КИБЕРНЕТИКА

А.А. ЛЕТИЧЕВСКИЙ, А.Б. ГОДЛЕВСКИЙ, А.А. ЛЕТИЧЕВСКИЙ (мл.), С.В. ПОТИЕНКО, В.С. ПЕСЧАНЕНКО

УДК 519.686.2

СВОЙСТВА ПРЕДИКАТНОГО ТРАНСФОРМЕРА СИСТЕМЫ VRS

Ключевые слова: *верификация, символьное моделирование, абстракции.*

ВВЕДЕНИЕ

Система VRS [1–5] предназначена для верификации требований к программным системам. В качестве формального языка требований и спецификации систем используется язык базовых протоколов. Общая теория базовых протоколов построена в работах [4, 5]. Для символьной верификации множества состояний представляются формулами многосортного исчисления предикатов первого порядка. На множестве этих формул определяется отношение переходов, размеченных базовыми протоколами, которые рассматриваются как действия, выполняемые в системе. В процессе осуществления переходов в пространстве формул постусловие рассматривается как оператор, определенный на множестве формул. Поскольку этот оператор преобразует одни формулы в другие, используем термин «предикатный трансформер», введенный Дикстрой в [6] и распространенный Лемпортом на параллельные программы в [7]. В системе VRS строится предикатный трансформер и доказываются его основные свойства, согласно которым он вычисляет сильнейшее постусловие для символьных состояний. Дается краткий обзор применений предикатного трансформера для верификации требований к программным системам. Частный случай предикатного трансформера для более простого логического языка (простые и функциональные типы данных, операторы присваивания в постусловиях) рассмотрен в работе [8]. В настоящей статье кроме простых и функциональных типов допускаются очереди, а в постусловиях кроме присваиваний допускаются операторы обновления очередей и произвольные формулы. Также, используется подход, основанный на раздельном рассмотрении переходов, определенных постусловиями и базовыми протоколами.

АТРИБУТНЫЕ ВЫРАЖЕНИЯ

Модель системы состоит из множества базовых протоколов B и описания среды E. Среда представляется в виде композиции своего ядра и совокупности агентов, погруженных в среду [9, 10].

Рассмотрим фрагмент описания среды, которая моделирует телефонную сеть (смысл самой модели в данном случае не имеет значения, пример используется для иллюстрации некоторых синтаксических конструкций).

```
environment (
    types:  obj (
        PhoneNumber:  ( phone_1,  phone_2,  phone_3,  phone_4,  NON_DEF),
            PhoneState:  (
```

```
                BUSY_STATE, RINGING_STATE, FREE_STATE,
                CONNECTED_STATE, DIAL_STATE, FASTBUSY_STATE, CWH,
                CALL_WAITING_STATE, THREE_WAY_STATE, HOLD_STATE
        )
    );
attributes: obj (
    State: PhoneNumber → PhoneState
);
agent_types: obj (
    Phone: obj (
            CW_Connected: PhoneNumber,
            CW_Hold: PhoneNumber,
            TW1: PhoneNumber,
            TW2: PhoneNumber,
            Three_Hold: PhoneNumber,
            rel: PhoneNumber
            )
    );
 .....
);
```

В этом описании среды определяются два перечислимых типа данных: PhoneNumber и PhoneState; перечисляются символьные константы, относящиеся к этим типам; определяются атрибут среды State функционального типа, отображающий номера телефонов в их состояния, а также тип агента Phone и его атрибуты перечислимого типа.

Базовый протокол представляет собой формулу динамической логики (тройку Хора) $\forall x(\alpha(r,x) \rightarrow < P(r,x) > \beta(r,x))$ и описывает локальные свойства системы в терминах пред- и постусловий α и β. Оба условия являются формулами многосортной логики первого порядка, интерпретированной на некоторых областях данных. Постусловие в отличие от предусловия кроме логических формул может содержать операторы присваивания. Процесс P представлен средствами MSC диаграмм [11–14] и описывает сценарий поведения специфицированной среды с погруженными в нее агентами, который может активизироваться при выполнении предусловия. Символ x представляет список типизированных переменных, r — список атрибутных выражений агентов и среды.

Описание среды E определяет сигнатуру языка, используемого для выражения пред- и постусловий в базовых протоколах и интерпретацию этой сигнатуры. Сигнатура языка включает сигнатуру типов (сортов многосортного исчисления) и сигнатуру функциональных и предикатных символов. Сигнатура типов содержит предопределенные простые типы — int, real, symb, перечислимые типы, которые определяются в описании среды, и функциональные типы вида $(\tau_1, \tau_2, \ldots) \rightarrow \tau$, где $\tau, \tau_1, \tau_2, \ldots$ — простые типы. Кроме того, в сигнатуре типов имеется предопределенный параметрический тип **queue of** τ, где τ — простой тип, обозначающий очередь элементов типа τ.

Предопределенные функциональные и предикатные символы включают символы арифметических операций $+, -, *$ и отношений $>, <, >=, <=, =$, а также конструкторы для символьного типа («,», «;» и т.д.). Областью значений символьного типа данных является множество свободных термов, образованных конструкторами из символьных констант и данных простых типов.

Простые атрибуты среды рассматриваются как неинтерпретированные функциональные символы арности 0, все остальные атрибуты являются неинтерпретированными функциональными символами арности больше нуля. Атрибутные выражения — это простые атрибуты или выражения вида $f(t_1, t_2, \ldots)$, где атрибут f имеет функциональный тип $(\tau_1, \tau_2, \ldots) \rightarrow \tau$, а $t_1, t_2, \ldots$ — параметры атрибутного функционального выражения, т.е. термы типов $\tau_1, \tau_2, \ldots$

Во входном языке VRS различаются атрибуты среды и атрибуты агентов. Соответственно различаются атрибутные выражения среды и агентные атрибутные выражения. Для атрибутных выражений среды используется префиксная запись (как показано выше), а для агентных атрибутных выражений используется более сложный синтаксис. Так, если τ — тип агента, а x — его простой атрибут, то соответствующее агентное выражение имеет вид $\tau(t).x$, где t есть имя агента, представленное выражением символьного типа (атрибут x агента t типа τ). Если же атрибут x имеет арность больше нуля, то атрибутное выражение имеет вид $\tau(t).x(t_1, t_2, \dots)$.

В теории целесообразно унифицировать обозначения для атрибутных выражений, сведя их все к префиксной записи. Для этого перекодируем агентные атрибутные выражения, повысив их арность на единицу, и будем рассматривать имя агента как дополнительный аргумент следующим образом. Атрибутное выражение $\tau(t).x$ заменим $(\tau().x)(t)$, а выражение $\tau(t).x(t_1, t_2, \dots)$ — выражением $(\tau().x)(t, t_1, t_2, \dots)$. При этом выражение $(\tau().x)$ рассматривается как новый символ атрибута x агента типа τ, имя которого является первым аргументом атрибутного выражения. Исходя из этих соглашений, произвольное атрибутное выражение будем обозначать как $f(t_1, t_2, \dots)$, допуская в качестве частного случая пустой список параметров $f() = f$ для простых атрибутов.

