ПРОГРАММИРОВАНИЕ, 2013, № 6, с. 12–24

УДК 681.3.06

СОГЛАСОВАНИЕ КОНФОРМНОСТИ И КОМПОЗИЦИИ *

© 2013 г. И.Б. Бурдонов, А.С. Косачев

Институт системного программирования РАН

109004 Москва, ул. А.Солженицына, 25

E-mail: igor@ispras.ru, kos@ispras.ru

Поступила в редакцию 12.06.2013

В нашей предыдущей статье [1] предлагалась новая модель реализации типа LTS. В обычной LTS переходы помечаются действиями, поэтому её можно назвать LTS действий. Новая модель – LTS наблюдений, в ней вместо действий используются наблюдения и тестовые воздействия *(кнопки)*. Эта модель обобщает многие семантики тестирования, которые опираются на LTS действий, но используют дополнительные наблюдения (отказы, множества готовности и т.п.). Кроме того, единообразно моделируются системы с приоритетами, которые не описываются с помощью LTS действий. В данной статье мы развиваем этот подход, концентрируя внимание на композиции систем. Дело в том, что на трассах наблюдений невозможно определить такую композицию, по отношению к которой композиция LTS обладала бы свойством *аддитивности:* множество трасс композиции LTS совпадает с множеством всех попарных композиций трасс LTS-операндов. Это объясняется тем, что наблюдение в композиционном состоянии не вычисляется по наблюдениям в состояниях-операндах. В данной статье мы предлагаем подход, устраняющий этот недостаток. Для этого переходы LTS помечаются такими символами *(событиями)*, которые, с одной стороны, можно компоновать для обеспечения свойства аддитивности, а, с другой стороны, можно использовать для генерации наблюдений при тестировании: переход по событию приводит к возникновению связанного с этим событием наблюдения. Эта модель названа LTS событий. В статье определяется 1) преобразование LTS событий в LTS наблюдений для согласования с положениями нашей предыдущей статьи [1], 2) композиция LTS событий, 3) композиция спецификаций, сохраняющая конформность: композиция конформных реализаций конформна композиции спецификаций, 4) единообразное моделирование LTS действий через LTS событий, которое позволяет рассматривать реализацию в любой семантике взаимодействия, допустимой для LTS действий. При этом композиция LTS событий, полученных в результате моделирования исходных LTS действий, эквивалентна LTS событий, полученной в результате моделирования композиции этих LTS действий.

1. ВВЕДЕНИЕ

Основной моделью взаимодействующих процессов является LTS *(Labelled Transition System)*. Для неё были определены операции алгебры процессов и, прежде всего, операция параллельной композиции LTS. Тестирование понимается как взаимодействие реализации с тестовой системой, формализуемое как параллельная композиция LTS-реализации с LTS-тестом. При этом первоначально предполагалось, что единственными наблюдениями над LTS являются внешние (наблюдаемые) действия, которыми помечены переходы LTS. Допускается также ненаблюдаемая внутренняя активность, которая в LTS изображается переходами по внутренним (ненаблюдаемым) действиям, которые только меняют состояние LTS. Поскольку такие действия не наблюдаемы, они не различимы при тестировании и обозначаются одним символом τ. Такую модель можно назвать LTS действий или ATS[1]. Результатом тестирования является последова-

*Работа выполнена при частичной поддержке гранта РФФИ №13-07-00915.

[1]“A” – от англ. “action” – действие.

тельность наблюдаемых действий – трасса действий. На трассах действий можно так определить композицию трасс, что тестирование и композиция LTS оказываются согласованными: множество трасс композиции LTS совпадает с множеством всех попарных композиций трасс LTS-операндов. Это свойство можно назвать *аддитивностью* трасс относительно композиции.

Другая (но тесно связанная с алгеброй процессов) формализация тестирования опирается на так называемую машину тестирования [7, 8, 9], которая фиксирует те или иные возможности тестовой системы по управлению и наблюдению за поведением реализации. Предполагается, что с помощью машины тестирования можно разрешать или запрещать реализации выполнение переходов по тем или иным внешним действиям. При этом τ-активность всегда разрешена. Сразу же выяснилось, что, кроме действий, при тестировании возможны и другие наблюдения. Так или иначе, все они связаны с наблюдением остановки LTS-реализации, которая возникает, когда текущее состояние стабильно (в нём нет τ-активности, то есть τ-переходов) и невозможно выполнение определённых в состоянии переходов по внешним действиям, поскольку все они в данный момент времени запрещены машиной тестирования. Такими наблюдениями, кроме самого факта остановки, являются, в частности, отказ *(refusal set)* и множество готовности *(ready set)*. Отказ – это множество действий, которые машина тестирования разрешает выполнять, но в текущем состоянии реализации нет переходов по этим действиям. Если мы наблюдаем остановку машины при том или ином множестве разрешённых внешних действий, то тем самым мы наблюдаем соответствующий отказ. Множество готовности – это множество всех внешних действий, по которым есть переходы в текущем состоянии реализации. Такое наблюдение характерно, например, для систем с графическим интерфейсом, когда на экран выдаётся “меню” возможных действий, и система ждёт от пользователя выбора того или иного действия. Теперь результатом тестирования оказывается трасса наблюдений, в которую могут входить не только внешние действия, но и другие наблюдения. Набор допустимых тестовых воздействий и наблюдений формализует возможности тестовой системы и задаётся семантикой взаимодействия. Например, хорошо известное отношение конформности *ioco (Input Output COnformance)* [11, 12], опирающееся на разделение действий на стимулы *(input)* и реакции *(output)*, допускает два вида тестовых воздействий – посылку одного стимула или приём всех реакций, и единственный вид отказа – отсутствие реакций *(quiescence)*.

Недостатком такой формализации является то, что все наблюдения, кроме внешних действий, описываются LTS-моделью неявно. Кроме того, тестовое воздействие, по сути, сводится к множеству разрешаемых им внешних действий. Всё это, во-первых, усложняет интерпретацию модели при тестировании, во-вторых, не даёт свободы в рассмотрении новых видов наблюдений и тестовых воздействий для той или иной семантики взаимодействия и, в-третьих, не даёт возможности в самой модели реализации явно определять те тестовые воздействия, на которые она реагирует теми или иными явно задаваемыми наблюдениями.

В нашей предыдущей статье [1] мы попытались избавиться от этого недостатка, определив новую обобщённую модель LTS. В этой модели переходы помечены не внешними действиями, а наблюдениями или тестовыми воздействиями, которые названы *кнопками*. Кроме свободы в соотношении кнопок и наблюдений в реализации, это дало возможность единообразно моделировать также системы с приоритетами, которые не описываются с помощью LTS действий. Эту новую модель можно назвать LTS *наблюдений* или OTS[2].

Однако остался ещё один существенный общий недостаток как LTS наблюдений, так и LTS действий при рассмотрении трасс не только действий, но и других наблюдений. Дело в том, что переход от действий к наблюдениям приводит к рассогласованию тестирования с композицией LTS: на трассах наблюдений невозможно определить композицию так, чтобы выполнялось свойство аддитивности. Это объясняется тем, что, вообще говоря, наблюдение в композиционном состоянии не вычисляется по наблюдениям в состояниях-операндах. Единственным ис-

[2] “O” – от англ. “observation” – наблюдение.

ключением, кроме внешнего действия, является множество готовности.

В данной статье мы предлагаем подход, устраняющий этот недостаток. Идея заключается в том, чтобы помечать переходы LTS такими символами (мы их назвали *событиями*), которые, с одной стороны, можно компоновать для обеспечения свойства аддитивности, а, с другой стороны, можно использовать для генерации наблюдений при тестировании: переход по событию приводит к возникновению того или иного связанного с этим событием наблюдения. Последнее свойство можно назвать *генеративностью* трасс событий: по ним вычисляются все трассы наблюдений реализации. Такую модель можно назвать LTS *событий* или ETS[3].

Мы определим 1) преобразование ETS в OTS для согласования с положениями нашей предыдущей статьи [1], 2) композицию ETS, 3) композицию спецификаций, сохраняющую конформность: композиция конформных реализаций конформна композиции спецификаций, 4) единообразное моделирование ATS через ETS, которое позволяет рассматривать реализацию в любой семантике взаимодействия, допустимой для ATS. Здесь мы ставим себе целью обеспечить следующее важное свойство *согласованности* композиции и моделирования: композиция ETS-операндов, полученных в результате моделирования исходных ATS-операндов, эквивалентна ETS, полученной в результате моделирования композиции этих ATS-операндов. Здесь эквивалентность понимается как равенство множества трасс событий.

