Leftrightarrow & (left(f) = left(g) \wedge right(f) = right(g) \wedge \\ & & (eof(f) \Rightarrow buffer(f) = buffer(g))); \end{array}$$

– для $buffer$:

$$buffer(file(x, \{u\} \cup y, v)) = u;$$
$$buffer(file(x, \{\}, \omega_s)) = \omega;$$
$$\neg(u = \omega_s) \Rightarrow buffer(file(x, \{\}, u)) = u;$$

– для get:

$$get(file(x, \{u\} \cup y, v)) = file(x \cup \{u\}, y, \omega_s);$$
$$get(file(x, \{\}, u)) = \omega;$$

– для put:

$$\neg(u = \omega_s) \Rightarrow put(file(x, \{\}, u)) = file(x \cup \{u\}, \{\}, \omega_s);$$
$$put(file(x, \{\}, \omega_s)) = \omega;$$
$$put(file(x, \{u\} \cup y, v)) = \omega;$$

– для res, rew:

$$res(file(x, y, u)) = file(\{\}, xcupy, \omega_s);$$
$$rew(file(x, y, u)) = file(\{\}, \{\}, \omega_s);$$

– для buf:

$$buf(file(x, \{\}, u), v) = file(x, \{\}, v);$$
$$buf(file(x, \{u\} \cup y, v), w) = \omega.$$

СПФ, элиминирующая операции над файлами, приведена в [6].

4. МОДУЛЬ ПЕРЕПИСЫВАНИЯ ФОРМУЛ В СИСТЕМЕ ВЕРИФИКАЦИИ

Основная цель реализации модуля переписывания формул — создание прототипа, с помощью которого можно было бы убедиться в силе и практичности аппарата систем переписывания формул для задачи автоматической верификации программ. Модуль предполагается использовать в качестве одного из компонентов блока доказательства проблемно-ориентированной системы верификации СПЕКТР, новая версия которой создается в настоящее время в лаборатории теоретического программирования ИСИ СО РАН. В качестве языка реализации выбран язык SML, позволяющий быстро создавать подобные прототипы.

В п.4.1 рассматривается логическая структура модуля — основные компоненты и связи между ними. В п.4.2 показано, как в модуле создаются упрощающие процедуры, основанные на СПФ. В п.4.3 описан язык комбинирования систем переписывания формул с упрощающими и разрешающими процедурами. В п.4.4 приводятся результаты применения модуля для доказательства условий корректности, возникающих при верификации ряда программ редактирования текстов, сортировки массивов и файлов.

В дальнейшем будем часто использовать такой тип данных языка SML, как списки. Поэтому опишем кратко синтаксис списков. Конструкция $[e_1, ..., e_k]$ обозначает список, состоящий из элементов e_1, ..., e_k. В частности, пустой список обозначается []. Операция добавления элемента в начало списка обозначается ::, а операция конкатенации (объединения) двух списков — @.

4.1. Общая структура модуля

Основными понятиями, используемыми в модуле переписывания формул, являются понятия формулы, контекстной формулы и тактики. Формулы определяются как элементы рекурсивного типа formulas языка SML. Каждой формуле приписывается некоторое слово из алфавита *Sort*, элементы которого называются сортами формул. Это слово содержит информацию о возможных способах обработки данной формулы и называется контекстом применения формулы. Формулы с приписанными им контекстами называются контекстными формулами. В качестве сортов формул используются произвольные буквенно-цифровые идентификаторы. Приписывание контекстов каждой отдельной формуле приводит к нерациональному расходованию памяти машины. Поэтому в реализации информация о формулах и их контекстах хранится крупными блоками, называемыми иерархическими контекстными формулами (ИК-формулами). ИК-формулы являются элементами рекурсивного типа contextformulas языка SML, определяемого конструкторами empcontext и context следующим образом:

– если l — список формул, то empcontext(l) — ИК-формула;

– если l — список ИК-формул, s — сорт, то context(s, l) — ИК-формула.

Сопоставим ИК-формуле следующее помеченное дерево:

– формуле empcontext(l) сопоставляется непомеченный лист;

– формуле context($s, [l_1, ..., l_k]$) сопоставляется дерево, корень кото-

рого помечен сортом s и из корня исходят k дуг к поддеревьям, сопоставленным формулам $l_1, ..., l_k$

В этом случае контекст любой формулы, входящей в ИК-формулу A, является путем, проходящим по вершинам из корня A в лист, в который входит данная формула.

Под тактикой понимается функция, написанная на языке SML и преобразующая наборы контекстных формул в наборы контекстных формул в определенном формате. Входом тактики является ИК-формула A, а выходом пара (l, l'), где l — список формул (без контекстов), l' — список ИК-формул. Такой выход позволяет проводить частичные преобразования ИК-формул. В l попадают формулы, на которых данная тактика закончила свою работу, а в l' — промежуточный выход, который в дальнейшем может либо подаваться, либо не подаваться на вход данной тактики в зависимости от стратегий проведения вывода. На логическом уровне тактика переписывает дизъюнкцию всех формул, входящих в листы формулы A, в дизъюнкцию всех формул, входящих или в листы формул из списка l', или в список формул l. Тактика называется корректной, если она переписывает формулы в формулы с сохранением выполнимости, т. е. входная дизъюнкция формул выполнима тогда и только тогда, когда выполнима выходная дизъюнкция формул. Если тактика не применима к входной формуле, то выдается исключение tac_not_applied. Каждая тактика хранится в отдельном файле с таким же именем и с расширением tac.

В модуле предусмотрены средства создания и комбинирования тактик. Функция на языке SML, которая в качестве выхода имеет некоторую тактику, называется тактикалом. В модуль встроены два тактикала FRS и CRS.

Тактикал FRS получает на вход систему переписывания формул и выдает тактику, применяющую данную систему в соответствии со стратегией "левый самый внутренний". Полученная тактика корректна, если соответствующая система переписывания формул сохраняет выполнимость. Доказательство сохранения выполнимости системой сводится к проверке достаточных условий теоремы о сохранении выполнимости и в настоящее время производится вручную. Выбор такой стратегии применения СПФ обусловлен тем, что для этого случая разработан ряд техник доказательства завершимости.

