УДК 519.686.2

ОБРАТНАЯ ТРАНСФОРМАЦИЯ ФОРМУЛ В СИМВОЛЬНОМ МОДЕЛИРОВАНИИ: ОТ РЕЗУЛЬТАТА К ИСХОДНОЙ ФОРМУЛЕ

А.Б. Годлевский, С.В. Потиенко

Институт кибернетики имени В.М. Глушкова НАН Украины,
03680, Киев, проспект Академика Глушкова, 40.
Тел.: (+38) (044) 526 0058.
stepan@iss.org.ua

В символьном моделировании транзиционных систем аналогом их состояний являются логические формулы над множеством атрибутов. Правила переходов исходной системы служат основой для построения моделирующих правил, которые действуют на формульных состояниях и определяют переход от исходного состояния к последующему или предшествующему ему состоянию. Предикатный трансформер – это алгоритм, моделирующий переходы в пространстве формульных состояний. В работе описан обратный предикатный трансформер, отличительной чертой которого является то, что моделируемые им переходы допускают как операторы присваивания над простыми и составными структурами данных (массивами и списками), так и констрейнты, задающие изменения атрибутов в неявном виде.

Logical formulas over attributes set are analog of system states in symbolic modeling of transition systems. Transition rules of source system are the base for building of modeling rules which operate on formula state and define transition from source state to the next or previous state. Predicate transformer is an algorithm which models transitions in the formula state space. We describe backward predicate transformer which has distinguishing feature that modeled transitions allow as assignment operators over simple and complex data structures (arrays and lists) as constraints which define attribute changes implicitly.

Введение

Трансформация формул под действием операторов программ активно изучалась в рамках подхода к верификации программ, предложенного Флойдом [1] и развитого затем Хоаром [2]. Рассматривались логические формулы над переменными программы, привязываемые к ее состоянию и описывающие свойства программы при передаче управления в это состояние. По отношению к оператору, исполняемому в заданном состоянии программы, требовалось определить, каким соотношениям будут удовлетворять значения переменных после его выполнения, или, в символьной постановке, каким образом оператор трансформирует логическую формулу, имевшую место до его выполнения. Эта задача известна как построение постусловия по заданному условию и оператору. Двойственной к ней является задача построения предусловия по условию и оператору, когда исходная формула выражает соотношение переменных после выполнения оператора и требуется определить, каким соотношениям должны удовлетворять переменные программы, чтобы этот оператор приводил к заданной формуле. Сложность этих задач по трансформации формул зависела как от языка представления формул, в частности используемых в них предикатов и структур данных, так и от семантики самих операторов.

Интерес авторов к задаче трансформации формул связан с технологией проверки свойств программ на моделях (model checking), когда для исследования достижимых состояний применяется символьное моделирование, позволяющее существенно редуцировать пространство поиска. В данной работе рассматривается обратное моделирование, которое соответствует построению предусловий. Рассмотрение проводится в рамках модели, заданной набором базовых протоколов [3], что, как формализм, соответствует правилам перехода в транзиционных системах. Предлагаемый здесь механизм преобразования формул будем называть (обратным) предикатным трансформером.

Отличительной чертой постановки являются язык формул и семантика трансформаций. Формулы рассматриваются как над простыми атрибутами, так и над атрибутами функционального типа (массивами) и списками. Допускаются чтение, добавление и удаление элементов как по отношению к началу, так и к концу каждого списка. Базовые протоколы синтаксически состоят из двух частей, предусловия, определяющего возможность применения к состоянию среды, и постусловия, которое определяет преобразование атрибутов[1]. Постусловия задают высокоуровневые преобразования данных и включают в себя три компоненты, исполняемые параллельно: присваивания, обработку списков и констрейнты, специфицирующие часть атрибутов. Таким образом, изменение одних атрибутов явно задается присваиваниями, для других же их новые значения задаются неявно в виде ограничений.

Работа является развитием [4], здесь устранены имевшиеся ошибки, детально рассмотрен алгоритм отождествления аргументов функциональных символов и расширено описание примеров.

[1] Мы используем термин атрибут системы вместо термина переменная программы, а также термины предусловие и постусловие как структурные составляющие базового протокола, но не как результаты трансформации одних формул в другие.