Множество всех атрибутных выражений для заданной среды E обозначим A_E. Атрибутное выражение $f(t_1, t_2, \dots)$ назовем константным, если все его параметры суть константные термы, т.е. не содержат ни атрибутных выражений, ни переменных. Множество всех константных атрибутных выражений для среды E обозначим A_E^{const}.

ОЧЕРЕДИ ЭЛЕМЕНТОВ ПРОСТЫХ ТИПОВ

Списком назовем символьную структуру данных, которая образуется с помощью конструктора ((),()). Список называется терминальным, если он образован по следующим правилам:

1) символьная константа Nil есть терминальный список (пустой терминальный список);

2) если a — символьный терм, а l — терминальный список, то символьный терм (a, l) также есть терминальный список, а его элементами являются терм a и элементы терминального списка l.

Пример терминального списка из трех элементов: ((x1,10), r2 + a2, 999, Nil). Обозначим L_E множество всех терминальных списков, а L_E^{const} — множество всех константных терминальных списков, т.е. терминальных списков, составленных из константных термов.

Теперь можно определить тип данных **queue of** τ. Значениями этого типа являются терминальные константные списки, элементами которых есть константы типа τ. На очередях типа **queue of** τ определены функции доступа **get_from_head** x и **get from tail** x, которые могут использоваться только в предусловиях базовых протоколов, и операторы обновления списков **add_to_head** (x, t), **add_to_tail** (x, t), **remove_from_head** (x) и **remove_from_tail** (x). Здесь x — это атрибутное выражение типа очередь, а t — терм соответствующего типа.

Операторы присваивания и обновления списков могут использоваться только в постусловиях. Для унификации обозначений будем считать, что оператор удаления (remove) также имеет два аргумента, но в качестве второго аргумента используется символьная константа Nil.

ОПЕРАТОР ПОСТУСЛОВИЯ

Базовые протоколы определяют отношение переходов на множестве состояний специфицируемой системы. При этом предусловия используются для определения применимости базового протокола, а постусловия — для выполнения преобразований. Далее будем рассматривать свойства постусловий как операторов,

определенных на состояниях среды, независимо от предусловий. Синтаксически оператор постусловия представляет собой конъюнкцию $\beta = R \wedge U \wedge C$, где

$$R = (r_1 := t_1 \wedge r_2 := t_2 \wedge \dots)$$

является конъюнкцией операторов присваивания,

$$U = p_1(g_1, h_1) \wedge p_2(g_2, h_2) \wedge \dots$$

является конъюнкцией операторов обновления очередей, а C — логическая формула, построенная из атрибутных выражений, отношений равенств (они определены для всех типов), арифметических отношений, логических связок и кванторов (в языке VRS используется только квантор существования по переменным простого типа, допускается также использование ограниченных кванторов общности).

Определим семантику операторов постусловий как преобразований на множестве состояний среды. Операторы постусловий задают переходы на множестве состояний среды, размеченные этими операторами, преобразуя среду в транзиционную систему. При этом различаются два класса моделей среды: конкретные и символьные модели (соответственно конкретная и символьная семантика операторов постусловий). Состояние конкретной модели определяется явно заданными значениями константных атрибутных выражений, состояние символьной модели определяется логической формулой, которая «покрывает» множество конкретных состояний. Для построения символьной семантики следует различать значения атрибутных выражений в состоянии модели до и после выполнения преобразования. Если состояние модели после выполнения преобразования определяется логической формулой, связывающей атрибутные выражения и накладывающей ограничения на их возможные конкретные значения в текущем состоянии, то значения этих атрибутных выражений до применения преобразования будут представлены переменными, взаимно однозначно соответствующими этим выражениям.

Рассмотрим четыре основных множества атрибутных выражений и соответствующих им переменных.

Первое множество представлено списком $\boldsymbol{r} = (r_1, r_2, \dots)$ и состоит из левых частей операторов присваивания из конъюнкции R. В него входят те атрибутные выражения, которые изменяют свое значение в результате выполнения присваиваний. Атрибутные выражения из списка $\boldsymbol{r}$ имеют простые типы. Список переменных, взаимно однозначно соответствующих атрибутам из списка $\boldsymbol{r}$, обозначим $\boldsymbol{x} = (x_1, x_2, \dots)$.

Второе множество представлено списком $\boldsymbol{g} = (g_1, g_2, \dots)$ и состоит из атрибутных выражений, значения которых обновляются операторами обновления очередей из конъюнкции U. Элементы этого списка имеют тип очереди. Список переменных, взаимно однозначно соответствующих атрибутам из списка $\boldsymbol{g}$, обозначим $\boldsymbol{w} = (w_1, w_2, \dots)$.

Третье множество представлено списком $\boldsymbol{s} = (s_1, s_2, \dots)$ и состоит из внешних вхождений атрибутных выражений в формулу C, отличных от атрибутных выражений из $\boldsymbol{r}$ (атрибутные выражения типа очереди в формулу C входить не могут). Этим атрибутным выражениям взаимно однозначно соответствуют переменные из списка $\boldsymbol{y} = (y_1, y_2, \dots)$.

Четвертое множество представлено списком $\boldsymbol{z} = (z_1, z_2, \dots)$ и формируется для использования оператором постусловия β вместе с формулой γ символьного состояния среды. Список включает все внешние вхождения атрибутных выражений в формулу γ и в правые части присваиваний из R, отличные от элементов уже построенных списков.

В дальнейшем для оператора постусловия $\beta = R \wedge U \wedge C$ будут использованы следующие сокращения:

$$R = (r_1 := t_1 \wedge r_2 := t_2 \wedge \dots) = (\boldsymbol{r} := t),\ U = p_1(g_1, h_1) \wedge p_2(g_2, h_2) \wedge \dots = p(\boldsymbol{g}, h).$$

КОНКРЕТНЫЕ МОДЕЛИ

Конкретная модель среды — это транзиционная система с конкретными состояниями. Конкретным состоянием $\sigma: A_E^{\text{const}} \to D$ называется отображение из множества константных термов в область их значений. Поскольку в системе VRS типизированы и атрибутные выражения, и их значения, то это отображение должно согласовываться с описаниями типов. Так, если атрибут f имеет тип $(\tau_1, \tau_2, \ldots) \to \tau$, то $f(c_1, c_2, \ldots)$ имеет тип τ. Множество D включает в себя все целые и вещественные числа, символьные значения (свободные термы), все значения перечислимых типов и все значения типа очереди. Множество всех конкретных состояний обозначим через C_E.

Конкретные состояния — это отображения, определенные на константных термах. Распространим эти отображения на произвольные термы. Заметим, что термы — это выражения, построенные из констант, атрибутных выражений и переменных. Рассмотрим систему переписывающих правил

$$S_\sigma = \{x = \sigma(x) \mid x \in A_E^{\text{const}}\}.$$

Пусть F — это терм или формула. Тогда $\sigma(F)$ есть результат нормализации выражения F путем применения системы переписывающих правил S_σ к этому выражению. Применение правил происходит с выполнением всех необходимых вычислений для интерпретированных операций и отношений (например, выражение $3+5$ упрощается до 8, выражение $x+0$ — до x, а $3<5$ преобразуется в истинностное значение 1). Система переписывающих правил канонична, поэтому результат нормализации всегда определен и не зависит от стратегии. Результатом применения системы S_σ к выражению F будет константа или выражение, не содержащее константных термов, отличных от константы.