В модуле используется специальный язык комбинирования тактик. Управление выводом с помощью этого языка основано на перепиcы-

вании контекстов применения формул, осуществляемом специальными правилами переписывания контекстов. Конечное множество правил переписывания контекстов образует систему переписывания контекстов. Тактикал CRS строит по системе переписывания контекстов соответствующую этой системе корректную тактику.

Кроме тактик, создаваемых с помощью тактикалов FRS и CRS, пользователь имеет возможность писать на языке свои собственные тактики, называемые базовыми. Проверка корректности базовых тактик ложится на пользователя. Библиотека базовых тактик в настоящее время содержит тактики supinf, ordering, equal, mseteq, проверяющие выполнимость бескванторных формул арифметики Пресбургера, теории частичного порядка, теории равенства и теории равенства на мультимножествах соответственно.

Основной функцией в модуле, проверяющей выполнимость формул, является функция prover. Она получает на вход формулу из файла с расширением frm и корректную тактику из файла с расширением tac и выдает исключение satisfiable, если формула выполнима, и список формул в противном случае. Из корректности тактики следует, что исходная формула выполнима тогда и только тогда, когда выполнима хотя бы одна из формул полученного списка. В частности, если список имеет вид [false], то исходная формула невыполнима. Пусть A — исходная формула, τ — подаваемая на вход тактика. Действие функции prover заключается в приведении формулы A к формату входа, а именно $B = \mathrm{empcontext}([A])$, применении тактики τ к B и преобразовании результата (l, l') в список формул, полученный добавлением к списку формул l всех формул, входящих в листы ИК-формул из списка l'.

4.2. Тактикал для систем переписывания формул

Как уже было сказано выше, тактикал FRS по системе переписывания формул строит соответствующую тактику. Пусть R — СПФ, τ — тактика, полученная по СПФ R, A — ИК-формула вида $\mathrm{empcontext}(l)$. Тогда тактика τ применяется к формуле A следующим образом. На вход СПФ R подается список формул l. Если R не применима к самому левому элементу списка l или список l — пустой, то результатом тактики τ будет исключение tac_not_applied. В противном случае СПФ R применяется к самому левому элементу списка формул в соответствии со стратегией "левый самый внутренний", заменяя элемент каждый раз на новый список формул до тех пор, пока не получится такой список

формул $B :: l'$, что СПФ R не применима к формуле B. Тогда выходом тактики τ является пара $([B], [\text{empcontext}(l')])$.

Системы переписывания формул подаются на вход тактикалу через файл с расширением frs и имеют следующий синтаксис:

```
<СПФ>         := <правило>;
              := <правило>; <СПФ>
<правило>     :=  rule(<образец>;base(<ветви>))
<образец>     := <выражение>
<ветви>       := <ветвь>
              := <ветвь>; <ветви>
<ветвь>       := branch(<посылка>;<лев.часть>;<прав.часть>)
<посылка>     := <формула>
<лев.часть>   := <выражение>
<прав.часть>  := <выражение>
<выражение>   := <формула> | <терм>.
```

Элементы синтаксических категорий `<формула>` и `<терм>` — обычные формулы и термы в префиксной записи.

4.3. Тактикал для систем переписывания контекстов формул

В модуле предусмотрено средство построения по существующим тактикам новых тактик, основанное на системах переписывания контекстов формул (СПКФ). СПКФ переписывают ИК-формулы в списки ИК-формул, меняя при этом контекст, — таким образом через контексты осуществляется управление выводом. СПКФ состоит из конечного числа правил переписывания контекстов формул, представляющих собой четверки $(s_i, s_o, s_{,e}, \tau)$, где s_i, s_o, s_e — сорта, τ — тактика. Сорт s_i называется входным сортом правила, s_o — выходным, s_e — выходным по исключению. Правило называется начальным, если s_i совпадает со специально выделенным сортом *start*. Любая СПКФ должна содержать начальное правило, и любые два правила, входящие в нее, должны отличаться первыми элементами.

Рассмотрим применение СПКФ к ИК-формуле. Пусть R — СПКФ, A — ИК-формула вида $\text{context}(s, l)$. Система R применима к ИК-формуле A, если найдется такое правило в R, первым элементом которого является сорт s. Согласно ограничениям, наложенным на СПКФ, если такое правило найдется, то оно будет единственным. Пусть $\rho = (s, s_o, s_e, \tau) \in R$. Если l — пустой список, то формула A переписывается в пустой список ИК-формул. Пусть $l = B :: l'$. Тогда к формуле B применяется

тактика τ. Если (l_1, l_2) — результат применения тактики τ, то формула A переписывается в список

$$[\text{context}(s_o, [\text{empcontext}(l_1)]), \text{context}(s, l_2@l')]$$

ИК-формул. Если тактика τ не применима к формуле B и выдает исключение tac_not_applied, то формула A переписывается в список

$$[\text{context}(s_e, [\text{empcontext}([B])]), \text{context}(s, l')]$$

ИК-формул.

Пусть R — СПКФ, τ — тактика, полученная по СПКФ R, A — ИК-формула вида context(l). Тогда тактика τ применяется к формуле A следующим образом. На вход СПКФ R подается список формул l. Если R не применима к самому левому элементу списка l или список l — пустой, то результатом тактики τ будет исключение tac_not_applied. В противном случае, СПКФ R применяется к самому левому элементу списка ИК-формул, заменяя элемент каждый раз на новый список ИК-формул до тех пор, пока не получится такой список ИК-формул $B :: l'$, что СПКФ R не применима к формуле B. Пусть формула B имеет вид context$(s, [\text{empcontext}(l'')])$ Тогда выходом тактики τ является пара $(l'', l'])$.

Рассмотрим примеры комбинирования тактик с помощью СПКФ.

