Об одном методе анализа поведения формальных моделей распределенных программных систем

Александр Колчин

Институт кибернетики им. В.М. Глушкова НАН Украины, проспект Академика Глушкова 40, Киев, Украина

kolchin_av@yahoo.com

Аннотация. *В основе предложенного метода лежит алгоритм отсечения избыточных ветвей поведения формальной модели. Факт избыточности устанавливается на основании доказательства изоморфизма графа информационных зависимостей модели. Такой подход существенно сокращает эффект "комбинаторного взрыва" количества состояний модели.*

Ключевые слова

Верификация, проверка модели, интерливинг.

1 Введение

Комбинаторный взрыв вариантов поведения – основная проблема, с которой сталкиваются методы проверки правильности формальных моделей. Универсального решения данной проблемы не существует, однако активно создаются и развиваются методы, позволяющие минимизировать количество вариантов поведения модели (ее состояний), необходимых для проверки тех или иных свойств.

Цели предлагаемого метода близки с целями методов частичного порядка, которые служат для устранения избыточного интерливинга. Суть этих методов заключается в том, чтобы при проверке свойств рассматривать не все перестановки параллельных действий в модели, а лишь некоторые из них. Такие методы показывают высокую эффективность, когда удается синтаксически (на основании структурного описания модели) локализовать места доступа к глобальным атрибутам и каналам связи процессов [1].

Но, к сожалению, такой подход не всегда эффективен – статический анализ преувеличивает фактические информационные связи, и как следствие, существенная часть избыточных перестановок не устраняется. Более того, интерливинг часто может быть промоделирован недетерминизмом (например, когда в модели «синтаксически» всего один процесс, но его поведение представляет большой цикл недетерминированных действий).

Проблема элиминации избыточных перестановок актуальна не только для верификации, но и для повышения параллелизма – выявление независимых участков поведений дает возможность построения адекватной распределенной архитектуры разрабатываемой системы.

2 Описание метода

Описываемый метод развивает предложенный ранее метод динамической абстракции [2]. Главная отличительная особенность предлагаемого подхода заключается в способе представления состояний модели. Следует отметить, что существующие верификаторы оперируют «монолитными» состояниями (иногда [1] разделенными на части по принципу принадлежности процессу, но это скорее для экономии памяти, а не для сокращения перебора).

Так, поведение модели представляется не "плоскими" состояниями, а графом информационных зависимостей. Соответственно, процедура отсечения ветвей поведения устанавливает факт их избыточности на основании доказательства изоморфизма уже построенной части графа и графа, который может быть построен

из текущего состояния (в отличие от существующих методов, которые производят сравнение текущего состояния с ранее пройденными).

Рассмотрим пример модели, переходы которого приведены в табл.1. Такая модель допускает 20 трасс (t1={A0,A1,A2,B0,B1,B2,T}; t2={A0,A1,B0,A2,B1,B2,T}; …; t20={B0,B1,B2,A0,A1,A2,T}).

Таблица 1. Переходы примера 1

переход	предусловие	постусловие
A0	x=0	x:=1
A1	x=1	x:=2
A2	x=2	x:=3
B0	y=0	y:=1
B1	y=1	y:=2
B2	y=2	y:=3
T	x=3 & y=3	z:=1

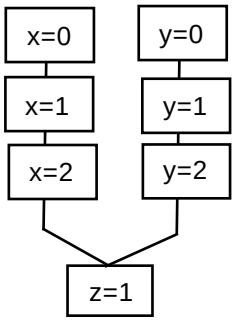

Рис 2. Граф зависимости для примера 1

При этом граф информационных зависимостей всего один (см. рис.1).

2.1 Формальная модель

Операционная семантика формальной модели может быть определена в терминах транзиционных систем. Пусть задано конечное множество атрибутов $A = v_1, v_2, ..., v_n$ и пусть также для каждого атрибута $v_i \in A$ определена конечная область допустимых значений $D(v_i)$.

***Определение* 1.** Состоянием называется множество пар атрибутов и их значений вида $\{\bigcup_{i=0..|A|}(v_i = d_i) \mid v_i \in A,\ d_i \in D(v_i)\}$.