2. LTS СОБЫТИЙ И ГЕНЕРАЦИЯ НАБЛЮДЕНИЙ ПО СОБЫТИЯМ

Под LTS *(Labelled Transition System)* понимается ориентированный граф с непустым множеством вершин, которые называются состояниями, одно из состояний выделено как начальное, дуги называются переходами и помечены символами из некоторого универсума символов $\boldsymbol{L}$ или специальным символом внутреннего действия $\tau \notin \boldsymbol{L}$. Мы также вводим специальный дополнительный символ разрушения $\gamma \in \boldsymbol{L}$, моделирующий любое поведение LTS, которое нежелательно при взаимодействии с ней [2,3,6]. LTS S задаётся четвёркой $S = Lts(V_S, E_S, s_0)$, где V_S – множество состояний, $E_S \subseteq V_S \times (\boldsymbol{L} \cup \{\tau\}) \times V_S$ – множество переходов, s_0 – начальное состояние. Запись $a \xrightarrow{x} a'$ означает, что $(a, x, a') \in E_S$. Трассой LTS называется последовательность пометок на маршруте, начинающемся в начальном состоянии, с пропуском символа τ. Множество трасс LTS S обозначим $\boldsymbol{tr}(S)$.

При определении ATS (LTS действий) считается, что задан универсум внешних действий $\boldsymbol{A} \subseteq \boldsymbol{L}$. Разрушение будем считать внешним действием $\gamma \in \boldsymbol{A}$. ATS – это такая LTS $S = Lts(V_S, E_S, s_0)$, в которой каждый переход помечен символом τ или действием из $\boldsymbol{A}$, то есть $E_S \subseteq V_S \times (\boldsymbol{A} \cup \{\tau\}) \times V_S$. Мы будем писать $S = Ats(V_S, E_S, s_0)$. Трассы ATS будем называть трассами действий; они содержат только действия.

Для определения OTS (LTS наблюдений), введённой в нашей предыдущей статье [5], считается, что заданы два непересекающихся универсума $\boldsymbol{B} \subseteq \boldsymbol{L}$ – универсум кнопок (тестовых воздействий) и $\boldsymbol{O} \subseteq \boldsymbol{L}$ – универсум наблюдений, $\boldsymbol{B} \cap \boldsymbol{O} = \varnothing$. Разрушение считается наблюдением $\gamma \in \boldsymbol{O}$. OTS – это такая LTS $S = Lts(V_S, E_S, s_0)$, в которой каждый переход помечен символом τ, кнопкой из $\boldsymbol{B}$ или наблюдением из $\boldsymbol{O}$, то есть $E_S \subseteq V_S \times (\boldsymbol{B} \cup \boldsymbol{O} \cup \{\tau\}) \times V_S$. Кроме того, отсутствие в состоянии перехода по кнопке трактуется как переход-петля по этой кнопке. Для OTS S мы будем писать $S = Ots(V_S, E_S, s_0)$. Трассы OTS будем называть трассами наблюдений; они содержат только наблюдения и кнопки.

Для определения ETS (LTS событий) будем считать, что, кроме универсума кнопок $\boldsymbol{B} \subseteq \boldsymbol{L}$, задан непересекающийся с ним универсум событий $\boldsymbol{E} \subseteq \boldsymbol{L}$, $\boldsymbol{B} \cap \boldsymbol{E} = \varnothing$. Разрушение будем считать событием $\gamma \in \boldsymbol{E}$. ETS – это такая LTS $S = Lts(V_S, E_S, s_0)$, в которой каждый переход помечен символом τ, кнопкой из $\boldsymbol{B}$ или событием из $\boldsymbol{E}$, то есть $E_S \subseteq V_S \times (\boldsymbol{B} \cup \boldsymbol{E} \cup \{\tau\}) \times V_S$. Кроме того, отсутствие в состоянии перехода по кнопке трактуется как переход-петля по этой кнопке. Для ETS S мы будем писать $S = Ets(V_S, E_S, s_0)$. Трассы ETS будем называть трассами событий; они содержат только события и кнопки.

В [1] пара $\boldsymbol{B}/\boldsymbol{O}$ определяет машину тестирования, в "чёрном ящике" которой находится OTS, каждому тестовому воздействию соответствует

[3] "E" – от англ. "event" – событие.

кнопка из $\boldsymbol{B}$ на клавиатуре машины, а наблюдения из $\boldsymbol{O}$ выводятся на экран дисплея. Если никакая кнопка не нажата, то реализация может выполнять любой τ-переход, γ-переход или переход по наблюдению $x \in \boldsymbol{O}$. В последнем случае на экране дисплея высвечивается наблюдение x. Кнопка $p \in \boldsymbol{B}$ высвечивается на экране дисплея при её нажатии. При нажатой кнопке p реализация может выполнять конечное число τ-переходов, после чего выполняет γ-переход или p-переход.

В данной статье мы рассматриваем реализацию не как OTS, а как ETS, поэтому будем считать, что в "чёрном ящике" машины тестирования находится не OTS, а ETS. Переход по кнопке в ETS соответствует переходу по той же кнопке в OTS, τ-переход в ETS – τ переходу в OTS. Переходу по событию в ETS поставим в соответствие переход по тому или иному наблюдению в OTS. Будем считать, что отображение событий в наблюдения задаётся универсальной частично-определённой однозначной функцией $\boldsymbol{f}$:$\boldsymbol{E} \to \boldsymbol{O}$. Потребуем, чтобы событие разрушения отображалось в наблюдение разрушения: $\gamma \in \boldsymbol{Dom(f)}$ и $\boldsymbol{f}(\gamma) = \gamma$. Событие $a \in \boldsymbol{Dom(f)}$ будем называть *наблюдаемым* событием.

Функционирование ETS определяется не только её устройством, но и функцией $\boldsymbol{f}$. Когда нажимается кнопка $p \in \boldsymbol{B}$, то реализация может выполнять конечное число τ-переходов, после чего выполняет γ-переход или p-переход. После p-перехода (до нажатия следующей кнопки), а также в начале тестирования до нажатия первой кнопки, реализация может выполнять любой τ-переход или переход по наблюдаемому событию $a \in \boldsymbol{Dom(f)}$, в последнем случае имеем наблюдение $\boldsymbol{f}(a)$. Заметим, что переход по разрушению всегда может быть выполнен, также как τ-переход, но в отличие от τ-перехода, не дающего наблюдений, при γ-переходе наблюдается γ. Также заметим, что переходы по ненаблюдаемым событиям (вне домена функции $\boldsymbol{f}$) не выполняются при тестировании. Однако такие переходы нам будут нужны для того, чтобы композиция ETS и трасс событий была аддитивной. Более того, композиция двух событий, одно или каждый из которых не наблюдаемо, может быть наблюдаемым событием.

Заметим, что ETS с одним множеством трасс событий можно считать эквивалентными, поскольку трасс событий достаточно для вычисления трасс наблюдений и в силу свойства аддитивности (которое мы покажем в следующем разделе) достаточно для композиции.

Функция $\boldsymbol{f}$ определяет естественное отображение ETS в OTS, когда переход по событию заменяется переходом по соответствующему наблюдению или удаляется, если такого наблюдения нет. Такое отображение мы также будем обозначать $\boldsymbol{f}$:ETS$\to$ OTS. Для ETS S оно определяется следующим образом. Каждый переход $s \xrightarrow{a} t$, где $a \in \boldsymbol{Dom(f)}$, заменяется переходом $s \xrightarrow{f(a)} t$, а каждый переход $s \xrightarrow{a} t$, где $a \in \boldsymbol{E} \backslash \boldsymbol{Dom(f)}$, удаляется, переходы по кнопкам и τ-переходы сохраняются.

Тестирование ETS-реализации S эквивалентно тестированию OTS-реализации $f(S)$.