© А.Б. Годлевский, С.В. Потиенко, 2010

1. Обозначения

На каждом шаге моделирования известно текущее состояние среды (на первом шаге известно начальное состояние). Применимые в этом состоянии протоколы модифицируют состояние согласно своим постусловиям. Каждый из одновременно применимых протоколов создает новую ветвь в дереве поведения системы.

Описанные далее пред- и постусловия, а так же формулы, определяющие состояния среды, могут содержать связанные переменные ($\exists x$). Здесь они явным образом не участвуют, так как обрабатываются в других процедурах и в данный алгоритм изменений не вносят.

Пред- и постусловие каждого базового протокола представляется в таком виде:

– предусловие: $A(r,l,s,z)$;
– постусловие: $B(r,l,s,z) = R(r,l,s,z) \wedge U(l,r,s,z) \wedge C(r,s)$.

Здесь:

– l – вектор списковых атрибутов;
– r,s,z – вектора атрибутов и атрибутных выражений, кроме списков;
– $A(r,l,s,z)$ и $C(r,s)$ – формулы базового языка; A может содержать операторы доступа к спискам *get_from_head*, *get_from_tail* и *empty*, C – нет;
– $U(l,r,s,z)$ – конъюнкция операторов обновления списков l;
– $R(r,l,s,z)$ – конъюнкция присваиваний новых значений атрибутам r, может содержать операторы доступа к спискам *get_from_head* и *get_from_tail* в правых частях:
– $(r_1(r,l,s,z) := t_1(r,l,s,z)) \wedge (r_2(r,l,s,z) := t_2(r,l,s,z)) \wedge \ldots$;
– $r_i(r,s,z) := t_i(r,s,z)$ – присваивания после подстановки элементов списков вместо операторов доступа (*get_from_head*(l), *get_from_tail*(l)); подстановки осуществляются после восстановления списков (см. раздел 2), так как здесь нужно использовать "старые" значения;
– z – вектор переменных, отсутствующих в атрибутных выражениях верхнего уровня формулы $C(r,s)$ постусловия и левых частей присваивания.

Таким образом, r – это левые части присваиваний и функциональные выражения, которые рекурсивно зависят от левых частей. Они представляют собой атрибутные выражения, значения которых меняются присваиваниями. Значения s могут измениться недетерминированным образом, поскольку входят в условие C. Значения выражений из z не меняются.

Пусть E и E' – формулы, определяющие символьные состояния среды, которые, в свою очередь, покрывают множества конкретных состояний. Для простоты, формулы E будем называть состояниями. Они представляются в таком виде:

– $E = D(r,s,z) \wedge L(r,l,s,z)$,
– $E' = D'(r,s,z) \wedge L'(r,l,s,z)$,

где

– $D(r,s,z)$, $D'(r,s,z)$ – формулы базового языка [4] (могут содержать связанные переменные);
– $L(r,l,s,z)$, $L'(r,l,s,z)$ – списковые равенства вида:
  - $(l_1 = list(h_1(r,s,z), h_2(r,s,z), ..., Nil; q_1(r,s,z), q_2(r,s,z), ..., Nil)) \wedge \ldots \wedge$
    $\wedge \ldots \wedge (l_2 = list(c_1(r,s,z), c_2(r,s,z), ..., Nil)) \wedge \ldots$;
  - $h_i(r,s,z)$, $q_i(r,s,z)$, и $c_i(r,s,z)$ – выражения соответствующего типа;
  - списки могут содержать абстрактную (неизвестную) часть, как l_1, или иметь конкретную длину, как l_2.

Базовый протокол применим на состоянии среды E, если формула $E \wedge A(r,l,s,z)$ не ложна. Применимый базовый протокол осуществляет переход:

$$E \rightarrow E' .$$

Во время прямого моделирования известно состояние E, с помощью прямого предикатного трансформера необходимо найти состояние E':

$$E' = pt(E \wedge A(r,l,s,z), B(r,l,s,z)).$$

Во время обратного моделирования известно состояние E', с помощью обратного предикатного трансформера необходимо найти состояние E:

$$E = pt^{-1}(E', A(r,l,s,z), B(r,l,s,z)).$$

2. Формульная часть

Рассмотрим формульную часть постусловия $B(r,l,s,z)$ без обновления списков:

$$R(r,l,s,z) \wedge C(r,s).$$

Постусловие кроме операторов присваивания и обновления списков содержит формулу базового языка. Обновление списков производится по правилам, описанным далее. Рассмотрим влияние полученной формулы $C(r,s)$ на формулу D.

