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

И. С. Ануреев

СИСТЕМА МАШИННОЙ ПОДДЕРЖКИ ДОКАЗАТЕЛЬСТВА: ОТ ТАКТИКАЛОВ К ГЕНЕРАТОРУ ТАКТИКАЛОВ

Препринт
96

Новосибирск 2002

В работе предложена новая трехуровневая схема доказателя, в которую к обычным двум уровням тактик и тактикалов добавлен новый третий уровень — генератор тактикалов. Описан язык выполнимых спецификаций тактикалов, определяющий класс программ, реализующих тактикалы. Для него разработаны операционная и аксиоматическая семантики. Аксиоматическая семантика основана на языке логических спецификаций тактикалов, позволяющем задавать предусловие, постусловие и инварианты в логике Хоара. Приведены примеры представления на языке выполнимых спецификаций тактикалов систем переписывания термов, систем переписывания контекстных формул и систем переписывания формул. Системы переписывания контекстных формул, позволяющие представлять все известные комбинации тактик (последовательное применение, попеременное применение, циклическое применение и т. п.), рассматриваются впервые в этой работе.

© Институт систем информатики им. А. П. Ершова СО РАН, 2002

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

I. S. Anureev

A SOFTWARE SUPPORT FOR FORMAL REASONING: FROM TACTICALS TO TACTICAL GENERATOR

Preprint
96

Novosibirsk 2002

A new three level scheme of a prover is suggested in this article. A new third level that presents a tactical generator has been introduced in addition to the usual levels of tactics and tacticals. An executable specification language specifying a class of programs which implement tacticals is described. Operational and axiomatical semantics of this language have been developed. The axiomatical semantics is based on the logical specification language which allows giving precondition, postcondition and invariants in Hoare logic. Examples of presentation of term rewriting systems, context formula rewriting systems, and formula rewriting systems in the executable specification language are given. Context formula rewriting systems allowing presentation of all known combinations of tactics (sequential application, alternate application, cyclic application and so on) have been first considered in this article.

© A. P. Ershov Institute of Informatics Systems, 2002

ВВЕДЕНИЕ

Известные современные доказатели [4, 5, 7, 8, 12] придерживаются двухуровневой схемы.

Первый уровень образуют тактики. С их помощью представляются разрешающие процедуры для конкретных теорий.

Второй уровень представлен тактикалами. Тактикалы позволяют по дополнительной информации строить тактики.

Примером тактикала является функция, которая по системе равенств строит тактику, применяющую их как систему переписывания термов. Другой пример тактикала — функция, которая по двум тактикам строит новую тактику, применяющую их последовательно.

Недостатком двухуровневой схемы является то, что число тактикалов фиксировано.

В работе предлагается трехуровневая схема доказателя. К двум существующим уровням добавляется третий, представленный генератором тактикалов. Генератор по программе на языке выполнимых спецификаций тактикалов строит тактикал. Таким образом, генератор позволяет снять ограничение на число тактикалов.

Язык выполнимых спецификаций тактикалов должен быть достаточно выразительным, позволяя строить все известные тактикалы. Это требование выполняется для любого современного языка программирования, например C++.

Однако помимо этого, язык должен иметь средства проверки корректности построенных тактикалов. Для этого ему необходимо иметь формальную операционную семантику.

Компромиссом между этими двумя требованиями является модельный императивный язык с небольшим числом операторов.

Для обеспечения гибкости и выразительности язык допускает расширение системы типов и добавление новых библиотечных функций над типами данных, используемыми в автоматическом доказательстве (термами, формулами, подстановками, наиболее общими унификаторами и т.п.).

В качестве средства проверки корректности тактикалов, реализованных на этом языке, используется логика Хоара. Операторы модельного языка выбраны так, чтобы упрощать верификацию программ в логике Хоара. Так использование специального вида циклов по последовательностям позволяет применять символический метод элиминации инвариантов цикла [3, 11].

Работа имеет следующую структуру. Язык логических спецификаций тактикалов, позволяющий задавать предусловие, постусловие и инварианты циклов в логике Хоара, дан в разд. 1 Язык выполнимых спецификаций тактикалов приведен в разд. 2 Его операционная семантика приведена в разд. 3 Аксиоматическая семантика языка рассматривается в разд. 4 Представление известных тактикалов на этом языке демонстрируется в разд. 5 Тактикал для систем переписывания термов [6, 9] рассматривается в разд. 5.1 Формализм, позволяющий представлять все известные комбинации тактик (последовательное применение, попеременное применение, циклическое применение и т. п.), предложен в разд. 5.2 Этот формализм — системы переписывания контекстных формул — также задается программой на языке выполнимых спецификаций тактикалов. Стратегия применения систем переписывания формул “самый внутренний” [1] реализована в разд. 5.3.

Работа частично поддержана грантами РФФИ 00-01-00909 и 01-01-06141.

1. ЯЗЫК ЛОГИЧЕСКИХ СПЕЦИФИКАЦИЙ ТАКТИКАЛОВ

Язык логических спецификаций тактикалов предназначен для описания свойств тактикалов, представленных на языке выполнимых спецификаций тактикалов. Он рассматривается раньше языка выполнимых спецификаций тактикалов, потому что последний наследует от него базовые операции и систему типов. Основу языка составляет бестиповое лямбда-исчисление [2]. Также формулы языка могут снабжаться метками, позволяющими расширять язык. С помощью меток вводятся, например, ограниченные кванторы, система типов, сорта формул.

1.1. Синтаксис языка

Сигнатура языка Σ определяется как тройка (F, U, L), где F — множество операций, U — универсум, L — множество меток. Множество операций F содержит специальное значение $false$.

Мы не вводим понятие λ-терма, считая любой λ-терм формулой, истинной тогда и только тогда, когда ее значение не равно $false$.

Формулы сигнатуры Σ над множеством переменных X строятся следующим образом:

1) если $A \in F \cup X$, то A — формула;

2) если A, A' — формулы, то $A\ A'$ — формула;

3) если $A_1, \ldots, A_n$ — формулы, то $(A_1, \ldots, A_n)$ — формула;

4) если $x \in X$, A — формула, то $x.A$ — формула;

5) если $x_1, \ldots, x_n \in X$, A — формула, то $(x_1, \ldots, x_n).A$ — формула;

6) если A — формула, s — метка, то $lab(s, A)$ — (помеченная) формула.