Отображение $\boldsymbol{f} : \boldsymbol{E} \to \boldsymbol{O}$ естественно распространяется на трассы событий $\boldsymbol{f} : (\boldsymbol{E} \cup \boldsymbol{B})^* \to (\boldsymbol{O} \cup \boldsymbol{B})^*$ и множества таких трасс $\boldsymbol{f}$:$2^{(\boldsymbol{E} \cup \boldsymbol{B})^*} \to 2^{(\boldsymbol{O} \cup \boldsymbol{B})^*}$.

Пусть трасса событий $\sigma = u_1, u_2, \ldots, u_n$, где $u_i \in \boldsymbol{E} \cup \boldsymbol{B}$. Тогда $\sigma \in \boldsymbol{Dom(f)}$, если для каждого $i = 1..n$ либо $u_i \in \boldsymbol{B}$, либо $u_i \in \boldsymbol{Dom(f)}$. В этом случае $\boldsymbol{f}(\sigma) = v_1, v_2, \ldots, v_n$, где $v_i = u_i$, если $u_i \in \boldsymbol{B}$, и $v_i = \boldsymbol{f}(u_i)$, если $u_i \in \boldsymbol{E}$. Для множества $\sum$ трасс событий $\boldsymbol{f}(\sum)$ определяется как множество соответствующих трасс наблюдений: $f(\sum) \triangleq \{\boldsymbol{f}(\sigma) | \sigma \in \sum \cap \boldsymbol{Dom(f)}\}$.

Очевидно, что для любой ETS S имеет место $\boldsymbol{tr}(\boldsymbol{f}(S)) = \boldsymbol{f}(\boldsymbol{tr}(S))$.

3. ПАРАЛЛЕЛЬНАЯ КОМПОЗИЦИЯ

Мы определим параллельную композицию ETS в духе CSP *(Communicating Sequential Processes)* [10]. В CSP результатом композиции двух ATS является ATS, состояния которой – это пары состояний операндов, начальное состояние – это пара начальных состояний, а переход либо асинхронный и соответствует переходу в одном из операндов (меняется состояние только этого операнда), либо синхронный и соответствует паре переходов по одному и тому же – синхронному – действию в обоих операндах (измениться могут состояния обоих операндов). При этом синхронность или асинхронность переходов однозначно определяется действиями, которыми эти переходы помечены, то есть все

внешние действия разбиваются на синхронные и асинхронные, а внутреннее действие τ всегда считается асинхронным. Мы также будем считать, что разрешение всегда асинхронно. В обоих случаях действие на результирующем переходе совпадает с действием на одном переходе-операнде (асинхронный случай) или на обоих переходах-операндах (синхронный случай). После этого с помощью оператора *hide* некоторые[4] синхронные действия на переходах "скрываются", то есть заменяются символом τ.

Для композиции ETS мы будем всегда считать кнопки из $\boldsymbol{B}$ – синхронными, а событие γ-асинхронным. Остальные события из $\boldsymbol{E}\backslash\{\gamma\}$ могут быть как синхронными, так и асинхронными, что задаётся алфавитом синхронных событий $U \subseteq \boldsymbol{E}\backslash\{\gamma\}$. Кроме того, событие на синхронном переходе композиции не обязательно совпадает с событиями на переходах-операндах. Мы будем считать, что задана частично определённая композиция синхронных событий, результатом которой также является синхронное событие $|_U| : U \times U \to U$, $\boldsymbol{Dom}(|_U|) \subseteq U \times U$. Эта композиция должна быть коммутативной: если $(a,b) \in \boldsymbol{Dom}(|_U|)$, то $(b,a) \in \boldsymbol{Dom}(|_U|)$ и $a|_U|b = b|_U|a$.

Формально определим композицию ETS $|_U| : ETS \times ETS \to ETS$. Для $S = Ets(V_S, E_S, s_0)$ и $T = Ets(V_T, E_T, t_0)$ имеем $S|_U|T = Ets(V_S \times V_T, E, s_0, t_0)$, где E –наименьшее множество переходов, порождаемое следующими правилами вывода: $\forall s, t, s', t', a, b, p$

1.1: $a \in (\boldsymbol{E}\backslash U) \cup \{\tau\}$ $\&s \xrightarrow{a} s'$ $\vdash st \xrightarrow{a} s't$,

1.2: $b \in (\boldsymbol{E}\backslash U) \cup \{\tau\}$ $\&t \xrightarrow{b} t'$ $\vdash st \xrightarrow{b} st'$,

1.3: $p \in \boldsymbol{B}$ $\&s \xrightarrow{p} s'$ $\&t \xrightarrow{p} t'$ $\vdash st \xrightarrow{p} s't'$,

1.4: $(a,b) \in \boldsymbol{Dom}(|_U|)$ $\&s \xrightarrow{a} s'$ $\&t \xrightarrow{b} t'$ $\vdash st \xrightarrow{a|_U|b} s't'$.

На основе композиции событий определим композицию трасс событий
$|_U| : (\boldsymbol{B} \cup \boldsymbol{E})^* \times (\boldsymbol{B} \cup \boldsymbol{E})^* \to (\boldsymbol{B} \cup \boldsymbol{E})^*$
как наименьшее множество трасс, получаемое с помощью следующих правил вывода: $\forall \sigma, \sigma_1, \sigma_2 \in (\boldsymbol{B} \cup \boldsymbol{E})^* \forall a, b, p$.

2.0: $\vdash \epsilon \in \epsilon|_U|\epsilon$,

2.1: $\sigma = \sigma_1|_U|\sigma_2$ $\&a \in \boldsymbol{E}\backslash U$ $\vdash \sigma \cdot a \in (\sigma_1 \cdot a)|_U|\sigma_2$,

2.2: $\sigma = \sigma_1|_U|\sigma_2$ $\&b \in \boldsymbol{E}\backslash U$ $\vdash \sigma \cdot b \in \sigma_1|_U|(\sigma_2 \cdot b)$,

2.3: $\sigma = \sigma_1|_U|\sigma_2$ $\&p \in \boldsymbol{B}$
$\vdash \sigma \cdot p \in (\sigma_1 \cdot p)|_U|(\sigma_2 \cdot p)$,

2.5: $\sigma = \sigma_1|_U|\sigma_2$ $\&(a,b) \in \boldsymbol{Dom}(|_U|)$
$\vdash \sigma \cdot (a|_U|b) \in (\sigma_1 \cdot a)|_U|(\sigma_2 \cdot b)$.

Если $\sigma_1|_U|\sigma_2 = \varnothing$, то будем говорить, что эти трассы не компонуются. Композиция трасс событий естественно распространяется на композицию множеств трасс событий, определяемую как объединение множеств всех попарных композиций:
$\sum_1 |_U| \sum_2 \triangleq \cup\{\sigma_1|_U|\sigma_2 \mid \sigma_1 \in \sum \ \& \ \sigma_2 \in \sum\}$.

Такая композиция обладает свойством аддитивности: для любых двух ETS S и T имеет место $\boldsymbol{tr}(S|_U|T) = \boldsymbol{tr}(S)|_U|\boldsymbol{tr}(T)$.

Композиция ATS и трасс действий является частным случаем композиции ETS и трасс событий, если считать, что $\boldsymbol{A} \subseteq \boldsymbol{E}$, $U \subseteq \boldsymbol{A}\backslash\{\gamma\}$, $\boldsymbol{Dom}(|_U|) = \{(a,a) | a \in U\}$ и, если $a \in U$, то $a|_U|a = a$.

После композиции с помощью оператора ***hide*** скрываются некоторые переходы по синхронным событиям, переходы по кнопкам не скрываются. Множество скрываемых символов обозначим $H \subseteq U$[5].

Для $S = Ets(V_S, E, s_0)$ формально $\boldsymbol{hide}(S, H) = Ets(V_S, E, s_0)$, где E – наименьшее множество, порождаемое следующими правилами вывода: $\forall s, s', a$

3.1: $a \notin H$ $\&s \xrightarrow{a} s'$ $\vdash s \xrightarrow{a} s'$,

3.2: $a \in H$ $\&s \xrightarrow{a} s'$ $\vdash s \xrightarrow{\tau} s'$.