Пример(последовательное применение) 4ю1. Тактика, состоящая в последовательном применении тактик τ_1 и τ_2, задается СПКФ, состоящей из правил:

$$(s, s_1, s_1, \tau_1);$$
$$(s_1, s_2, s_2, \tau_2).$$

Пример(попеременное применение) 4ю2. Тактика, состоящая в попеременном применении тактик τ_1 и τ_2 до тех пор, пока хотя бы одна из них применяется, задается СПКФ, состоящей из правил:

$$(s, s_2, s_{e1}, \tau_1);$$
$$(s_{e2}, s_2, s_e, \tau_1);$$
$$(s_1, s_2, s_{end1}, \tau_1);$$
$$(s_{e1}, s_1, s_e, \tau_2);$$
$$(s_2, s_1, s_{e2}, \tau_2).$$

Перевод СПКФ в соответствующую тактику производится тактикалом CRS. СПКФ подаются на вход тактикалу через файлы с расширением *srs* и имеют следующий синтаксис:

```
<СПКФ>          := <СПКФ-правило>;
                := <СПКФ-правило>;<СПКФ>
<СПКФ-правило> := rule(<сорт>;<сорт>;<сорт>;<тактика>).
```

Элементами синтаксических категорий `<сорт>` и `<тактика>` являются произвольные буквенно-цифровые идентификаторы.

5. ПРИМЕРЫ АВТОМАТИЧЕСКОЙ ВЕРИФИКАЦИИ ПРОГРАММ

Рассмотрим применение СПФ для автоматического доказательства условий корректности, появляющихся при верификации программ. В качестве примеров верифицируемых программ выбраны программы редактирования текстов, сортировки массивов и сортировки файлов.

Примеры простых программ редактирования текстов — копирование из одного файла в другой и линейный поиск — даются в п.5.1. Программы сортировки массивов можно разбить на два класса. Первый класс образуют простые программы сортировки типа пузырьковой сортировки или сортировки вставкой.

В п.5.2.1 на примере программы сортировки вставкой показано, что для условий корректности этих программ можно построить полную корректную тактику, комбинирующую применение СПФ и базовых тактик, описанных в п.3. Второй класс образуют улучшенные программы сортировки. В качестве примера программы из этого класса в п.5.2.2 выбрана быстрая сортировка.

Сортировка последовательных файлов — более сложная задача по сравнению с сортировкой массивов, так как в связи с только последовательным доступом к элементам файла требуется разрабатывать более сложный алгоритм перегруппировки и упорядочивания его элементов. В п.5.3 дан пример программы сортировки файлов естественным слиянием.

Примеры программ взяты из [8].

Во всех примерах буквы x, y, z представляют кортежи, u, v, w — элементы кортежей, f, g, h — файлы, a, b, c — массивы, k, l, r, m, n, i, j — целые числа.

5.1. Программы редактирования текстов

Верификация программ редактирования текстов — хороший способ проверки выразительности методов автоматической верификации. Рас-

смотрим примеры верификации программ копирования файла и линейного поиска в массиве.

5.1.1. Копирование файла

Равенство на кортежах часто встречается в программах редактирования текстов. Приведем пример одной из таких программ — программы копирования из одного файла в другой. Аннотированная программа копирования из файла f в файл g имеет следующий вид:

$\{in\}$**function** $copy : (f, g) \to (f, g);$
$f := res(f);\ g := rew(g);$
repeat $g := buf(g, buffer(f));\ f := get(f);\ g := put(g);$
until $eof(f); \{inv\}\{out\},$

где

$$\begin{array}{ll} in: & \neg(left(f) \cup right(f) = \{\}), \\ out: & eof(f) \wedge eof(g) \wedge left(f) = left(g), \\ inv: & left(f) = left(g) \wedge eof(g). \end{array}$$

Требуется доказать истинность условий корректности данной программы, а тактики позволяют проверять выполнимость (невыполнимость) формул. Поэтому перейдем от доказательства истинности к проверке выполнимости. Формула истинна тогда и только тогда, когда ее отрицание невыполнимо. Следовательно, доказательство истинности условий корректности можно свести к установлению невыполнимости их отрицаний. Построим тактику для условий корректности этой программы. На первом шаге элиминируются операции над файлами. В результате отрицания условий корректности сводятся к формулам, содержащим только равенство на кортежах и элементах кортежей. На втором шаге применяется приближенная тактика τ для равенства на кортежах. Эта тактика строится на основе СПФ R, состоящей из правил:

– элиминации отрицания над $=$:

$$\neg(x = y) \to x \neq y;$$

– левой ассоциативности:

$$(x \cup y) \cup z \to x \cup (y \cup z);$$

– элиминации совпадающих префиксов:

$$x = x \to true;$$
$$x \neq x \to false;$$
$$x \cup y = x \cup z \to y = z;$$
$$x \cup y \neq x \cup z \to y \neq z;$$
$$x = x \cup y \to \{\} = y;$$
$$x \neq x \cup y \to \{\} \neq y;$$
$$x \cup y = x \to y = \{\};$$
$$x \cup y \neq x \to y \neq \{\};$$

– элиминации одноэлементных префиксов:

$$\{u\} \cup x = \{v\} \cup y \to u = v \wedge x = y;$$
$$(\{u = v | \{u\} \cup x = \{v\} \cup y \to x \neq y,$$
$$u \neq v | \{u\} \cup x = \{v\} \cup y \to true\},$$
$$\{u\} \cup x \neq \{v\} \cup y);$$

– просачивания и элиминации пустых кортежей:

$$\{\} \cup x \to x;$$
$$x \cup \{\} \to x;$$
$$\{\} = \{u\} \to false;$$
$$\{u\} = \{\} \to false;$$
$$\{\} = \{u\} \cup y \to false;$$
$$\{u\} \cup y = \{\} \to false;$$
$$(\{\{\} = \{\} \to true\}, \{\} = x);$$
$$(\{\{\} = \{\} \to true\}, x = \{\});$$
$$(\{\{\} = \{\} \cup y \to \{\} = y\}, \{\} = x \cup y);$$
$$(\{\{\} \cup y = \{\} \to y = \{\}\}, x \cup y = \{\});$$

