Модел. и анализ информ. систем. Т. **19**, № **2** (2012) 115–137

УДК 517.51+514.17

Полиномиальный алгоритм верификации цепей Маркова для подмножества логики PLTL

Лебедев П.В.[1]

Тверской государственный университет

e-mail: pavelvlebedev@gmail.com

получена 27 февраля 2012 года

Ключевые слова: цепи Маркова, проверка на моделях, темпоральные логики, PLTL, верификация динамических свойств

Описывается полиномиальный алгоритм верификации цепей Маркова, динамические свойства которых описываются формулами некоторого подмножества темпоральной логики PLTL (propositional temporal logic of linear time). Алгоритм позволяет найти вероятность истинности формулы на заданной цепи Маркова, а также множество траекторий, на которых истинна верифицируемая формула.

1. Введение

Настоящая работа посвящена вопросам верификации дискретных стохастических динамических систем. Математическими моделями таких систем являются конечные цепи Маркова.

Основной подход для верификации такого рода систем основан на так называемом методе проверки на моделях [8], который широко используется в настоящее время для верификации детерминированных и недетерминированных систем переходов.

При таком подходе спецификации представляются формулами некоторой пропозициональной темпоральной логики, а электронные схемы и протоколы моделируются системами переходов. Чтобы проверить, верна ли спецификация на системе переходов, система переходов проверяется на предмет того, является ли она моделью для предъявленной спецификации.

Этот подход к верификации был распространён также на вероятностные динамические системы. В частности, в работе [2] был предложен алгоритм для нахождения вероятности выполнения темпоральной формулы языка PLTL на траекториях конечной цепи Маркова. В работах [3, 4] было показано, как проблема верификации

[1]Работа поддержана грантом РФФИ №10-01-00532а

вероятностных многоагентных систем может быть сведена к верификации конечных цепей Маркова. Обзоры некоторых других подходов к верификации цепей Маркова для различных темпоральных логик приводятся в работах [5, 6, 7].

В настоящей статье предлагается эффективный алгоритм верификации для подмножества $PLTL^{-}$ логики PLTL, работающий за полиномиальное время от длины формулы, что позволяет верифицировать формулы этого подмножества более эффективно, чем в [2]. Программная реализация этого алгоритма описана в [9].

2. Основные понятия

В этом разделе мы приведём определения основных понятий, используемых в статье.

2.1. Цепь Маркова

Цепи Маркова – это математические модели вероятностных динамических систем, определяемых множествами состояний и вероятностными переходами между ними. Цепь Маркова – это четвёрка $M = (S, R, \{p_{vw} | (v, w) \in R\}, \{p_0(v) \mid v \in S\})$, где

1. S – конечное множество состояний;
2. $R \subseteq S \times S$ – множество переходов между состояниями;
3. $p(v, w)$ – вероятность перехода из состояния v в состояние w;
4. $p_0(v)$ – начальная вероятность состояния v.

При этом предполагается, что $p(v, w) = 0$ для всех $(v, w) \notin R$, для каждого $v \in S$ вероятность перехода из v (т.е. $\sum_{w \in S} p(v, w)$) равна 1. Для начального распределения $p_0 \sum_{v \in S} p_0(v) = 1$. Траектория в цепи Маркова M – это конечная или бесконечная последовательность состояний $\pi = s_0, s_1, \ldots$, такая что $(s_i, s_{i+1}) \in R$ для каждого $i \geq 0$.

Будем рассматривать цепи Маркова, в которых из каждого состояния есть хотя бы один переход, т. е. для всех $v \in S$ существует $w \in S$, для которого выполнено $(v, w) \in R$, и только бесконечные траектории цепи Маркова.

Дерево вычислений для состояния s_0 – это множество всех траекторий цепи Маркова, начинающихся с s_0.

2.2. Темпоральная логика PLTL

В работе [2] для формализации свойств поведения вероятностных программ (цепей Маркова) предложена пропозициональная темпоральная логика линейного времени PLTL.

2.2.1. Синтаксис

Формулы PLTL описывают свойства траекторий деревьев вычислений. Они строятся над конечным множеством пропозициональных переменных (элементарных утверждений) AP с помощью булевых и темпоральных операторов.

Основные темпоральные операторы:

1. Оператор сдвига по времени X (neXttime, «в следующий момент») требует, чтобы свойство соблюдалось во втором состоянии траектории.

2. Оператор условного ожидания U (Until, «до тех пор пока») чуть более изощрённый, поскольку в нём содержится комбинация двух свойств. Оператор выполняется, если на траектории имеется состояние, в котором выполняется второе свойство, и при этом каждое из предшествующих ему состояний обладает первым свойством.

2.2.2. Семантика

Семантика PLTL определяется на конечной цепи Маркова M, состояния которой помечены подмножеством AP. Пусть задана цепь Маркова $M = (S, R, \{p_{vw} | (v, w) \in R\}, \{p_0(v) \mid v \in S\})$ с функцией разметки L над пропозициональными переменными AP. $L : S \mapsto 2^{AP}$ – это отображение множества состояний во множество истинных в этом состоянии пропозициональных переменных.

Для траектории цепи Маркова $\pi = s_0, s_1, ...$ используем запись π^i для обозначения суффикса π, начинающегося с состояния s_i, т. е. $\pi^i = s_i, s_{i+1}, ...$ Запись $M, \pi \models f$ означает, что формула f истинна (выполняется) на траектории π на цепи Маркова M.

Пусть f_1 и f_2 – формулы PLTL, p – пропозициональная переменная из AP, тогда

1. $M, \pi \models p \Leftrightarrow p \in L(s_0)$
2. $M, \pi \models \neg f_1 \Leftrightarrow M, \pi \not\models f_1$
3. $M, \pi \models f_1 \vee f_2 \Leftrightarrow M, \pi \models f_1$ или $M, \pi \models f_2$
4. $M, \pi \models f_1 \wedge f_2 \Leftrightarrow M, \pi \models f_1$ и $M, \pi \models f_2$
5. $M, \pi \models X f_1 \Leftrightarrow M, \pi^1 \models f_1$
6. $M, \pi \models f_1 U f_2 \Leftrightarrow$ существует такое $k \geq 0$, что $M, \pi^k \models f_2$ и для каждого $0 \leq j < k$ верно соотношение $M, \pi^j \models f_1$

2.3. Вероятность формулы PLTL

Цепь Маркова M стандартным образом определяет вероятностное пространство (S^w, P_M) , где S^w – множество ее бесконечных траекторий M, а вероятностная мера P_M генерируется базисными открытыми цилиндрическими множествами вида

$cyl(s_0, \ldots, s_k) = \{\pi \in S^w | \pi = s_0, \ldots, s_k, \pi^{k+1}\}$. Вероятность (мера) такого цилиндра с учетом начального распределения равна $P_M(cyl(s_0, \ldots, s_k)) = p_0(s_0) \cdot p(s_0, s_1) \cdot p(s_1, s_2) \cdot \ldots \cdot p(s_{k-1}, s_k)$. Тогда вероятность $P_M(C)$ множества траекторий C, являющегося объединением непересекающихся цилиндров $C = \bigcup_{i=1}^{\infty} C_i$, определяется как сумма вероятностей этих цилиндров: $P_M(C) = \sum_{i=1}^{\infty} P_M(C_i)$. Из этого определения непосредственно следует, что для любого такого C выполнено $P_M(C) \leq 1$.

Обозначим через $L_M(f)$ множество траекторий цепи Маркова M, на которых истинна формула f логики PLTL. Это множество является *измеримым* подмножеством пространства (S^w, P_M). Его мера $P_M(L_M(f))$ и есть вероятность истинности формулы f на цепи Маркова M.

3. Алгоритм верификации Куркубетиса и Яннакакиса

Алгоритм, описанный в [2], шаг за шагом преобразует формулу f и цепь Маркова M, исключая каждый раз один темпоральный оператор (Until или neXt), но сохраняя при этом вероятность полученной новой формулы на новой построенной цепи Маркова. Определены два преобразования, C_U и C_X, позволяющие последовательно заменить темпоральные операторы U и X на пропозициональные переменные и перестроить соответствующим образом цепь Маркова. При этом применение каждого из этих преобразований может увеличить размер цепи в 2 раза.

Если в формуле φ содержится k темпоральных операторов, то $P_M(L_M(f))$ вычисляется следующим образом: k раз применяются преобразования C_U или C_X и строится последовательность $(f^0 = f, M^0), \ldots, (f^k, M^k)$, в которой заключительная формула f^k не содержит темпоральных операторов. Тогда $P_M(L_M(f)) = P_M\left(L_{M^k}\left(f^k\right)\right)$, при этом вероятность $P_M\left(L_{M^k}\left(f^k\right)\right)$ вычисляется просто как сумма начальных вероятностей всех состояний, удовлетворяющих f^k в M^k.