Операторы присваивания и формула $C(r,s)$ изменяют формулу состояния среды как описано в прямом предикатном трансформере [4]. Далее речь идет об обратном моделировании.

Условие валидности постусловия определяется такой формулой: $R(r,s,z) \wedge C(r,s)$. Условие применимости – $D' \wedge R \wedge C$. Так же необходимо учитывать списки: если в части постусловия U есть операторы *add_to_head* или *add_to_tail*, то соответствующие списки не должны быть пустыми в L' (но могут быть абстрактными). Пусть эти условия выполняются.

Построим формулу

$$D(r,l,s,z) = \exists(y,u)\ (\ D'(u,l,y,z) \wedge u = t(r,l,s,z) \wedge C(u,y) \wedge P(r,s,z,y)\ ) \wedge A(r,l,s,z),$$
$$P(r,s,z,y) = P_1(r,s,z,y) \vee P_2(r,s,z,y) \vee \ldots \qquad (1)$$

Здесь y и u – векторы связанных переменных, введенных для обозначения значений, которые имеют атрибуты s и r до выполнения обратного трансформера.

Выше упоминалось, что r, s, z – векторы атрибутов и атрибутных выражений не списковых типов, т.е. простых типов – числовые (целые и действительные), перечислимые, символьные, а так же функциональных типов. Массивы могут рассматриваться как функциональные типы с целочисленными или перечислимыми параметрами, целочисленные ограничены размерностями массива. Эти ограничения имеют вид неравенств: $i \geq 0 \wedge i < N$, где i – индексное выражение, N – размерность массива, и должны учитываться при проверке выполнимости формул. Если один атрибут функционального типа встречается в пред- или постусловии и/или состоянии E' (включая списки) более одного раза, должны быть рассмотрены все комбинации возможных отождествлений его аргументов. Формула $P_i(r,s,z,y)$ определяет одну из таких возможностей. Так как P – это дизъюнкция всех P_i, состоящих из равенств и неравенств аргументов, то она тождественно равна 1. Т.е. присутствие P в формуле (1) не влияет на состояние среды, описываемое этой формулой.

Рассмотрим метод построения P с учетом такого ограничения: разрешается только один уровень вложенности аргументов, т.е. выражение в аргументе не может содержать функциональных символов (например, $a(a(x))$ запрещено).

Сначала рассмотрим все аргументы функциональных символов, входящих в B и E'. Предусловие A пока не рассматриваем, так как оно остается неизменным. Обозначим эти аргументы как w_i^k, где индекс k обозначает место аргумента и тип содержащего функционального символа, индекс i – перечисляет все вхождения этого аргумента. Например, для формулы $g(a,b,c) > g(d,e,f)$, обозначения аргументов будут выглядеть так: $w_1^1 = a,\ w_1^2 = b,\ w_1^3 = c,\ w_2^1 = d,\ w_2^2 = e,\ w_2^3 = f$.

1. Сделаем подстановки атрибутов из r на $t(r,l,s,z)$ согласно присваиваниям в постусловии, атрибутов из s – на связанные переменные y_i, так как новое состояние E' содержит новые значения всех атрибутов.

2. Построим отождествление P как дизъюнкцию всех возможных комбинаций P_i равенств и неравенств аргументов. Для этого надо раскрыть скобки в формуле: $P = \forall(i,j,k) \bigwedge_{i \neq j} (w_i^k = w_j^k \vee w_i^k \neq w_j^k)$.

3. Далее, к первой части формулы (1) без предусловия A будем применять функцию отождествления $\eta(F(r,s,z),\ P_i(r,s,z))$ для каждого P_i. Определим функцию $\eta(F,\ P_i)$ как замену в формуле F аргументов функциональных символов, исходя из равенств, присутствующих в P_i. Замена происходит следующим образом: пусть в F есть два функциональных символа $a(x)$ и $a(v)$, а в P_i есть равенство $x = v$. В случае, когда оба символа $a(x)$ и $a(v) \in r$, отождествлять $x = v$ нельзя, так как в постусловии запрещено делать более одного присваивания одному атрибуту. Если же один из символов $a(x) \in r$, то v заменить на x, что бы остались вхождения атрибутного выражения из r, вместо которого потом можно будет подставить $t(\ldots)$. Если же $a(x) \in s$, $a(v) \in z$, то будет аналогичная замена, останутся вхождения атрибутного выражения из s, вместо которых потом можно будет подставить подкванторные переменные y. Если символы $a(x)$ и $a(v)$ из одного вектора s или z, то можно сделать любую замену. Обозначим: $D_i' = \eta(D', P_i)$, $C_i = \eta(C, P_i)$.