– просачивания пар префиксов переменная — одноэлементный кортеж:

$$(\{\{u\} = \{u\} \to true\}, x = \{u\});$$
$$(\{\{u\} = \{u\} \to true\}, \{u\} = x);$$
$$(\{\{u\} = \{u\} \cup y \to \{\} = y\}, x = \{u\} \cup y);$$
$$(\{\{u\} \cup y = \{u\} \to y = \{\}\}, \{u\} \cup y = x);$$
$$(\{\{u\} \cup z = \{u\} \cup y \to z = y,$$
$$\{\} \cup z = \{u\} \cup y \to z = \{u\} \cup y\},$$
$$x \cup z = \{u\} \cup y);$$
$$(\{\{u\} \cup y = \{u\} \cup z \to y = z,$$
$$\{u\} \cup y = \{\} \cup z \to \{u\} \cup y = z\},$$
$$\{u\} \cup y = x \cup z);$$

– просачивания пар префиксов переменная — переменная:

$$
\begin{array}{l}
(\{x = x \to true\}, x = y);\\
(\{x = x \cup z \to \{\} = z,\\
\quad x = \{\} \cup z \to x = z,\\
\quad \{\} = y \cup z \to \{\} = y \cup z\},\\
\ x = y \cup z);\\
(\{x \cup z = x \to z = \{\},\\
\quad \{\} \cup z = x \to z = x,\\
\quad y \cup z = \{\} \to y \cup z = \{\}\},\\
\ y \cup z = x);\\
(\{x \cup z = x \cup z' \to z = z',\\
\quad \{\} \cup z = y \cup z' \to z = y \cup z',\\
\quad x \cup z = \{\} \cup z' \to x \cup z = z'\},\\
\ x \cup z = y \cup z');
\end{array}
$$

– элиминации $\neq$:

$$x \neq y \to true.$$

Стратегия применения системы R заключается в том, что сначала применяются правила элиминации отрицания над равенством, затем все остальные правила, кроме последнего, и, наконец, последнее правило. Перед применением правила элиминации $\neq$ равенство уже элиминировано из формул, а все оставшиеся атомарные формулы с $\neq$ выполнимы, причем выполнимость их сохраняется в конъюнкции с любой выполнимой формулой теории равенства. Поэтому применение последнего правила сохраняет выполнимость. Применение всех остальных правил, кроме правил просачивания пар префиксов, сохраняет выполнимость, так как эти правила удовлетворяют условиям теоремы о сохранении выполнимости. Правила просачивания пар префиксов в общем случае не сохраняют выполнимость. Но с учетом эвристического принципа, что кортежи в посылке условия корректности отличаются от соответствующих кортежей в заключении условия корректности только заранее определенным конечным числом элементов, относительная полнота этих правил обеспечивается. Данный принцип является следствием того, что за один шаг выполнения программы в массиве или файле изменяется только один элемент (исключением являются только операции res и rew, но результат этих операций жестко определен). Так, например, правило

$$(\{\{u\} = \{u\} \cup y \to \{\} = y\}, x = \{u\} \cup y)$$

не сохраняет выполнимость, так как возможен вариант, когда $y \neq \{\}$ и $x = \{u\} \cup y$, не учтенный в этом правиле. Но с учетом принципа при таком варианте x оказалось бы уже разбито и имело вид $\{v\} \cup z$, а для равенства $\{v\} \cup z = \{u\} \cup y$ имеется соответствующее правило. Таким образом, тактика τ не корректна, но достаточна для проверки выполнимости условий корректности программ данного вида. Заметим, что в одну сторону выполнимость все-таки сохраняется. Если формула выполнима, то переписанная с помощью системы R формула также выполнима. Значит, если в результате получена невыполнимая формула, то исходная формула (отрицание условия корректности) также невыполнима, а условие корректности истинно.

В результате применения тактики τ получим формулы теории равенства, выполнимость которых проверяется с помощью базовой тактики, описанной в п.2.2 Таким образом, получена тактика для условий корректности программы копирования файла. Следовательно, эту программу можно верифицировать автоматически.

5.1.2. Линейный поиск

Еще одной часто встречающейся в программах редактирования текстов операцией является операция принадлежности $\in$. Приведем пример такой программы. Пусть t — сокращение для терма

$$tuple(nar(a, l, r)).$$

Аннотированная программа линейного поиска элемента u в массиве a между границами l и r имеет вид

$$\{in\}\textbf{function } search : (a, l, r, u) \to (i);$$
$$a := upd(a, r + 1, u);\ i := 1;$$
$$\{inv\}\textbf{while } a[i] \neq u \textbf{ do } i := i + 1 \textbf{ od}; \{out\},$$

где

$$\begin{array}{ll} in: & left(a) \leq l \leq r \wedge r + 1 \leq right(a) \wedge \omega_s \notin x \wedge t = x; \\ out: & (i = r + 1 \vee x[i - l + 1] = u) \wedge in; \\ inv: & (l < i \Rightarrow u \notin tuple(nar(a, l, i - 1))) \wedge in. \end{array}$$

Тактика для условий корректности этой программы строится следующим образом. На первом шаге элиминируются операции над массивами. В результате отрицания условий корректности сводятся к формулам

на кортежах, содержащим операции принадлежности $\in$, равенства на кортежах и элементах кортежей, выборки элемента из кортежа .[.] и операции арифметики Пресбургера. Тактика для проверки выполнимости формул с равенством на кортежах и элементах кортежей дана в п.5.1.1 Тактика для формул с операциями $\in$ и .[.] рассмотрена в [6]. Формулы арифметики Пресбургера проверяются с помощью базовой тактики, описанной в п.2.1.1 Таким образом, тактика проверки выполнимости отрицаний условий корректности программы линейного поиска получена. Значит, эту программу тоже можно верифицировать автоматически.

5.2. Сортировка массивов

Программы сортировки массивов делятся на простые и улучшенные. К простым программам относятся пузырьковая и шейкерная сортировки, сортировка с выбором максимального элемента, сортировка простой вставкой и другие программы, не требующие вычислений над сложными стуктурами данных. К улучшенным программам сортировки относятся сортировка Шелла, пирамидальная и быстрая сортировки, а также другие программы, использующие вычисления над более сложными структурами данных, чем массивы. Так, вычисления проводятся в пирамидальной сортировки на пирамиде, закодированной в массиве, в сортировке Шелла — над выборками из массива с произвольным целочисленным шагом, а в быстрой сортировке — над упорядоченными разбиениями массива. Чтобы показать возможности применения СПФ к программам сортировки массивов, возьмем в качестве представителя простых программ сортировку простой вставкой, а в качестве представителя улучшенных программ — быструю сортировку.