Пусть $FVar(A)$ обозначает множество свободных переменных формулы A.

1.2. Семантика языка

Интерпретацией I сигнатуры Σ называется отображение, сопоставляющее универсуму U множество $I(U)$ и каждой операции из F семейство тотальных функций из $I(U)^k$ в $I(U)$, где k — неотрицательное целое число.

Говорят, что интерпретация I сигнатуры Σ расширена на множество переменных X, если она сопоставляет каждой переменной $x \in X$ элемент $I(x)$ множества $I(U)$.

Интерпретация, расширенная на множество переменных X, распространяется на формулы сигнатуры Σ.

1. Если $A \in F \cup X$, то $I(A)$ задано по определению I.

2. Если A, A' — формулы, то $I(A\ A') = I(A)(I(A'))$ (применение функции $I(A)$ к аргументу $I(A')$).

3. Если $A_1, \ldots, A_n$ — формулы, то

$$I(A_1, \ldots, A_n) = (I(A_1), \ldots, I(A_n))$$

(кортеж формул $I(A_1), \ldots, I(A_n)$).

4. Если $x \in X$ — переменная, A — формула, то $I(x.A)$ — тотальная функция f из $I(U)$ в $I(U)$ такая, что для любого $a \in I(U)$ выполнено равенство $f(a) = I'(A)$, где интерпретация I' отличается от I только, возможно, значением на переменной x, причем $I'(x) = a$.

5. Если $x_1, \ldots, x_n \in X$, A — формула, то $I((x_1, \ldots, x_n).A)$ — тотальная функция f из $I(U)^n$ в $I(U)$ такая, что для любых $a_1, \ldots, a_n \in I(U)$ выполнено равенство $f(a_1, \ldots, a_n) = I'(A)$, где интерпретация I' отличается от I только, возможно, значением на переменных $x_1, \ldots, x_n$, причем все $I'(x_i) = a_i$.

6. Если A — формула, s — метка, то $I(lab(s, A)) = I(A)$ (метка не влияет на семантику формулы).

Формула A ложна при интерпретации I, расширенной на множество переменных X, если $I(A) = I(false)$. В противном случае формула A истинна при интерпретации I.

Формула A истинна при интерпретации I, если $I'(A) \neq I'(false)$ для любого расширения I' интерпретации I на множество переменных X. В противном случае формула A ложна при интерпретации I.

Формула A истинна, если A истинна при любой интерпретации I. В противном случае формула A ложна.

1.3. Типы данных

Типы в языке задаются формулами. Множество элементов типа, задаваемого формулой — это множество значений, на которых формула истинна. Система базовых типов данных языка логических спецификаций тактикалов является расширяемой.

Для описания примеров тактикалов нам потребуются функции int, $rational$, $formula$, $label$, pos, $subst$, $tactic$, $tactical$ для задания базовых типов данных.

Формула $int(x)$ ($rational(x)$, $formula(x)$, $label(x)$, $pos(x)$, $subst(x)$, $tactic(x)$, $tactical(x)$) истинна тогда и только тогда, когда x — целое число (рациональное число, формула, метка, позиция, подстановка, тактика, тактикал).

Из базовых типов данных могут образовываться новые типы с помощью конструкторов типов данных $(\_, \ldots, \_)$, $seq(\_)$ и $+$.

Пусть $t, t_1, \ldots, t_n$ — типы данных. Формула $(t_1, \ldots, t_n)(x)$ ($seq(t)(x)$, $(t_1 + t_2)(x)$) истинна тогда и только тогда, когда x — кортеж из n элементов типов $t_1, \ldots, t_n$ (последовательность элементов типа t, элемент объединения типов t_1 и t_2).

Кортежи образуются с помощью конструктора $(\_, \ldots, \_)$, описанного в определении терма сигнатуры Σ. Кортеж $(x_1, \ldots, x_n)$ имеет тип $(t_1, \ldots, t_n)$, если элементы $x_1, \ldots, x_n$ имеют типы $t_1, \ldots, t_n$ соответственно.

1.4. Функции на типах данных

На типах данных определены функции, набор которых может расширяться. В дальнейшем для примеров нам потребуются следующие функции:

- функции над целыми: $\ldots,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ \ldots, +, \leq$;
- функции над формулами: $true$, $false$, || (дизъюнкция), && (конъюнкция), ! (отрицание), $\Rightarrow$ (импликация), $\forall$, $\exists$;
- функции над позициями:
  - $replace(A, q, B)$ — формула A, в позицию q которой подставлена формула B;
  - $posterm(A, q)$ — подформула, находящаяся в позиции q формулы A;
  - $pos(A)$ — последовательность позиций формулы A сверху-вниз, слева-направо;
  - $innerpos(A)$ — последовательность позиций формулы A снизу-вверх, слева-направо;
- функции над метками:
  - $labp(A)$ — $true$, если $A = lab(s, A')$ для некоторых формулы A' и метки s и $false$ иначе;
  - $lab(A)$ — метка s такая, что $A = lab(s, A')$ для некоторой формулы A';
  - $unlab(A)$ — формула A' такая, что $A = lab(s, A')$ для некоторой метки s;
  - $instance(A, B)$ — подстановка σ такая, что $\sigma(A) = B$;
  - $mgu(A, B)$ — наиболее общий унификатор формул A и B;
- функции над последовательностями:
  - nil — пустая последовательность;
  - $cons(b, A)$ — последовательность, полученная из последовательности A добавлением b в начало A;
  - $rcons(A, b)$ — последовательность, полученная из последовательности A добавлением b в конец A;
  - $A + B$ — конкатенация последовательностей A и B;
  - $choose(A)$ — первый элемент A;
  - $rest(A)$ — последовательность, полученная из последовательности A удалением первого элемента;
  - $empty(A)$ — $true$, если A — пустая последовательность (не содержит элементов) и $false$ иначе.

Заметим, что для удобства некоторые функции задаются не в префиксной форме.

1.5. Тактики и тактикалы

Тактики определяются как функции, преобразующие формулы в последовательности формул. Если тактика неприменима, то она выдает специальное неопределенное значение ω.

Тактика τ называется корректной, если для любой формулы A из $\tau(A) \neq \omega$ следует, что формула A выполнима тогда и только тогда, когда выполнима хотя бы одна из формул последовательности $\tau(A)$.