Для трассы событий $\sigma \in \boldsymbol{E}^*$ определяется трасса $\boldsymbol{hide}(\sigma, H)$, получаемая удалением из σ всех событий, принадлежащих H:
$\forall \sigma \in \boldsymbol{E}^* \ \forall a \in \boldsymbol{E}$

4.0: $\vdash \boldsymbol{hide}(\epsilon, H) = \epsilon$,

4.1: $a \notin H$ $\vdash \boldsymbol{hide}(\sigma \cdot a, H) = \boldsymbol{hide}(\sigma, H) \cdot a$,

4.2: $a \in H$ $\vdash \boldsymbol{hide}(\sigma \cdot a, H) = \boldsymbol{hide}(\sigma, H)$.

Оператор сокрытия для трасс естественно распространяется на множества трасс: для $H \subseteq U$ и множества трасс $\sum \subseteq \boldsymbol{E}^*$ имеем $\boldsymbol{hide}(\sum, H) \triangleq \{\boldsymbol{hide}(\sigma, H) | \sigma \in \sum\}$.

[4] Причина, по которой может скрываться только часть синхронных символов, – возможность моделирования "широковещания" (broadcasting): переход в композиции нескольких ATS соответствует переходам в нескольких (не обязательно двух) операндах.

[5] На самом деле условие $H \subseteq U$ для оператора сокрытия излишне, однако после композиции сокрытие асинхронных событий вряд ли имеет смысл. Более того, после композиции достаточно скрывать только синхронные события из $\boldsymbol{Im}(|_U|) \subseteq U$, поскольку другие синхронные события в композиции заведомо отсутствуют.

Оператор сокрытия обладает следующим важным свойством: для любой ETS S и любого $H \subseteq U$ имеет место

$\boldsymbol{tr}(\boldsymbol{hide}(S,H)) = \boldsymbol{hide}(\boldsymbol{tr}(S),H)$.

Вместе со свойством аддитивности композиции ETS имеем:

$\boldsymbol{hide}(\boldsymbol{tr}(S|_U|T),H)) = \boldsymbol{tr}(\boldsymbol{hide}(S|_U|T,H)) =$
$= \boldsymbol{hide}(\cup(\boldsymbol{tr}(S)|_U|\boldsymbol{tr}(T)),H)$.

Очевидно, оператор сокрытия для ATS и трасс действий является частным случаем оператора сокрытия для ETS и трасс событий.

4. КОМПОЗИЦИЯ СПЕЦИФИКАЦИЙ

В общем виде проблема композиции – это проблема согласования спецификации композиционной системы со спецификациями её компонентов. Спецификацию системы называют *корректной* [3, 6], если композиция реализаций, конформных спецификациям компонентов, конформна спецификации системы. Это называется сохранением конформности при композиции [13, 14] или монотонностью композиции [2, 3, 6, 15]. Как проверить, что заданная спецификация системы корректна? Понятно, что если спецификация не предъявляет никаких требований к реализациям, то ей конформна любая реализация и, следовательно, она является корректной спецификацией любой композиционной системы. Поэтому интерес представляет поиск такой корректной спецификации системы, которая предъявляла бы к реализациям наибольшие требования по сравнению со всеми корректными спецификациями системы. Такую «наибольшую» корректную спецификацию системы называют композицией спецификаций компонентов [3, 6]. Основная проблема композиции, таким образом, – это построение композиции спецификаций.

В [1] спецификация в $\boldsymbol{B/O}$-семантике определялась как множество конечных трасс наблюдений, понимаемых как ошибки (1-го рода). Реализация конформна, если в ней нет ошибок. Более строго: OTS-реализация I конформна спецификации S, если $\boldsymbol{tr}(I) \cap S = \varnothing$. Множество OTS-реализаций, конформных спецификации S, обозначим $\boldsymbol{Oconf}(S)$.[6] Соответственно, ETS-реализация I конформна спецификации S, если $\boldsymbol{tr}(\boldsymbol{f}(I)) \cap S = \varnothing$. Множество ETS-реализаций, конформных спецификации S, обозначим $\boldsymbol{Econf}(S)$. Очевидно, что $\boldsymbol{f}(\boldsymbol{Econf}(S)) \subseteq \boldsymbol{Oconf}(S)$.

Для любой LTS S (в частности, OTS и ETS) существует префиксная зависимость между её трассами: множество трасс LTS вместе с каждой трассой содержит и все её префиксы. Также для OTS и ETS в каждом состоянии имеется переход по каждой кнопке, поскольку отсутствие в состоянии перехода по кнопке трактуется как наличие перехода-петли по этой кнопке. Это даёт ещё одну зависимость между трассами: если есть трасса σ, то есть и трасса $\sigma \cdot p$ для любой кнопки p. Наличие этих двух тривиальных зависимостей определяет, кроме ошибок 1-го рода, непосредственно задаваемых спецификацией, другие ошибки – *неконформные* трассы, то есть трассы, не встречающиеся в конформных реализациях. Ошибки, не являющиеся ошибками 1-го рода, называются ошибками 2-го рода. *Первичными* ошибками называются ошибки, все строгие префиксы которых конформны. Это даёт возможность как минимизации, так и максимизации спецификации. Процедура минимизации[7] строит множество первичных ошибок: 1) добавляет ошибку σ, если ошибкой является трасса $\sigma \cdot p$, где p – кнопка, и 2) удаляет из спецификации каждую ошибку, некоторый строгий префикс которой также является ошибкой. После систематического применения эти двух правил к спецификации S получается минимальная спецификация, которую обозначим $\min(S)$. Процедура максимизации строит множество всех ошибок, она также систематически применяет два правила: такое же правило 1 и обратное правило 2, то есть добавляет в спецификацию каждое продолжение каждой ошибки. Для спецификации S максимальную спецификацию, обозначим $\max(S)$.

Рассмотрим композиционную систему, получаемую в результате композиции двух компонентов. Пусть заданы спецификации компонентов $S_1 \subseteq \boldsymbol{O}^*$ и $S_2 \subseteq \boldsymbol{O}^*$, синхронный алфавит $U \subseteq \boldsymbol{E}\backslash\{\gamma\}$ и множество сокрытия $H \subseteq U$. Формальное определение корректности спецификации системы $S \subseteq \boldsymbol{O}^*$ следующее:

S *корректна* $\triangleq \forall I_1 \in \boldsymbol{Econf}(S_1) \forall \boldsymbol{I}_2 \in$

[6] В [1], где рассматривались только OTS-реализации, вместо $O(S)$ мы писали C_S.

[7] В [1] эта процедура называлась нормализацией.

$\in \boldsymbol{Econf}(S_2)\boldsymbol{hide}(I_1|_U|I_2, H) \in \boldsymbol{Econf}(S)$.
Если реализации-операнды берутся из фиксированных подклассов реализаций $\boldsymbol{I}_1$ и $\boldsymbol{I}_2$, то это определение соответствующим образом меняется:
S *корректна* $\triangleq \forall I_1 \in \boldsymbol{Econf}(S_1) \cap \boldsymbol{I}_1 \forall I_2 \in$
$\in \boldsymbol{Econf}(S_2) \cap \boldsymbol{I}_2\ \boldsymbol{hide}(I_1|_U|I_2, H) \in \boldsymbol{Econf}(S)$.
Формальное определение композиции спецификаций $S_1|_U|S_2 \subseteq O^*$ следующее:
$S = S_1|_U|S_2 \triangleq S$ *корректна* $\&\forall S'$ (S' *корректна* $\Rightarrow \max(S') \subseteq \max(S)$).

Заметим, что спецификация системы, корректная на классах всех реализаций, очевидно, корректна на любых подклассах реализаций. Но спецификация системы, корректная на подклассах реализаций, может быть некорректной на классах всех реализаций и, следовательно, может предъявлять к реализации системы большие требования, чем композиция спецификаций на классах всех реализаций.

Достаточно очевидно следующее утверждение: множество трасс наблюдений, которые не генерируются трассами событий, встречающимися в композициях конформных реализаций, является композицией спецификаций. Это множество трасс является дополнением до $(\boldsymbol{B} \cup \boldsymbol{O})^*$ множества трасс наблюдений, которое генерируются множеством трасс событий, встречающихся в композициях конформных реализаций. Последнее множество трасс событий, как показано в разделе 3, совпадает с множеством попарных композиций трасс событий, которые встречаются в конформных реализациях 1-го и 2-го операндов, соответственно. Трасса событий встречается в конформных реализациях i-го операнда тогда и только тогда, когда генерируемая ею трасса наблюдений конформна относительно спецификации S_i. Такие трассы событий получаются при помощи функции f^{-1} из множества конформных трасс наблюдений, а это множество конформных трасс наблюдений является дополнением до $(\boldsymbol{B} \cup \boldsymbol{O})^*$ множества всех ошибок $\max(S_i)$.