5.2.1. Сортировка простой вставкой

Пусть A — сокращение для формулы

$$\neg(\omega_s \in tuple(a)) \wedge left(a) = 1 \wedge right(a) = n \wedge 1 \leq n.$$

Пусть на элементах массива определен некоторый частичный порядок $\leq$. Все перечисленные выше программы сортировки массивов используют операции: упорядоченности кортежа ord, отношение $\leq$ на кортежах, определяемое аксиомой

$$x \leq y \Leftrightarrow (\forall u \in x \forall v \in y(u \leq v)),$$

и операцию *tuplemset*, преобразующую кортеж в мультимножество с теми же элементами. Аннотированная программа сортировки массива a простой вставкой имеет следующий вид:

$$\begin{array}{l}\{in\}\textbf{function } straightinsertion : (a) \to (a);\\ \{inv_1\}\textbf{for } i := 2 \textbf{ to } n \textbf{ do}\\ \ u := a[i];\ j := i;\\ \ \{inv_2\}\textbf{while } u < a[j-1] \wedge 2 \le j \textbf{ do}\\ \ \ a := upd(a, j, a[j-1]);\ j := j-1;\ \textbf{od};\\ \ a := upd(a, j, u);\ \textbf{od}; \{out\},\end{array}$$

где

$$\begin{array}{ll} in: & A \wedge tuplemset(tuple(a)) = x,\\ out: & in \wedge ord(tuple(a)),\\ inv_1: & ord(tuple(nar(a, 1, i-1))) \wedge input,\\ inv_2: & A \wedge 2 \le i \le n \wedge 1 \le j \le i \wedge \{u\} \le tuple(nar(a, j, i)) \wedge\\ & tuplemset(tuple(upd(a, j, u))) = x \wedge\\ & ord(tuple(nar(a, 1, i-1))) \wedge\\ & (j < i \Rightarrow ord(tuple(nar(a, 1, i)))).\end{array}$$

Общий принцип, облегчающий верификацию программ сортировки массивов и файлов, состоит в том, что факт сохранения тех же самых элементов в массиве или файле можно доказывать отдельно от факта получения на выходе программы упорядоченного массива и файла.

Сохранение одних и тех же элементов в массиве доказывается с помощью следующей тактики. На первом шаге элиминируются операции над массивами. В результате условия корректности сводятся к формулам арифметики Пресбургера и формулам равенства на мультимножествах. Для тех и других базовые тактики рассмотрены в п.2

Построим тактику, проверяющую, что выходом программы является упорядоченный массив. На первом шаге элиминируются операции над массивами. Для всех операций, входящих в полученные формулы, соответствующие тактики описаны в [6]. Поэтому нетрудно построить тактику проверки выполнимости отрицаний условий корректности как комбинацию тактик для этих операций. Следовательно, программу сортировки простой вставкой можно верифицировать автоматически.

Условия корректности других простых программ сортировки массивов включают те же операции с добавлением только операции взятия целой части от деления на два — div_2 — которую можно элиминировать

с помощью тактики, описанной в [6]. Таким образом, все простые программы сортировки массивов можно верифицировать автоматически.

5.2.2. Быстрая сортировка

Пусть $t(m,n)$ — сокращение для терма $tuple(nar(a,m,n))$. Аннотированная программа быстрой сортировки массива a в границах от l до r имеет следующий вид:

$$
\begin{array}{l}
\{in\}\textbf{function } sort : (a,l,r) \to (a);\\
i := l;\ j := r;\ u := a[div_2(l+r)];\\
\textbf{repeat}\\
\ \{inv_1\}\textbf{while } a[i] < u \textbf{ do } i := i+1 \textbf{ od};\\
\ \{inv_2\}\textbf{while } u < a[j] \textbf{ do } j := j-1 \textbf{ od};\\
\ \textbf{if } i \le j \textbf{ then}\\
\ \ w := a[i];\ a := upd(a,i,a[j]);\ a := upd(a,j,w);\\
\ \ i := i+1;\ j := j-1 \textbf{ fi};\\
\textbf{until } i > j; \{inv_3\}\\
\textbf{if } l < j \textbf{ then } sort(a,l,j) \textbf{ fi};\\
\textbf{if } i < r \textbf{ then } sort(a,i,r) \textbf{ fi}; \{out\},
\end{array}
$$

где

$$
\begin{array}{ll}
in: & \neg(\omega_s \in t(l,r)) \wedge left(a) \le l \le r \le right(a);\\
out: & ord(t(l,r)) \wedge in;\\
inv_1: & (l < i \Rightarrow t(l,i-1) \le \{u\}) \wedge l \le i \wedge in;\\
inv_2: & (l < i \Rightarrow t(l,i-1) \le \{u\}) \wedge (j < r \Rightarrow \{u\} \le t(j+1,r)) \wedge\\
 & l \le i \wedge j \le r \wedge in;\\
inv_3: & (l < i \Rightarrow t(l,i-1) \le \{u\}) \wedge (j < r \Rightarrow \{u\} \le t(j+1,r)) \wedge\\
 & l \le i \wedge j \le r \wedge in.
\end{array}
$$

Условия корректности данной программы содержит те же операции, что и программы сортировки простой вставкой, за исключением операции div_2. Для операции div_2 тактика, элиминирующая ее, описана в [6]. Таким образом, программу быстрой сортировки можно верифицировать автоматически.

5.3. Сортировка файлов естественным слиянием

Для спецификации программ сортировки файлов естественным слиянием потребуются две новые операции — $ordnum$ и $couple$. Выражение

$ordnum(x)$ обозначает максимальное число упорядоченных кортежей, на которые можно разбить кортеж x так, чтобы они не накладывались друг на друга. Формула $couple(x, y)$ истинна тогда и только тогда, когда кортежи x и y непустые и последний элемент кортежа x меньше или равен (относительно частичного порядка $\leq_e$) первого элемента кортежа y. Пусть $A(w), B, C, D$ — сокращения для формул

$$\neg eof(w) \Rightarrow x \leq buffer(w) \wedge couple(left(f), \{buffer(w)\}),$$
$$A(g) \wedge A(h),$$
$$a = tuplemset(right(f) \cup left(g) \cup left(h)) \wedge eof(g) \wedge eof(h),$$
$$0 \leq l \wedge eof(f) \wedge a = tuplemset(left(f) \cup right(g) \cup right(h))$$

соответственно.