Тактикал — это функция, которая возвращает в качестве результата тактику. На вход тактикал может получать любую информацию, необходимую для вычисления тактики. В рассматриваемом языке логических спецификаций тактикалов тип аргумента тактикала строится только с помощью конструкторов типов из базовых типов.

2. ЯЗЫК ВЫПОЛНИМЫХ СПЕЦИФИКАЦИЙ ТАКТИКАЛОВ

Переменные и система типов данных языка выполнимых спецификаций тактикалов и языка логических спецификаций тактикалов совпадают. Поэтому требуется рассмотреть только небольшое число операторов языка выполнимых спецификаций тактикалов, изменяющих значения переменных. Для каждого оператора дается неформальная семантика. Формальная семантика рассматривается в разделе 3

2.1. Программа

Программа на языке выполнимых спецификаций тактикалов имеет вид

`tactical(s); OL`

где `s` — имя тактикала, `OL` — последовательность операторов.

2.2. Оператор-формула

Оператор-формула имеет вид

`A;`

где `A` — формула. Помимо переменных программы в A допускаются свободные вхождения означиваемых переменных вида $x?$, где x — переменная программы или новая переменная. Выполнение оператора состоит в проверке выполнимости формулы A. Если она невыполнима,

то программа заканчивает свою работу с текущими значениями переменных. В противном случае выбирается любой набор значений означиваемых переменных, при которых формула A истинна и переменные, для которых имеются соответствующие означиваемые переменные в A, принимают значения этих означиваемых переменных.

Пример. Пусть переменные x и y определены в программе и имеют значения 0 и 1 соответственно. Тогда выполнение оператора $x? = x + y;$ присвоит переменной x значение 1. Действительно, формула $x? = 0 + 1$ истинна, когда $x? = 1$. □

Пример. Выполнение оператора $int(x?)\&\&0 \leq x? \leq 1;$ присвоит переменной x целочисленное значение 0 или 1. □

2.3. Оператор-блок

Оператор блок имеет вид

$$\{OL\},$$

где OL — последовательность операторов. Выполнение оператора блока состоит в последовательном выполнении операторов из последовательности OL. В отличие от таких императивных языков, как C++ и Java, имеется только одна глобальная область видимости имен. С помощью блока нельзя переопределять переменные.

Пример. На примере блока $\{int(x?)\&\&0 \leq x? \leq 1;\ rational(x);\}$ покажем, что язык выполнимых спецификаций тактикалов является бестиповым относительно имен переменных. Выполнение первого оператора блока присвоит переменной x целочисленное значение 0 или 1. Выполнение второго оператора присвоит переменной x любое рациональное число. □

2.4. Условный оператор

Условный оператор имеет вид

`if(A) I else J`

где I, J — операторы (ветви условного оператора), A — формула (условие условного оператора) такого же вида, как и формула в операторе-формуле. Ветвь else может опускаться. Если формула A выполнима, то выполняется ветвь I, в противном случае — ветвь J.

2.5. Оператор цикла

Оператор цикла имеет вид

$$\texttt{for(c;A)I}$$

где c — формула, построенная только из переменных и конструктора $(\_, \ldots, \_)$ (счетчик цикла), A — формула такая, что $seq(t)(A)$ истинна для некоторого типа t (последовательность просмотра), I — оператор (тело цикла).

Последовательность просмотра имеет такой же вид, как и формула в операторе-формуле.

На счетчики циклов накладываются следующие ограничения:

1) означиваемые переменные не входят в счетчик;

2) счетчики могут иметь не более одного вхождения любой переменной;

3) переменные разных счетчиков различны.

При первом выполнении цикла осуществляется инициализация переменных цикла c, $\hat{c}$ и $\hat{c}$ с помощью последовательности операторов

$$\texttt{empty(\^{}c);\,c?} = \texttt{choose(A);\,\^{}c?} = \texttt{rest(A);}$$

Здесь $c?$ означает, что все переменные образца c становятся означиваемыми (после них ставится ?). Счетчик c соответствует счетчику цикла в таких языках программирования, как C++ и Pascal. Переменная цикла $\hat{c}$ задает последовательность, из которой счетчик c берет свои значения (начиная с головы последовательности). Переменная $\hat{c}$ используется для хранения последовательности значений, выдаваемых при каждой итерации цикла.

Если счетчик имеет сложный вид, то вместо него в переменных цикла $\hat{c}$ и $\hat{c}$ можно использовать любую переменную, входящую в этот счетчик.

После каждого выполнения тела цикла значения переменных c и $\hat{c}$ изменяются с помощью последовательности операторов

$$\texttt{c?} = \texttt{choose(\^{}c);\,\^{}c?} = \texttt{rest(\^{}c);}$$

и значение переменной $\hat{c}$ проверяется на пустоту.

В случае пустой последовательности цикл завершает свою работу. В противном случае, тело цикла выполняется очередной раз.

Переменные цикла $\hat{c}$ и $\hat{c}$ могут изменяться в теле цикла, а счетчик c нет.

Пример. Цикл, который по последовательности формул L строит последовательность отрицаний этих формул и сохраняет новую последовательность в той же переменной L, выглядит следующим образом:

$$\texttt{for(A;L)\^{}A? = rcons(\^{}A, A); L? = \^{}A;}$$

□

2.6. Оператор цикла с несколькими счетчиками

Оператор цикла с несколькими счетчиками имеет вид:

$$\texttt{for}(\texttt{c}_1;\texttt{A}_1;\ldots;\texttt{c}_\texttt{n};\texttt{A}_\texttt{n})\ \texttt{I}$$

где c_i — счетчики, A_i — соответствующие им последовательности просмотра, i — тело цикла.

Все ограничения на счетчики, переменные цикла и последовательности просмотра и соглашения для них, имеющие место для циклов с одним счетчиком, сохраняются.