В [2] доказано,что алгоритм Куркубетиса и Яннакакиса для цепи Маркова M и формулы PLTL f вычисляет вероятность $P_M(L_M(f))$ за время, полиномиальное от размера графа переходов M и экспоненциальное от длины формулы f.

4. Полиномиальный алгоритм верификации для подмножества PLTL

В этом разделе мы представляем эффективный алгоритм верификации формул подмножества PLTL$^-$ логики PLTL на цепях Маркова.

4.1. Язык $PLTL^-$

Формулы языка $PLTL^-$ строятся над конечным множеством пропозициональных переменных AP с помощью булевых и темпоральных операторов $X(\theta)$ («neXt») и $\phi U^{\leq k}\psi$ (ограниченный «Until»), где аргументы ϕ и ψ не содержат больше операто-

ров $U^{\leq k}$ (нет вложенности операторов ограниченного «Until»). Более формально, синтаксис $PLTL^{-}$ определяется следующей грамматикой:

1. $\Phi ::= true|a|\Phi \wedge \Phi|\Phi \vee \Phi|\neg\Phi|X(\Phi)|\Psi U^{\leq k}\Psi$
2. $\Psi ::= true|a|\Psi \wedge \Psi|\Psi \vee \Psi|\neg\Psi|X(\Psi)$

Здесь a – пропозициональная переменная, $\wedge, \vee, \neg$ – стандартные булевы операторы, $X(f)$ – темпоральный оператор Next из логики PLTL, $f_1 U^{\leq k} f_2$ – темпоральный оператор «ограниченный» Until $U^{\leq k}$. Формула $f_1 U^{\leq k} f_2$ истинна на траектории, если на ее префиксе длины k имеется состояние, в котором выполняется второе свойство f_2, и при этом на каждом из предшествующих состояний выполняется первое свойство f_1. Более формально:

$M, \pi \models f_1 U^{\leq k} f_2 \Leftrightarrow$ существует такое $0 \leq i \leq k$, что $M, \pi^i \models f_2$ и для каждого $0 \leq j < i$ имеет место $M, \pi^j \models f_1$.

Следующая простая лемма показывает, что любую формулу из $PLTL^{-}$ можно привести к нормальному виду без операторов $U^{\leq k}$ и отрицаний перед сложными формулами, при этом длина новой формулы останется полиномиальной длины от размера первоначальной.

Лемма 1. *Всякую темпоральную формулу f из $PLTL^{-}$ можно эффективно преобразовать в эквивалентную формулу f', такую что:*
1) f' не содержит операторов вида $U^{\leq k}$;
2) в формуле f' отрицание может стоять только перед подформулами, не содержащими темпоральных операторов (позитивная форма).

Доказательство.

1) Каждую подформулу $U^{\leq k}$ можно заменить эквивалентной подформулой:
$\phi U^{\leq 0}\psi = \psi$,
$\phi U^{\leq k}\psi \equiv \psi \vee (\phi \wedge X(\phi U^{\leq k-1}\psi))$.
Это позволяет индукцией по k устранить все подформулы вида $U^{\leq k}$. Так как подформулы $U^{\leq k}$ не вложенные, то размер новой формулы будет $O(n \cdot K)$, где n – количество операторов $U^{\leq k}$ в старой формуле, K – максимальное k среди всех $U^{\leq k}$ в старой формуле.

Замечение. Полиномиальность размера формулы сохраняется при ограниченной вложенности операторов $U^{\leq k}$, т. е. если в формуле оператор $U^{\leq k}$ вложен не более m раз, то размер новой формулы будет $O(n \cdot K \cdot m)$.

2) Формулы с отрицаниями перед подформулами с темпоральными операторами можно преобразовать индукцией по вложенности операторов, используя эквивалентности:
$\neg(f_1 \wedge f_2) \equiv (\neg f_1 \vee \neg f_2)$,
$\neg(f_1 \vee f_2) \equiv (\neg f_1 \wedge \neg f_2)$,
$\neg(\neg f_1) \equiv f_1$,
$\neg X f_1 \equiv X(\neg f_1)$.
После применения конечного числа преобразований из п. 1 и 2 к формуле f получим требуемую формулу f'.

Мы рассматриваем следующую задачу верификации:

по заданным конечной цепи Маркова $M = (S, R, \{p_{vw} | (v,w) \in R,\ p_0(v) \mid v \in S\})$ *с функцией разметки* $L : S \mapsto 2^{AP}$ *и верифицируемой формуле* f *логики* $PLTL^-$ *найти вероятность* $P_M(L_M(f))$ *выполнения* f *на* M, *где* f *преобразована по лемме 1.*

4.2. Граф истинных префиксов (ГИП)

Работа предлагаемого алгоритма верификации состоит в построении по цепи Маркова и верифицируемой формуле некоторого специального графа истинных префиксов (ГИП), по которому можно просто получить все траектории, на которых истинна формула, и вычислить вероятность ее выполнения.

Определение 1. *Назовём графом истинных префиксов (ГИП)* $G(f, M)$ *для цепи Маркова* $M = (S,\ R,\ \{p(v,w) \mid (v,\ w) \in R,\ p_0(v) \mid v \in S\})$ *и формулы* f *логики* $PLTL^-$ *ациклический ориентированный помеченный граф* $G = (V, E,\ T[\,],\ V_{START},\ V_{END},\ P^G(v,w),\ P_0^G(v))$, *где:*
V *– множество вершин графа,* $E \subseteq V \times V$ *– множество его рёбер,* $T[\,]$ *– функция* $T :\ V \mapsto S$, *помечающая вершину из* V *состоянием из* S, *такая, что*

1. $\forall\ (v,u) \in E \quad (T[v], T[u]) \in R$;
2. $\forall\ (u,v) \in E, (u,w) \in E \rightarrow T[v] \neq T[w]$;
3. $\forall\ s \in S\ (\ p_0(s) > 0 \rightarrow \exists! v \in V_{START} : T[v] = s)$,
4. *Для всех путей* $v_1, \ldots, v_k$ *графа* G, *в которых* $v_1 \in V_{START}$, $v_k \in V_{END}$ *и* $v_2, \ldots, v_{k-1} \notin V_{START} \cup V_{END}$, *формула* f *истинна на всех траекториях из* $cyl(T[v_1], \ldots, T[v_k])$.

V_{START} *– подмножество вершин графа, помеченных как «начальные»,* V_{END} *– подмножество вершин графа, помеченных как «конечные», при этом* $\forall v \in V_{END}\ \nexists u :\ (v,u) \in E$, $P^G(v,\ w)$ *– метка ребра* (v,w), *означающая его вероятность и равная* $p(T[v],\ T[w])$, $P_0^G(v)$ *– метка вершины* v, *означающая её начальную вероятность и равная* $p_0(T[v])$.

Определение 2. *Траекторией в графе* $G = (V, E, T[\,], V_{START}, V_{END}, P^G(v,w), P_0^G(v))$ *назовём путь* $v_1, \ldots, v_k$ *в графе* G, *в котором* $v_1 \in V_{START}$, $v_k \in V_{END}$ *и* $v_2, \ldots, v_{k-1} \notin V_{START} \cup V_{END}$.

Определение 3. *Пути (траектории)* $v_1, \ldots, v_k$ *в ГИП* G_1 *для цепи Маркова* M *и* $w_1, \ldots, w_k$ *в ГИП* G_2 *для той же* M *назовём соответствующими, если* $T_1[v_1] = T_2[w_1], \ldots, T_1[v_k] = T_2[w_k]$.

Определение 4. *Множеством* $M(G)$ *траекторий цепи Маркова* M, *порождённых ГИП* G, *назовём множество таких траекторий, что каждая траектория* $s_1, s_2, \ldots, s_k, \ldots \in M(G)$ *тогда и только тогда, когда существует такое* k *и существует (порождающая) траектория* $v_1, \ldots, v_k$ *в* G, *что* $T[v_1] = s_1, \ldots, T[v_k] = s_k$.

Определение 5. *Объединением* $G = G_1 \cup G_2$ *двух графов истинных префиксов* G_1 *и* G_2 *назовём ГИП* G *такой, что в* G *есть траектория* $w_1, \ldots, w_k$ *тогда и только тогда, когда существует* $v_1, \ldots, v_k$ *– траектория в* G_1 *или* G_2 *(пусть в* G_ξ *) такая, что* $T[w_1] = T_\xi[v_1], \ldots, T[w_k] = T_\xi[v_k]$, *и не существует траектории* $u_1, \ldots, u_l$ *в* $G_{3-\xi}$ *такой, что* $l < k$ *и* $T_\xi[v_1] = T_{3-\xi}[u_1], \ldots, T_\xi[v_l] = T_{3-\xi}[u_l]$.