См. примеры в разделе 4.

Если формула $D(r,s,z)$ ложна, то данный базовый протокол не мог быть применен и соответствующая ветвь поведения не рассматривается.

Таким образом:

$$pt^{-1}(\ D'(r,s,z)\ ,\ A(r,l,s,z),\ B(r,l,s,z)\ ) =$$
$$= \exists y \bigvee_i (\ D_i'(t(r,l,s,z),y,z) \wedge C_i(\ t(r,l,s,z),y) \wedge P_i(r,s,z,y)\ ) \wedge A(r,l,s,z).$$

3. Восстановление списков

Операторы обновления списков $U(r,l,s,z)$ изменяют списковые равенства в состоянии среды:

$$pt(L(r,l,s,z),\ U(r,l,s,z)) = L'(r,l,s,z),$$

$$pt^{-1}(L'(r,l,s,z),\ A(r,l,s,z),\ U(r,l,s,z)) = L(r,l,s,z).$$

$U(r,l,s,z)$ содержит операторы add_to_tail(l, f(r,s,z)), add_to_head(l, f(r,s,z)), remove_from_tail(l, f(r,s,z)), remove_from_head(l, f(r,s,z)).

$L(r,l,s,z)$ содержит равенства вида (см. раздел 1):

$$l = list(h_1(r,s,z),\ \ldots,\ h_m(r,s,z),\ Nil;\ q_1(r,s,z),\ \ldots,\ q_n(r,s,z),\ Nil).$$

Голова или хвост списка могут быть пусты, тогда $l = list(Nil;\ q_1(r,s,z),\ \ldots,\ Nil)$ или $l = list(h_1(r,s,z),\ \ldots,\ Nil;\ Nil)$ соответственно.

Полностью абстрактный список содержит только неизвестную часть: $l = list(Nil;\ Nil)$.

Восстановление списков и генерация списковых равенств $L(l,r,s,z)$ осуществляется по правилам:

1) для оператора $add\_to\_tail(l,f(r,s,z))$:

– если $L'(l,r,s,z)$ содержит равенство $l = list(h_1(r,s,z), \ldots, Nil; q_1(r,s,z), \ldots, q_n(r,s,z), Nil)$, то применяем *remove_from_tail*(l);
тогда $L(l,r,s,z)$ содержит равенство
$\exists y\ (\ l = list(h_1(t(r,s,z),y,z), \ldots, Nil; q_1(t(r,s,z),y,z), \ldots, q_{n-1}(t(r,s,z),y,z), Nil)\ )$;
в формулу F необходимо добавить равенство $f(r,s,z) = q_n(t(r,s,z),y,z)$;
– если $L'(l,r,s,z)$ содержит равенство $l = list(h_1(r,s,z), \ldots, Nil; Nil)$, то неизвестная часть остается неизвестной;
тогда $L(l,r,s,z)$ содержит равенство $\exists y\ (\ l = list(h_1(t(r,s,z),y,z), \ldots, Nil; Nil)\ )$;

2) для оператора *add_to_head*(l,$f(r,s,z)$):

– если $L'(l,r,s,z)$ содержит равенство $l = list(h_1(r,s,z), \ldots, Nil; q_1(r,s,z), \ldots, Nil)$, то применяем *remove_from_head*(l);
тогда $L(l,r,s,z)$ содержит равенство
$\exists y\ (\ l = list(h_2(t(r,s,z),y,z), \ldots, Nil; q_1(t(r,s,z),y,z), \ldots, Nil)\ )$;
в формулу F необходимо добавить равенство $f(r,s,z) = h_1(t(r,s,z),y,z)$;
– если $L'(l,r,s,z)$ содержит равенство $l = list(Nil; q_1(r,s,z), \ldots, Nil)$, то неизвестная часть остается неизвестной;
тогда $L(l,r,s,z)$ содержит то же равенство $\exists y\ (\ l = list(Nil; q_1(t(r,s,z),y,z), \ldots, Nil)\ )$;

3) для оператора *remove_from_tail*(l):

– пусть $L'(l,r,s,z)$ содержит равенство $l = list(h_1(r,s,z), \ldots, Nil; q_1(r,s,z), \ldots, q_n(r,s,z), Nil)$;
вводим новую переменную v и применяем *add_to_tail*(l,v);
тогда $L(l,r,s,z)$ содержит равенство
$\exists(y,v)\ (\ l = list(h_1(t(r,s,z),y,z), \ldots, Nil; q_1(t(r,s,z),y,z), \ldots, q_n(t(r,s,z),y,z), v, Nil)\ )$.