Назовем предусловием некоторую бесквантерную формулу логики предикатов над атрибутами множества *A* и константами из соответствующих областей допустимых значений *D*. Назовем постусловием множество присваиваний вида *v* := *expr*(*s*), где $v \in A$, *s* – состояние, а $expr{:}s \to D(v)$ – некоторая функция над атрибутами состояния *s*.

***Определение* 2.** Переходом называется тройка вида (t, α, β), где *t* – имя перехода, α – его предусловие, а β – постусловие.

Семантика перехода такова: если в некотором состоянии `s` предусловие перехода `t` выполнимо, то модель может выполнить этот переход и перейти в новое состояние `s'=next(t,s)`, которое отличается от предыдущего значениями тех атрибутов, которым было выполнено присваивание новых значений в постусловии.

***Определение* 3**. Атрибутной транзиционной системой (АТС) называется шестерка вида *M* = <*S*, s_0, *T*, *A*, *D*, *F*>, где *S* – конечное множество состояний, $s_0 \in S$ – начальное состояние, *T* – конечное множество переходов, *A* – атрибутов, *D* – соответствующие области допустимых значений, $F{:}(s,v) \to D(v)$ – функция, вычисляющая значение атрибута *v* в состоянии *s*.

Аналогично охраняемым командам Дейкстры, с помощью отношения *T* задаются возможные варианты дискретных детерминированных переходов из одного множества состояний в другое. АТС является удобным математическим аппаратом для описания поведения широкого спектра формальных моделей систем асинхронно-взаимодействующих процессов и их абстракций.

***Определение* 4.** Трассой в *M* из состояния s_i в состояние s_j называется такая последовательность состояний и переходов $s_i \xrightarrow{t_i} s_{i+1} \xrightarrow{t_{i+1}} s_{i+2} \ldots s_j$, что $s_k \in S \wedge t_k \in T$ для всех $k \in i..j$.

***Определение* 5.** Состояние `s` достижимо в модели *M*, если существует трасса, ведущая из начального состояния s_0 в `s`. Множество всех достижимых состояний будем обозначать `Reachable`(*M*, s_0).

Для выполнения перехода `t` из состояния `s` в новое состояние `s'` данная работа использует процедуру `transit`; ее подробное описание приведено в [2], а модификация выполнения предусловия в [3]. Так, выполнение этой процедуры построит множество `R` атрибутов, значений которых (согласно лемме 1 [3]) достаточно для вычисления предусловия `t`, и (согласно лемме 4 [2]), если переход выполним, множество `W` атрибутов, которым осуществлялось присваивание, а так же для каждого атрибута `w`∈`W` множество `V[w]` атрибутов, которые входят в выражение, формирующее значение `w`: `(result,s',R,W,V[]) ← transit(s,t)`. Примеры ее работы с выполнимыми переходами продемонстрированы на рис. 2, 3.

Необходимо отметить, что в результате работы процедуры transit множество R-атрибутов будет содержать не обязательно все атрибуты, входящие в предусловие. Например, для вычисления $f_1 \wedge f_2 \wedge \ldots f_n$ при истинных $f_1, \ldots, f_{n-1}$ и ложном f_n, множество R будет содержать только атрибуты, входящие в f_n; аналогично, только атрибуты, входящие в f_n, будут в множестве R при вычислении $\sim(f_1 \vee f_2 \vee \ldots f_n)$ при ложных $f_1, \ldots, f_{n-1}$ и истинном f_n. На рис. 4, 5 приведены примеры таких множеств, построенных для невыполнимых переходов.

