وارسی ویژگی دسترس‌پذیری در سامانه‌های تبدیل گراف با رویکرد کشف وابستگی شرطی بین قوانین

جعفر پرتابیان

دانشکدهٔ مهندسی کامپیوتر- واحد لامرد- دانشگاه آزاد اسلامی-لامرد- ایران

عضو باشگاه پژوهشگران جوان و نخبگان - واحد لامرد- دانشگاه آزاد اسلامی-لامرد- ایران

چکیده

وارسی الگو[1] یکی از مؤثرترین شیوه‌های صحت‌سنجی خودکار ویژگی‌های سامانه‌های سخت‌افزاری و نرم‌افزاری است. در حالت کلی، در این روش، الگویی از سامانه موردنظر تولید می‌شود و تمام حالات ممکن در گراف فضای حالت کاوش می‌شود تا بتواند خطاها و الگوهای نامطلوب را پیدا کند. در سامانه‌های بزرگ و پیچیده تولید تمام فضای حالت منجر به مشکل انفجار فضای حالت[2] می‌شود. پژوهش‌های اخیر نشان می‌دهند که کاوش در فضای حالت با استفاده از روش‌های هوشمندانه، می‌تواند پیشنهاد امیدوارکننده‌ای باشد؛ ازاین‌رو، در این پژوهش، نخست، الگویی از سامانه موردنظر ایجاد می‌شود، سپس، بخشی از فضای حالت الگو، تولید شده و با استفاده از احتمالات شرطی، وابستگی بین قوانین موجود در فضای حالت کشف می‌شوند. پس از آن، با کمک وابستگی‌های کشف‌شده، باقی فضای حالت الگو به‌طور هوشمندانه کاوش می‌شود. در این مقاله روشی برای وارسی ویژگی دسترس‌پذیری[3] در سامانه‌های نرم‌افزاری پیچیده و بزرگ که به زبان رسمی، تبدیل گراف[4] (GTS) الگو شده‌اند، ارائه می‌شود. روش پیشنهادی در GROOVE که یک مجموعه ابزار منبع باز برای طراحی و بررسی وارسی الگوی سامانه‌های تبدیل گراف است، پیاده‌سازی شده‌است. نتایج آزمایش‌های تجربی نشان می‌دهند رویکرد پیشنهادی نسبت‌به روش‌های قبلی سریع‌تر بوده و مثال‌های نقض[5]/شاهد[6] کوتاه‌تری تولید می‌کند.

واژگان کلیدی: وارسی الگو، انفجار فضای حالت، سامانه تبدیل گراف، جدول وابستگی شرطی، ویژگی دسترس‌پذیری

Checking Reachability Property of Systems Specified through Graph Transformation with the Approach of Discovery Conditional Dependency Between the Rules

Jaafar Partabaian

Department of Computer Engineering, Lamerd Branch,
Islamic Azad University, Lamerd, Iran.
Young Researchers and Elite club Lamerd branch,
Islamic Azad University, Lamerd, Iran.

Abstract

Model checking is among the most effective techniques for automatic verification of hardware and software systems' properties. Generally, in this method, a model of the desired system is generated and all possible states are explored in the space state graph to find errors and undesirable patterns. In models of large, complex systems, if the size of the generated state space is too extensive, so that not all

[1] Model checking
[2] State space explosion
[3] Reachability property
[4] Graph transformation system
[5] Counterexample
[6] Witness

* نویسندهٔ عهده‌دار مکاتبات

* Corresponding author

• تاریخ ارسال مقاله: ۱۴۰۰/۳/۲۰ • تاریخ پذیرش: ۱۴۰۲/۳/۱۲ • تاریخ انتشار: ۱۴۰۲/۷/۳۰ • نوع مطالعه: پژوهشی

[DOI: 10.61186/jsdp.20.2.175]

available states can be explored due to computational restrictions, the problem of state space explosion occurs. In fact, this problem confines the validation process in model verification systems. To use the model checking technique, the system must be described in a formal language. Graphs are very beneficial and intuitive tools for describing and modelling software systems. Correspondingly, graph transformation system provides a proper tool for formal description of software system features as well as their automatic verification.