При первом выполнении цикла осуществляется инициализация переменных цикла

$$c_1, \hat{c}_1, c_1\hat{}, \ldots, c_n, \hat{c}_n, c_n\hat{}$$

с помощью последовательности операторов

$$\texttt{empty(\^{}c}_1\texttt{?); c}_1\texttt{\^{} ? = A}_1\texttt{; }\ldots\texttt{; empty(\^{}c}_\texttt{n}\texttt{?); c}_\texttt{n}\texttt{\^{} ? = A}_\texttt{n}\texttt{;}$$

После каждого выполнения тела цикла значения переменных

$$c_1, \ldots, c_n, c_1\hat{}, \ldots, c_n\hat{}$$

изменяются с помощью последовательности операторов

$$\texttt{c}_1\texttt{? = choose(c}_1\texttt{\^{}); c}_1\texttt{\^{}? = rest(c}_1\texttt{\^{}); }\ldots\texttt{; c}_\texttt{n}\texttt{? = choose(c}_\texttt{n}\texttt{\^{}); c}_\texttt{n}\texttt{\^{}? = rest(c}_\texttt{n}\texttt{\^{});}$$

и значения переменных

$$c_1\hat{}, \ldots, c_n\hat{}$$

проверяются на пустоту.

Цикл заканчивается, когда значением хотя бы одной из переменных цикла $c_i\hat{}$ становится пустая последовательность. В противном случае тело цикла выполняется очередной раз.

Пример. Цикл, который по последовательностям формул L и M строит последовательности конъюнкций и дизъюнкций соответствующих формул в исходных последовательностях и сохраняет последовательность конъюнкций в переменной L, а последовательность дизъюнкций в M, выглядит следующим образом:

```
for(A;L;B;M){A? = rcons(A, A&&B);B? = rcons(B, A||B); }
L? =A; M? =B;
```

□

2.7. Операторы передачи управления

Операторы передачи управления *continue* и *break* во многом аналогичны соответствующим операторам в других императивных языках программирования.

Оператор *continue* имеет вид

```
continue(c);
```

Он производит передачу управления на следующий шаг цикла со счетчиком c.

Оператор *break* имеет вид

```
break(c);
```

Он выполняет выход из цикла со счетчиком c и передачу управления на первый оператор после этого цикла.

В случае сложного счетчика его можно заменять в операторах передачи управления на любую переменную этого счетчика. В силу уникальности переменных счетчика такая замена не приведет к неоднозначности.

Примеры операторов *continue* и *break* приведены в разд. 5

2.8. Специальные переменные

Среди переменных выделяются три переменные со специальными именами.

Любая программа на языке выполнимых спецификаций тактикалов задает реализацию некоторого тактикала.

Переменная с именем *input* определяет входную формулу для тактики, порождаемой этим тактикалом.

Переменная с именем *output* определяет выходную последовательность формул для тактики, порождаемой этим тактикалом.

Эти переменные имеют тип $seq(formula)$ и не требуют объявления.

Переменная с именем inf задает информацию, подаваемую на вход тактикалу, на основе которой строится тактика. Тип переменной inf определяет, в каком формате должны подаваться данные на вход тактикалу.

3. ОПЕРАЦИОННАЯ СЕМАНТИКА ЯЗЫКА ВЫПОЛНИМЫХ СПЕЦИФИКАЦИЙ ТАКТИКАЛОВ

Операционная семантика языка выполнимых спецификаций тактикалов представляется в виде множества пар конфигураций абстрактной вычислительной машины, связанных отношением перехода $\rightarrow$. Конфигурация — это пара (P, s), где P — программа (или ее фрагмент), s — состояние памяти абстрактной машины (отображение из множества переменных программы в множество значений).

Помимо программы или ее фрагмента в качестве первого элемента конфигурации допускаются специальные значения:

$$OK,\ STOP,\ CONTINUE(c),\ BREAK(c).$$

Значение OK означает завершение программы. Значение $STOP$ служит для передачи управления на конец программы.

Константы $CONTINUE(c)$ и $BREAK(c)$ появляются в конфигурации при обработке операторов $continue(c)$; и $break(c)$; и просачиваются через другие операторы до оператора цикла. Таким образом, с их помощью в операционной семантике определяется передача управления соответствующими операторами.

Распространим понятие состояния на формулы. Пусть $s(A)$ обозначает результат замены всех свободных вхождений переменных в формуле A на их значения в состоянии s.

Отношение перехода $\rightarrow$ определяется индукцией по программе.

3.1. Программа

Отношение перехода для программы имеет вид:

$$(tactical(n); OL, s) \to (OK, s'),$$

где s' — состояние такое, что $(OL, s) \to (VAL, s')$, VAL имеет вид OK, $STOP$, $BREAK(c)$ или $CONTINUE(c)$.

3.2. Оператор-формула

Пусть $del?(A)$ — результат удаления знака ? в формуле A у означиваемых переменных.

Отношение перехода для оператора-формулы имеет вид:

- $(A; s) \to (OK, s')$, где s' — состояние такое, что $s'(x) = s(x)$ для любой переменной, не входящей в множество $FVar(del?(s(A))$, и формула $s'(del?(s(A)))$ истинна;
- $(A; s) \to (STOP, s)$, если нет такого состояния s'.

Пример. Пусть $s(x) = 0$. Тогда $(x? = x + 1, s) \to (OK, s')$, где s' — состояние такое, что $s'(y) = s(y)$ для любой переменной y, отличной от x, и $s'(x) = s(x) + 1 = 1$. □

3.3. Оператор-блок

Пусть OL, OL' — последовательности операторов.

Отношение перехода для оператора-блока сводит оператор блока к последовательности операторов:

$$(\{OL\}\ OL', s) \to (OL\ OL', s).$$

В свою очередь, отношение перехода для последовательности операторов имеет вид:

- $(I\ OL, s) \to (OL, s')$, если $(I, s) \to (OK, s')$;
- $(I\ OL, s) \to (STOP, s')$, если $(I, s) \to (STOP, s')$;
- $(I\ OL, s) \to (BREAK(c), s')$, если $(I, s) \to (BREAK(c), s')$;
- $(I\ OL, s) \to (CONTINUE(c), s')$, если

$$(I, s) \to (CONTINUE(c), s');$$

- $(, s) \to (OK, s)$.

В последнем правиле рассматривается пустая последовательность перехода.

3.4. Условный оператор

Отношение перехода для условного оператора имеет вид:

- $(if(A)\ I\ else\ I', s) \to (I, s')$, если $(A; s) \to (OK, s')$;
- $(if(A)\ I\ else\ I', s) \to (I', s')$, если $(A; s) \to (STOP, s')$.