Определение 6. *Пересечением* $G = G_1 \cap G_2$ *двух ГИП* G_1 *и* G_2 *назовём ГИП* G *такой, что в* G *есть траектория* $w_1, \ldots, w_k$ *тогда и только тогда, когда существует* $v_1, \ldots, v_k$ *– траектория в* G_1 *или* G_2 *(например, в* $G_\xi, \xi = 1, 2$*) такая, что* $T[w_1] = T_\xi[v_1], \ldots, T[w_k] = T_\xi[v_k]$, *и существует траектория* $t_1, \ldots, t_l$ *в* $G_{3-\xi}$ *такая, что* $l \leq k$ *и* $T_\xi[u_1] = T_{3-\xi}[t_1], \ldots, T_\xi[u_l] = T_{3-\xi}[t_l]$.

Существование объединения и пересечения ГИП доказывается приведенными ниже алгоритмами Join и Intersection, которые строят объединение и пересечение соответственно по двум заданным ГИП G_1 и G_2.

Теорема 1. $M(G_1 \cup G_2) = M(G_1) \cup M(G_2)$, *где* G_1, G_2 *– графы истинных префиксов.*

Доказательство. Пусть $G \equiv G_1 \cup G_2$.

1) Пусть траектория $X = s_1, s_2, \ldots \in M(G_1 \cup G_2)$. Докажем, что $X \in M(G_1)$ или $X \in M(G_2)$.

$X = s_1, s_2, \ldots \in M(G) \Rightarrow$ существует траектория $v_1, \ldots, v_k$ в G, которая её порождает (т. е. $T[v_1] = s_1, \ldots, T[v_k] = s_k$). Так как $v_1, \ldots, v_k$ есть в объединении, то по определению в G_1 или G_2 есть соответствующая траектория $u_1, \ldots, u_k$, но тогда она порождает множество траекторий вида $T[u_1], \ldots, T[u_k], \ldots$, в котором содержится X, т. е. $X \in G_1$ или $X \in G_2$.

2) Пусть траектория $X = s_1, s_2, \ldots \in M(G_1)$(или $M(G_2)$). Докажем, что $X \in M(G)$.

$X \in M(G_1) \Rightarrow$ существует траектория $v_1, \ldots, v_k$ в G_1, которая её порождает.

а) Если не существует траектории $w_1, \ldots, w_{l<k}$ в G_2, соответствующей $v_1, \ldots, v_l$, то по определению объединения в G есть траектория $u_1, \ldots, u_k$, соответствующая $v_1, \ldots, v_k$, поэтому $u_1, \ldots, u_k$ порождает множество траекторий вида $T[u_1], \ldots, T[u_k], \ldots$, в котором содержится X, т. е. $X \in M(G)$.

б) Если в G_2 существует траектория $w_1, \ldots, w_l$ для $l < k$, соответствующая $v_1, \ldots, v_l$, то в G_1 нет более короткой траектории $v'_1, \ldots v'_r$, $(r < l)$, соответствующей префиксу $w_1, \ldots, w_r$. Поэтому в G есть траектория $t_1, \ldots, t_l$, соответствующая $w_1, \ldots, w_l$. Она порождает множество траекторий вида $s_1, \ldots, s_{l<k}, \ldots$, в которое входит траектория X, т. е. $X \in M(G)$.

Теорема 2. $M(G_1 \cap G_2) = M(G_1) \cap M(G_2)$, *где* G_1, G_2*– графы истинных префиксов.*

Доказательство.

Пусть $G \equiv G_1 \cap G_2$.

1) Пусть траектория $X = s_1, s_2, \ldots \in M(G)$. Докажем, что $X \in M(G_1)$ и $X \in M(G_2)$.

$X = s_1, s_2, \ldots \in M(G) \Rightarrow$ существует траектория $v_1, \ldots, v_k$ в G, которая её порождает (т. е. $T[v_1] = s_1, \ldots, T[v_k] = s_k$). Так как $v_1, \ldots, v_k$ есть в пересечении, то по определению в G_1 или G_2 (пусть в G_1) есть соответствующая траектория $u_1, \ldots, u_k$, а в G_2 есть траектория $w_1, \ldots, w_{l \leq k}$, соответствующая $u_1, \ldots, u_l$. Тогда первая траектория порождает множество траекторий вида $s_1, \ldots, s_k, \ldots$, а вторая - множество траекторий вида $s_1, \ldots, s_{l \leq k}, \ldots$. И в том, и другом множестве содержится X, т. е. $X \in M(G_1)$ и $X \in M(G_2)$.

2) Пусть траектория $X = s_1, s_2, \ldots \in M(G_1)$ и $X \in M(G_2)$. Докажем, что $X \in M(G)$.

$X \in M(G_1) \Rightarrow$ существует траектория $v_1, \ldots, v_k$ в G_1, которая её порождает.

$X \in M(G_2) \Rightarrow$ существует траектория $u_1, \ldots, u_l$ в G_2, которая её порождает.

Пусть $l \leq k$ для определённости. Так как $T_1[v_1] = s_1, \ldots, T_1[v_k] = s_k$ и $T_2[u_1] = s_1, \ldots, T_2[u_l] = s_l$, то $T_1[v_1] = T_2[u_1], \ldots, T_1[v_l] = T_2[u_l]$. Таким образом, в G_1 существует траектория $v_1, \ldots, v_k$, в G_2 существует траектория $u_1, \ldots, u_l$ такая, что $l \leq k$ и $u_1, \ldots, u_l$ соответствует $v_1, \ldots, v_l$, поэтому по определению пересечения в G существует траектория $w_1, \ldots, w_k$, соответствующая $v_1, \ldots, v_k$, которая порождает множество траекторий вида $s_1, \ldots, s_k, \ldots$, в которое входит X, т. е. $X \in M(G)$.

4.3. Алгоритм VerificationPLTLMinus

Алгоритм VerificationPLTLMinus – главный алгоритм, который по цепи Маркова M и верифицируемой формуле f строит ГИП $G(M, f)$ и вычисляет вероятность истинности формулы f на M. Этот алгоритм создаёт стартовый граф, заполняет его начальными состояниями, запускает рекурсивный алгоритм VerificationStep, который по f строит неразмеченный ГИП, сокращает его с помощью процедуры Reduse, затем заполняет P^G и P_0^G графа G и вычисляет вероятность формулы алгоритмом ProbCalculate.

VerificationPLTLMinus *(f, M)*

Вход:

1. *Верифицируемая формула f, в которой нет отрицаний над подформулами с темпоральными операторами и операторов $U^{\leq k}$.*

2. *Цепь Маркова $M = (S, R, \{p_{vw} | (v, w) \in R,\ p_0(v) \mid v \in S\})$.*

Выход:

1. *Граф истинных префиксов $G = (V, E, T[\,], V_{START}, V_{END}, P^G(v, w), P_0^G(v))$, задающий все траектории цепи Маркова M, на которых истинна заданная формула f.*

2. *Вероятность p истинности формулы f.*

Создать граф $G = (V = \emptyset, E = \emptyset, T[\,], V_{START} = \emptyset, V_{END} = \emptyset, \forall (v, w) P^G(v, w) = 0, \forall v P_0^G(v) = 0)$;

Создать множество вершин D;

// Создаём вершину в G для каждого начального состояния M

foreach $s : p_0(s) > 0$ ***do*** {

Добавить вершину v в V;

$T[v] = s$; $V_{START} := V_{START} \cup \{v\}$; $V_{END} := V_{END} \cup \{v\}$;

}

// Построение графа G в рекурсивной процедуре.

$(G,\ D)$:= VerificationStep(f, M, G) ;
Reduce(G, D);
foreach $v \in D : \nexists (v, u) \in E$ ***do***
$V := V \setminus \{v\}$;
// *Получение вероятностей для графа* G *по* M.
foreach $v \in V$ ***do***
$P_0^G(v) := p_0(T[v])$;
foreach $(u, v) \in E$ ***do***
$P^G(u, v) := p(T[u], T[v])$;
p := ProbCalculate(G) ;
return (G, p) ;

4.4. Алгоритм VerificationStep

Алгоритм VerificationStep рекурсивно обрабатывает каждый оператор формулы.

VerificationStep *(f, M, G)*
Вход:
1. *Верифицируемая формула* f, *в которой нет отрицаний над подформулами с темпоральными операторами и операторов* $U^{\leq k}$.
2. *Цепь Маркова* M.
3. *ГИП* $G = \big(V,\ E,\ T[\,], V_{START}, V_{END},\ P^G(v,\ w),\ P_0^G(v)\big)$.

Выход:
1. *Изменённый граф* G.
2. *Множество вершин* D, *служащие для удаления лишних путей в графе* G.