Various techniques have been investigated in the researches to reduce the effect of state space explosion problem in the model checking process. Some of these methods try to reduce the required memory by reducing the number of cases explored. Among others are symbolic model checking, partial-order reduction, symmetry reduction, and scenario-driven model checking. In a complex system, these algorithms, along with conventional methods such as DFS or BFS search algorithms may not afford any complete answer due to the explosion of state space. Hence, the use of intelligent methods such as knowledge-based techniques, datamining, machine learning, and meta-heuristic algorithms which do not entail full state space exploration could be advantageous.

Recent researches attest that exploring the state space using intelligent methods could be a promising idea. Therefore, an intelligent method is used in this research to explore the state space of large and complex systems. Accordingly, in this paper, first a model of the desired system is created using graph conversion system. Then, a portion of the state space of the model is generated. Afterwards, using the conditional probability table, the dependencies between the rules in the paths toward the goal state are discovered. Finally, by means of the discovered dependencies, the rest of the model state space is intelligently explored. In other words, only promising paths, i.e., those who match the detected dependencies are explored to reach the goal state. It is worth noting that the first goal of the proposed approach is to find a goal state, i.e., one in which either the safety property is violated, or the reachability property is satisfied in the shortest possible time. The second less important goal is to reduce the number of explored states in the graph of the state space until reaching the goal state. This paper provides a way to check the availability feature in complex, large software systems modelled in the official graph transformation language. The suggested method is implemented in GROOVE, which is an open-source toolset for designing and model checking graph transformation systems. The results of experimental tests indicate that the proposed approach is faster than the previous methods and produces a shorter counterexample/witness.

Keywords: Model checking, State space explosion, Graph transformation system, Conditional probability table, Reachability property

۱- مقدمه

امروزه سامانه‌های نرم‌افزاری بسیار پیشرفته‌اند و طراحان نرم‌افزار علاقه‌مند به استفاده از روش‌هایی هستند که پژوهش بر روی صحت سامانه‌های پیچیده را ممکن سازند. وارسی الگو یکی از روش‌های موفق برای بررسی صحت ویژگی‌های سامانه است[۱]. در این روش، نخست، الگویی از سامانه ایجاد می‌شود. سپس، ویژگی‌های موردنظر در فضای حالت الگو به‌صورت خودکار کاوش شده و درستی یا نادرستی ویژگی‌های داده‌شده بررسی و گزارش می‌شود[۲].

برای استفاده از این روش، سامانه موردنظر باید به‌وسیلهٔ یک زبان رسمی توصیف شود. گراف‌ها ابزارهای بسیار مفید و شهودی برای توصیف و الگوسازی سامانه‌های نرم‌افزاری هستند. سامانه‌های تبدیل گراف (GTS) ابزار مناسبی برای توصیف رسمی ویژگی‌های سامانه‌های نرم‌افزاری و نیز درستی‌یابی خودکار آن‌ها فراهم می‌آورند. همچنین، سامانه‌های تبدیل گراف در بسیاری از فعالیت‌های توسعهٔ نرم‌افزار استفاده می‌شوند؛ از جمله تبدیلِ الگوی[۷] [۳]، نمایش سبک‌های معماری[۸] [۴]، پالایش[۹] [۵]، آبَرالگوسازی[۱۰] [۶]، بازسازی[۱۱] [۷]، الگوسازی و تحلیل گردش کار [۸] و تجزیه‌وتحلیل عملکرد معماری نرم‌افزار [۹]. به دلیل ویژگی‌های پیش‌گفته، سامانه تبدیل گراف را به‌عنوان بستری برای اجرای ایده‌های خود در این مقاله در نظر می‌گیریم.