Эти рассуждения (прочитанные в обратном порядке) дают следующую процедуру построения композиции спецификаций на классе всех реализаций.

1. Строим множество A_i трасс наблюдений, не встречающихся в конформных OTS-реализациях i-го компонента: $A_i = \max(S_i)$, где S_i – спецификация i-го компонента. Заметим, что это множество постфикс-замкнуто (вместе с каждой трассой содержит и все её продолжения).

2. Строим множество B_i трасс наблюдений, встречающихся в конформных OTS-реализациях i-го компонента: $B_i = (\boldsymbol{B} \cup \boldsymbol{O})^* \backslash A_i$. Заметим, что это множество префикс-замкнуто (вместе с каждой трассой содержит и все её префиксы).

3. Строим множество C_i трасс событий, встречающихся в конформных ETS-реализациях i-го компонента: $C_i = \cup\{\boldsymbol{f}^{-1}(\sigma)\ |\sigma \in B_i\}$.

4. Строим множество D попарных композиций трасс событий, которые встречаются в конформных ETS-реализациях 1-го и 2-го операндов: $D = C_1|_U|C_2$.

5. Выполняем сокрытие: $E = \boldsymbol{hide}(D, H)$.

6. Строим множество F трасс наблюдений, которые генерируются трассами событий, встречающимися в композициях конформных ETS-реализаций операндов: $F = \boldsymbol{f}(E)$. Заметим, что это множество префикс-замкнуто.

7. Строим множество G трасс наблюдений, которые не генерируются трассами событий, встречающимися в композициях конформных реализаций: $G = (\boldsymbol{B} \cup \boldsymbol{O})^* \backslash F$. Заметим, что это множество постфикс-замкнуто.

Полученное множество G является максимальной композицией спецификаций, то есть $G = \max(S_1|_U|S_2)$. Далее его можно минимизировать, чтобы искомая спецификация системы содержала только первичные ошибки:
$S = \min(G)$.

Заметим, что если рассматриваются подклассы реализаций-операндов $\boldsymbol{I}_1$ и $\boldsymbol{I}_2$, то меняются процедуры максимизации и минимизации спецификации, поскольку они должны опираться на ту нетривиальную зависимость между ошибками, которая имеется для этих подклассов. Это относится к пункту 1, в котором выполняется максимизация спецификации, и к минимизации

построенной композиции спецификаций. Но даже после этого композиция спецификаций может содержать "лишние" ошибки, то есть трассы, которые не встречаются в композициях реализаций из классов $\boldsymbol{I}_1$ и $\boldsymbol{I}_2$. Конечно, такие ошибки не влияют на результат тестирования, но их удаление могло бы оптимизировать генерацию тестов: не нужны тесты, которые ловят "лишние" ошибки. Для того чтобы это сделать в процедуре композиции спецификаций меняются только пункты 2 и 7, в которых строится дополнение множества трасс. В п. 2 множество A_i нужно дополнять не до множества всех трасс наблюдений $(\boldsymbol{B}\cup\boldsymbol{O})^*$, а до множества трасс наблюдений, встречающихся в реализациях из $\boldsymbol{I}_i$: $\cup\{\boldsymbol{tr}(\boldsymbol{f}(I))|I \in \boldsymbol{I}_i\}$. В п. 7 множество F нужно дополнять не до множества всех трасс наблюдений $(\boldsymbol{B} \cup \boldsymbol{O})^*$, а до множества трасс наблюдений, встречающихся в композициях реализаций из $\boldsymbol{I}_1$ и $\boldsymbol{I}_2$: $\cup\{\boldsymbol{tr}(\boldsymbol{f}(\boldsymbol{hide}(I_1|_U|I_2, H))) \ |I_1 \in \boldsymbol{I}_1 \ \& \ I_2 \in \boldsymbol{I}_2\} = \cup\{\boldsymbol{f}(\boldsymbol{hide}(\boldsymbol{tr}(I_1)|_U|\boldsymbol{tr}(I_2), H)) \ |I_1 \in \boldsymbol{I}_1 \ \& \ I_2 \in \boldsymbol{I}_2\}$.

Для практического применения описанная процедура обладает тем недостатком, что некоторые получающиеся в ней множества трасс бесконечны. Например, если спецификация компонента S_i конечна, то после максимизации множество $\max(S_i)$ бесконечно (кроме вырожденного случая, когда универсумы кнопок и наблюдений пусты). В то же время, если множество трасс регулярно, его можно представить конечным образом в виде конечного порождающего графа. Такой порождающий граф отличается от LTS $S = Lts(V_S, E_S, s_0)$ только тем[8], что выделяется подмножество конечных состояний $T_S \subseteq V_S$. Мы будем обозначать этот граф как S^g, а множество порождаемых им трасс как $\boldsymbol{tr}(S^g)$. LTS – это порождающий граф, все состояния которого конечны.

Как известно, любой порождающий граф S^g можно сделать детерминированным с сохранением множества порождаемых им трасс $\boldsymbol{tr}(S^g)$. Это делается с помощью процедуры детерминизации, строящей новый детерминированный порождающий граф $\boldsymbol{d}(S^g)$. Его состояниями будут все непустые множества состояний графа S^g, начальное состояние – множество состояний, достижимых из начального состояния графа S^g по пустым переходам. Множество состояний U объявляется конечным, если хотя один его элемент $u \in U$ является конечным состоянием графа S^g. Переход $U \xrightarrow{a} U'$ проводится тогда и только тогда, когда существует $u \in U$ такое, что в графе S^g есть переход $u \xrightarrow{a} u'$, и U' – множество концов u' всех таких переходов.

Будем считать, что универсумы $\boldsymbol{B}$, $\boldsymbol{O}$ и $\boldsymbol{E}$ конечны. Перепишем нашу процедуру для случая, когда спецификации компонентов S_1 и S_2 заданы конечными порождающими графами S_1^g и S_2^g: $\boldsymbol{tr}(S_1^g) = S1$ и $\boldsymbol{tr}(S_2^g) = S_2$.

1. Строим граф A_i^g, порождающий множество $A_i = \max(S_i)$.
   a. Для этого в S_i^g начальное состояние каждого перехода по кнопке, ведущего в конечное состояние, объявляем также конечным.
   b. Добавляем новое конечное состояние ω.
   c. В каждом конечном состоянии (включая ω) проводим переход в состояние ω по всем кнопкам и наблюдениям. Заметим, что это соответствует постфикс-замкнутости множества ошибок.
2. Строим граф B_i^g, порождающий множество $B_i = (\boldsymbol{B} \cup \boldsymbol{O})^* \backslash A_i$.
   a. Для этого сначала применяем процедуру детерминизации, то есть строим граф $\boldsymbol{d}(A_i^g)$.
   b. Добавляем в $\boldsymbol{d}(A_i^g)$ новое конечное состояние ω', в котором проводим переходы-петли по всем кнопкам и наблюдениям.
   c. В каждом неконечном состоянии проводим переход в состояние ω' по всем кнопкам и наблюдениям, по которым не было переходов из этого состояния.
   d. Удаляем все конечные состояния вместе с ведущими в них переходами.
   e. Все оставшиеся состояния объявляем конечными. Заметим, что это соответствует префикс-замкнутости множества конформных трасс наблюдений.

[8] Еще одно отличие чисто формальное: в порождающем графе τ-переходу соответствует пустой переход (непомеченная дуга графа).

3. Строим граф C_i^g, порождающий множество $C_i = \cup\{\boldsymbol{f}^{-1}(\sigma) \mid \sigma \in B_i\}$.

   Для этого в B_i^g каждый переход из состояния s по наблюдению x превращаем во множество кратных переходов по каждому событию a такому, что $x = \boldsymbol{f}(a)$. Если таких событий a нет, то переход по наблюдению x просто удаляется.

4. Строим граф D^g, порождающий множество $D = C_1|_U|C_2$.