3.5. Оператор цикла

Отношение перехода для оператора цикла имеет вид:

$$(for(c; A)I, s) \to (\{empty(\hat{c}?); \hat{c}\ ? = A; for1(c; I); \}, s),$$

где $for1$ — вспомогательный оператор, отношение перехода для которого, в свою очередь, имеет вид:

- $(for1(c; I);, s) \to (OK, s')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (STOP, s');$$

- $(for1(c; I);, s) \to (for1(c; I);, s'')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (OK, s')$$

  и $(I, s') \to (OK, s'')$;
- $(for1(c; I);, s) \to (STOP, s'')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (OK, s')$$

  и $(I, s') \to (STOP, s'')$;
- $(for1(c; I);, s) \to (OK, s'')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (OK, s')$$

  и $(I, s') \to (BREAK(c), s'')$;
- $(for1(c; I);, s) \to (BREAK(c'), s'')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (OK, s'),$$

  $(I, s') \to (BREAK(c'), s'')$ и $c' \neq c$;
- $(for1(c; I);, s) \to (for1(c; I);, s'')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (OK, s')$$

  и $(I, s') \to (CONTINUE(c), s'')$;
- $(for1(c; I);, s) \to (CONTINUE(c'), s'')$, если

  $$(\{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c}); \}, s) \to (OK, s'),$$

  $(I, s') \to (CONTINUE(c'), s'')$ и $c' \neq c$.

3.6. Оператор цикла с несколькими счетчиками

Отношение перехода для оператора цикла с несколькими счетчиками определяется аналогично отношению перехода для оператора цикла с одним счетчиком, но выглядит более громоздко.

Пусть $\overrightarrow{c;}$ и $\overrightarrow{c;A}$ обозначают последовательности счетчиков

$$c_1 \dots c_n;$$

и пар "счетчик—последовательность просмотра"

$$c_1; A_1; \dots; c_n; A_n$$

Пусть

$$\overrightarrow{empty(\hat{c}?); \hat{c}\ ? = A;}$$

и

$$\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}$$

обозначают последовательности операторов

$$\mathtt{empty(\hat{c}_1?); \hat{c_1}\ ? = A_1; \dots; empty(\hat{c}_n?); \hat{c_n}\ ? = A_n;}$$

и

$$\begin{array}{c} \mathtt{c_1? = choose(\hat{c_1});\ \ \hat{c_1}\ ? = rest(\hat{c_1});\ \ !empty(\hat{c_1});} \\ \dots \\ \mathtt{c_n? = choose(\hat{c_n});\ \ \hat{c_n}\ ? = rest(\hat{c_n});\ \ !empty(\hat{c_n});} \end{array}$$

Отношение перехода для оператора цикла с несколькими счетчиками имеет вид:

$$(for(\overrightarrow{c;A})I, s) \to (\{\overrightarrow{empty(\hat{c}?); \hat{c}\ ? = A;} for1(\overrightarrow{c;}I);\}, s),$$

где $for1$ — вспомогательный оператор, отношение перехода для которого, в свою очередь, имеет вид:

- $(for1(\overrightarrow{c;}I);, s) \to (OK, s')$, если

  $$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (STOP, s');$$

- $(for1(\overrightarrow{c;}I);, s) \to (for1(\overrightarrow{c;}I);, s'')$, если

  $$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (OK, s')$$

  и $(I, s') \to (OK, s'')$;

- $(for1(\overrightarrow{c;}I);,s) \to (STOP, s'')$, если

$$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (OK, s')$$

  и $(I, s') \to (STOP, s'')$;
- $(for1(\overrightarrow{c;}I);,s) \to (OK, s'')$, если

$$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (OK, s')$$

  и $(I, s') \to (BREAK(c_i), s'')$ для некоторого $1 \le i \le n$;
- $(for1(\overrightarrow{c;}I);,s) \to (BREAK(c), s'')$, если

$$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (OK, s'),$$

  $(I, s') \to (BREAK(c), s'')$ и $c \ne c_i$ для любого $1 \le i \le n$;
- $(for1(\overrightarrow{c;}I);,s) \to (for1(\overrightarrow{c;}I);,s'')$, если

$$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (OK, s')$$

  и $(I, s') \to (CONTINUE(c_i), s'')$ для некоторого $1 \le i \le n$;
- $(for1(\overrightarrow{c;}I);,s) \to (CONTINUE(c), s'')$, если

$$(\{\overrightarrow{c? = choose(\hat{c});\ \hat{c}\ ? = rest(\hat{c});\ !empty(\hat{c});}\}, s) \to (OK, s')$$

  и $c \ne c_i$ для любого $1 \le i \le n$.

3.7. Операторы передачи управления

Отношение перехода для операторов передачи управления имеет вид:

- $(continue(c);,s) \to (CONTINUE(c), s)$;
- $(break(c);,s) \to (BREAK(c), s)$.

Значения $CONTINUE(c)$ и $BREAK(c)$ затем просачиваются наверх через другие операторы до тех пор, пока не встретится цикл с таким счетчиком или не будет достигнут уровень программы.

4. АКСИОМАТИЧЕСКАЯ СЕМАНТИКА

Аксиоматическая семантика для языка выполнимых спецификаций тактикалов определяется как система вывода, роль формул в которой

играют тройки Хоара. Тройка Хоара $\{P\}A\{Q\}$, где P, Q — формулы языка логических спецификаций тактикалов (предусловие и постусловие), A — программа, истинна, если для любого состояния, в котором предусловие P истинно и на котором программа A завершается, постусловие Q истинно после выполнения программы.

При выводе тройки Хоара в этой системе происходит элиминация операторов, и истинность тройки сводится к истинности набора лемм — условий корректности, которые не содержат операторов. Истинность условий корректности определяет истинность тройки Хоара для исходной программы, что позволяет доказывать частичную корректность программы. Методы доказательства завершимости программы, обеспечивающей наряду с частичной корректностью ее тотальную корректность, в данной работе не рассматриваются.

Опишем систему вывода Хоара для языка выполнимых спецификаций тактикалов.

4.1. Программа

Правило вывода для программы имеет вид:

$$\frac{\{PREDCOND\}OL\{POSTCOND\}}{\{PREDCOND\}tactical(n);\ OL\{POSTCOND\}}.$$

4.2. Оператор-формула

Пусть A — формула, $\bar{x}?$ — вектор означиваемых переменных формулы A, $\bar{x}$ — вектор обычных переменных, соответствующих $\bar{x}?$, $\bar{y}$ — вектор свободных неозначиваемых переменных формулы A.