اگر در الگوهای مربوط به یک سامانه مفروض، اندازهٔ فضای حالت تولیدشده بیش‌ازاندازه بزرگ باشد؛ به‌طوری‌که تمام حالات قابل‌دسترسی نتوانند به دلیل محدودیت‌های محاسباتی کاوش شوند، مشکل انفجار فضای حالت پیش می‌آید. درواقع، این مشکل فرایند صحت‌سنجی را در سامانه‌های وارسی‌کنندهٔ الگو با محدودیت مواجه می‌کند. در این پژوهش، از چنین سامانه‌ای با عنوان یک سامانه پیچیده نام برده‌می‌شود.

[7] Model transformation
[8] Architectural styles representation
[9] Refinement
[10] Meta-modeling
[11] Refactoring

[DOI: 10.61186/jsdp.20.2.175]

در پژوهش‌های انجام‌شده روش‌های متنوعی برای کاهش اثر این مشکل در فرایند وارسی الگو بررسی‌شده‌اند. بعضی از این روش‌ها با کم‌کردن تعداد حالت‌های کاوش‌شده در تلاشند حافظهٔ موردنیاز را کاهش دهند. از جملهٔ آن‌ها، روش وارسی الگوی نمادین[12] [۱۰]، کاهش ترتیب جزئی[13] [۱۱]، کاهش تقارنی[14] [۱۲] و بررسی الگو بر مبنای سناریو[15] [۱۳] هستند. در یک سامانه پیچیده، این الگوریتم‌ها به‌همراه روش‌های مرسوم مانند جستجوی اول عمق (DFS)، یا جستجوی اول بهترین (BFS) ممکن است به دلیل انفجار فضای حالت نتوانند هیچ جواب کاملی ارائه دهند. ازاین‌رو، استفاده از روش‌های هوشمند که نیازمند کاوش کامل فضای حالت نباشند، می‌توانند مفید و مؤثر واقع شوند.

به‌طور معمول، روش وارسی الگو بر مبنای کاوش کامل فضای حالت الگوست و یک مسئلهٔ بهینه‌سازی نیست. درواقع، هدف اصلی وارسی الگو بیشتر، یافتن یک جواب (به‌عنوان مثال، یک حالت خطای دست‌یافتنی) است تا پیداکردن یک جواب بهینه. باوجوداین، الگوریتم‌های هوشمند تلاش دارند که فضای حالت را به‌طور هوشمندانه جستجو کنند. بدین منظور، آن‌ها از اطلاعات مربوط به فضای حالت که پیشتر کشف‌شده‌اند، استفاده کرده و این اطلاعات را برای کشف باقیماندهٔ فضای حالت به‌کارمی‌برند. البته الگوریتم‌های هوشمند نمی‌توانند برای صحت‌سنجی ویژگی‌هایی نظیر ایمنی استفاده شوند؛ زیرا نمی‌توانند فضای حالت را به‌طور کامل کاوش کنند. به‌بیان‌دیگر، اگر یک الگوریتم هوشمند در مسئله‌ای استفاده شود و هیچ خطایی حاصل نشود، با قطعیت نمی‌توان نتیجه گرفت در سامانه خطایی وجود ندارد. باوجوداین، هرگاه خطایی یافت شود، آن الگوریتم زمان کمتری را در مقایسه با الگوریتم‌های سنتی مانند DFS و BFS صرف خواهدکرد [۱۴].

پژوهش‌های اخیر نشان می‌دهند استفاده از رویکرد داده‌کاوی و یادگیری ماشین راه‌حل‌های مناسبی برای کشف نقض ویژگی‌های سامانه در مقایسه با الگوریتم‌های کلاسیک هستند. برای مثال، الگوریتم هیبرید بهینه‌سازی ازدحام ذرات (PSO) و الگوریتم جستجوی گرانشی (GSA) در پژوهش‌های رافع و همکاران جهت کشف بن‌بست [۱۵]، ترکیب الگوریتم‌های تکاملی و بیزین در پژوهش‌های پیرا جهت وارسی ویژگی ایمنی [۱۶] و الگوریتم‌های یادگیری ماشین [۱۷] و [۱۸] جهت وارسی ویژگی دسترس‌پذیری، در سامانه‌های توصیف‌شده توسط سامانه تبدیل گراف استفاده‌شده‌اند. درواقع، آن‌ها گراف الگوهای سامانه‌ای بزرگ را پیمایش می‌کنند تا حالت‌های قابل‌قبول را پیدا کنند، درحالی‌که روش‌های سنتی مانند DFS و BFS در این مورد کارایی ندارند.