Входное, выходное условия и инварианты цикла аннотированной программы сортировки файлов естественным слиянием (см. таблицу) имеют следующий вид:

$$\begin{aligned}
in: &\quad a = tuplemset(left(f) \cup right(f)),\\
out: &\quad a = tuplemset(left(f)) \wedge eof(f) \wedge ord(left(f)),\\
inv_1: &\quad D \wedge ordnum(left(f)) \leq l,\\
inv_2: &\quad D \wedge ordnum(left(f)) \leq l + 1 \wedge\\
&\quad (ordnum(left(f)) = l + 1 \Rightarrow B),\\
inv_3: &\quad D \wedge ordnum(left(f)) \leq l + 1 \wedge\\
&\quad (ordnum(left(f)) = l + 1 \Rightarrow B) \wedge\\
&\quad (\neg(x \leq buffer(g)) \vee \neg(y \leq buffer(h)) \vee eof(g) \vee eof(h)),\\
inv_4: &\quad D \wedge ordnum(left(f)) \leq l + 1 \wedge\\
&\quad (ordnum(left(f)) = l + 1 \Rightarrow B) \wedge\\
&\quad (\neg(x \leq buffer(g)) \vee eof(g)),\\
inv_5: &\quad D \wedge ordnum(left(f)) \leq l \wedge (eof(g) \vee eof(h)),\\
inv_6: &\quad D \wedge ordnum(left(f)) \leq l + 1 \wedge\\
&\quad (ordnum(left(f)) = l + 1 \Rightarrow B) \wedge eof(h),\\
inv_7: &\quad D \wedge ordnum(left(f)) \leq l \wedge eof(g),\\
inv_8: &\quad D \wedge ordnum(left(f)) \leq l + 1 \wedge\\
&\quad (ordnum(left(f)) = l + 1 \Rightarrow B) \wedge eof(g),\\
inv_9: &\quad D \wedge ordnum(left(f)) \leq l.
\end{aligned}$$

Все операции, входящие в условия корректности программы, за исключением операций *ordnum* и *couple*, уже рассматривались ранее, и для формул с этими операциями были предложены соответствующие тактики. Тактика τ для операций *ordnum* и *couple* основана на СПФ.

Прежде чем выписать соответствующие СПФ, введем подтип

$$ord[k] = \{x | ordnum(x) = k \wedge x \text{ — кортеж}\}$$

типа всех кортежей, где $k > 0$. Пусть q и q' — элементы типов $ord[k]$ и $ord[k']$ соответственно.

СПФ R, по которой строится тактика τ, состоит из следующих правил:

– для операции $ordnum$:

$$\begin{array}{l}
(\{\neg couple(x,y) | ordnum(x \cup y) \to ordnum(x) + ordnum(y),\\
\quad couple(x,y) | ordnum(x \cup y) \to ordnum(x) + ordnum(y) - 1\},\\
\ ordnum(x \cup y));\\
ordnum(\{u\}) \to 1;\\
ordnum(\{\}) \to 0;\\
ordnum(q) \to k;\\
(\{k > 0 | ordnum(q) \to k, ordnum(\{\}) \to 0\}, ordnum(x));
\end{array}$$

– для операци $couple$:

$$\begin{array}{l}
(\{ordnum(y) = 0 | couple(x \cup y, z) \to couple(x,z),\\
\quad \neg(ordnum(y) = 0) | couple(x \cup y, z) \to couple(y,z)\},\\
\ couple(x \cup y, z));\\
(\{ordnum(y) = 0 | couple(x, y \cup z) \to couple(x,z),\\
\quad \neg(ordnum(y) = 0) | couple(x, y \cup z) \to couple(y,z)\},\\
\ couple(x, y \cup z));\\
couple(\{\}, x) \to false;\\
couple(x, \{\}) \to false;\\
(\{couple(\{\}, q) \to false, k' > 0 | couple(q', q) \to couple(q', q)\},\\
\ couple(x, q));\\
(\{couple(q, \{\}) \to false, k' > 0 | couple(q, q') \to couple(q, q')\},\\
\ couple(q, x));\\
(\{couple(\{\}, y) \to false,\\
\quad couple(x, \{\}) \to false,\\
\quad k' > 0 \wedge k > 0 | couple(q', q) \to couple(q', q)\}, couple(x, y)).
\end{array}$$

Построим теперь тактику для условий корректности данной программы. На первом шаге применяется тактика, элиминирующая операции над файлами. На втором шаге применяется тактика τ. Бинарное