Правило вывода для оператора-формулы имеет вид:

$$\frac{\begin{array}{c}\{P\}OL\{A[\bar{x}? \leftarrow \bar{n}] \Rightarrow Q[\bar{x} \leftarrow \bar{n}]\},\\ \{P\}OL\{(\forall \bar{n}(!A[\bar{x}? \leftarrow \bar{n}])) \Rightarrow POSTCOND\}\end{array}}{\{P\}OL\ A;\{Q\}},$$

где $\bar{n}$ — вектор новых переменных.

4.3. Оператор-блок

Пусть OL — последовательность операторов.

Правило вывода для оператора-блока сводит оператор блока к последовательности операторов:

$$\frac{\{P\}OL\ OL'\{Q\}}{\{P\}OL\{OL'\}\{Q\}}.$$

Правило вывода для пустой последовательности операторов имеет вид:

$$\frac{P \Rightarrow Q}{\{P\}\{Q\}}.$$

4.4. Условный оператор

Пусть A — формула, $\bar{y}$ — вектор свободных неозначиваемых переменных формулы A.

Правило вывода для условного оператора имеет вид:

$$\frac{\begin{array}{c}\{P\}OLn? = A; [n]I\{Q\},\\ \{P\}OLn? = A; [!n]I'\{Q\}\end{array}}{\{P\}OL\ if(A)I\ else\ I'\{Q\}},$$

где n — новая переменная.

Правило вывода для вспомогательного оператора [ ] имеет вид:

$$\frac{\{P\}OL\{n \Rightarrow Q\}}{\{P\}OL[n]\{Q\}}.$$

4.5. Оператор цикла

Правило вывода для оператора цикла имеет вид:

$$\frac{\begin{array}{c}\{P\}OL\ empty(\hat{c}?); \hat{c}\ ? = A; c? = choose(\hat{c}); \{INV\},\\ \{INV\}for1(c; I); \{Q\}\end{array}}{\{P\}OL\{INV\}for(c; A)I\{Q\}},$$

а для вспомогательного оператора $for1$:

$$\frac{\begin{array}{c}\{INV \&\& !empty(\hat{c})\}I\ c? = choose(\hat{c}); \hat{c}? = rest(\hat{c}); \{INV\},\\ INV \&\& empty(\hat{c}) \Rightarrow Q\end{array}}{\{P\}OL\{INV\}for1(c; I); \{Q\}}.$$

Дополнительная формула INV (инвариант цикла) требуется для того, чтобы свести с помощью правила вывода тройку Хоара с циклом к конечному числу троек Хоара. Инвариант цикла должен быть истинным при входе в цикл и сохранять истинность при любом количестве итераций цикла.

4.6. Оператор цикла с несколькими счетчиками

Правило вывода для оператора цикла с несколькими счетчиками определяется аналогично правилу вывода для оператора цикла с одним счетчиком, но имеют более громоздкий вид.

Пусть

$$\overrightarrow{empty(\hat{c}?); \hat{c}\,? = A; c? = choose(\hat{c})};$$

и

$$\overrightarrow{c? = choose(\hat{c}); \hat{c}? = rest(\hat{c})};$$

обозначают последовательности операторов

$$\mathtt{empty(\hat{c}_1?); \hat{c_1}\,? = A_1; c_1? = choose(\hat{c_1})};$$
$$\dots$$
$$\mathtt{empty(\hat{c}_n?); \hat{c_n}\,? = A_n; c_n? = choose(\hat{c_n})};$$

и

$$\mathtt{c_1? = choose(\hat{c_1}); \hat{c_1}? = rest(\hat{c_1})};$$
$$\dots$$
$$\mathtt{c_n? = choose(\hat{c_n}); \hat{c_n}? = rest(\hat{c_n})};$$

Пусть

$$\overrightarrow{!empty(\hat{c})}$$

и

$$\overrightarrow{empty(\hat{c_1})}$$

обозначают формулы

$$!empty(\hat{c_1})\&\& \dots \&\&!empty(\hat{c_n})$$

и

$$empty(\hat{c_1})|| \dots ||empty(\hat{c_n}).$$

Правило вывода для оператора цикла с несколькими счетчиками имеет вид:

$$\frac{\{P\}OL\ \overrightarrow{empty(\hat{c}?); \hat{c}\,? = A; c? = choose(\hat{c})};\{INV\},\ \{INV\}for1(c_1; \dots ; c_n; I);\{Q\}}{\{P\}OL\{INV\}for(c; A)I\{Q\}},$$

а для вспомогательного оператора $for1$:

$$\frac{\{INV\&\&\overrightarrow{!empty(\hat{c})}\}I\ \overrightarrow{c? = choose(\hat{c}); \hat{c}? = rest(\hat{c})};\{INV\},\ INV\&\&\overrightarrow{empty(\hat{c_1})} \Rightarrow Q}{\{P\}OL\{INV\}for1(c_1; \dots ; c_n; I);\{Q\}}.$$

4.7. Операторы передачи управления

Правила вывода для оператора $continue(c)$; имеет вид:

$$\frac{\{P\}OL\ c? = choose(\hat{c}); \hat{c}? = rest(\hat{c}); \{INV(c)\}}{\{P\}OL\ continue(c); \{Q\}},$$

если существует цикл со счетчиком c, охватывающий данное вхождение $continue(c)$; в программу, и

$$\frac{\{P\}OL\{POSTCOND\}}{\{P\}OL\ continue(c); \{Q\}},$$

если такого цикла не существует.

Здесь $INV(c)$ — инвариант такого цикла (единственного в силу уникальности счетчиков).

Правила вывода для оператора $break(c)$; имеет вид:

$$\frac{\{P\}OL\{POSTCOND(c)\}}{\{P\}OL\ break(c); \{Q\}},$$

если существует цикл со счетчиком c, охватывающий данное вхождение $break(c)$; в программу, и

$$\frac{\{P\}OL\{POSTCOND\}}{\{P\}OL\ break(c); \{Q\}},$$

если такого цикла не существует.