رویکرد پیشنهادی در نرم‌افزار GROOVE [۱۹] اجراشده است که ابزاری متن‌باز برای الگوسازی و صحت‌سنجی سامانه‌های تعیین‌شده توسط GTS است. مقالهٔ حاضر به وارسی ویژگی دسترس‌پذیری می‌پردازد. ازآنجاکه بررسی ویژگی پیشروی[16] الگوریتم‌های پیچیده‌تری را می‌طلبد، بیشتر مطالعات موجود تلاش داشته‌اند فقط به ویژگی‌های ایمنی[17] و دسترس‌پذیری بپردازند و ویژگی پیشروی را نادیده بگیرند. قابل‌ذکر است که اولین هدف رویکرد پیشنهادی پیداکردن یک حالت هدف[18] (حالتی که در آن یا ویژگی ایمنی نقض شده‌اند، یا ویژگی دسترس‌پذیری برقرار باشد)، در کوتاه‌ترین زمان ممکن، است. هدف دوم که از اهمیت کمتری برخوردار است، تعداد حالت‌های پیمایش‌شده در گراف فضای حالت تا رسیدن به حالت هدف است.

ادامهٔ مقاله به‌صورت زیر سازمان‌دهی شده‌است: کارهای مرتبط در بخش ۲ توضیح داده‌شده‌اند. در بخش ۳ به‌طور خلاصه پیش‌زمینه‌های لازم مانند وارسی الگو و شکل GTS توصیف‌شده‌اند. در بخش ۴ رویکرد پیشنهادی با جزئیات کامل شرح داده شده، در بخش ۵ اجرای روش پیشنهادی به‌همراه نتایج تجربی گزارش شده، در بخش ۶ مزایا و محدودیت‌های روش پیشنهادی بحث شده‌اند؛ و در نهایت، بخش ۷ که پایان‌دهندهٔ مقاله است، به نتیجه‌گیری و پژوهش‌های آینده اختصاص دارد.

۲- کارهای مرتبط

برای پرداختن به مسئلهٔ انفجار فضای حالت در وارسی الگو، می‌توان به راه‌حل‌های کلاسیک مانند درستی‌یابی ترکیبی [۲۰ و ۲۱]، کاهش ترتیب جزئی [۲۵-۲۲] و کاهش تقارنی [۲۶] اشاره کرد. این راهکارها در تلاشند که اندازهٔ فضای حالت را کوچک کنند. به‌علاوه، راهکارهای

[12] Symbolic model checking
[13] Partial-order reduction
[14] Symmetry reduction
[15] Scenario-driven model checking

[16] Liveness
[17] Security
[18] Goal state

[DOI: 10.61186/jsdp.20.2.175]

دیگری نیز موجودند که هدفشان کاهش حافظهٔ موردنیاز برای ذخیرهٔ حالت‌های پیمایش شده هستند [۳۲-۲۷].