Пусть теперь F — это формула, не содержащая свободных переменных (т.е. все переменные связаны кванторами). Состояние σ полностью задает интерпретацию сигнатуры формулы F, поэтому можно говорить об истинности формулы F на этом состоянии. Тот факт, что формула F истинна, будем записывать как $\sigma \mid = F$.

Внешняя подстановка. В отличие от полной подстановки $\sigma(F)$ при внешней подстановке $out(\sigma, F)$ полностью вычисляются только внешние вхождения атрибутных выражений в F (формула или терм). Эта подстановка определяется следующей системой правил:

$$out(\sigma, \exists x p(x)) = \exists x\, out(\sigma,\, p(x)),$$

$$out(\sigma,\, f(t)) = f(\sigma(t)),$$

$$out(\sigma, \omega(t)) = \omega(out(\sigma, t)),$$

$$out(\sigma,\, f) = f,$$

$$out(\sigma, \omega) = \omega.$$

В этих правилах $x = (x_1, x_2, \ldots)$ — список переменных, $t = (t_1, t_2, \ldots)$ — список аргументов атрибутного выражения $f(t)$, f — атрибут, ω — символ интерпретированной функции либо логическая связка, либо предикатный символ, либо переменная.

Далее определим отношение переходов на множестве C_E конкретных состояний среды E. Действиями транзиционной системы C_E являются операторы постусловий. Пусть $\beta = R \wedge U \wedge C$ — оператор постусловия, а σ и σ' — конкретные состояния. Переход

$$\sigma \xrightarrow{\beta} \sigma'$$

возможен, если выполняются следующие условия:

1) $\sigma'(out(\sigma, \boldsymbol{r})) = \sigma(t)$;

2) $\sigma'(out(\sigma, \boldsymbol{g})) = p(\sigma(\boldsymbol{g}), \sigma(h))$;

3) очереди, к которым применяются операторы удаления элементов, должны быть непустыми;

4) $\sigma' \mid = \sigma'(C)$;

5) $x \notin \{out(\sigma, y) \mid y \in \{\boldsymbol{r}\} \cup \{\boldsymbol{g}\} \cup \{\boldsymbol{s}\}\} \Rightarrow \sigma'(x) = \sigma(x),\ x \in A_E^{\text{const}}$.

В этих определениях списки термов рассматриваются как символьные структуры данных. Так, например, первое условие можно представить следующим образом:

$$\sigma'(out(\sigma, \boldsymbol{r})) = (\sigma'(out(\sigma, r_1)),\ \sigma'(out(\sigma, r_2)),\ \ldots) = (\sigma(t_1),\ \sigma(t_2),\ldots) \Leftrightarrow$$

$$\Leftrightarrow \sigma'(out(\sigma, r_1)) = \sigma(t_1),\ \sigma'(out(\sigma, r_2)) = \sigma(t_2),\ldots$$

Если $x = (x_1, x_2, \ldots)$, то $\{x\} = \{x_1, x_2, \ldots\}$ обозначает множество его элементов. На основании условия 5 в новом состоянии могут получить новые значения только атрибутные выражения из множества $\{\boldsymbol{r}\} \cup \{\boldsymbol{g}\} \cup \{\boldsymbol{s}\}$.

Система C_E может быть недетерминированной. Однако этот недетерминизм определяется только наличием формулы C в постусловии. Если $C = 1$, то система C_E детерминирована.

СИМВОЛЬНЫЕ МОДЕЛИ

Символьные модели среды рассматриваются как абстракции конкретных моделей, а их состояния — как абстракции конкретных состояний. Состояниями символьных моделей являются формулы вида $\gamma = \exists v(L(v) \wedge F(v))$, где $v = (v_1, v_2, \ldots)$ — список переменных, $L(v) = (l_1 = L_1 \wedge l_2 = L_2 \wedge \ldots)$ — список определяющих равенств для очередей, а $F(v)$ — формула, не содержащая атрибутных выражений типа очереди. Атрибутные выражения $l_1, l_2, \ldots$ имеют тип очереди, а $L_1, L_2, \ldots$ — абстрактные очереди, типы которых согласуются с типами атрибутных выражений $l_1, l_2, \ldots$ соответственно. Абстрактная очередь типа τ — это либо терминальный список термов типа τ, либо выражение вида $(P;Q)$, где P и Q — терминальные списки термов типа τ. Абстрактная очередь L согласуется с типом **queue of** τ, если все составляющие ее элементы имеют тип τ.

Семантика абстрактных очередей выражается через конкатенацию и кванторы по переменным типа очереди следующим образом: равенство $(P;Q) = R$ эквивалентно формуле $\exists w(R = P * w * Q)$, где $*$ обозначает конкатенацию терминальных списков

$$(a_1, a_2, \ldots, a_n, \text{Nil}) * (b_1, b_2, \ldots, b_n, \text{Nil}) = (a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n, \text{Nil}),$$

$$\text{Nil} * P = P * \text{Nil} = P .$$

Если R — атрибутный терм, то равенство является определяющим равенством для этого терма. Если R — абстрактная очередь, то равенство сводится к формуле без очередей. Действительно, $(P;Q) = (P';Q') \Leftrightarrow \exists (w, w')(P * w * Q = P' * w' * Q') \Leftrightarrow$ $\Leftrightarrow R \wedge S$, где R не содержит очередей, $S \Leftrightarrow \exists (w, w')(p * w = w' * q)$ или $S \Leftrightarrow \exists (w, w')(w * q = p * w')$. В обоих случаях $S \Leftrightarrow 1$.

Множество всех состояний символьной модели, определенной описанием среды E, обозначим S_E. Это множество становится транзиционной системой путем определения отношения переходов по правилу

$$\gamma \xrightarrow{\beta} \text{pt}_\beta(\gamma) .$$

Функция $\text{pt}_\beta(\gamma) = \text{pt}(\gamma, \beta)$ (предикатный трансформер) будет определена ниже. Если $\text{pt}_\beta(\gamma) = 0$, переход считается неопределенным. Система S_E детерминирована.

Символьные подстановки. Пусть $\sigma: A_E \to T_E$ — отображение множества всех (символьных и константных) атрибутных термов в множество всех термов (для заданного описания среды E). Определим символьную подстановку $\sigma(F)$ с помощью системы переписывающих правил $\{x = \sigma(x) \mid x \in A_E\}$. В отличие от конкретных состояний результат переписывания зависит от стратегии. Примем стратегию

однократного переписывания внешних вхождений атрибутных термов. Она определяется следующими правилами, которые добавляются к правилам переписывания:

$$\sigma(\exists x p(x)) = \exists x\, \sigma(p(x)),$$

$$\sigma(\omega(t)) = \omega(\sigma(t)),$$

$$\sigma(\omega) = \omega.$$

Значения символов, используемых в этих правилах, такие же, как и в определении внешней подстановки для конкретных состояний. Для определения внешней символьной подстановки используем правила, которые применялись выше для конкретных состояний σ.