В зависимости от формулы f могут возникнуть четыре случая:
Случай 1: $f = X\ \varphi$**.**
Создать вспомогательный массив $F : S \mapsto V$;
Создать множество вершин W;
$W := \emptyset$;
// *Находим все состояния, достижимые за один шаг из всех «конечных», и*
// *добавляем соответствующую ему вершину в граф* G.
foreach $s \in S : (T[u],\ s) \in R$, где $u \in V_{END}$ ***do*** {
Добавить вершину v в V;
$W := W \cup \{v\}$; $T[v] := s$; $F[s] := v$;
}
// *Добавляем рёбра*
foreach $v \in V_{END}$ ***do***
foreach $(T[v], s) \in R$ ***do***
$E := E \cup (v,\ F[s])$;
$V_{END} := W$;
(G, D) := VerificationStep(φ, M, G) ;
return $(G,\ D)$;
Случай 2: f **– булева формула.**
// *Находим среди конечных вершин графа* G *те, соответствующие состояния которых не удовлетворяют булевой формуле.*
$D := \emptyset$;

foreach $v \in V_{END}$ ***do*** {
if $(T[v] \not\models f)$ {
$D := D \cup \{v\}$; $V_{END} := V_{END} \setminus \{v\}$;
}
}
return (G, D);
Случай 3: $f = \varphi \vee \psi$.
Создать два ГИП G_1, G_2:
$G_1 = (V_1 = \emptyset,\ E_1 = \emptyset,\ T_1[\,],\ V^1_{START} = \emptyset,\ V^1_{END} = \emptyset, \forall (v,w)\, P^{G1}(v,w) = 0, \forall\, v P_0^{G1}(v) = 0)$;
$G_2 = (V_2 = \emptyset,\ E_2 = \emptyset,\ T_2[\,],\ V^2_{START} = \emptyset,\ V^2_{END} = \emptyset, \forall (v,w)\, P^{G2}(v,w) = 0, \forall\, v P_0^{G2}(v) = 0)$;
foreach $(v \in V_{END})$ ***do*** {
Создать v_1 в V_1;
$T[v_1] := T[v]$; $V^1_{START} := V^1_{START} \cup \{v_1\}$; $V^1_{END} := V^1_{END} \cup \{v_1\}$;
Создать v_2 в V_2;
$T[v_2] := T[v]$; $V^2_{START} := V^2_{START} \cup \{v_2\}$; $V^2_{END} := V^2_{END} \cup \{v_2\}$;
}
$(G_1, D_1) = VerificationStep(\varphi, M, G_1)$;
$(G_2, D_2) = VerificationStep(\psi, M, G_2)$;
$Reduce(G_1, D_1)$; $Reduce(G_2, D_2)$;
// Найти объединение двух ГИП.
$(G', D') := Join(G_1, G_2, D_1, D_2)$;
Присоединить граф G' к графу G, заменив каждую вершину $v \in V_{END}$ на $v' \in V'_{START}$ такую, что $T[v] = T[v']$ (по построению, такие пары вершин всегда есть);
$D := D'$;
$return\ (G, D)$;
Случай 4: $f = \varphi \wedge \psi$.
Создать два ГИП G_1, G_2:
$G_1 = (V_1 = \emptyset,\ E_1 = \emptyset,\ T_1[\,],\ V^1_{START} = \emptyset,\ V^1_{END} = \emptyset, \forall (v,w)\, P^{G1}(v,w) = 0, \forall\, v P_0^{G1}(v) = 0)$;
$G_2 = (V_2 = \emptyset,\ E_2 = \emptyset,\ T_2[\,],\ V^2_{START} = \emptyset,\ V^2_{END} = \emptyset, \forall (v,w)\, P^{G2}(v,w) = 0, \forall\, v P_0^{G2}(v) = 0)$;
foreach $(v \in V_{END})$ ***do*** {
Создать v_1 в V_1 ;
$T[v_1] := T[v]$; $V^1_{START} := V^1_{START} \cup \{v_1\}$; $V^1_{END} := V^1_{END} \cup \{v_1\}$;
Создать v_2 в V_2;
$T[v_2] := T[v]$; $V^2_{START} := V^2_{START} \cup \{v_2\}$; $V^2_{END} := V^2_{END} \cup \{v_2\}$;
}
$(G_1, D_1) = VerificationStep(\varphi, M, G_1)$;
$(G_2, D_2) = VerificationStep(\psi, M, G_2)$;
$Reduce(G_1, D_1)$; $Reduce(G_2, D_2)$;
// Найти пересечение двух ГИП.
$(G', D') :=$ Intersection(G_1, G_2, D_1, D_2);
Присоединить граф G' к графу G, заменив каждую вершину $v \in V_{END}$ на $v' \in V'_{START}$ такую, что $T[v] = T[v']$;
$D := D'$;
$return\ (G, D)$;

4.5. Алгоритм Reduce

Данный алгоритм по графу G и множеству вершин D графа G удаляет те вершины, из которых нет пути в конечную вершину.

Reduce(G, D)
Вход:
1. $G = \big(V,\ E,\ T[\,], V_{START}, V_{END},\ P^G(v,\ w),\ P_0^G(v)\big)$.
2. $D \subseteq V$ – *множество удаляемых вершин.*

while $(\exists\ v \in\ D \setminus V_{START})$
{
Пусть $v\ \in\ D \setminus V_{START}$;
// *Находим множество вершин, из которых есть дуга*
// *только в вершину* v.
$W\ :=\ \{u \mid (u,v) \in E$ и $(\nexists\ w \neq v) \mid (u,w) \in\ E\}$;
// *Удаляем все рёбра с вершиной* v.
$E\ :=\ E \setminus\ \{(v',v) :\ (v',v) \in\ E\}$;
$V\ :=\ V \setminus\ \{v\}$; $D :=\ D \setminus\ \{v\}$; $D :=\ D\ \cup\ W$;
}

4.6. Алгоритм Join

Алгоритм Join строит объединение ГИП G_1 и G_2.

Join$(G_1,\ G_2, D_1, D_2)$
Вход:
1. *ГИП* $G_1 = \big(V_1,\ E_1,\ T_1[\,],\ V^1_{START},\ V^1_{END},\ P^{G1}(v,w),\ P_0^{G1}(v)\big)$.
2. *ГИП* $G_2 = \big(V_2,\ E_2,\ T_2[\,],\ V^2_{START},\ V^2_{END},\ P^{G2}(v,w),\ P_0^{G2}(v)\big)$.
3. D_1- *подмножество вершин* V_1.
4. D_2- *подмножество вершин* V_2.

Выход:
1. *ГИП* $G = \big(V,\ E,\ T\ [\,],\ V_{START},\ V_{END},\ P^G(v,w),\ P_0^G(v)\big)$
2. D – *подмножество вершин* V.

Создать множества $S_1,\ S_2,\ S_{ALL}$;
$D := \emptyset$;
foreach $(v_1,\ v_2) :\ v_1 \in\ V^1_{START},\ v_2 \in\ V^2_{START},\ T_1\ [v_1] = T_2\ [v_2]$ ***do*** {
Добавить вершину v в V; $V_{START} :=\ V_{START} \cup\ \{v\}$;
if $(v_1 \in\ D_1 \wedge\ v_2 \in\ D_2)$ { $D :=\ D\ \cup\ \{v\}$; ***continue*** };
else if $(v_1 \in\ D_1)$ { $S_2 :=\ S_2 \cup\ \{(v_2,v)\}$; ***continue*** };
else if $(v_2 \in\ D_2)$ { $S_1 :=\ S_1 \cup\ \{(v_1,v)\}$; ***continue*** };
if $(v_1 \in\ V^1_{END} \vee\ v_2 \in\ V^2_{END})\ V_{END} :=\ V_{END} \cup\ \{v\}$;
else $S_{ALL} :=\ S_{ALL} \cup\ \{(v_1,\ v_2, v)\}$;
}
while $(S_{ALL} \neq\ \emptyset\ )$ {
$(v_1,\ v_2, v)$ – некоторый элемент из S_{ALL};
foreach $(u_1,u_2) :\ (v_1,u_1) \in\ E_1,\ (v_2,u_2) \in\ E_2,\ T_1\ [u_1] = T_2\ [u_2]$ ***do*** {