   Для этого применяем композицию $D^g = C_1^g|_U|C_2^g$, описанную в разделе 3.

5. Строим граф E^g, порождающий множество $E = \boldsymbol{hide}(D, H)$.

   Для этого применяем операцию сокрытия $E^g = \boldsymbol{hide}(D^g, H)$, описанную в разделе 3.

6. Строим граф F^g, порождающий множество $F = \boldsymbol{f}(E)$.

   Для этого в E^g каждый переход из состояния s по событию $a \in \boldsymbol{Dom}(f)$ превращаем в переход по наблюдению $f(a)$. Если $a \notin \boldsymbol{Dom}(f)$, то переход по такому событию просто удаляется.

7. Строим граф G^g, порождающий множество $G = (\boldsymbol{B} \cup \boldsymbol{O})^* \setminus F$.

   a. Для этого сначала применяем процедуру детерминизации, то есть строим граф $\boldsymbol{d}(F^g)$.

   b. Добавляем в $\boldsymbol{d}(F^g)$ новое состояние ω', в котором проводим переходы-петли по всем кнопкам и наблюдениям. Это состояние и только его объявляем конечным.

   c. В каждом состоянии проводим переход в состояние ω' по всем кнопкам и наблюдениям, по которым не было переходов из этого состояния. Заметим, что это соответствует постфикс-замкнутости множества ошибок. Также отметим, что у нас могут получиться состояния, из которых не достижимо состояние ω'. Такие состояния можно удалить.

Полученный граф G^g порождает максимальную композицию спецификаций, то есть $\boldsymbol{tr}(G^g) = \max(S_1|_U|S_2)$. Этот граф можно минимизировать, чтобы получить граф S^g, порождающий минимальную композицию спецификаций (множество первичных ошибок), то есть $\boldsymbol{tr}(S^g) = \min(S_1|_U|S_2)$. Для этого систематически применяем следующие два правила: 1) если переход по кнопке ведет из состояния s в конечное состояние s', то объявляем состояние s конечным, 2) переход из конечного состояния удаляется. В конце можно удалить получившиеся изолированные состояния.

5. МОДЕЛИРОВАНИЕ ДРУГИХ СЕМАНТИК

В [1] рассматривались различные семантики тестирования, задаваемые соответствующими машинами тестирования, и моделирование каждой такой семантики T_i в $\boldsymbol{B/O}$-семантике, определяемое как соответствующее этой семантике преобразование реализации $\boldsymbol{M}_i : ATS \to OTS$. В этой статье мы будем моделировать эти семантики в $\boldsymbol{B/E}$-семантике, то есть рассмотрим преобразование $\boldsymbol{W} : ATS \to ETS$, после чего далее можно преобразовать ETS в OTS, задав подходящую функцию $\boldsymbol{f}$, как описано в разделе 2.

В рассматриваемых семантиках реализация задаётся ATS, то есть LTS, в которой переходы помечены внешними действиями из универсума $\boldsymbol{A}$ или символом τ. Тестовое воздействие разрешает реализации выполнять внешние действия из некоторого множества разрешённых действий, разрушение γ и внутреннее τ-действие всегда разрешены. Машину называют генеративной или реактивной в зависимости от того, может ли реализация выполнять последовательность разрешённых внешних действий или только одно такое действие, после чего выполнение внешних действий блокируется до поступления следующего (быть может, того же самого) тестового воздействия. Если в текущем состоянии реализации нет разрешённых внешних действий и нет τ-активности, реализация останавливается. Эту остановку можно наблюдать, если активирована, так называемая, зелёная лампочка:

она горит, пока реализация выполняет какие-то действия, и гаснет в момент остановки. После остановки тестирование либо заканчивается, либо может быть продолжено тем или иным тестовым воздействием. Наконец, при остановке машины можно наблюдать действия, переходы по которым определены в текущем состоянии реализации. Для этого предназначены лампочки меню, соответствующие действиям. Если такая лампочка активирована, то в момент остановки она горит, если в реализации есть переход по соответствующему действию. Все эти дополнительные наблюдения систематизированы в [8], семантики с ограничениями на множество разрешаемых действий рассматривались в [2, 3, 6, 11, 12].

В данной статье мы будем считать все эти возможности атрибутами кнопки, что позволяет рассматривать широкий класс семантик, основанных на ATS, в том числе "смешанные" семантики, когда, например, зелёная лампочка активирована для одних кнопок и не работает для других кнопок. Формально, будем считать, что фиксировано подмножество универсума кнопок $\boldsymbol{P} \subseteq \boldsymbol{B}$, для которых определены следующие функции, всюду определённые на $\boldsymbol{P}$:

- $\boldsymbol{acts}$: $\boldsymbol{P} \to 2^A$, разрешаемые внешние действия, кроме всегда разрешённого разрушения.
- $\boldsymbol{reactive}$: $\boldsymbol{P} \to \boldsymbol{Bool}$, если $true$, то кнопка реактивная, то есть следующую кнопку можно нажимать только после выполнения внешнего действия или после наблюдаемой остановки реализации; если $false$, то кнопка генеративная, что не налагает ограничений на нажатие следующей кнопки.
- $\boldsymbol{green}$: $\boldsymbol{B} \to \boldsymbol{Bool}$, если $true$, то можно наблюдать остановку реализации.
- $\boldsymbol{continue}$: $\boldsymbol{P} \to \boldsymbol{Bool}$, если $true$, то можно продолжать тестирование после остановки.
- $\boldsymbol{menu}$: $\boldsymbol{P} \to 2^{\boldsymbol{A}}$, активированные лампочки меню.

Говорят, что в системе нет приоритетов, если правило недетерминированного выбора предписывает реализации выбирать на выполнение любое действие a (внешнее $a \in \boldsymbol{A}$ или внутреннее $a = \tau$), которое в ней определено, то есть в текущем состоянии имеется переход по a, и которое разрешено нажатой кнопкой $p \in \boldsymbol{P}$, то есть $a \in \boldsymbol{acts}(p) \cup \{\tau, \gamma\}$. В [1] рассматривались также системы с приоритетами, введённые в [4, 5], когда выполнимость действия a зависит от нажатой кнопки p, а не только от условия $a \in \boldsymbol{acts}(p) \cup \{\tau, \gamma\}$. Такие системы моделируются ATS, в которых переход помечается не действием a, а парой (a, p), где $p \in \boldsymbol{P}$ и $a \in \boldsymbol{acts}(p) \cup \{\tau, \gamma\}$. Такой переход выполняется с наблюдением действия a только в том случае, когда нажата кнопка p. Понятно, что систему без приоритетов можно понимать как частный случай системы с приоритетами, если заменить в ней каждый переход по действию a на множество кратных переходов по парам (a, p), где p пробегает множество $\{p \in \boldsymbol{P} | a \in \boldsymbol{acts}(p) \cup \{\tau, \gamma\}\}$. В дальнейшем будем рассматривать только системы с приоритетами. Для таких систем лампочка меню должна соответствовать уже не действию, а паре (действие, кнопка). Теперь τ-активность также управляется кнопками, то есть вместо τ-переходов теперь есть (τ, p)-переходы, где $p \in \boldsymbol{P}$. Поэтому следует говорить не о стабильности, а p-стабильности состояния, означающей отсутствие в состоянии переходов, помеченных кнопкой p, то есть парой вида (a, p). Соответственно, парам вида (τ, p) также соответствуют лампочки меню. Поэтому будем считать, что $\boldsymbol{menu} : \boldsymbol{P} \to 2^{(\boldsymbol{A} \cup \{\tau\}) \times \boldsymbol{P}}$.

В таблице 1 приведены семантики, рассматривавшиеся в [1, 2, 3, 6, 8, 11, 12], с указанием допустимых для них кнопок. Все действия из $\boldsymbol{A}$ будем считать событиями: $\boldsymbol{A} \subseteq \boldsymbol{E}$. Кроме этого вводятся события вида (r, p), где $r \in 2^{(\boldsymbol{A} \cup \{\tau\}) \times \boldsymbol{P}}$ – множество готовности, а $p \in \boldsymbol{P}$ кнопка. Обозначим множество готовности в состоянии s через $r(s) = \{(a, p) | s \xrightarrow{a,p}\}$. Итак, мы вводим множество событий $\boldsymbol{A} \cup (2^{(\boldsymbol{A} \cup \{\tau\}) \times \boldsymbol{P}} \times \boldsymbol{P}) \subseteq \boldsymbol{E}$.