Здесь $POSTCOND(c)$ — постусловие в тройке Хоара

$$\{INV\}for1(c; I); \{POSTCOND(c)\},$$

встречавшейся ранее в дереве вывода.

5. ПРИМЕРЫ ТАКТИКАЛОВ

В качестве тактикалов, реализуемых на языке выполнимых спецификаций тактикалов, рассмотрим системы переписывания термов, системы переписывания контекстных формул и системы переписывания формул.

5.1. Системы переписывания термов

Системы переписывания термов (СПТ) — часто применяемая в автоматическом доказательстве техника переписывания формул. Подробно СПТ рассматриваются в [6, 9]. Здесь приводятся только некоторые необходимые определения.

СПТ определяются следующим образом.

Пара $\rho = (l, r)$ называется правилом переписывания термов, если $FVar(l) \subseteq FVar(r)$. Конечное множество правил переписывания термов образует СПТ.

Отношение редукции $\rightarrow_\rho$, порождаемое правилом переписывания формул $\rho = (l, r)$, определяется как множество пар

$$(A, replace(A, q, \sigma(r))),$$

где A — формула, q — позиция в формуле A, σ — подстановка такая, что $\sigma(l) = posterm(A, q)$.

Отношение переписывания, порождаемое СПТ, определяется как объединение отношений переписывания, порождаемых всеми ее правилами.

Опишем тактикал, получающий на вход СПТ и выдающий тактику, применяющую эту СПТ в соответствии со стратегией сверху-вниз, слева-направо:

```
tactical(TRS); seq(term,term) inf; for(A;cons(input,nil))
  {for(q;pos(A))
    {for((l,r);inf)
      {sigma?=instance(l,posterm(A,q));
       if(sigma=omega) continue(r);
       A^?=replace(A,q,sigma(r))+A^;
       continue(A);
      }
    }
   output?=output+A;
  }
```

Программа определяет тактикал с именем TRS. Тактикал получает на вход систему переписывания термов inf. Правила inf применяются до тех пор, пока применимо хотя бы одно правило inf.

5.2. Системы переписывания контекстных формул

Система переписывания контекстных формул — это формализм для описания комбинирования тактик. Каждой формуле в этом формализме приписывается сорт — контекст, который показывает, какие тактики применяются к ней. Формула с приписанным ей сортом называется контекстной формулой. Контекстные формулы переписываются с помощью специальных правил, которые указывают, какая тактика применяется в данном контексте и какие контексты приписываются формулам, полученным в результате применения тактики. Набор таких правил позволяет описать план доказательства — какие тактики и в каком порядке следует применять. С помощью этих правил моделируются многие распространенные операции комбинирования тактик.

Для описания классов (сортов) формул используются метки.

Правилом переписывания контекстных формул ρ называется четверка (s_i, s_o, s_e, τ), где s_i, s_o, s_e — метки, τ — тактика. Конечное множество таких правил образует систему переписывания контекстных формул (СПКФ).

Отношение редукции, порождаемое правилом $\rho = (s_i, s_o, s_e, \tau)$, определяется как множество пар вида

$$(A, lab(s_o, unlab(A_1)) + ... + lab(s_o, unlab(A_n)))$$

для $A_1 + ... + A_n = \tau(A)$ и

$$(A, lab(s_e, unlab(A)))$$

для $\tau(A) = \omega$, где A — формула такая, что $lab(A) = s_0$.

Отношение редукции, порождаемое СПКФ, задается как объединение отношений редукций, порождаемой всеми ее правилами.

Задание плана доказательства с помощью СПКФ позволяет выразить многие часто встречающиеся на практике комбинации тактик.

Пример. Последовательное выполнение тактик τ и τ' представляется системой

$$\{(s_0, s_1, s_1, \tau), (s_1, s, s, \tau')\}.$$ □

Пример. Недетерминированное применение тактик τ и τ' представляется системой

$$\{(s_0, s, s, \tau), (s_0, s, s, \tau')\}.$$

Производится недетерминированный выбор одной из тактик. Затем выбранная тактика применяется. □

Пример. Итерация тактики τ представляется системой

$$\{(s_0, s_1, s, \tau), (s_1, s_1, s, \tau)\}.$$

Она означает применение тактики до тех пор, пока тактика применяется. □

Пример. Попеременное выполнение тактик τ и τ' представляется системой

$$\{(s_0, s_1, s_2, \tau), (s_1, s_0, s_3, \tau'), (s_3, s_1, s, \tau), (s_2, s_0, s, \tau')\}.$$

Тактики выполняются попеременно до тех пор, пока хотя бы одна из них применима. □

Пример. Условное применение тактик τ и τ' представляется системой

$$\{(s_0, s, s_1, \tau), (s_1, s, s, \tau')\}.$$

Если тактика τ применима, то она применяется. Иначе применяется тактика τ'. □

Опишем тактикал, получающий на вход СПКФ и выдающий тактику, применяющую эту СПКФ:

```
tactical(SRS); seq(label,label,label,tactic) inf;
for(A;cons(lab(start,input),nil))
  {for((s,s1,s2,tau);inf)
       {if (!(s=lab(A))) continue(s);
         Bs?=tau(A);
         if (Bs=omega)
            {A^?=lab(s2,unlab(A))+A^;
             continue(A);}
         for(B;Bs) A^?=lab(s1,unlab(B))+A^;
         continue(A);
        }
   output?=output+unlab(A);
  }
```

Программа определяет тактикал с именем SRS. Тактикал получает на вход систему переписывания контекстных формул inf. Входной формуле *input* для тактики, порождаемой этим тактикалом, сначала приписывается метка (сорт) start. Поэтому СПКФ inf должна содержать правило, начинающееся с метки start. С этого правила начинает применяться inf. Оператор программы

$$output? \ = \ output \ + \ unlab(A);,$$

прежде чем помещать формулы, к которым неприменимо ни одно из правил inf, в выходную последовательность output, удаляет из них метки, расставленные при применении правил inf.

5.3. Системы переписывания формул

Системы переписывания формул (СПФ) представляют формализм переписывания формул, комбинирующий такие известные техники переписывания, как системы переписывания термов и сужение. Подробно СПФ рассматриваются в [1]. Здесь приводятся только некоторые необходимые определения. Под формулами ниже понимаются формулы языка логических спецификаций тактикалов, имеющие только свободные вхождения переменных, так как системы переписывания формул применяются только к таким формулам.