Добавить вершину u в V;
$T[u] := T_1[u_1]$; $E := E \cup \{(v,u)\}$;
if $(u_1 \notin V^1_{END} \wedge u_2 \notin V^2_{END})$ $S_{ALL} := S_{ALL} \cup \{(u_1,u_2,u)\}$;
else $V_{END} := V_{END} \cup \{u\}$;
}
foreach $u_1 : u_1 \in V_1, (v_1,u_1) \in E_1 \wedge \nexists u_2 : (v_2,u_2) \in E_2 \wedge T_2[u_2] = T_1[u_1]$ {
Добавить вершину u в V ;
$T[u] := T_1[u_1]$; $E := E \cup \{(v,u)\}$; $S_1 := S_1 \cup \{(u_1,u)\}$;
}
foreach $u_2 : u_2 \in V_2, (v_2,u_2) \in E_2, \nexists u_1 : (v_1,u_1) \in E_1 \wedge T_1[u_1] = T_2[u_2]$ {
Добавить вершину u в V;
$T[u] := T_2[u_2]$; $E := E \cup \{(v,u)\}$; $S_2 := S_2 \cup \{(u_2,u)\}$;
}
$S_{ALL} := S_{ALL} \setminus \{(v_1,v_2,v)\}$;
}
while$(S_1 \neq \emptyset)$ {
(v_1, v) – некоторый элемент из S_1;
if $(v_1 \in V^1_{END})$ $V_{END} := V_{END} \cup \{v\}$;
foreach $(u_1 : (v_1, u_1) \in E_1)$ {
Добавить вершину u в V ;
$T[u] := T_1[u_1]$; $E := E \cup \{(v,u)\}$; $S_1 := S_1 \cup \{(u_1,u)\}$;
}
$S_1 := S_1 \setminus \{(v_1,v)\}$;
}
while$(S_2 \neq \emptyset)$ {
(v_2, v) – некоторый элемент из S_2;
if $(v_2 \in V^2_{END})$ $V_{END} := V_{END} \cup \{v\}$;
foreach $(u_2 : (v_2, u_2) \in E_2)$ ***do*** {
Добавить вершину u в V;
$T[u] := T_2[u_2]$; $E := E \cup \{(v,u)\}$; $S_2 := S_2 \cup \{(u_2,u)\}$;
}
$S_2 := S_2 \setminus \{(v_2, v)\}$;
}

Теорема 3. *Алгоритм Join строит* $G_1 \cup G_2$.

Доказательство.

Рассмотрим алгоритм $\mathrm{Join}(G_1, G_2)$. В начале массив S_{ALL} заполняется тройками (v_1, v_2, v) , где $T_1[v_1] = T_2[v_2] = T[v]$. После этого в первом цикле, пока не пуст S_{ALL}, для тройки (v_1, v_2, v) выбирается пара вершин (u_1, u_2), соединённых ребрами с v_1 и v_2 соответственно, удовлетворяющих условию $T_1[u_1] = T_2[u_2]$. Если $u_1 \notin V^1_{END}$ и $u_2 \notin V^2_{END}$, то создаётся вершина u в V такая, что $T[u] = T_1[u_1]$ и тройка (u_1, u_2, u) записывается в S_{ALL}. Таким образом, каждый элемент (v_1, v_2, v) в S_{ALL} означает, что в G_1 есть путь $u^1_1, \ldots, u^k_1, v_1$, где $u^1_1 \in V^1_{START}$, а в G_2 есть путь $u^1_2, \ldots, u^k_2, v_2$, где $u^1_2 \in V^2_{START}$ такие, что $T_1[u^1_1] = T_2[u^1_2], \ldots, T_1[u^k_1] = T_2[u^k_2], T_1[v_1] = T_2[v_2]$ и в G построен путь $u_1, \ldots, u_k, v$, где $u_1 \in V_{START}$ такой, что $T[u_1] = T_1[u^1_1], \ldots, T[u_k] = T_1[u^k_1], T[v] = T_1[u_1]$. Если в процессе заполнения S_{ALL} выполнилось $u_1 \notin V^1_{END}$ и $u_2 \notin V^2_{END}$, то траектория в G заканчивается, т. е.

соответствующая траектория есть в G_1 или G_2 и нет соответствующей траектории меньшей длины, что удовлетворяет определению объединения.

В массив S_1 попадают пары $(v_1,\ v)$, означающие, что для пути $u_1, \ldots, u_l, v_1$ в G_1 построен соответствующий путь $w_1, \ldots, w_l, v$ в G. Если $v_1 \in\ V^1_{END}$, то и $v\ \in\ V_{END}$, поэтому для некоторой траектории из G_1 есть соответствующая траектория в G, при этом в G_2 нет траектории $t_1,\ \ldots, t_{r<l+1}$, соответствующей $u_1,\ \ldots,\ u_r$, иначе бы пара $(v_1,\ v)$ не попала в S_1, поэтому такая траектория в G удовлетворяет определению объединения.

Утверждения о парах массива S_2 аналогичны утверждениям о парах S_1.

4.7. Алгоритм Intersection

Алгоритм Intersection строит пересечение ГИП G_1 и G_2.

Intersection(G_1, G_2, D_1, D_2)

Вход:

1. *ГИП* $G_1 = \big(V_1,\ E_1,\ T_1[\,],\ V^1_{START},\ V^1_{END},\ P^{G1}(v,w),\ P^{G1}_0(v)\big)$.
2. *ГИП* $G_2 = \big(V_2,\ E_2,\ T_2[\,],\ V^2_{START},\ V^2_{END},\ P^{G2}(v,w),\ P^{G2}_0(v)\big)$.
3. D_1– *подмножество вершин* V_1.
4. D_2– *подмножество вершин* V_2.

Выход:

1. *ГИП* $G = \big(V,\ E,\ T\ [\,],\ V_{START},\ V_{END},\ P^G(v,w),\ P^G_0(v)\big)$
2. D – *подмножество вершин* V.

Создать множество $S_1, S_2,\ S_{ALL}$;
$D := \emptyset$;
foreach $(v_1,\ v_2):\ v_1 \in\ V^1_{START},\ v_2 \in\ V^2_{START},\ T_1\ [v_1] = T_2\ [v_2]$ ***do*** {
Добавить вершину v в V;
$V_{START} :=\ V_{START} \cup\ \{v\}$;
if $(v_1 \in\ D_1 \vee\ v_2 \in\ D_2)$ { $D\ :=\ D\ \cup\ \{v\}$; ***continue*** ; }
if $(v_1 \in\ V^1_{START})\ S_2 :=\ S_2 \cup\ \{(v_2, v)\}$;
else if $(v_1 \in\ V^1_{START})\ S_1 :=\ S_1 \cup\ \{(v_1, v)\}$;
else $S_{ALL} :=\ S_{ALL} \cup\ \{(v_1,\ v_2, v)\}$;
}
while $(S_{ALL} \neq\ \emptyset\ )$ ***do*** {
$(v_1,\ v_2, v)$ – некоторый элемент из S_{ALL};
foreach $(u_1, u_2):\ (v_1, u_1) \in\ E_1,\ (v_2, u_2) \in\ E_2,\ T_1\ [u_1] = T_2\ [u_2]$ {
Добавить вершину u в V ;
$T[u] := T[u_1]$; $E := E \cup\ \{(v, u)\}$;
if$(u_1 \in\ V^1_{END}\ )\ S_2 :=\ S_2 \cup\ \{(u_2, u)\}$;
else if$(u_2 \in\ V^2_{END}\ )\ S_1 :=\ S_1 \cup\ \{(u_1\ , u)\}$;
else $S_{ALL} :=\ S_{ALL} \cup\ \{(u_1, u_2, u)\}$;
}
$S_{ALL} :=\ S_{ALL} \setminus\ \{(v_1, v_2, v)\}$;
}
while$(S_1 \neq\ \emptyset)$ ***do*** {
$(v_1,\ v)$ – некоторый элемент из S_1;

```
        if (v_1 ∈ V^1_END ) V_END := V_END ∪ {v};
        foreach ( u_1 : (v_1, u_1) ∈ E_1)        {
                Добавить вершину u в V ;
                T[u] := T_1 [u_1]; E := E ∪ {(v,u)}; S_1 := S_1 ∪ {(u_1,u)};
        }
        S_1 := S_1 \ {(v_1,v)};
    }
    while(S_2 ≠ ∅) do {
        (v_2, v) − некоторый элемент из S_2;
        if (v_2 ∈ V^2_END ) V_END := V_END ∪ {v};
        foreach( u_2 : (v_2, u_2) ∈ E_2) do        {
                Добавить вершину u в V ;
                T[u] := T_2 [u_2]; E := E ∪ {(v,u)}; S_2 := S_2 ∪ {(u_2,u)};
        }
        S_2 := S_2 \ {(v_2 ,v)};
    }
```

Теорема 4. *Алгоритм Intersection*(G_1, G_2) *строит* $G_1 \cap G_2$.

Доказательство.