ОПРЕДЕЛЕНИЕ ПРЕДИКАТНОГО ТРАНСФОРМЕРА

Рассмотрим случай, когда все атрибутные выражения, которые находятся в постусловии, имеют арность 0, а обновление списков и логическая формула C отсутствуют ($C = \mathrm{l}$). В этом случае предполагается, что свои значения могут изменить только атрибуты из списка $\boldsymbol{r}$ левых частей присваиваний. Предикатный трансформер определяется простой формулой

$$\mathrm{pt}\,(\gamma, \beta) = \exists \boldsymbol{x}\, (\varphi(\gamma) \wedge \boldsymbol{r} = \varphi(t)),$$

где φ — подстановка переменных из списка $\boldsymbol{x}$ вместо переменных из списка $\boldsymbol{r}$, т.е. $\varphi(\boldsymbol{r}) = \boldsymbol{x}$, t — список правых частей, $\boldsymbol{x}$ — список переменных, соответствующих левым частям, как было определено выше. Содержательно эта формула утверждает, что после выполнения операций присваивания значения левых частей присваиваний равны значениям правых частей, которые они имели до выполнения присваиваний.

В случае $C \neq 1$ предполагается, что изменить свои значения могут все атрибутные выражения из списка $\boldsymbol{s}$ (внешние вхождения в формулу C). Подстановка φ распространяется на атрибутные выражения $\boldsymbol{s}$ таким образом, что $\varphi(\boldsymbol{s}) = \boldsymbol{y}$, и список переменных $\boldsymbol{y}$ добавляется под квантор существования. Если в постусловия входят операторы обновления очередей, то подстановка φ распространяется на атрибутные выражения $\boldsymbol{g}$ таким образом, что $\varphi(\boldsymbol{g}) = \boldsymbol{w}$, список переменных $\boldsymbol{w}$ заносится под квантор существования и добавляются равенства для очередей

$$\mathrm{pt}(\gamma, \beta) = \exists (\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{y})(\varphi(\gamma) \wedge (\boldsymbol{r} = \varphi(t)) \wedge (\boldsymbol{g} = p\,(\boldsymbol{w}, \varphi\,(h)))) \wedge C\,.$$

Операторы обновления очередей определяются следующим образом:

$$\boldsymbol{g} = p(\boldsymbol{w}, \varphi(h)) \Leftrightarrow ((g_1 = p_1(w_1, \varphi(h_1))) \wedge (g_2 = p_2(w_2, \varphi(h_2))) \wedge \ldots);$$

$$\boldsymbol{g} = \textbf{add\_ to\_ head}\,(\boldsymbol{w}, \varphi(h)) \Leftrightarrow \boldsymbol{g} = (\varphi(h), \mathrm{Nil}) * \boldsymbol{w};$$

$$\boldsymbol{g} = \textbf{add\_ to\_ tail}\,(\boldsymbol{w}, \varphi(h)) \Leftrightarrow \boldsymbol{g} = \boldsymbol{w} * (\varphi(h),\ \mathrm{Nil});$$

$$\boldsymbol{g} = \textbf{remove\_ from\_ head}\,(\boldsymbol{w}, \mathrm{Nil}) \Leftrightarrow \exists (u, v)((\boldsymbol{w} = (u, \mathrm{Nil}) * v) \wedge \boldsymbol{g} = v);$$

$$\boldsymbol{g} = \textbf{remove\_ from\_ tail}\,(\boldsymbol{w}, \mathrm{Nil}) \Leftrightarrow \exists\,(u, v)((\boldsymbol{w} = v * (u, \mathrm{Nil})) \wedge \boldsymbol{g} = v).$$

Переменная u в двух последних определениях имеет тип τ, если w имеет тип **queue of** τ, а переменная v имеет тот же тип, что и w. Операция конкатенации, как и кванторы по переменным типа очереди, не входит в сигнатуру логического языка моделей. Определение предикатного трансформера дано в расширенном языке. Но это расширение легко элиминируется следующим образом. Если среди определяющих равенств для списков в формуле γ имеется равенство $g = L$, то квантор по переменной $\varphi(g)$ элиминируется после подстановки абстрактного списка L вместо всех вхождений переменной $\varphi(g)$ и элиминации конкатенации.

Отождествление атрибутных термов. Появление атрибутных термов арности больше нуля приводит к необходимости отождествлений: если $u = v$, то $f(u) = f(v)$ (подстановочность равенства). Для того чтобы описать все возможные

способы отождествлений, которые следует учитывать при построении предикатного трансформера в общем случае, проиндексируем все атрибутные выражения, которые входят в постусловие и символьное состояние, множеством индексов I. Если атрибутное выражение $f(u)$ имеет индекс k, то его атрибутный символ f обозначим через f_k, а список аргументов u — через $u^{(k)}$. Таким образом, k-й атрибутный символ в этой индексации имеет вид $f_k(u^{(k)})$. Пусть $\boldsymbol{Q} = \{\boldsymbol{r}\} \cup \{\boldsymbol{g}\} \cup \{\boldsymbol{s}\}$. Рассмотрим множество

$$M = \{f_k(u^{(k)}) = f_k(v^{(k)}) | k \in I,\ f_k(u^{(k)}) \in \boldsymbol{Q},\ f_k(v^{(k)}) \in \{\boldsymbol{z}\}\},$$

состоящее из всех равенств атрибутных выражений, которые могут появиться в результате отождествления аргументов атрибутных выражений из множества $\{\boldsymbol{z}\}$ с аргументами атрибутных выражений из множества $\boldsymbol{Q}$. Если аргументы атрибутного выражения из $\{\boldsymbol{z}\}$ не отождествляются с аргументами атрибутных выражений из $\boldsymbol{Q}$, то атрибутное выражение из $\{\boldsymbol{z}\}$ не меняет своего значения в результате применения трансформера. Если же имеет место отождествление $u^{(k)} = v^{(k)}$, из которого следует $f_k(u^{(k)}) = f_k(v^{(k)})$, то атрибутное выражение $f_k(v^{(k)})$ в формуле для предикатного трансформера следует заменить на переменную, соответствующую атрибутному выражению $f_k(u^{(k)})$.