الگوریتم‌های جستجوی ابتکاری ساده‌ای مانند A*، IDA*[19] و جستجوی پرتو[20] (BS) گروه‌های دیگری از روش‌ها هستند که برای مدیریت مسئلهٔ انفجار فضای حالت در وارسی الگو استفاده می‌شوند. راه‌حل‌های ابتکاری در وارسی ترجیحی الگو [۳۳] و وارسی صریح الگو [۳۴] استفده می‌شوند. در این راه‌حل‌ها، نخست، حالت‌هایی پیمایش می‌شوند که احتمال رسیدن به خطا در آن‌ها بیشتر باشد. در تحقیق یانگ و دیل [۳۵]، در ابزار مورف[21] از جستجوی اول- بهترین همراه با نمودار تصمیم دودویی (BDD)- برای وارسی الگو استفاده شده‌است. در پژوهش ادلکامپ و همکاران [۳۴] روشی با استفاده از A* و IDA* ارائه شده که می‌تواند جهت تشخیص بن‌بست و ایمنی در سامانه‌های هم‌زمان مانند پروتکل‌های ارتباطی استفاده شود. با فرض دانستن حالت هدف، این روش از فاصلهٔ همینگ بین نمایش دودویی حالت‌های هدف و جاری به‌عنوان یک تابع ابتکاری استفاده می‌کند. درصورتی‌که حالت هدف مجهول باشد، تعداد جریان‌های فعال در یک حالت یا تعداد انتقال‌های خروجی یک حالت به‌عنوان تابع ابتکاری درنظرگرفته‌می‌شود. برای ارزیابی این الگو، آن را در HSF-SPIN [۳۴] که یک ابزار توسعه‌یافتهٔ SPIN است، اجرا می‌شود. نتایج اجراهای HSF-SPIN بر روی پروتکل‌های متعدد نشان می‌دهد که این روش جدید در مقایسه با بررسی‌کنندهٔ الگوی SPIN می‌تواند حالت هدف را با پیمایش تعداد کمتری از حالت‌ها کشف کند.

در تحقیق گروس و ویسر [۳۶] مؤلفان چند تابع ابتکاری را بر اساس تعداد نخ‌ها ارائه دادند تا A* را هدایت کنند. آنان همچنین، الگوریتم‌های جستجوی پرتوی را برای یافتن خطا در برنامه‌های جاوا استفاده کردند. نتایج تجربی اجرای این ابتکارها در Pathfinder (JPF) کارایی بالای آن‌ها را در مقایسه با DFS، BFS و الگوریتم‌های تکاملی در برخی مطالعات موردی نشان می‌دهند. در پژوهش ادلکامپ و رفل [۳۷] رویکردی تحت عنوان BDDA* با استفاده از نمودارهای تصمیم‌گیری دودویی، الگوریتم BFS را با الگوریتم سنتی A* ترکیب می‌کند تا حالت‌های بن‌بست را در برنامه‌های جاوا کشف کند. نتایج تجربی، عملکرد بهتر BDDA* را در مقایسه با الگوریتم BFS که از OBDD استفاده می‌کند، نشان می‌دهند.

علاوه‌بر کارهای گروس و ویسر [۳۶ و ۳۷]، پژوهش انجام‌شده توسط میوکا و همکاران [۳۸] روشی را معرفی می‌کند که بتواند نقض ویژگی‌های ایمنی را در برنامه‌های جاوا آشکار کند. این روش امکان عقب‌گرد به الگوریتم DFS را نیز در نظر می‌گیرد. در مورد عقب‌گرد باید گفت عقب‌گرد به حالت‌هایی که منجر به خطا می‌شوند، دارای احتمال زیادی است. نتایج تجربی اجرای این روش در JPF حاکی از کارایی بالای آن در برابر DFS و BFS در بسیاری از مطالعات موردی است.