Рассмотрим алгоритм Intersection(G_1, G_2). Как и в алгоритме Join, каждый элемент (v_1, v_2, v) в S_{ALL} означает, что есть соответствующие пути $u_1^1, \ldots, u_1^k$, v_1 в G_1, где $u_1^1 \in V^1_{START}$, $u_2^1, \ldots, u_2^k, v_2$ в G_2, где $u_2^1 \in V^2_{START}$, и построен соответствующий им путь в G. Когда выполняется условие $v_1 \in V^1_{END}$, это означает, что найдена траектория в G_1, и пара (v_2, v) помещается в S_2. Пара (v_2, v) из S_2 означает, что есть путь $w_1, \ldots, w_k, v_2$ в G_2, для него построен соответствующий путь $t_1, \ldots, t_k, v$ в G. Если $v_2 \in V^2_{END}$, то и $v \in V_{END}$, поэтому для траектории $w_1, \ldots, w_k, v_2$ в G_2 построена соответствующая траектория $t_1, \ldots, t_k, v$ в G. Так как перед занесением пары (v_2, v) выполнялось условие $v_1 \in V^1_{END}$, то в G_1 существует траектория $b_1, \ldots, b_{r \le k+1}$, соответствующая траектории $w_1, \ldots, w_r$ в G_2, поэтому траектория $t_1, \ldots, t_k, v$ в G удовлетворяет определению пересечения.

Корректность S_1 устанавливается аналогично.

4.8. Вычисление вероятности

Алгоритм ProbCalculate по заданному ГИП G вычисляет вероятность, которая является вероятностью истинности формулы на цепи Маркова.

ProbCalculate(G)

Вход:

ГИП $G = \left(V,\ E,\ T\ [\,],\ V_{START},\ V_{END},\ P^G(v,w),\ P_0^G(v)\right)$.

Выход:

Prob – вероятность исходной формулы.

$Prob := \sum_{v \in V_{START}} P_0^G(v) \cdot P(v)$, где:

$$P(v) = \begin{cases} 1, & \text{если } v \in V_{END} \\ \sum_{(v,w) \in E} P^G(v,w) \cdot P(w), & \text{если } v \notin V_{END} \end{cases};$$

return *Prob*;

Лемма 2. *Для каждой вершины* $v \in V_{START}$ *алгоритм ProbCalculate(G) вычисляет значение* $P(v)$*, которое равно вероятности траекторий цепи Маркова, порождённых* G *и начинающихся из единственного начального состояния* $T[v]$.

Доказательство.

Каждая траектория цепи Маркова, начинающаяся в $T[v]$, порождена некоторой траекторией в G, начинающейся в v, поэтому из каждой траектории v, v_1^i, $\ldots$, $v_{k_i}^i$ порождается множество всех траекторий, начинающихся с $T[v]$, $T[v_1^i]$, $\ldots$, $T\left[v_{k_i}^i\right]$. Вероятность каждого из таких множеств равна $p(T[v], T[v_1^i]) \cdot \ldots \cdot p\left(T\left[v_{k_i-1}^i\right], T\left[v_{k_i}^i\right]\right)$, поэтому вероятность траекторий, порождённых G и начинающихся в $T[v]$, равна $prob = \sum_i p(T[v], T[v_1^i]) \cdot \ldots \cdot p\left(T\left[v_{k_i-1}^i\right], T\left[v_{k_i}^i\right]\right)$. По определению G получаем: $prob = \sum_i P^G(v, v_1^i) \cdot \ldots \cdot P^G\left(v_{k_i-1}^i, v_{k_i}^i\right)$. Если записать это выражение рекурсивно, то получим $P(v) = \begin{cases} 1, & \text{если } v \in V_{END} \\ \sum_{(v,w) \in E} P^G(v,w) \cdot P(w), & \text{если } v \notin V_{END} \end{cases}$

Теорема 5. *Вероятность, вычисляемая алгоритмом ProbCalculate(G), равна вероятности траекторий* $M(G)$ *на цепи Маркова* M.

Доказательство.

Обозначим вероятность траекторий, порождённых G и начинающихся из единственного начального состояния s как q_s. Тогда вероятность $M(G)$ на цепи Маркова вычисляется как $\mathrm{p} = \sum_{s \in S} p_0(s) \cdot q_s$. Вероятность в алгоритме ProbCalculate вычисляется как $Prob := \sum_{v \in V_{START}} P_0^G(v) \cdot P(v)$. Так как между вершинами из V_{START} и начальными состояниями цепи Маркова взаимооднозначное соответствие и из леммы 2 $q_s = P(v)$, где $T[v] = s$, то получаем $\mathrm{p} = \mathrm{Prob}$.

4.9. Корректность и время работы

Теорема 6. *Алгоритм VerificationPLTLMinus(*f, M*) строит ГИП* $G = \left(V, E, T, V_{START}, V_{END}, P^G, P_0^G\right)$*, который порождает множество всех траекторий цепи Маркова* M*, на которых истинна* f.

Доказательство.

Рассмотрим некоторую подформулу f' формулы f и подграф $G_0 = \left(V_0, E_0, T_0, V_{START}^0, V_{END}^0, P^{G_0}, P_0^{G_0}\right)$ графа G, для которых в процессе выполнения алгоритма верификации должен выполниться некоторый шаг алгоритма VerificationStep. На каждом шаге VerificationStep граф G лишь «достраивается», т. е. используются вершины из множества V_{END}, к которым добавляются новые рёбра и вершины. Поэтому рассмотрим подграф G' графа G, который содержит вершины из V_{END} в качестве «начальных», а также всё, что будет построено в процессе обработки f', т. е. $G' = (V', E', T', V'_{START}, V'_{END}, P'^{G'}, P_0'^{G'})$, где

1. $V' = V \setminus V_0 \cup V_{END}^0$.
2. $E' = E \setminus E_0$.

3. $T'[v] = \begin{cases} T_0[v] & \text{если } v \in V_0 \\ T[v] & \text{если } v \notin V_0 \end{cases}$.

4. $V'_{START} = V^0_{END}$.

5. $V'_{END} = V_{END}$.

6. $P'^G(v,w) = p(T'[v], T'[w])$.

7. $P'^G_0(v) = p_0(T'[v])$.

Докажем утверждение теоремы для ГИП G' и формулы f'. Тогда при $f' = f$ получим утверждение теоремы для графа $G' = G$.

1) Докажем, что если $\tau = s_1, s_2, \ldots \models f'$**, то** $\tau \in M(G')$**.**

Индукция по сложности формулы f'.

Базис.f' – булева формула, тогда $s_1 \models f'$.

В начале алгоритма VerificationPLTLMinus в G' включаются вершины, соответствующие начальным ($p_0(v) > 0$) состояниям цепи Маркова. Далее выполняется процедура VerificationStep, где выполняется *Случай 4*. Во множество D записываются такие v, что $T[v] \not\models f'$. Так как $\exists u \in V_{END} : T[u] = s_1$ по начальному заполнению G' и $s_1 \models f'$, то $u \notin D$. Поэтому в процедуре Reduce вершина u не будет удалена из G'. Так как $u \in V_{END}$ и $u \in V_{START}$, то u является траекторией в графе G' , которая порождает множество траекторий вида $s_1, \ldots$, в которое входит τ, т. е. $\tau \in M(G')$.

Индукционный шаг.

Случай 1. f'=$X\ \varphi$.

Обозначим подграф графа G' как $G'' = G(\varphi, M)$, который получается при обработке алгоритмом VerificationStep формулы φ.

$\tau' := f' \Rightarrow s_2, s_3, \ldots \Rightarrow \varphi$. По индукционному предположению тогда $s_2, s_3, \ldots \in M(G'')$, т. е. в G''существует траектория $v_2, \ldots, v_k$, порождающая $s_2, s_3, \ldots$. Так как $(s_1, s_2) \in S$, то на шаге $f' = X\ \varphi$ алгоритма VerificationStep добавилось ребро $(T[s_1], T[s_2])$, поэтому в G'' есть траектория $T[s_1], v_2, \ldots, v_k$, которая порождает τ, поэтому $\tau \in M(G'')$.

Случай 2. $f' = \varphi \vee \psi$**.**

Пусть траектория $\tau \models \varphi$ или $\tau \models \psi$. Тогда по индукционному предположению $\tau \in M(G_1)$ или $\tau \in M(G_2)$. Алгоритм Join по теоремам 1 и 3 строит граф G' такой, что $M(G') = M(G_1) \cup M(G_2)$, поэтому $\tau \in M(G')$. После присоединения G' к G получаем $\tau \in M(G)$.

Случай 3. $f' = \varphi \wedge \psi$**.**

Аналогично, как и случай 2, с использованием теорем 2 и 4.

2) Докажем, что если $\tau \in M(G')$**, то** $\tau \models f'$**.**

Базис.f' – булева формула.

В начале алгоритма VerificationPLTLMinus в G' включаются вершины, соответствующие начальным ($p_0(v) > 0$) состояниям цепи Маркова. Во множество D записываются такие v, что $T[v] \not\models f'$.