Чтобы получить формальное определение этой процедуры, перенумеруем все подмножества J_i, $J_i \subseteq I$, $i = 1, 2, \ldots$, множества I, включая пустое множество, и пусть при этом

$$N_i = \{f_k(u^{(k)}) = f_k(v^{(k)}) | k \in J_i, J_i \subseteq I,\ i = 1, 2, \ldots\}.$$

Обозначим

$$E_i = \bigwedge_{k \in J_i} (u^{(k)} = v^{(k)}) \wedge \bigwedge_{k \in I \setminus J_i} (u^{(k)} \neq v^{(k)}).$$

Каждая формула определяет один из возможных способов отождествления. Пусть теперь $\xi_i = (\xi_{i1}, \xi_{i2}, \ldots)$, где $\xi_{ij} = \varphi(f_k(u^{(k)}))$, если $z_j = f_k(v^{(k)})$, $k \in J_i$, а $\xi_{ij} = z_j$ в противном случае. Определим символьную подстановку φ_i таким образом, что на множестве $\boldsymbol{Q}$ она действует как φ, а $\varphi_i(\boldsymbol{z}) = \xi_i$. Теперь можно определить общую формулу для предикатного трансформера

$$\mathrm{pt}(\gamma, \beta) = q_1 \vee q_2 \vee \ldots,$$

$$q_i = \exists(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{y})(\gamma'_i \wedge R'_i \wedge U'_i \wedge E'_i) \wedge C,$$

где $\gamma'_i = \varphi_i(\gamma)$, $R'_i = (out(\varphi_i, \boldsymbol{r}) = \varphi_i(t))$, $U'_i = (out(\varphi_i, \boldsymbol{g}) = p(\varphi_i(\boldsymbol{g}), \varphi_i(h)))$, $E'_i = \varphi_i(E_i)$.

Кванторы по переменным типа очереди элиминируются так же, как и в случае простых атрибутов. Однако в результате отождествлений могут появиться равенства двух абстрактных очередей. Такие равенства элиминируются аналогично показанным выше. Перед выполнением этих преобразований кванторы в формуле γ должны быть вынесены в начало формулы.

Некоторые из формул $q_1, q_2, \ldots$ могут быть невыполнимы, а значит, они не покрывают ни одного конкретного состояния. Если все формулы невыполнимы, то переход не определен. Следовательно, все конкретные состояния, которые покрываются формулой γ, являются тупиковыми.

СВОЙСТВА ПРЕДИКАТНОГО ТРАНСФОРМЕРА

Задача символьного моделирования состоит в исследовании свойства конкретных систем на их символьных моделях. Между состояниями символьных и конкретных систем определено отношение $\sigma \models \gamma$, которое будем рассматривать как отношение покрытия: символьное состояние γ «покрывает» состояние σ конкрет-

ной системы. Для того чтобы моделирование было корректным, переходы конкретной системы должны также «покрываться» переходами символьной системы: если $\sigma \xrightarrow{\beta} \sigma'$ в конкретной системе и $\sigma \models \gamma$, то $\sigma' \models \mathrm{pt}(\gamma, \beta)$. Чтобы условие $\mathrm{pt}(\gamma, \beta)$ было сильнейшим постусловием для предусловия γ и оператора β, необходимо выполнение обратного условия: если $\sigma' \models \mathrm{pt}(\gamma, \beta)$ и $\sigma \xrightarrow{\beta} \sigma'$, то $\sigma \models \gamma$.

Введем следующие обозначения:

$$\mathrm{State}(\gamma) = \{\sigma \in C_E \mid \sigma \models \gamma\},$$

$$\mathrm{Nstate}(S, \beta) = \{\sigma' \mid \exists\, (\sigma \in S)\, (\sigma \xrightarrow{\beta} \sigma')\}\ ,\ S \subseteq C_E .$$

Теперь основное свойство предикатного трансформера, определенного в предыдущем разделе, может быть сформулировано в виде следующей теоремы.

Теорема 1. Для предикатного транформера pt выполняется $\mathrm{State}(\mathrm{pt}(\gamma, \beta)) = \mathrm{Nstate}(\mathrm{State}(\gamma), \beta)$.

Эквивалентная формулировка этой теоремы:

$$\sigma' \models \mathrm{pt}(\gamma, \beta) \Leftrightarrow \exists (\sigma \in C_E)(\sigma \models \gamma \wedge \sigma \xrightarrow{\beta} \sigma'),\ \sigma' \in C_E .$$

Здесь выражено необходимое и достаточное условие покрытия конкретного состояния σ' состоянием $\mathrm{pt}(\gamma, \beta)$.

Достаточность. Докажем, что $\sigma \models \gamma \wedge \sigma \xrightarrow{\beta} \sigma' \Rightarrow \sigma' \models \mathrm{pt}(\gamma, \beta)$. Предположим, что посылка истинна. Имеем

$$\mathrm{pt}(\gamma, \beta) = q_1 \vee q_2 \vee \ldots$$

Пусть $N = \{(f(u) = f(v)) \in M \mid \sigma'(u) = \sigma'(v)\}$, а $J = \{k \mid (f_k(u^{(k)}) = f_k(v^{(k)})) \in N\}$. Поскольку J_i, $i = 1, 2, \ldots$, пробегает все подмножества множества M, то найдется такое i, что $J_i = J$, $N_i = N$. Покажем, что $\sigma' \models q_i = \exists (\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{y})(\gamma'_i \wedge R'_i \wedge U'_i \wedge E'_i) \wedge C$. Имеем $\sigma' \models C$ по определению оператора β. Для доказательства истинности первого конъюнктивного члена необходимо найти такие значения переменных $\boldsymbol{x}$, $\boldsymbol{w}$ и $\boldsymbol{y}$, для которых подкванторная формула истинна. Положим $\boldsymbol{x} = \sigma'(\boldsymbol{r})$, $\boldsymbol{w} = \sigma'(\boldsymbol{g})$, $\boldsymbol{y} = \sigma'(\boldsymbol{s})$. Соответствующую подстановку обозначим μ. Покажем, что

$$\sigma' \models \mu(\gamma'_i \wedge R'_i \wedge U'_i \wedge E'_i) = \mu(\gamma'_i) \wedge \mu(R'_i) \wedge \mu(U'_i) \wedge \mu(E'_i) .$$

Вначале докажем, что $\sigma' \models E_i$. Для этого вычислим $\mu(E'_i) = \mu(\varphi_i(E_i)) = \nu_i(E_i)$, где через ν_i обозначена композиция подстановок φ_i и μ. Имеем $\nu_i(\boldsymbol{r}) = \sigma'(\boldsymbol{r})$, $\nu_i(\boldsymbol{s}) = \sigma'(\boldsymbol{s})$, $\nu_i(\boldsymbol{z}) = \eta_i$, где $\eta_i = (\eta_{i1}, \eta_{i2}, \ldots)$, $\eta_{ij} = \sigma(f_k(u^{(k)}))$, если $z_j = f_k(v^{(k)})$, $k \in J_i$, а $\eta_{ij} = \sigma(z_j)$ в противном случае. Из определения множеств N_i и J_i непосредственно следует $\nu_i(E_i) = \sigma'(E_i)$ и $\sigma' \models E_i$. Из этого соотношения и сказанного выше вытекает $\sigma' \models \mu(E'_i)$. Докажем лемму.

Лемма 1. Для любого терма $t \in T_E$ имеет место $\sigma' \models E_i \Rightarrow \sigma' \models \nu_i(t) = \sigma'(t)$.