Таблица 1

Программа сортировки естественным слиянием

$\{in\}$**function** $naturalmerge : (f, g, h) \to (f, g, h)$;
repeat
$f := res(f)$; $g := rew(g)$; $h := rew(h)$;
$\{C\}$**while** $\neg eof(f)$ **do**
$x := buffer(f)$;
$\{C\}$**while** $\neg eof(f) \wedge x \leq buffer(f)$ **do**
$g := buf(g, buffer(f))$; $g := put(g)$; $x := buffer(f)$; $f := get(f)$ **od**;
if $\neg eof(f)$ **then** $x := buffer(f)$ **fi**;
$\{C\}$**while** $\neg eof(f) \wedge y \leq buffer(f)$ **do**
$h := buf(h, buffer(f))$; $h := put(h)$;
$y := buffer(f)$; $f := get(f)$ **od**; **od**;
$l := 0$; $f := rew(f)$; $g := res(g)$; $h := res(h)$;
$\{inv_1\}$**while** $\neg eof(g) \wedge \neg eof(h)$ **do**
$x := buffer(g)$; $y := buffer(h)$;
$\{inv_2\}$**while** $\neg eof(g) \wedge \neg eof(h) \wedge x \leq buffer(g) \wedge y \leq buffer(h)$ **do**
if $buffer(g) \leq buffer(h)$
then $f := buf(f, buffer(g))$; $x := buffer(g)$;
$f := put(f)$; $g := get(g)$
else $f := buf(f, buffer(h))$; $y := buffer(h)$;
$f := put(f)$; $h := get(h)$ **fi od**;
$\{inv_3\}$**while** $\neg eof(g) \wedge x \leq buffer(g)$ **do**
$f := buf(f, buffer(g))$; $x := buffer(g)$; $f := put(f)$; $g := get(g)$ **od**;
$\{inv_4\}$**while** $\neg eof(h) \wedge y \leq buffer(h)$ **do**
$f := buf(f, buffer(h))$; $y := buffer(h)$; $f := put(f)$; $g := get(h)$ **od**;
$l := l + 1$;
$\{inv_5\}$**while** $\neg eof(g)$ **do**
$x := buffer(g)$;
$\{inv_6\}$**while** $0\neg eof(g) \wedge x \leq buffer(g)$
$f := buf(f, buffer(g))$; $x := buffer(g)$; $f := put(f)$; $g := get(g)$ **od**
$l := l + 1$ **od**
$\{inv_7\}$**while** $\neg eof(h)$ **do**
$y := buffer(h)$;
$\{inv_8\}$**while** $\neg eof(h) \wedge y \leq buffer(h)$ **do**
$f := buf(f, buffer(h))$; $y := buffer(h)$; $f := put(f)$; $h := get(h)$ **od**;
$l := l + 1$**od**;
until $l = 1; \{inv_9\}\{out\}$

отношение $\preceq$, объединяющее частичный порядок $\leq$ на элементах кортежей и отношение $couple$, суженное на множество $\cup_{k>0} ord[k]$, является частичным порядком. На третьем шаге тактика, соответствующая СПФ, состоящей из правил

$$u \leq v \to u \preceq v;$$
$$couple(q, q') \to q \preceq q';$$
$$couple(\{u\}, q) \to u \preceq q;$$
$$couple(q, \{v\}) \to q \preceq v;$$
$$couple(\{u\}, \{v\}) \to u \preceq v,$$

элиминирует операции $\leq$ и $couple$.

Полученные после третьего шага формулы кроме операций арифметики Пресбургера включают только операцию $\preceq$. Поэтому последним шагом будет применение базовых тактик для арифметики Пресбургера и частичного порядка (в данном случае для $\preceq$). Следовательно, программу сортировки файлов естественным слиянием можно верифицировать автоматически.

ЗАКЛЮЧЕНИЕ

В работе получены следующие результаты.

1. Предложена методология применения систем переписывания формул в автоматической верификации программ, включающая разрешающие процедуры для условий корректности, возникающих при верификации программ редактирования текстов, сортировки массивов и сортировки файлов.

2. Разработан алгоритм проверки выполнимости формул теории равенства мультимножеств.

3. В рамках системы верификации СПЕКТР реализован модуль переписывания формул, базирующийся на СПФ и позволяющий комбинировать системы переписывания формул с другими разрешающими и упрощающими процедурами.

4. Модуль переписывания формул апробирован на примерах верификации из трех классов программ: редактирования текстов, сортировки массивов и сортировки файлов. В качестве программ редактирования текстов были рассмотрены программы копирования файла, линейного и двоичного поиска. Сортировка массивов включала следующие программы: пузырьковую сортировку, сортировку вставкой, шей-

керную сортировку, сортировку Шелла, быструю сортировку и сортировку слиянием. Сортировка файлов проводилась на основе алгоритма простого естественного слияния. Все эти программы верифицируемы автоматически. Доказательство отдельных условий корректности для данных программ занимало от нескольких секунд до нескольких минут в зависимости от их сложности. Полученные результаты показывают, что техника автоматического доказательства, основанная на системах переписывания формул, является достаточно мощным средством автоматизации доказательства и, в частности, автоматизации процесса верификации программ.

В перспективе планируется добавить к модулю переписывания формул еще две части:

– генератор условий сохранения выполнимости, который получал бы на вход систему переписывания формул, а выдавал файл с расширением con, содержащий условия сохранения выполнимости для данной СПФ. Полученные формулы первого порядка могут доказываться вручную или с помощью некоторого универсального доказателя. В настоящее время такой доказатель резолюционного типа разрабатывается в качестве еще одного компонента блока доказательства системы верификации СПЕКТР;

– модуль проверки нетеровости СПФ, который получал бы на вход систему переписывания формул, а выдавал либо ответ, что все СПФ, входящие в этот план, являются нетеровыми относительно стратегии "любой самый внутренний"(в случае, если эти СПФ являются УСЭА с аргументным статусом или униформными), либо ответ, что в план входит СПФ, которая не является ни УСЭА с аргументным статусом, ни униформной СПФ.

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов С.А. Элементы анализа программ. — М.: Наука, 1986. — 128 с.
2. Агафонов В.Н. Спецификация программ: понятийные средства и их организация. — Новосибирск: Наука, 1987. — 240 с.
3. Алагич С., Арбиб М. Построение корректных структурированных программ. — М.: Радио и связь, 1984. — 264 с.
4. Анурeев И. С. Интегрированные правила переписывания термов и их применение в автоматической верификации программ // Проблемы спецификации и верификации параллельных систем. — Новосибирск, 1995. — С. 185—213.
5. Анурeев И. С. Системы переписывания формул. — Новосибирск, 1997. — 22 с. — (Препр./РАН. Сиб. отд-ние. ИСИ; №40).
6. Анурeев И.С. Упрощающие процедуры для типов данных, основанные на систе-