4) для оператора *remove_from_head*(l):

– пусть $L'(l,r,s,z)$ содержит равенство $l = list(h_1(r,s,z), \ldots, Nil; q_1(r,s,z), \ldots, Nil)$;
вводим новую переменную v и применяем *add_to_head*(l,v);
тогда $L(l,r,s,z)$ содержит равенство
$\exists(y,v)\ (\ l = list(v, h_1(t(r,s,z),y,z), \ldots, Nil; q_1(t(r,s,z),y,z), \ldots, Nil)\ )$.

У списков конкретной длины отсутствует неизвестная часть, правила обновления применяются те же. Необходимо лишь отдельно рассмотреть случай, когда состояние E' содержит пустой список $l = list()$, а постусловие базового протокола оператор *add_to_tail*(l) или *add_to_head*(l). Тогда этот протокол не мог быть применен и соответствующая ветвь поведения не рассматривается.

$$pt^{-1}(\ D'(r,s,z) \wedge L'(l,r,s,z), A(r,l,s,z), B(r,l,s,z)\ ) =$$
$$= D(r,l,s,z) \wedge \exists y(L(l,t(r,s,z),y,z)) \wedge M(r,s,z).$$

Здесь $M(r,s,z)$ – конъюнкция равенств, добавленных во время восстановления списков для операторов *add_to_tail* и *add_to_head*.

4. Примеры с отождествлением аргументов

Пример 1. Столбцы в таблице описывают: (A) – предусловие, (R) – оператор присваивания, (C) – условие, E' – состояние среды, (P) – соотношения между атрибутами, задающие конкретные формулы. Будем говорить о «новых» и «старых» значениях атрибутов, полагая, что новые – результат применения базового протокола к состоянию со старыми значениями. Однако, поскольку мы рассматриваем обратное моделирование, то исходными для предикатного трансформера будут «новые» значения, а результатом моделирования будет реконструкция «старых» значений, точнее тех соотношений, которые должны были выполняться, чтобы базовый протокол был применимым.

Трасформацию условий будем представлять в виде таблицы, в столбцах которой будут соответственно представлены: (A) – предусловие базового протокола, (R) – оператор присваивания, (U) – оператор обновления списков, (E') исходное условие, (P) равенства и их отрицания, описывающие разные случаи отождествления аргументов функциональных выражений. Условимся также новые значения в таблицах выделять жирным шрифтом. В рассматриваемом примере отсутствуют операторы обработки списков, а сам пример задается в первой строке табл. 1. К «новым» значениям относятся: формула E' и значения атрибутов a[i] и a[k], формируемые постусловием базового протокола. Заметим, что в (R) и (C) обновляются только значения массива, но не значения индексов у элементов этого массива.

Описание среды для этого примера будет следующим: массив a из трех элементов, целочисленные атрибуты i, j и k. Интерпретация строк таблицы (кроме первой, в которой представлено описание базового протокола и исходного условия) соответствует разбору разных случаев интерпретации исходной формулы E'. Эти случаи соответствуют разным соотношениям индексов i, j и k, представленным в столбце P.

Каким образом «новые» значения должны подставляться в формулу E'? Во-первых, если $i=k$, то результат присваивания может подставляться в условие (C); во-вторых, поскольку значение индекса i не менялось ни в присваивании R, ни в условии C, то в формуле E' новым значением элемента $\mathbf{a}[i]$ будет значение $a[k]+1$, где $a[k]$ – это старое значение; в-третьих, $\mathbf{a}[j]$ может заменяться на $\boldsymbol{a}[k]$ при $j=k$, но не на $\boldsymbol{a}[i]$ (поскольку из $i=j$ должно следовать $a[i]=a[j]$, но эти элементы в E' не совпадают).