Таким образом, после операции Reduce в графе остаются вершины v, которые являются траекториями и для которых $T[v] \models f'$. Так как $\tau \in M(G)$, то одна из таких v порождает τ, поэтому $s_1 \models f'$, т. е. $\tau \models f'$.

Индукционный шаг.

Случай 1. $f'{=}X\ \varphi$**.**

$\tau \in M(G') \Rightarrow \exists\ v_1, \ldots, v_k$ в G', порождающая τ. Так как в VerificationStep было построено $(v_1, v_2) \in G'$, то $v_2, \ldots, v_k$ – это траектория в $G'' = G_\varphi$, тогда по индукционному предположению $s_2, s_3, \ldots \models \varphi$, но тогда $\tau \models X\ \varphi$ так как есть ребро (s_1, s_2).

Случай 2. $f' = \varphi \vee \psi$**.**

Пусть $\tau \in M(G')$. По теоремам 1 и 3 $M(G') = M(G_1 \cup G_2)$, поэтому $\tau \in M(G_1)$ или $\tau \in M(G_2)$, тогда по индукционному предположению $\tau \models \varphi$ или $\tau \models \psi$, т. е. $\tau \models \varphi \vee \psi$.

Случай 3. $f' = \varphi \vee \psi$**.**

Случай 3 аналогичен случаю 2 (используются теоремы 1 и 3).

Следствие 7. *Алгоритм VerificationPLTLMinus(f, M) вычисляет вероятность истинности формулы f на цепи Маркова M.*

Доказательство.

По теореме 6 алгоритм VerificationPLTLMinus(f, M) строит ГИП G, который порождает множество траекторий цепи Маркова M, на которых истинна формула f. Далее с помощью алгоритма ProbCalculate вычисляется вероятность, которая по теореме 5 равна вероятности траекторий из $M(G)$. Таким образом, алгоритм VerificationPLTLMinus(f, M) вычисляет вероятность траекторий M, на которых истинна f.

Теорема 8. *Алгоритм VerificationPLTLMinus вычисляет вероятность истинности формулы f логики $PLTL^-$ на цепи Маркова M за время, полиномиальное от размера пространства состояний и длины формулы.*

Доказательство.

Пусть f' – формула, эквивалентная f, построенная по лемме 1, для которой используется алгоритм VerificationPLTLMinus. Введём следующие обозначения:
$n \equiv |M| = |S| + |R|$,
$m \equiv |G| = |V| + |E|$,
$x(x')$ – количество операторов X в формуле $f(f')$,
u – количество операторов $U^{\leq k}$ в формуле f,
l (l') – количество операторов $\wedge$ и $\vee$ в формуле f (f'), хотя бы в одном из аргументов которых есть подформула с темпоральным оператором,
K – максимальное число i среди операторов $f_1 U^{\leq i}$ в формуле f.

По формуле эквивалентной замены $U^{\leq K}$ из леммы 1 получаем, что $x' \leq x + K \cdot u$.

В начале алгоритма VerificationPLTLMinus размер графа G равен n (после начального заполнения). Далее рекурсивно выполняется алгоритм VerificationStep в зависимости от текущего оператора:

Случай 1. $f' = X\ \varphi$

В этом случае в граф G добавляется не более $|S|$ новых вершин и не более $|R|$ новых рёбер. Выполняется этот шаг за время $\mathrm{O}(n)$.

Случай 2. $f' = \varphi \wedge \psi$, $f' = \varphi \vee \psi$

Рассмотрим процедуру Join. Пусть $m_1 = |G_1|, m_2 = |G_2|$. Каждая вершина проходится по одному разу, также в первом цикле для пары (v_1, v_2) находится пара соседей (u_1, u_2), где $T_1[u_1] = T_2[u_2]$, $(v_1, u_1) \in E_1$, $(v_2, u_2) \in E_2$, что занимает O(n) времени. Поэтому время работы процедуры $O\left((m_1 + m_2)^2\right)$. Для каждой вершины $v \in V_1 \cup V_2$ добавляется не более одной вершины в G, и для каждого ребра $(u, v) \in E_1 \cup E_2$ добавляется не более одного ребра в G. Таким образом, размер результирующего объединения $|G| \leq |G_1| + |G_2|$. Если одна из формул φ или ψ (пусть ψ) является булевой (без темпоральных операторов), то граф G_2 будет состоять лишь из вершин V^2_{START}, соответствующих начальным состояниям цепи Маркова, и для этого множества есть соответствующее V^1_{START} в G_1. По алгоритму Join для каждой пары $(u_1 \in G_1, u_2 \in G_2) : u_1 \in V^1_{START}, u_2 \in V^2_{START}, T_1[u_1] = T_2[u_2]$ добавляется одна вершина в G. Таким образом, алгоритм Join проходит лишь каждую вершину графа G_1, что занимает $O(m_1)$ времени, и результирующий граф будет иметь размер $|G| \leq |G_1|$.

Для процедуры Intersection оценка аналогична.

Процедура Reduce проходит каждую вершину не более одного раза и поэтому время ее работы $O(n)$.

Таким образом, размер получаемого графа G $\mathrm{O}(n + x' \cdot n + (l'+1) \cdot n) = O((x' + l' + 2) \cdot n)$. Поэтому каждый шаг выполняется за время $O\left((x' + l' + 2)^2 \cdot n^2\right)$, а все шаги – за время $O\left((x' + l' + 2)^3 \cdot n^2\right)$. Формулу f' можно преобразовать в эквивалентную, в которой $l' \leq x'$. Таким образом, размер графа равен $O((2x' + 2) \cdot n)$. Тогда каждый шаг будет выполняться за $O\left((2x' + 2)^2 \cdot n^2\right)$ или $O(2(x + K \cdot u) + 1)^2 \cdot n^2)$, а все шаги – за время $O\left((2x' + 1)^3 \cdot n^2\right)$ или $O(2(x + K \cdot u) + 1)^3 \cdot n^2)$.

5. Сравнение алгоритмов VerificationPLTLMinus и Куркубетиса–Яннакакиса на примерах

5.1. Пример применения алгоритма VerificationPLTLMinus

Рассмотрим задачу верификации формулы $f = X(\neg y \vee z) \vee XX(x \wedge y)$ языка $PLTL^-$ на цепи Маркова M (рис. 1). Состояния: q1, q2, q3. Подмножества истинных элементарных утверждений на состояниях:

$$L(q1) = \{x\}, L(q2) = \{x, y\}, L(q3) = \{y, z\}.$$

Начальное распределение вероятностей:

$$P0 = \begin{pmatrix} 0.5 & 0.5 & 0 \end{pmatrix}$$

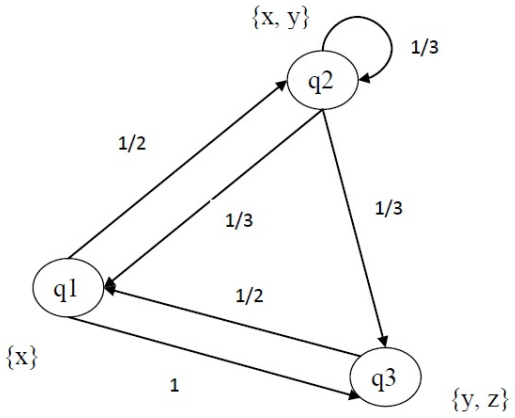

Рис. 1. Цепь Маркова

Матрица вероятностей переходов:

$$P = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 1/3 & 1/3 & 1/3 \\ 1 & 0 & 0 \end{pmatrix}$$

На рисунках 2 – 8 изображён процесс построения графа истинных префиксов G, который будет построен в результате рекурсивного выполнения алгоритма. Для наглядности, метка рядом с вершиной v – это значение $T[v]$, т. е. соответствующее v состояние цепи Маркова. Вершины, расположенные в верхней горизонтали графа, соответствуют начальным вершинам, т. е. $\in V_{START}$. Вершины, из которых не выходят дуги, соответствуют конечным вершинам, т. е. $\in V_{END}$.

1. Применение алгоритма VerificationPLTLMinus(f, M): стартовое заполнение графа $G(M, f)$ – две вершины v_1 и v_2 без рёбер, для которых выполнено $T[v_1] = q1$, $T[v_2] = q2$, $v_1, v_2 \in V_{START}$, $v_1, v_2 \in V_{END}$.

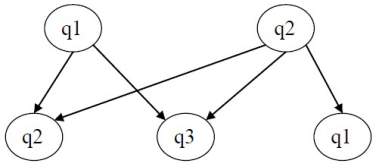

Рис. 2. Граф G_1, случай $f = X\,\varphi = X\,(\neg y \vee z)$

2. После этого запускается рекурсивный алгоритм VerificationStep($f = X\,(\neg y \vee z) \vee XX\,(x \wedge y)$, M, G), в котором выполняется случай 3.