мах переписывания формул. — Новосибирск, 1998. — 38 с. — (Препр./РАН. Сиб. отд-ние. ИСИ; №54).
7. Ануреев И.С. Теория систем переписывания формул. — Новосибирск, 1998. — 30 с. — (Препр./РАН. Сиб. отд-ние. ИСИ; №55).
8. Вирт Н. Алгоримы и структуры данных. — М.: Мир, 1989. — 360 с.
9. Вульф В., Лондон Р., Шоу М. Введение в построение и верификацию программ на языке Альфард // Данные в языках программирования. — М.: Мир, 1982. — С. 123—153.
10. Грис Д. Наука программирования. — М.: Мир, 1984. — 416 с.
11. Дейкстра Э. Дисциплина программирования. — М.: Мир, 1978. — 275 с.
12. Компьютерная алгебра. Символьные и алгебраические вычисления — М.: Мир, 1986. — 392 с.
13. Непомнящий В.А. Верификация программ над массивами // Системная информатика. Вып.3. Программные и вычислительные системы: Методы и языки анализа. — Новосибирск: ВО Наука. Сибирская издательская фирма, 1993. — С. 68—98.
14. Непомнящий В.А., Рякин О.М. Прикладные методы верификации программ. — М.: Радио и связь, 1988. — 256 с.
15. Непомнящий В. А. Сулимов А. А. Верификация программ линейной алгебры в системе СПЕКТР // Кибернетика и системный анализ. — 1992. — №5. — С. 136—144.
16. Непомнящий В. А. Сулимов А. А. Проблемно-ориентированные базы знаний и их применение в системе верификации программ СПЕКТР // Изв. РАН. Сер. Теория и системы управления. — 1997. — №2. — С. 169—175.
17. Рабин М.О. Разрешимые теории // Справочная книга по математической логике. — М.: Наука, 1982. — Т. 3. — С. 77—111.
18. Семенов А.Л. Разрешающие алгоритмы для логических теорий // Кибернетика и вычислительная техника. — М.: Наука, 1986. — Т. 2. — С. 134—146.
19. Ющенко Е.Л., Касаткина И.В. Современные методы доказательства правильности программ // Кибернетика, 1980. — №6. — С. 37—62.
20. Anureev I.S. A method for simplification procedures design based on formula rewriting system // Joint Bulletin of IIS and Computer Center. — 1998. — №8. — (To appear).
21. Apt K.R. Ten years of Hoare's logic: a survey. Part I // ACM Trans. Progr. Lang. Syst. — 1981. — Vol. 3, — №4. — P. 431—483.
22. Burstall R.M., Goguen J.A. The semantics of CLEAR, a specification language // Lect. Notes Comput. Sci. — 1980. — Vol. 86. — P. 292—332.
23. Clarke E.M. Programming language constructs for which it is possible to obtain good Hoare axiom system // J. ACM. — 1979. — Vol. 26, №1. — P. 129—147.
24. Downey P.J., Sethi R., Tarjan R.E. Variations on the common subexpression problem // J. ACM. — 1980. — Vol. 27, №4. — P. 758—771.
25. Ferro A., Omodeo E.G., Schwartz J.T. Decision Procedures for some fragments of set theory // Lect. Notes Comput. Sci. — 1980. — Vol. 87. — P. 88—96.
26. Gerhart S.L. An overview of AFFFIRM: a specification and verification system // Proc. IFIP Congress (Tokyo, Melbourne, 1980). — Amsterdam: North-Holland, 1980. — P. 343—347.
27. Gerhart S.L. Program verification in the 1980's // Proc. Conf. Computing in the 1980's. — Oregon, IEEE, 1978. — P. 80—98.

28. Igarashi S., London R.L., Luckham D.C. Automatic program verification I: a logical basis and its implementation // Acta Informatica. — 1975. — Vol. 4, №2. — P. 145—182.
29. Manna Z., Waldinger R. The logic of computer programming // IEEE Trans. Software Eng. — 1978. — Vol. SE-4, №3. — P. 199—229.
30. Martelli A., Montanari U. An efficient unification algorithm // ACM Trans. Prog. Lang. and Syst. — 1982. — Vol. 4, №2. — P. 258—282.
31. Mateti P. A decision procedure for the correctness of a class of programs // J. ACM. — 1981. — Vol. 28, №2. — P. 215—232.
32. Moriconi M., Schwartz R.L. Automatic construction of verification condition generators from Hoare's logics // Lect. Notes Comput. Sci. — 1981. — Vol. 115. — P. 363—377.
33. Nelson G. Combining satisfiability procedures by equality sharing // Contemporary Mathematics. — 1984. — Vol. 29. — P. 201—211.
34. Nelson G., Oppen D.C. Simplification by cooperating decision procedures // ACM Trans. Progr. Lang. Syst. — 1979. — Vol. 1, №2. — P. 245—257.
35. Nelson G., Oppen D.C. Fast desition procedures based on congruence closure // J. ACM. — 1980. — Vol. 27, №2. — P. 356—364.
36. Nepomniaschy V.A., Sulimov A.A. Problem-Oriented Means of Program Specification and Verification in Project SPECTRUM // Lect. Notes Comput. Sci. — 1993. — Vol. 722. — P. 374—378.
37. Nepomniaschy V.A., Sulimov A.A. Towards Automatic Program Verification: Problem-Oriented Knowledge Bases // Specification, verification and net models of concurrent systems. — Novosibirsk, 1994. — P. 138—150
38. Oppen D.C. Reasoning about recursively defined data structures // J. ACM. — 1980. — Vol. 27, №3. — P. 403—411.
39. Schwartz J.T. A survey of program proof technology. — New York Univ., 1978. — (Dep. Comput. Sci.; Rep. №001).
40. Shostak R.E. Deciding linear inequalitities by computing loop residues // J. ACM. — 1981. — Vol. 28, №4. — P. 769—779.
41. Shostak R.E., Schwartz R., Melliar-Smith P.M. STP: A mechanized logic for specification and verification // Lect. Notes Comput. Sci. — 1984. — Vol. 170. — P. 1—42.
42. Suzuki N., Jefferson D. Verification decidability of Presburger array programs // J. ACM. — 1980. — Vol. 27, №1. — P. 191—205.

И. С. Ануреев

ПРИМЕНЕНИЕ СИСТЕМ ПЕРЕПИСЫВАНИЯ ФОРМУЛ В АВТОМАТИЧЕСКОЙ ВЕРИФИКАЦИИ ПРОГРАММ

Препринт
55

Рукопись поступила в редакцию 26.06.98
Рецензент Ф. А. Мурзин
Редактор Л. А. Карева

Подписано в печать 10.08.98
Формат бумаги 60×84 1/16 Объем 2,3 уч.-изд.л., 2,5 п.л.
Тираж 100 экз.

Отпечатано на ризографе “AL Group”, 630090, пр. Акад. Лаврентьева, 6