Итак, относительно значений индексов, встречающихся в выражениях для элементов массива допустимы три комбинации, перечисленные в столбце P. Для каждого значения в столбце P осуществляем подстановку в C и E'.

Таблица 1

(*A*) Precond	(*R*) Assign	(*C*) Cond	(*U*) List	*E'*	(*P*) Cases
a[i]+a[j]+a[k]=5	**a**[i]:=a[k]+1	**a**[k]<=0		**a[i]>a[j]**	
	a[i]=a[k]+1	**a**[k]<=0		a[k]+1>**a[j]**	i≠k, j≠k, i≠j
	a[i]=a[k]+1	**a**[k]<=0		a[k]+1>a[k]	i≠k, j=k, i≠j
	a[i]=a[k]+1	a[k]+1<=0		a[k]+1>**a[j]**	i=k, i≠j

Рассмотрим подробно случай, когда i≠k, j≠k, i≠j, соответствующий второй строке табл. 1. Новые значения элементов **a**[i] и **a**[k] определяются присваиванием, для **a**[i], и условием для **a**[k]. При этом новое значение **a**[i] выражается как a[k]+1 через старые, а новое значение **a**[k] от старых не зависит и лишь ограничено сверху числом 0. Значение a[j] базовым протоколом не изменяется. Поэтому неравенство **a[i]>a[j]**, присутствующее в E′ для новых значений трансформируется в a[k]+1>a[j] для старых, неравенство **a**[k]<=0 заменяется на формулу (Exist y)(y<0), отражающую замену **a**[k] связанной переменной y. С учетом формулы предусловия, выполнимость которой есть условие применимости базового протокола, предикатный трансформер строит формулу

(Exist y)(a[i]+a[j]+a[k]=5) &((y<0)&(a[k]+1>a[j])&(i≠j)&(i≠k)&(j≠k),

соответствующую данному соотношению индексов.

Обратим внимание на последнюю строку табл. 1 (случай i=k & i≠j). Поскольку i=k, то в условие (*C*) подставляется значение **a**[i], порождаемое присваиванием, и условие (*C*) трансформируется к виду a[k]+1<0, которому должны удовлетворять старые значения атрибутов. С учетом всех трех случаев формула будет иметь вид:

```
(Exist y)(a[i]+a[j]+a[k]=5)&(i≠j)&(
        (y<0)&(a[k]+1>a[j])&(i≠k)&(j≠k) |/
        (y<0)&(a[k]+1>y)   &(i≠k)&(j=k) |/
        (a[k]+1<0)&(a[k]+1>a[j]) & (i=k)
        )
```

Замечание 1. Предикатный трансформер, должен был бы вначале построить полную дизъюнкцию из 5 членов, включая 2 члена, когда i=j, а затем удалить эти два члена должна уже процедура проверки выполнимости, которая встроена в трансформер, чтобы проверять не тривиальность результата и упрощать результирующую формулу.

Замечание 2. Предусловие базового протокола попадает в формулу в неизменном виде и может измениться только в результате упрощений, если раскрыть скобки дизъюнкции.

Замечание 3. Если присваивания базового протокола будут изменять атрибуты, входящие в индексные выражения, то выражения в столбце (*P*) определяющие частные случаи трансформации формулы, могут зависеть не только от атрибутов типа r,s,z, но и от связанных переменных, которыми заменяются атрибуты типа s.

Пример 2. Особенность этого примера в том, что здесь присутствует обработка списков. Обратим внимание на то, что формула *E′* характеризует значение элемента, лежащего в списке.

Таблица 2

(*A*) Precond	(*R*) Assign	(*C*) Cond	(*U*) List	*E′*	(*P*) Cases
a[i]+a[j]=5	**a**[i]:=get_from_head(l)	**a**[j]<=0	remove_from_hd(l)	**a[i]+a[j]>0 & & a[2]>=1 & l=list(a[2],Nil)**	
a[i]+a[j]=5	**a**[i]:=v	**a**[j]<=0		**(Exist v) (** **a**[i]+a[j]>0 & **a**[2]>=1 & **l=list(v,a[2],**Nil))	
	a[i]=v	**a**[j]<=0		**(Exist v) (a[2]+a**[j]>0 & **a[2]**>=1 & **l**=list(a[2],a[2],Nil))	i=2, i≠j
	a[i]=v	**a**[j]<=0		**(Exist v) (v+a**[j]>0 & **a[2]>=**1 & l=list(**v,a[2]**,Nil))	i≠2, i≠j

Как и ранее, в первой строке табл. 2 представлены исходные данные для предикатного трансформера. Во второй строке описано преобразование, связанное с восстановлением списка l , и в третьей и четвертой рассматриваются два нетривиальных случая трансформации условия.