Рекурсивно строятся графы G_1 и G_2 аргументов ($X\,(\neg y \vee z)$ и $XX\,(x \wedge y)$) (пункты 2.1 и 2.2).

2.1. Выполнение алгоритма VerificationStep($X\,(\neg y \vee z)$, M, G_1).

2.1.1. Случай $f = X\,\varphi = X\,(\neg y \vee z)$

Добавляем рёбра из текущих конечных вершин (рис. 2).

2.1.2. Выполнение алгоритма VerificationStep($f = \neg y \vee z$, M, G_1) – случай булевой формулы (рис. 3).

$$V_{END} = \{q1, q2, q3\};$$

$q2 \not\models y \vee z$, поэтому $D = \{q2\}$.

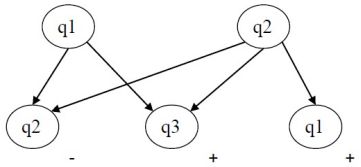

Рис. 3. Граф G_1, случай – булева формула: $f = \neg y \vee z$, плюсами помечены конечные состояния, на которых истинна f, минусами – на которых ложна.

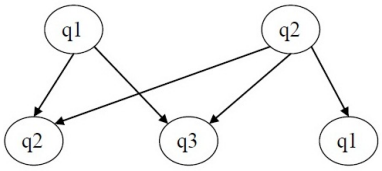

Рис. 4. Случай $f = X\,\varphi = X\,(X\,(x \wedge y))$

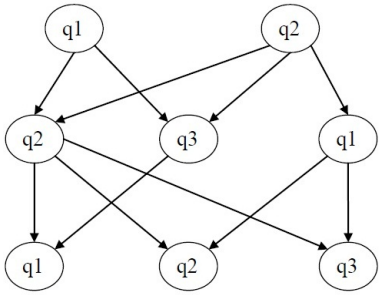

Рис. 5. Граф G_2, случай $f = X\varphi = X\,(x \wedge y)$

2.2. Выполнение алгоритма VerificationStep($\psi = XX\,(x \wedge y)$, M, G_2).

2.2.1. Случай $f = X\,\varphi = X\,(X\,(x \wedge y))$ (рис. 4).

2.2.2. Выполнение алгоритма VerificationStep($f = X\,(x \wedge y)$, M, G_2), случай $f = X\varphi$ (рис. 5).

2.2.3. Выполнение алгоритма VerificationStep($f = x \wedge y$, M, G_2), случай f - булева формула.

$q1 \not\models x \wedge y$ и $q3 \not\models x \wedge y \Rightarrow D = \{q1,\ q3\}$

3. Применяем операцию Reduce к графам G_1 и G_2, получившимся в пункте 2 (рис. 6).

4. Строится объединение $G_1 \cup G_2$ по алгоритму $G' = \mathrm{Join}(G_1, G_2)$ (рис. 7).

5. Присоединяем конечные состояния графа G к начальным состояниям графа G', полученного в пункте 4. В данном случае результат будет совпадать с G'.

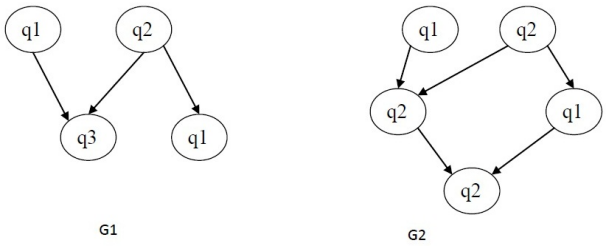

Рис. 6. Результат применения алгоритма Reduce к графам G_1 и G_2

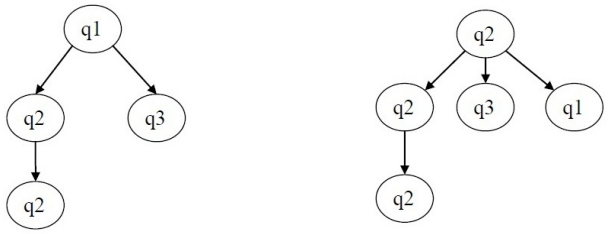

Рис. 7. $G' = \mathrm{Join}(G_1, G_2)$

6. Алгоритм VerificationStep завершён, выполняется завершающий этап алгоритма VerificationPLTLMinus: получение начального распределения вероятностей начальных вершин и вероятностей рёбер графа G по цепи Маркова M (рис. 8).

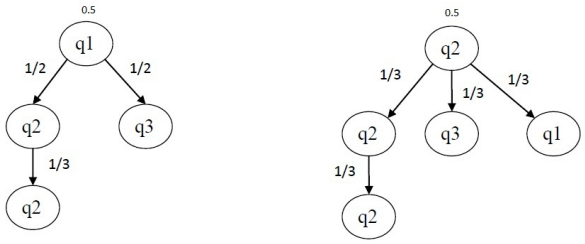

Рис. 8. Метки вершин и рёбер в графе G

Это и есть граф истинных префиксов для данной цепи Маркова M и верифицируемой формулы f. Множество всех истинных траекторий имеет вид

$L(f) = \{cyl(q1, q2, q2),\ cyl(q1, q3),\ cyl(q2, q3),\ cyl(q2, q1),\ cyl(q2, q2, q2)\}$

Вероятность истинности формулы тогда равна (по алгоритму ProbCalculate)

$$Prob(G) = 0.5 \cdot (\frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2}) + 0.5 \cdot (\frac{1}{3} + \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{3}) = \frac{13}{18}$$

5.2. Сравнение алгоритма VerificationPLTLMinus с алгоритмом Куркубетиса–Яннакакиса

В таблице 1 приведены размеры конечных систем переходов, получающихся в процессе верификации четырех формул языка $PLTL^-$ различного размера двумя алгоритмами на цепи Маркова M, приведенной в предыдущем примере на рис. 1.

Формула	**Алг. Куркубетиса–Яннакакиса**		**Алг. VerificationPLTLMinus**	
	Состояния	Переходы	Состояния	Переходы
$X^2(\neg y \vee z)$	9	18	7	9
$X^3(\neg y \vee z)$	17	34	10	15
$X^4(\neg y \vee z)$	33	66	13	21
$X^5(\neg y \vee z)$	65	130	16	27

Таблица 1. Сравнение алгоритмов

Как видно из таблицы 1, в этих примерах при увеличении числа темпоральных операторов X размер графа переходов и, следовательно, время верификации алгоритмом Куркубетиса–Яннакакиса растёт в геометрической прогрессии, а алгоритмом VerificationPLTLMinus – в арифметической.

Список литературы

1. Baier C., Katoen J.-P., Principles of model checking. The MIT Press. Cambridge, Massachusetts. London, England, 2008.

2. Courcoubetis C., Yannakakis M., The complexity of probabilistic verification // J. ACM. 1995. V. 42, 4. P. 857–907.

3. Dekhtyar M., Dikovsky A., Valiev M. On complexity of verification of interacting agents behavior // Annals of pure and applied logic. 2006. 141. P. 336 – 362.

4. Dekhtyar M., Dikovsky A., Valiev M. Temporal Verification of Probabilistic Multi-Agent Systems. Pillars of Computer Science: Essays Dedicated to Boris (Boaz)Trakhtenbrot on the Occasion of His 85th Birthday // LNCS. 2008. N 4800. P. 256–265.

5. Hansson H., Jonsson B. A logic for reasoning about time and reliability. Formal Aspects of Computing. 1994. P. 512–535.

6. Kwiatkowska Marta. Model checking for probability and time: from theory to practice // Proc. 18th IEEE Symposium on Logic in Computer Science (LICS'03), IEEE Computer Society Press, 2003. P. 351–360.

7. Luka de Alfaro. Formal verification of probabilistic systems. PhD, Stanford Univ., 1997.

8. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. МЦНМО, М., 2002.

9. Лебедев П.В. Программа верификации вероятностных многоагентных систем // Труды XII Национальной конференции по искусственному интеллекту с международным участием. Т. 4. М.: Физматлит, 2010. С. 81–88.

Polynomial Algorithm of Verification for Subset of PLTL Logic

Lebedev P.V.

Keywords: Markov chains, PLTL, temporal logics, verification of dynamic properties

In this article a polynomial algorithm is described of verification of dynamic properties of Markov chains described by formulas of a subset of temporal logic PLTL (propositional temporal logic of linear time). The algorithm allows to find probability of the validity of the formula on the Markov chain, and also set of trajectories on which the verified formula is true.

Сведения об авторе:
Лебедев Павел Валерьевич,
Тверской государственный университет,
аспирант