Действительно, если t — атрибутное выражение и $t \in \boldsymbol{Q}$, то $\nu_i(t) = \sigma'(t)$, если же $t \in \{\boldsymbol{z}\}$ и $t = z_j = f_k(v^{(k)}), k \in J_i$, то $\nu_i(t) = \sigma(f_k(u^{(k)}))$. Однако из посылки леммы следует, что $\sigma'(f_k(u^{(k)})) = \sigma'(f_k(v^{(k)})) = \sigma'(t)$ и, значит, $\sigma' \models \nu_i(t) = \sigma'(t)$. Если же $t = f_k(v^{(k)})$ и $k \notin J_i$, то из той же посылки следует, что $\nu_i(t) = \sigma'(t)$. Таким образом, лемма имеет место для атрибутных выражений и стандартным образом (индукцией по структуре выражения) распространяется на все термы.

Следствие. Для любой формулы F имеет место $\sigma' \models E_i \Rightarrow \sigma' \models \nu_i(F) = \sigma'(F)$.

Это очевидно. Аналогично структурной индукцией доказывается лемма 2.

Лемма 2. Выполняется $\sigma' \models E_i \Rightarrow \sigma' \models out(\nu_i, \boldsymbol{r}) = out(\sigma', \boldsymbol{r}),\ out(\nu_i, \boldsymbol{g}) = out(\sigma', \boldsymbol{g})$.

Теперь не рассмотренные фрагменты $\sigma' \models \mu(\gamma'_i), \sigma' \models \mu(R'_i), \sigma' \models \mu(u'_i)$ утверждения о достаточности условия теоремы следуют из определения отношения переходов конкретной модели.

Необходимость. Докажем, что $\sigma' \models \mathrm{pt}(\gamma, \beta) \Rightarrow \exists(\sigma \in C_E)(\sigma \models \gamma \wedge \sigma \xrightarrow{\beta} \sigma')$. Пусть истинна посылка. Построим σ такое, что $(\sigma \models \gamma \wedge \sigma \xrightarrow{\beta} \sigma')$. Для этого выберем i такое, что $\sigma' \models q_i = \exists(\boldsymbol{x}, \boldsymbol{y})(\gamma'_i \wedge R'_i \wedge U'_i \wedge E'_i) \wedge C$. Отсюда следует существование списков констант $\boldsymbol{c}$ и $\boldsymbol{d}$ таких, что $\sigma' \models \mu(\gamma'_i \wedge R'_i \wedge U'_i \wedge E'_i)$, где $\mu(\boldsymbol{x}) = \boldsymbol{c}$, $\mu(\boldsymbol{y}) = \boldsymbol{d}$. Положим $\sigma(\boldsymbol{r}) = \boldsymbol{c}$, $\sigma(\boldsymbol{s}) = \boldsymbol{d}$, $\sigma(\boldsymbol{z}) = \eta_i$, где $\eta_i = (\eta_{i1}, \eta_{i2}, \ldots)$, $\eta_{ij} = \sigma(f_k(u^{(k)}))$, если $z_j = f_k(v^{(k)})$ и $k \in J_i$, а $\eta_{ij} = \sigma'(z_j)$ в противном случае. Через ν_i, как и ранее, обозначим композицию подстановок φ_i и μ. Для состояния σ справедливы аналоги лемм 1 и 2. Имеем $\nu_i(E'_i) = \sigma(E_i) = 1 \Rightarrow \sigma \models E_i$. Используя лемму 1, получаем $\sigma \models \gamma$. Докажем, что $\sigma \xrightarrow{\beta} \sigma'$. Для этого необходимо доказать условия 1–5 определения отношения переходов конкретной модели. Условия 1–2 доказываются с помощью леммы 1, условие 3 следует из того, что предикатный трансформер отличен от нуля, условие 4 очевидно, а условие 5 следует из определения вектора η.

ПРИМЕНЕНИЯ ПРЕДИКАТНОГО ТРАНСФОРМЕРА

Одно из применений предикатного трансформера заключается в построении транзиционной системы для базовых протоколов.

Рассмотрим спецификацию $\Sigma = <B, E>$ системы, состоящую из множества базовых протоколов B и описания среды E. Пусть $b \in B$ — базовый протокол и $b = \forall x(\alpha(r,x) \rightarrow <P(r,x)> \beta(r,x))$. Протокол $b(p) = \alpha(r,p) \rightarrow <P(r,p)> \beta(r,p)$, где $p = (p_1, p_2, \ldots)$ — набор значений параметров $x = (x_1, x_2, \ldots)$, согласованный с их типами, называется конкретизацией протокола b. Состояниями конкретной системы C_Σ являются элементы множества C_E, действиями — конкретизированные базовые протоколы, а отношение переходов определяется следующим образом:

$$\sigma \xrightarrow{b(p)} \sigma' \Leftrightarrow \sigma \models \alpha(r,p),\ \sigma \xrightarrow{\beta(r,p)} \sigma'.$$

Состояниями символьной модели C_Σ являются элементы множества S_E, т.е. формулы базового (логического) языка системы VRS. Конкретизация протокола для символьной модели не требует задания конкретных значений параметров протокола. Они добавляются к описанию среды как простые ее атрибуты, а после применения протокола преобразуются в переменные, связанные квантором существования. Отношение переходов символьной модели определяется следующим образом. Вначале определяется тип символьной модели: универсальная или экзистенциальная (в работе [5] использовались термины «прямая» и «инверсная»). Эти модели отличаются способом применимости базового протокола. Отношение переходов для универсальной модели определяется как

$$\gamma \xrightarrow{b(p)} \exists p \gamma' \Leftrightarrow \gamma \models \alpha(r,p),\ \gamma \xrightarrow{\beta(r,p)} \gamma',$$

а для экзистенциальной модели как

$$\gamma \xrightarrow{b(p)} \exists p \gamma' \Leftrightarrow \neg(\gamma \models \neg\alpha(r,p)),\ \gamma \wedge \alpha(r,p) \xrightarrow{\beta(r,p)} \gamma'.$$

Условие применимости для универсальной модели эквивалентно общезначимости импликации $\gamma \rightarrow \alpha(r,p)$. Применимое предусловие покрывает все конкретные состояния, которые покрываются состоянием γ. Для экзистенциальной модели условие применимости эквивалентно выполнимости конъюнкции $\gamma \wedge \alpha(r,p)$. Пересечение множеств состояний, покрытых применимым предусловием и состоянием γ, не пусто.

Базовые протоколы системы VRS. Среда в системе представляется в виде композиции ядра среды и системы агентов, погруженных в среду. Формально со-

стояние такой среды можно представить в виде $s[m_1:u_1, m_2:u_2,\dots]$, где ядро состояния среды s есть отображение множества константных атрибутных выражений в их значения для конкретных моделей или формул базового языка для символьных моделей. Выражения $u_1, u_2, \dots$ представляют состояния агентов, $m_1, m_2, \dots$ — их имена. И то, и другое относительно базового языка — это символьные выражения, а агентная компонента $[m_1:u_1, m_2:u_2, \dots]$ состояния среды рассматривается как значение атрибута state типа символьной функции из имен агентов в их состояния. Эти состояния отождествляются с поведениями агентов (выражениями в алгебре поведений) либо символьными значениями атрибута state, который вычисляется операторами постусловий базовых протоколов. Базовые протоколы определяют функцию погружения и отношение переходов на множестве состояний среды.