Предусловие перехода t1 : (x=a+b ∧ y=0) **Постусловие перехода t1 :** a:=a-c; x:=0
Вход: transit((a=2,b=0,c=1,x=2,y=0), t1) **Выход**: (result=**T**; s'=(a=1,b=0,c=1,x=0,y=0); R={x,a,b,y};W={a,x};V[a]={a,c},V[x]=∅)

Рис. 2. Пример 2 работы процедуры transit

Предусловие перехода t2 : (a>0 ∨ b=1) **Постусловие перехода t2:** a:=a-1; c:=b
Вход: transit((a=2, b=0, c=1), t2) **Выход**: (result=**T**; s'=(a=1, b=0, c=0); R={a};W={a,c};V[a]={a},V[c]={b}

Рис.3. Пример 3 работы процедуры transit

Предусловие t3: ~(x=a+b) ∧ y=0	
Вход – состояние S	**Выход – множество R**
a=0, b=0, x=0, y=0	(F, {x,a,b})
a=0, b=0, x=1, y=1	(F, {y})

Рис. 4. Примеры множества R

Предусловие t4: x=0 ∨ ~(y=0 ∨ z=0)	
Вход – состояние S	**Выход – множество R**
x=1,y=0,z=0	(F, {x,y})
x=1,y=1,z=0	(F, {x,z})

Рис.5. Примеры множества R

Мы будем пользоваться спецификацией зависимости, определенной над парами вида (атрибут=значение), которое включает условие различности пар и учитывает существование перехода, порождающего зависимость.

***Определение* 7.** Пары $(a=v_a, b=v_b) \in DR(M)$ имеют информационную связь тогда и только тогда, когда

$(a \neq b \vee v_a \neq v_b) \wedge \exists (s_1, s_2) \in Reachable(M, s_0)$:

$\forall x (x \neq a \rightarrow s_1.x = s_2.x \wedge s_1.a = v_a \wedge \exists t (next(t, s_1).b = v_b \wedge (next(t, s_2) = \varnothing \vee next(t, s_2).b \neq v_b))$

***Определение* 8.** Графом зависимости называется ориентированный граф, вершинами которого являются пары вида (атрибут=значение), а дуги отмечают факт зависимости.

2.2 Метод построения графа зависимости

Для построения графа зависимости используется адаптированный алгоритм Тарьяна построения компонент сильной связности. Дуги и вершины графа добавляются на основании результатов (R, W, V[W]) выполнений процедуры transit(s,t,s') следующим образом: если переход выполним, то для каждого $w \in W$ пара (w, s'.w) будет соединена дугами, выходящими из всех пар (r, s.r), $r \in R$, и из пар (v, s.v), $v \in V[w]$.

В случае, если переход невыполним, будет порождена специальная вершина с отметкой (t=**F**), в которую будут входить дуги из всех пар (r,s.r) , $r \in R$. В [4] показано, что метод построения графа зависимости сохраняет информационные связи, порождаемые определением 7. Таким образом, метод гарантирует, что если зависимость существует, то граф будет ее содержать. Далее предполагается, что информационная зависимость атрибутов модели представлена графом, построенным согласно определению 8 (далее для краткости D-граф).

2.3 Проверка пройденных состояний

При обходе пространства поведения формальной модели актуальна задача проверки вхождения текущего состояния во множество ранее пройденных (так называемых visited states). Это позволяет избежать повторных посещений состояний и определять цикличность поведения. Для этих целей существующие методы эффективно используют хеширование состояний (например, [1]).

В этом подходе заключается суть различия предлагаемого метода от существующих. Предлагаемый в этой работе метод представляет пройденную часть пространства поведения модели в виде графа информационных связей, а процедура установления повторности посещения состояния сводится к доказательству изоморфизма уже построенной части графа и графа, который может быть построен из текущего состояния.

Согласно методу построения динамических абстракций [2], для установления факта повторности посещения состояния, достаточно сравнивать состояния с точностью до множества R-атрибутов (множество пар вида `атрибут=значение`, к которым осуществлялся «доступ на чтение»). В [2] предполагалось, что пройденные состояния хранятся как монолитные («плоские»).

Усовершенствование заключается в том, что R-множество не обязательно хранить в «плоском» виде, его можно построить динамически как транзитивное замыкание по компоненте сильной связности D-графа. Это утверждение основывается на том факте, что, будучи полностью построенной, компонента сильной связности уже никогда не изменится: внутри нее не добавятся ни вершины, ни дуги.