Для моделирования этих семантик в $\boldsymbol{B/E}$-семантике определим преобразование реализации $\boldsymbol{W}{:}ATS \to ETS$. Также как при преобразовании $\boldsymbol{M}{:}ATS \to OTS$ для каждого состояния s исходной ATS и каждой кнопки $p \in \boldsymbol{P}$ добавляется новое состояние s_p и переход $s \xrightarrow{p} s_p$. В каждом состоянии s_p определяется переход по действию $a \in \boldsymbol{acts}(p) \cup \{\tau, \gamma\}$, если в состоянии s

Таблица 1. $reactive = false$ в генеративной машине ван Глаббека, $reactive = true$ в реактивной R/Q-машине и *ioco*-машине. Для множества Z через $Fin(Z)$ обозначено семейство конечных подмножеств множества Z.

Семантика в алфавите $A \subseteq \boldsymbol{A}$		***допустимые кнопки из P***					***полное наблюдение*** (r^+, r^-)	***интерпретация наблюдения в семантике***
		reactive $=$	***acts*** $\in$	***green*** $=$	***menu*** $\in$	***continue*** $=$		
trace	T	$\times$ $(false)$	2^A	$false$	$\times$	*times*	–	–
	T^*	$\times$ $(false)$	$\{A\}$	$false$	$\times$	$\times$		
failure trace	FT	$\times$ $(false)$	2^A	*true*	$\{\varnothing\}$	*true*	$r^+ = \varnothing$, $r^- =$ $= \boldsymbol{acts}(p) \times$ $\times \{p\}$	refusal $\boldsymbol{acts}(p)$
failure	F	$\times$ $(false)$	2^A	*true*	$\{\varnothing\}$	$false$		
failure trace	FT	$\times$ $(false)$	2^{A-}	*true*	$\{\varnothing\}$	*true*		
failure	F	$\times$	2^A	*true*	$\{\varnothing\}$	$false$		
ready trace	RT	$\times$ $(false)$	2^A	*true*	$\{(\boldsymbol{A} \cup \{\tau\}) \times$ $\times \boldsymbol{P}\}$	*true*	$r^+ = r$ $r^- = (\boldsymbol{A} \times$ $\times \boldsymbol{P}) \backslash r$	ready r
readiness	R	$\times$ $(false)$	2^A	*true*	$\{(\boldsymbol{A} \cup \{\tau\}) \times$ $\times \boldsymbol{P}\}$	$false$		
ready trace	RT^-	$\times$ $(false)$	2^{A-}	*true*	$\boldsymbol{Fin}((\boldsymbol{A} \cup \{\tau\}) \times$ $\times \boldsymbol{P})$	*true*	r^+, r^-	(r^+, r^-)
readiness	R^-	$\times$	2^{A-}	*true*	$\boldsymbol{Fin}((\boldsymbol{A} \cup \{\tau\}) \times$ $\times \boldsymbol{P})$	$false$		
completed traces	TO	$\times$ $(false)$	$\{A\}$	*true*	$\{\varnothing\}$	$\times$	$r^+ = \varnothing$, $r^- = A \times$ $\times \{p\}$	refusal $0 = A =$ $= \boldsymbol{acts}(p)$
stability	$T0S$	$\times$ $(false)$	$\{A\}$	*true*	$\{\varnothing\}$	$\times$		
			$\{\varnothing\}$			*true*	$r^+ = \varnothing$, $r^- = \varnothing \times$ $\times \{p\} = \varnothing$	refusal $S = \varnothing =$ $= \boldsymbol{acts}(p)$
$\boldsymbol{R} \cap \boldsymbol{Q} =$ $= \varnothing$ и	$\boldsymbol{R/Q}$	$\times(true)$	$\boldsymbol{R}$	*true*	$\{\varnothing\}$	*true*	$r^+ = \varnothing$, $r^- =$ $= \boldsymbol{acts}(p) \times$ $\times \{p\}$	refusal $\boldsymbol{acts}(p)$
$\cup \boldsymbol{R} \cup \cup \boldsymbol{Q} =$ $= A$			$\boldsymbol{Q}$	$false$	$\times$	$\times$	–	–
$X \cap Y =$ $= \varnothing$, $X \cup Y =$ $= A$	*ioco*	$\times(true)$	$\{\delta\}$	*true*	$\{\varnothing\}$	*true*	$r^+ = \varnothing$, $r^- = \delta \times$ $\times \{p\} =$ $= \{(\delta, p)\}$	refusal $\delta = Y =$ $= \boldsymbol{acts}(p)$
и $\delta = Y$			$\{\{x\} \mid \in$ $\in X\}$	$false$	$\times$	*times*	–	–

имелся переход $s \xrightarrow{a,p} t$. Если $\boldsymbol{reactive}(p) = true$ и $a \in \boldsymbol{acts}(p)$, то добавленный переход тоже ведёт в состояние t. Если $\boldsymbol{reactive}(p) = false$ или $a \in \{\tau, \gamma\}$, то добавленный переход ведёт в состояние t_p. В каждом состоянии s_p, где $\boldsymbol{reactive}(p) = false$, для каждой кнопки $q \in \boldsymbol{P}$ определяется переход $s_p \xrightarrow{q} s_q$. Для каждого состояния s и каждой кнопки $p \in \boldsymbol{P}$ добавляется переход из s_p по событию (r, p), если s p-стабильно, $\boldsymbol{green}(p) = true$, и $r = r(s)$, где $r(s)$ вычисляется по исходной ATS. Условие p-стабильности состояния s такое: $(\tau, p) \notin r(s)$ & $(\tau, p) \notin r(s)$. Этот

переход ведёт в состояние s_p, если $\boldsymbol{continue}(p) = true$ и $\boldsymbol{reactive}(p) = false$, или в состояние s, если $\boldsymbol{continue}(p) = true$ и $\boldsymbol{reactive}(p) = true$, или в какое-нибудь терминальное состояние (при необходимости такое состояние добавляется), если $\boldsymbol{continue}(p) = false$. После этого удаляются все (a, p)-переходы из старых состояний, где $a \in \boldsymbol{A}$ и $a \neq \gamma$, а $(\tau, \varnothing)$- и $(\gamma, \varnothing)$-переходы заменяются τ- и γ-переходами, соответственно. Начальное состояние остаётся прежним, предполагается, что в начале тестирования никакая кнопка не нажата (все внешние действия, кроме разрушения, запрещены), а зелёная лампочка не активирована.

Теперь определим функцию $\boldsymbol{f}$ для событий из $\boldsymbol{A} \cup (2^{(\boldsymbol{A} \cup \{\tau\}) \times \boldsymbol{P}} \times \boldsymbol{P})$. Для действий будем считать, что $\boldsymbol{A} \subseteq \boldsymbol{Dom}(\boldsymbol{f})$ и $\boldsymbol{f}(a) = a$ для каждого $a \in \boldsymbol{A}$. Для события $(r, p) \in 2^{(\boldsymbol{A} \cup \{\tau\}) \times \boldsymbol{P}} \times \boldsymbol{P}$ условие наблюдения, то есть $(r, p) \in \boldsymbol{Dom}(\boldsymbol{f})$, следующее $\{(a, x) \in r | x = p\} = \varnothing$. Это условие наблюдаемой остановки машины в состоянии с множеством готовности r при нажатой кнопке p. Если $(r, p) \in \boldsymbol{Dom(f)}$, то $\boldsymbol{f}(r, p) = (r^+, r^-)$, где $r^+ = \boldsymbol{menu}(p) \cap r$, $r^- = (\boldsymbol{menu}(p) \backslash r) \cup (\boldsymbol{acts}(p) \times \{p\})$. Здесь r^+ – это множество пар (a, p), для которых лампочки меню активированы и горят. Про каждую пару $(a, p) \in r^+$ в момент остановки реализации мы точно знаем, что имеется переход, помеченный этой парой, и он может выполняться при нажатии кнопки p. Соответственно, r^- – это множество всех пар (a, p), где p – нажатая кнопка, а также таких пар (a, p), для которых лампочки меню активированы, но не горят. Про каждую пару $(a, p) \in r^-$ в момент остановки реализации мы точно знаем, что отсутствуют переходы, помеченные этой парой.