Система VRS позволяет работать как с конкретными, так и с символьными моделями. Верификацию конкретных моделей поддерживает генератор трасс CTG (Concrete State Generator), а верификацию символьных моделей — генератор символьных трасс STG (Symbolic Trace Generator). Основное отличие конкретных моделей от символьных состоит в детерминированном поведении. Это означает, что после выбора базового протокола (который может быть недетерминированным) преобразование состояния среды должно определяться однозначно. Поэтому в конкретных моделях запрещается использовать предикатные формулы в постусловиях. Такие постусловия состоят только из операторов присваивания и операторов обновления очередей.

Оба трассовых генератора осуществляют генерацию и исследование трасс в пространстве состояний модели, одновременно проверяя целостность и безопасность модели (safety conditions) и осуществляя поиск целевых состояний (проверка достижимости). Трассовые генераторы также определяют возможность тупиковых состояний и недетерминизма, возникающего при выборе одного из нескольких базовых протоколов.

Управление генерацией трасс осуществляется общим генерирующим механизмом (generating engine), который использует и изменяет атрибут state, осуществляет выбор базового протокола и агентов, которые могут быть задействованы в его применении. Сам атрибут state скрыт от базовых протоколов, которым доступно лишь определение в предусловии состояния, где может находиться один из участников базового протокола (ключевой агент). В постусловии базового протокола может присутствовать оператор изменения состояния ключевого агента. Имя ключевого агента, как правило, является одним из параметров протокола, значение которого определяется генерирующим механизмом. Генерирующий механизм обладает рядом различных методов и эвристик для сокращения пространства поиска и отбрасывания тех направлений, где не ожидается нарушений исследуемых условий.

Абстракции. Вследствие детерминированности конкретного трассового генератора недетерминизм внешних воздействий на систему может быть промоделирован только недетерминизмом выбора базовых протоколов. Некоторые из них могут предназначаться для представления модели внешней среды. В символьных моделях такой недетерминизм моделируется более просто. Он может быть представлен атрибутами, не имеющими конкретных значений, но удовлетворяющими определенным ограничениям. Иначе говоря, символьная модель представляет собой абстракцию некоторого класса конкретных моделей и может быть получена следующим образом. Рассмотрим конкретную модель и в качестве начального состояния определим формулу, которая покрывает некоторый класс конкретных состояний. Моделируя систему из этих начальных состояний, получаем покрытие целого класса конкретных трасс. Другие формы абстракции можно получить, модифицируя базовые протоколы, например заменяя в постусловиях конкретные операторы присваиваний или модификации очередей формулами, ограничивающими возможные произвольные изменения атрибутов. Различного рода абстракции широко применяются в современных системах верификации [15–18]. Те из них, которые связаны с общей теорией базовых протоколов, исследованы в [4, 5].

Статическая верификация. В некоторых случаях верификацию модели можно осуществить без явного порождения трасс, исследуя лишь взаимоотношения между условиями верификации и базовыми протоколами. В частности, применяя базовые протоколы к заданным условиям, можно проверять инвариантность этих условий. Статически можно также доказывать детерминированность поведения системы и отсутствие тупиков.

Система, описанная в языке базовых протоколов системы VRS, включает агенты, работающие параллельно и асинхронно, а каждый базовый протокол допускает изменение состояния (атрибута state) только одного агента (называемого ключевым для этого протокола). Поэтому практическую ценность имеет поиск недетерминированного поведения для каждого агента отдельно.

Состояние s какого-либо агента a называется недетерминированным, если из него существует более одного перехода, т.е. применимо более одного протокола с ключевым агентом a. Введем понятие непротиворечивости для систем базовых протоколов. Оно определяется отсутствием пересечений предусловий базовых протоколов для одного ключевого агента. Пусть $\alpha(a,x,r)$ и $\alpha'(a,y,r)$ — предусловия двух различных базовых протоколов b и b' с ключевыми агентами одного и того же типа, конкретизированные одним и тем же именем ключевого агента (в случае, если это имя является параметром), x и y — списки параметров, r — список атрибутных выражений. Протоколы b и b' непротиворечивы, если формула

$$\neg(\exists x\,\alpha(a,x,r) \wedge \exists y\,\alpha'(a,y,r))$$

тождественно истинна (отрицание невыполнимо). Система базовых протоколов называется непротиворечивой, если любая пара протоколов с однотипными ключевыми агентами непротиворечива. Для распознавания непротиворечивости системы базовых протоколов в среде, имеющей n типов агентов, поведение каждого из которых описано k_i $(i = 1,\ldots,n)$ базовыми протоколами, требует $\sum\limits_{i=1,\ \ldots,\ n} C_2^{k_i}$ проверок непротиворечивости пар протоколов.

Если система базовых протоколов непротиворечива, то в каждом состоянии существует не более одного применимого базового протокола, а следовательно, все агенты такой системы имеют детерминированное поведение. Многоагентная система базовых протоколов, как правило, имеет состояния, из которых возможны несколько переходов под действиями разных протоколов. Однако при доказанном свойстве непротиворечивости все такие переходы осуществляются разными агентами. В этом случае данные состояния не являются недетерминированными, а наличие множества возможных переходов вызвано перестановочностью (interleaving) параллельно выполняемых базовых протоколов.

Помимо условия детерминизма, критическим требованием к моделируемой системе является отсутствие тупиков. Введем понятие полноты для систем базовых протоколов. Оно определяется следующим условием: дизъюнкция предусловий всех базовых протоколов (точнее, условий применимости)

$$\exists x_1 \alpha_1(x_1, r_1) \vee \exists x_2 \alpha_2(x_2, r_2) \vee \ \ldots$$

должна быть тождественно истинной (отрицание невыполнимо). Здесь $\alpha_i(x_i, r_i)$ — предусловие базового протокола, который параметризирован списком переменных x_i и использует список r_i атрибутных выражений. В некоторых реализациях пользователю предоставляется возможность сформулировать свои ограничения во избежание ложных отрицательных результатов. Полнота с учетом пользовательских ограничений определяется формулой

$$R(r) \vee \exists x_1 \alpha_1(x_1, r_1) \vee \exists x_2 \alpha_2(x_2, r_2) \vee \ \ldots,$$

где R — формула базового языка, задающая пользовательские ограничения.

Если выполняется условие полноты и в любой момент времени для каждого типа агентов в системе существует хотя бы один агент, то в каждом состоянии системы найдется хотя бы один применимый протокол. Таким образом, в полной системе отсутствуют тупики.

Для многоагентных систем, в которых задано разбиение базовых протоколов по типам ключевых агентов, можно сначала проверять полноту для каждого агента (типа агентов) индивидуально, что существенно сокращает размер формул в доказательствах. Если неполнота найдена для всех типов агентов, то можно перейти к исследованию полноты системы в целом с помощью формул, представленных выше. Если в системе существует хотя бы один тип агентов, для которого доказана полнота, и в системе всегда есть по крайней мере один агент данного типа, то такая система не имеет тупиков, так как агенты этого типа всегда смогут осуществлять переходы.