Иными словами, для обнаружения полного множества R-атрибутов, совпадающих с текущим состоянием, предлагаемый метод повторит прохождение соответствующих участков поведения модели. Такая процедура, конечно, существенно медленнее существующих эффективных процедур поиска состояния в базе пройденных по хеш-функции. Но с другой стороны, такой метод позволяет эффективно устранять избыточные перестановки поведения модели (за счет того, что повторное прохождение осуществляется только по D-графу).

Для повышения эффективности этой процедуры разработан алгоритм разметки D-графа, так, что при установлении факта повторности, наносится специальная разметка, позволяющая последующий анализ подграфа сводить к анализу метки вершины. Например, если, однажды проанализировав подграф G, и установив, что он не будет изоморфным (по отношению к подграфу, который порождается текущим состоянием), его вершины будут помечены красным цветом.

Вершина, имеющая красный цвет не нуждается в обходе подграфа, достижимого из нее – отрицательный ответ в таком случае будет выдан мгновенно. Причем помимо своей разметки каждая вершина будет иметь информацию о «причине» – указатель на ту соседнюю вершину, без изменения разметки которой нет смысла менять свою разметку. Такая техника позволяет существенно повысить производительность новой процедуры проверки пройденных состояний.

2.4 Отсечение ветвей: статический и динамический подходы

Метод можно улучшить путем дополнительного отсечения ветвей в D-графе. Суть статического отсечения заключается в распознании фактической независимости некоторых вершин графа на основании анализа переходов модели до начала построения графа. Рассмотрим пример модели, переходы которой представлены в табл. 2.

Таблица 2. Переходы примера 4

переход	предусловие	постусловие
T1	control_flow=cf1 & a=0 & b=0	control_flow := cf2; a:=1
T2	control_flow=cf1 & c=0	control_flow := cf2; c := 0; b:=0
…		
Tn	control_flow=cf1 &…	control_flow := cf2;…

Можно сделать вывод, что вершина `control_flow=cf2` будет зависеть только от вершины `control_flow=cf1`, таким образом, остальные связи (`a=0`, `b=0`, `c=0`, …) в процессе построения графа можно будет игнорировать. Динамический подход заключается в отсечении всех связей, не влияющих на вершины, которые являются причинами невыполнения переходов (см. рис. 4, 5).

Такое отсечение выполняется при завершении обхода компоненты сильной связности и отсекает только избыточные поведения: действительно, для проверки факта вхождения нового подграфа в уже существующий достаточно убедиться в том, что новый не будет иметь дуг, выходящих за пределы существующего подграфа, при этом внутренние связи можно игнорировать.

Такой подход особенно эффективен при проверке моделей, содержащих много тупиков. Тупик в достаточной степени характеризуется причинами, по которым невозможно из него выйти, при этом, как правило, множество причинных вершин не велико.

3 Выводы

Усовершенствовано определение информационной зависимости в формальной модели. Уточнения позволяют сократить перебор при проверке свойств и более детально анализировать причинно-следственные связи значений атрибутов модели. Поведения (трассы) могут иметь одну изоморфную сущность – граф зависимостей.

На таком графе можно проверять факт повторности вхождения состояний. Алгоритм разметки графа позволяет существенно повысить эффективность этой процедуры. Предложены статическая и динамическая техники отсечения ветвей. На практике применение метода позволило получить такие многообещающие результаты, как существенное сокращение комбинаторного взрыва за счет устранения избыточного интерливинга.

Литература

[1] Ben-Ari. M. Principles of Spin // Springer Verlag. – 2008. – 216 P.
[2] Колчин А.В. Автоматический метод динамического построения абстракций состояний формальной модели // Кибернетика и системный анализ. – 2010. – № 4. – С. 70–90.
[3] Колчин А.В. Оптимизация проверки выполнимости переходов при верификации формальных моделей // Проблемы программирования. – 2012. – № 2–3. – С. 201–210.
[4] Колчин А.В., Четвертак Р.В. Интерактивная система для анализа поведения формальных моделей программных систем // Искусственный интеллект. –2012. – №4. – С. 330–341.