СПФ определяются следующим образом.

Пусть B — конечное множество троек формул, s — формула. Пара $\rho = (B, s)$ называется правилом переписывания формул, если для любой тройки $(p, l, r) \in B$ формулы l и s унифицируемы. Конечное множество правил переписывания формул образует систему переписывания формул.

Прежде чем определять отношение переписывания (редукции), порождаемое СПФ, введем понятие тривиального наиболее общего унификатора (ТНОУ) — специального случая наиболее общего унификатора (НОУ). Наиболее общий унификатор двух подстановок называется их тривиальным наиболее общим унификатором, если он находится с помощью алгоритма унификации [10] без применения правила редукции термов.

Отношение редукции $\rightarrow_\rho$, порождаемое правилом переписывания формул $\rho = (\mathcal{B}, s)$, определяется как множество пар

$$(A, \{\phi_l(\sigma(p) \wedge replace(A, q, \sigma(r)))|(p, l, r) \in B\}),$$

где A — формула, q — позиция в формуле A, ϕ_π — тривиальный унификатор подстановок σ и θ_l таких, что $\sigma(s) = posterm(A, q)$ и θ_l — наиболее общий унификатор l и s.

Отношение переписывания, порождаемое системой переписывания формул, определяется как объединение отношений переписывания, порождаемых всеми ее правилами.

Опишем тактикал, получающий на вход СПФ и выдающий тактику, применяющую эту СПФ в соответствии со стратегией снизу-вверх,

слева-направо.

Чтобы описать тактикал, расширим множество функций над подстановками функцией $tmgu$ такой, что $tmgu(\sigma, \sigma')$ обозначает тривиальный наиболее общий унификатор подстановок σ и σ'.

Тактикал имеет вид:

```
program(FRS); seq(seq(term,term,term),term) inf;
for(A;cons(input,nil))
  {for(q;innerpos(A))
    {for((B;s),inf)
      {sigma?=instance(l,posterm(A,q));
       if(sigma=omega) continue(B);
       empty(C?);
       for((p,l,r);B)
            {phi?=tmgu(mgu(l,s),sigma);
             if (phi=omega) continue(q);
             C?=C+phi(replace(A,q,sigma(r)));
            }
        A^?=C+A^;
        continue(A);
      }
    }
   output?=output+A;
  }
```

Программа определяет тактикал с именем FRS. Тактикал получает на вход систему переписывания формул inf. Правила inf применяются до тех пор, пока применимо хотя бы одно правило inf. В тактикале для систем переписывания формул использована стратегия снизу-вверх, слева-направо, так как для ряда классов таких систем эта стратегия гарантирует завершимость применения правил.

ЗАКЛЮЧЕНИЕ

В работе предложена трехуровневая схема доказателя, в которую к обычным двум уровням тактик и тактикалов добавлен новый третий уровень — генератор тактикалов. В рамках схемы разработаны:

- язык логических спецификаций тактикалов, позволяющий задавать предусловие, постусловие и инварианты циклов в логике Хоара;

- язык выполнимых спецификаций тактикалов, который определяет класс программ, реализующих тактикалы;
- формальная операционная и аксиоматическая семантика языка выполнимых спецификаций тактикалов;
- системы переписывания контекстных формул — новый формализм, позволяющий представлять все известные комбинации тактик (последовательное применение, попеременное применение, циклическое применение и т. п.);

и приведены примеры представления на языке выполнимых спецификаций тактикалов ряда средств автоматического доказательства (систем переписывания термов, систем переписывания контекстных формул, систем переписывания формул).

СПИСОК ЛИТЕРАТУРЫ

1. **Ануреев И. С.** Теория систем переписывания формул. — Новосибирск, 1998. — 35 с. — (Препр./РАН. Сиб. отд-ние. ИСИ; №54).
2. **Барендрехт Х.** Ламбда-исчисление. Его синтаксис и семантика. — М.: Мир, 1985. — 606 с.
3. **Непомнящий В. А.** Верификация финитной итерации над структурами данных // Кибернетика и системный анализ. — 1999. — №3. — С. 25–37.
4. **Boyer R. S., Moore J. S.** A Computational Logic Handbook. — Academic Press, 1988. — 385 p.
5. **Chen J., Han. J.** A Review of EVES. — St. Lucia, 1993. — (Tech. Rep. / Department of Computer Science/ University of Queensland; №93-5).
6. **Dershowitz N., Jouannaud J. P.** Rewrite systems. Handbook of Theoretical Computer Science. — 1990. — Vol. B(6). — P. 243–320.
7. **Gerhart S. L., Musser D. R. et al.** An overview of AFFIRM: A specification and verification system // Proc. of IFIP Congress 80. — IFIP Congress Ser. — 1980. — Vol. 8. — P. 343–347.
8. **Gordon M. J. C., Melham T. F.** Introduction to HOL: a theorem proving environment for higher order logic. — Cambridge University Press, 1993.
9. **Klop J. W.** Term rewriting systems. // Handbook of Logic in Computer Science. — 1993. — Vol. 2. — P. 1–116.
10. **Martelli A., Montanari U.** An efficient unification algorithm // ACM Trans. on Prog.Lang. and Syst. — 1982. — Vol. 4, №2. — P. 258–282.
11. **Nepomniaschy V. A.** Symbolic verification method for definite iteration over data structures // Information Processing Letters. — 1999. — Vol. 69, №4. — P. 207–213.
12. **Owre S., Shankar N., Rushby J. M.** User Guide for the PVS Specification and Verification System. — Computer Science Laboratory, SRI International, Menlo Park, CA, 1993.

И. С. Ануреев

СИСТЕМА МАШИННОЙ ПОДДЕРЖКИ ДОКАЗАТЕЛЬСТВА: ОТ ТАКТИКАЛОВ К ГЕНЕРАТОРУ ТАКТИКАЛОВ

Препринт
96

Рукопись поступила в редакцию 26.02.2002
Рецензент Ф. А. Мурзин
Редактор З. В. Скок

Подписано в печать 15.04.2002
Формат бумаги 60×84 1/16 Объем 1,7 уч.-изд.л., 1,8 п.л.
Тираж 50 экз.

НФ ООО ИПО "Эмари" РИЦ, 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6