Кроме полноты и непротиворечивости системы существует ряд других свойств, обусловленных требованиями к живучести (liveness), надежности, производительности моделируемой системы и т.д. Такие свойства формулируются для каждой системы индивидуально и представляются в формализме конкретной модели.

Рассмотрим свойства, которые можно записать в виде формул базового языка, и поставим требование истинности этих формул на всех состояниях системы. Назовем их условиями целостности. Построим алгоритм проверки условия целостности для системы базовых протоколов. Пусть $Q(r)$ — формула базового языка, задающая условие целостности, $s_0(r)$ — формула базового языка, характеризующая начальное состояние системы (r — список атрибутных выражений). В первую очередь, необходимо проверить целостность начальных состояний системы: $s_0(r) \to Q(r)$. Эта формула должна быть общезначимой, иначе начальное состояние нарушает условие целостности и нет смысла продолжать проверки, а алгоритм должен завершить работу. Далее следует проверить инвариантность условия целостности. В этом случае достаточно проверить для каждого протокола следующее условие: если протокол применим и условие целостности истинно, то после выполнения протокола оно также будет истинным. Формально это условие выражается в виде тождественной истинности формулы

$$\forall x(\mathrm{pt}(\alpha(x,r) \wedge Q(r), \beta(x,r)) \to Q(r)),$$

где $\alpha(x,r)$ и $\beta(x,r)$ — пред- и постусловия рассматриваемого протокола.

Статически найденные недетерминизмы, тупики и нарушения условий целостности являются лишь предположительными ошибками. Состояния среды с ошибками можно получить, анализируя формулы, которые выражают непротиворечивость, полноту или условия целостности. Достижимость этих состояний можно опровергать статическими методами, сформулировав соответствующие условия целостности, или доказывать с помощью генерации трасс и поиска в пространстве состояний.

Дедуктивные средства системы VRS. Используемая в VRS теория (линейная арифметика, символьные типы, перечислимые типы и очереди) алгоритмически неразрешима, поскольку числовые функции позволяют моделировать всю арифметику. Однако ограничение теории только экзистенциальными формулами с ограниченным использованием кванторов общности позволило получить разрешимый фрагмент, который применяется для вычисления предикатных трансформеров. Дедуктивная система решает две основные задачи: проверку выполнимости формул и их упрощение. Первая задача решается непосредственно после получения результата применения предикатного трансформера с целью исключения невыполнимых дизъюнктивных членов; вторая задача решается с целью получения окончательного результата применения трансформера.

Алгоритм проверки выполнимости основан на элиминации неинтерпретированных функциональных символов методом Шостака [19]. Упрощение формул заключается в элиминации кванторов, когда это возможно, а также в упрощении систем линейных неравенств.

ЗАКЛЮЧЕНИЕ

Построенные алгоритмы преобразований формул реализованы в виде предикатного трансформера в системе верификации требований VRS. Предикатный трансформер используется как для построения пространства состояний с последующей генерацией трасс, так и для статического анализа моделей. Эти методы позволяют использовать VRS для поиска нарушений динамических и статических свойств систем.

СПИСОК ЛИТЕРАТУРЫ

1. Baranov S., Jervis C., Kotlyarov V., Letichevsky A., and Weigert T. Leveraging UML to deliver correct telecom applications / L. Lavagno, G. Martin, and B.Selic, eds // UML for Real: Design of Embedded Real-Time Systems. — Amsterdam: Kluwer Academic Publishers, 2003. — P. 323–342.
2. Letichevsky A., Kapitonova J., Letichevsky A.(jr.), Volkov V., Baranov S., Kotlyarov V., and Weigert T. Basic protocols, message sequence charts, and the verification of requirements specifications // Computer Networks. — 2005. — N 47. — P. 662–675.
3. Kapitonova J., Letichevsky A., Volkov V., and Weigert T. Validation of embedded systems / R. Zurawski, ed. // The Embedded Systems Handbook. — Miami: CRC Press. – 2005. — 51 p.
4. Летичевский А.Ад., Капитонова Ю.В., Волков В.А., Летичевский А.А., Баранов С.Н., Котляров В.П., Вейгерт Т. Спецификация систем с помощью базовых протоколов // Кибернетика и системный анализ. — 2005. — № 4. — С. 3–21.
5. Letichevsky A., Kapitonova J., Kotlyarov V., Letichevsky A.(jr.), Nikitchenko N., Volkov V., and Weigert T. Insertion modeling in distributed system design // Problems in Programming (ISSN 1727-4907). — 2008. — N 4. — P. 13–39.
6. Dijkstra E.W. A discipline of programming // Prentice-Hall, Englewood Cliffs, N.J., 1976. — 217 p.
7. Lamport Leslie. Win and sin: predicate transformer for concurrency // ACM Translation on Programming Language and System (TOPLAS), New York: ACM. — 1990. — **12**, Issue 3 (July 1990). — P. 396–428.
8. Годлевский А.Б. Предикатные преобразователи в контексте символьного моделирования транзиционных систем // Кибернетика и системный анализ. — 2010. — № 4. — С..
9. Letichevsky A. and Gilbert D. A model for interaction of agents and environments / D. Bert, C. Choppy, P. Moses, eds. // Recent Trends in Algebraic Development Techniques. Lecture Notes in Computer Science 1827, Springe, 1999. — P. 311–328.
10. Letichevsky A. Algebra of behavior transformations and its applications. In V.B. Kudryavtsev and I.G. Rosenberg, eds. // Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry, Springer. — 2005. — **207.** — P. 241–272.
11. Reniers M. Message sequence chart: Syntax and semantics // Ph.D. Thesis. Eindhoven University of Technology, 1998.
12. International Telecommunications Union. ITU-T Recommendation Z.120: Message Sequence Charts // Geneva: ITU–T, 2002.
13. Letichevskii A.A., Kapitonova Yu.V., Kotlyarov V.P., Letichevskii A.A. Jr., Volkov V.A. Semantics of timed Message Sequence Charts / Кибернетика и системный анализ. — 2002. — № 4. — С. 3–14.
14. Letichevsky A., Kapitonova J., Kotlyarov V., Volkov V., Letichevsky A.(jr.), and Weigert T. Semantics of message sequence charts // SDL Forum. — 2005. — P. 117–132.
15. Quielle J. and Sifakis J. Specification and verification of concurrent systems in CESAR // Proc. 5th Intern. Symposium on Programming, 1981. — P. 142–158.
16. Burch J., Clarke E., McMillan K., Dill D., and Hwang L. Symbolic model checking: 10^{20} states and beyond // Information and Computation. — 1992. — N 98(2). — P. 142–170.
17. Lamport L. The temporal logic of actions // ACM Transactions on Programming Languages and Systems, 1994. — P. 872–923.
18. Bultan T. and Yavuz-Kahveci T. Action language verifier // Proc. of ASE 2001, 2001. — P. 382–386.
19. Shostak R. A practical decision procedure for arithmetic with function symbols // J. of the Association for Computing Machinery. — 1979. – **26**, N 2. — P. 351–360.

Поступила 15.04.2010