Те семантики, в которых имеется событие (r, p), наблюдение (r^+, r^-) может интерпретироваться тем или иными способом, если семантика (как набор кнопок) налагает ограничения на возможные значения множеств r^+ и r^-. Например, в семантике трасс с отказами *(FT – failure trace semantics)* лампочек меню нет (не активированы), поэтому для события (r, p) всегда будет $r^+ = \varnothing$ и $r^- = acts(p) \times \{p\}$, что однозначно задаётся множеством $acts(p)$, которое называется *отказом* $(refusal\ set)$ в момент остановки реализации при нажатой кнопке p. В семантике с конечными множествами разрешённых действий и конечными множествами активированных лампочек меню (RT^- и R^-) множества r^+ и r^- конечны. В семантике завершённых трасс *(T0 – completed trace semantics)* возможен только один вид отказа, возникающий в терминальных состояниях. Это наблюдение обозначается как 0, что совпадает с отказом, содержащим все внешние действия. Наблюдение S возникает при любой наблюдаемой остановке реализации, то есть ровно тогда, когда наблюдается пустой отказ. В $\mathbf{R}/\mathbf{Q}$-семантике наблюдаемые отказы – это элементы семейства $\boldsymbol{R}$, а в семантике конформности *ioco (Input Output COnformance)* имеется единственный отказ, обозначаемый символом δ и означающий отсутствие в реализации переходов по реакциям (из множества Y) в момент остановки.

Для конкретной семантики T_i определим преобразование $\boldsymbol{D}_i{:}ETS \to ETS$, которое удаляет все переходы по кнопкам, не допустимым в этой семантике. С учётом описанной выше интерпретации общего наблюдения (r^+, r^-) в той или иной семантике T_i легко видеть, что моделирование $\boldsymbol{M}_i$, описанное в [1], даёт тот же результат, что некоторое общее моделирование $\boldsymbol{M}{:}ATS \to OTS$, после которого удаляются переходы по кнопкам, не допустимым в семантике T_i, то есть применяется преобразование $\boldsymbol{D}_i$. Это универсальное преобразование $\boldsymbol{M}$ отличается от преобразования $\boldsymbol{W}$ только тем, что вместо события (r, p) переход помечается наблюдением $\boldsymbol{f}(r, p) = (r^+, r^-)$. Легко видеть, что при моделировании сохраняются трассы наблюдений: в ATS S множество трасс наблюдений и кнопок, соответствующих семантике T_i, равно $\boldsymbol{tr}(\boldsymbol{f}(\boldsymbol{D}_i(\boldsymbol{W}(S)))) = \boldsymbol{tr}(\boldsymbol{D}_i(\boldsymbol{M}(S)))$.

Нам осталось определить композицию введённых нами событий. Асинхронными могут быть только внешние действия, причём разрушение всегда асинхронно, а события вида (r, p) всегда синхронны. После композиции будем скрывать все синхронные переходы по действиям и только их. Итак: $H \subseteq \boldsymbol{A} \backslash \{\gamma\}$, $U = H \cup (2^{(\boldsymbol{A} \cup \{\tau\}) \times \boldsymbol{P}} \times \boldsymbol{P})$.

Определение композиции опирается на следующие основные свойства множества готовности. Если состояния s и t двух ATS-операндов p-стабильны, то композиционное состояние st p-стабильно тогда и только тогда, когда в состо-

яниях s и t нет пары синхронных переходов, то есть переходов по одной и той же кнопке q и одному и тому же синхронному действию a:$\{(a,q) \in r_1 \cap r_2 | a \in H\} = \varnothing$. Это даёт следующее определение домена композиции событий:

$$\boldsymbol{Dom}(|_U|) = \{(a,\ a) \mid a \in \boldsymbol{A}\} \cup$$
$$\cup\{((r_1,p),\ (r_2,p)) \mid r_1, r_2 \in 2^{(\boldsymbol{A}\cup\{\tau\})\times \boldsymbol{P}} \& p \in$$
$$\in \boldsymbol{P} \& \{(a,q) \in r_1 \cap r_2 | a \in H\} = \varnothing\}.$$

Если композиционное состояние st p-стабильно, то его множество готовности вычисляется по множествам готовности состояний-операндов, а именно: оно содержит пару (a,q) тогда и только тогда, когда эта пара есть во множестве готовности хотя бы одного из состояний-операндов, а действие a асинхронно:

$\boldsymbol{r}(st) = \{(a,q) \in \boldsymbol{r}(s) \cup \boldsymbol{r}(t) | a \notin H\}$. Это даёт следующее определение значения композиции событий:

$\forall (a,a) \in \boldsymbol{Dom}(|_U|)$
$a|_U|a = a,$
$\forall ((r_1,p),(r_2,p) \in \boldsymbol{Dom}(|_U|)$
$(r_1,p)|_U|(r_2,p) = (\{(a,q) \in r_1 \cup r_2 | a \notin H\}, p).$

Теперь, опираясь на определения композиции ATS и ETS, легко увидеть выполнение свойства согласованности композиции и моделирования $\boldsymbol{W}$:$ATS \to ETS$: для любых ATS S и T множество трасс событий ETS, моделирующей композицию этих ATS, совпадает с множеством трасс событий композиции ETS, моделирующих эти ATS:

$$\boldsymbol{tr}(\boldsymbol{W}(S|_U|T)) = \boldsymbol{tr}(\boldsymbol{W}(S)|_U|\boldsymbol{W}(T)).$$

СПИСОК ЛИТЕРАТУРЫ

1. *Бурдонов И.Б., Косачев А.С.* Формализация тестового эксперимента–II. “Программирование”. 2013. № 4. С. 3–27.
2. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Формализация тестового эксперимента. “Программирование”. 2007. № 5. С. 1–33.
3. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Теория соответствия для систем с блокировками и разрушением. М.: “Наука”, 2008.
4. *Бурдонов И.Б., Косачев А.С.* Системы с приоритетами: конформность, тестирование, композиция. Труды Института системного программирования РАН. 2008. № 14.1.
5. *Бурдонов И.Б., Косачев А.С.* Системы с приоритетами: конформность, тестирование, композиция. “Программирование”. 2009. № 4. С. 24–40.
6. *Бурдонов И.* Теория конформности (функциональное тестирование программных систем на основе формальных моделей). LAP LAMBERT Academic Publishing, Saarbrucken, Germany, 2011.
7. *van Glabbeek R.J.* The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, CONCUR’90, Lecture Notes in Computer Science 458, Springer-Verlag, 1990. P. 278–297.
8. *van Glabbeek R.J.* The linear time – branching time spectrum II; the semantics of sequential processes with silent moves. Proceedings CONCUR ’93, Hildesheim, Germany, August 1993 (E. Best, ed.), LNCS 715, Springer-Verlag, 1993. P. 66–81.
9. *Milner R.* Modal characterization of observable machine behaviour. In G. Astesiano & C. Bohm, editors: Proceedings CAAP 81, LNCS 112, Springer, P. 25–34.
10. *Hoare C.A.R.* Communicating sequential processes. In R.M. McKeag A.M. Macnaghten, editors, On the construction of programs – an advanced course. Cambridge University Press, 1980. P. 229–254.
11. *Tretmans J.* Conformance testing with labelled transition systems: implementation relations and test generation. Computer Networks and ISDN Systems, Dec., 1996. V. 29. № 1. P. 49–79.
12. *Tretmans J.* Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools. 1996. V. 17. I. 3.
13. *van der Bijl M., Rensink A., Tretmans J.* Compositional testing with ioco. Formal Approaches to Software Testing: Third International Workshop, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. Editors: A. Petrenko, A. Ulrich. Springer. LNCS V. 2931. P. 86–100.
14. *van der Bijl M., Rensink A., Tretmans J.* Component Based Testing with ioco. CTIT Technical Report TR-CTIT-03-34, University of Twente, 2003.
15. *Jard C., Jeron T., Tanguy L., Viho C.* Remote testing can be as powerful as local testing. In Formal methods for protocol engineering and distributed systems, FORTE XII/ PSTV XIX’ 99, Beijing, China, J. Wu, S. Chanson, Q. Gao (eds.), October, 1999. P. 25–40.