В первую очередь восстановим список l, добавив в него некий элемент v (переменную, связанную экзистенциальным квантором). После чего заменим оператор чтения из головы списка значением этого элемента. В результате, задача трансформера преобразуется к следующей:

- присваивание из (*R*) заменяется на a[i]:=v;
- оператор восстановления списков из *U* удаляется (как уже промоделированный);
- список l в *E′* увеличивается путем дописывания в его голову нового элемента.

Этой задаче соответствует вторая строка табл. 2.

Атрибуты типа r и s как и ранее – это a[i] и a[j]. Сопоставляться между собой могут их индексы и индекс 2 элемента из списка l. Рассмотрим возможные случаи.

Случай j=2 отбрасываем как невыполнимый, поскольку значение a[2], большее 1, несовместимо с ограничением a[j]<=0.

Случай i=2 позволяет определить значение связанной переменной v как a[2].

Случай i≠2 и i=j отбрасываем как невыполнимый, связанная переменная v должна быть не больше 0 (условие для a[j]), а тогда сумма a[i]+a[j], равная 2v, должна быть одновременно не больше 0 и больше 0 (условие a[i]+a[j]>0 из E′).

Если теперь в (*C*) и *E'* обновленный атрибут a[j] заменить связанной переменной y, то итоговая формула принимает вид:

(Exist v,y)((a[i]+a[j]=5 & i≠j & y<=0 & a[2]>=1) &
(a[2]+y>0 & l=list(a[2],a[2],Nil) |/ v+y>0 & l=list(v,a[2],Nil))

В эту формулу вошли равенство из предусловия (*A*), ограничение i≠j, которое верно для всех выполнимых случаев, условие y<=0 из (*C*), неравенство a[2]>=1 из *E′*, которое вынесено за скобки как общее для обоих дизъюнктов.

Выводы

В данной работе рассмотрена техника обратного символьного моделирования, которая используется для решения задач верификации систем базовых протоколов. Описан алгоритм обратного предикатного трансформера, моделирующего переход транзиционной системы от заданного формульного состояния к предшествующему ему состоянию. Этот алгоритм ориентирован на такие типы данных как integer, real и enumerated, а также на структуры функциональных и списочных типов данных. Обратный трансформер рассматривают как оператор, который по заданной формуле строит новую, определяющую класс состояний системы, из которых возможен переход, совершенный под действием заданного базового протокола.

Следующий важный этап в этой работе, который еще требует своего завершения, состоит в том, чтобы доказать свойство идеальности описанного предикатного трансформера. Суть этого свойства состоит в установлении адекватности формульной модели по отношению к конкретной модели, в которой состояния транзиционной системы задаются наборами значений всех своих атрибутов. Иными словами *s′* – состояние, удовлетворяющее формульному состоянию *E′*, *s* – состояние, к которому применим базовый протокол и которое он переводит в *s′*, то нужно показать, (1) что *s* будет удовлетворять формульному состоянию *E*, которое порождает трансформер, и (2) для каждого состояния *s*, удовлетворяющего формуле *E*, существует состояние *s′*, удовлетворяющее *E′*, и такое, что базовый протокол переводит *s* в *s′*.

1. *Floyd R. W.* «Assigning Meaning to Programs», in Proceedings of Symposium on Applied Mathematics, Vol. 19, J.T. Schwartz (Ed.), A.M.S., 1967. – P. 19–32.
2. *Hoare C. A. R.* «An axiomatic basis for computer programming». Communications of the ACM, 12(10): 576–580, 583 October 1969.
3. *Летичевский А.А., Капитонова Ю.В., Волков В.А., Летичевский А.А. (мл.), Баранов С.Н., Котляров В.П., Вейгарт Т.* Спецификация систем с помощью базовых протоколов // Кибернетика и системный анализ – № 4. –2005.
4. *Потиенко С.В.* Методы прямого и обратного символьного моделирования систем, заданных базовыми протоколами // Проблеми програмування. – 2008. – № 4. – С. 39–45.