در پژوهش ادلکامپ و همکاران [۳۹] الگوریتم A* در بررسی الگو، حالت صریح به‌کاررفته تا بتواند نقض خاصیت پیشروی را کشف کند. رویکردهای پژوهش استلر و ورهایم [۴۰] و اسنیپ [۴۱] از جمله مثال‌های دیگری است که در مورد استفاده از A* قابل‌ذکر است. این رویکردها از ابتکارهای بر پایهٔ تشابه و بر پایهٔ اختلاف استفاده می‌کنند؛ با این هدف که الگوریتم‌های A* و بهترین اول را هدایت کنند تا ویژگی‌های دسترس‌پذیری در سامانه‌های تعیین‌شده توسط GTS تحلیل شوند. ابتکار بر پایهٔ تشابه می‌تواند مقدار تشابه ساختاری (گره‌ها و لبه‌ها) را بین گراف‌های G_s و G_g مربوط به حالت فعلی s و یک ویژگی مفروض g محاسبه کند. علاوه‌بر این، در ابتکار بر پایهٔ اختلاف تعداد گره‌ها و لبه‌ها در G_s که باید برای رسیدن به G_g ایجاد یا حذف شوند، محاسبه می‌شوند. کارایی آن‌ها نسبت به BFS و DFS به‌وسیلهٔ نتایج تجربی تأیید شده‌است.

در پژوهش‌های زیگرت و الزینگا [۴۲ و ۴۳] چندین ابتکار بر پایهٔ انتزاع جهت هدایت A*، جستجوی بهترین اول و الگوریتم جستجوی تپه‌نوردی برای بررسی خاصیت دسترس‌پذیری ارائه شده‌اند؛ نتایج تجربی و ابتکارها هم نشان می‌دهند که این‌ها می‌توانند در بسیاری از مطالعات موردی، یک حالت هدف را سریع‌تر از دیگر روش‌ها بیابند.

روش‌های فراابتکاری نمونه‌های دیگری از روش‌ها هستند که بر روی انفجار فضای حالت کار می‌کنند. رویکردهای بر پایهٔ الگوریتم ACO در طی پیمایش مسیر و با هدف کشف حالت‌های خطا، مسیرهای کوتاه‌تر را به مسیرهای بلندتری که از رفتار مورچگان الهام گرفته‌شده‌اند، ترجیح می‌دهند. در این رویکرد، به دلیل نیاز کمتر به حافظه جهت ذخیره‌کردن حالت‌ها، انفجار فضای حالت رخ نمی‌دهد [۴۴-۴۶].

[19] Iterative deepening A*
[20] Beam search
[21] Murf

[DOI: 10.61186/jsdp.20.2.175]

آلبا و چیکانو [۴۵] یک نوع جدید از ACO به نام ACOhg مخفف (بهینه‌سازی کلونی مورچه برای گراف‌های بزرگ) به جهت کشف نقض ایمنی معرفی کردند. همچنین، در مقاله‌ای دیگر، آلبا و چیکانو [۴۷] کار خود را برای ردکردن ویژگی پیشروی تعمیم دادند. به‌علاوه، در پژوهش‌های کومازاوا و همکاران [۴۸] و تاکادا و همکاران [۴۹] مؤلفان توانسته‌اند ACOhg را تحت عنوان EACOhg تعمیم دهند تا از حرکت‌های بی‌فایدهٔ مورچگان جلوگیری شود. بدین منظور، آن‌ها فرومون‌هایی شبه‌بو ارائه کردند که متفاوت‌تر از فرومون‌های مرسوم منتشر می‌شوند. درواقع، این فرومون‌ها از غذاها منتشر می‌شوند و می‌توانند مورچگان را به‌طرف غذا هدایت کنند. EACOhg سریع‌تر و دقیق‌تر از ACOhg عمل می‌کند، ولی باوجوداین، EACOhg نیاز به حافظهٔ بیشتری دارد. برای ارزیابی کارایی، آن‌ها در LTSA که ابزاری برای وارسی الگو است و توسط فوستر و همکاران [۵۰] با هدف صحت‌سنجی سامانه‌های هم‌زمان بر پایهٔ عمل ابداع‌شده، اجرا کردند.

اگرچه GA به‌طور معمول، برای مسائل بهینه‌سازی به‌کارمی‌رود، برای وارسی الگو نیز استفاده می‌شود. برای انجام این کار، مفاهیمی مانند کروموزوم‌ها باید دوباره تعریف شوند: یک کروموزوم به‌عنوان یک مسیر تصادفی در فضای حالت الگو با یک طول ویژه که از یک حالت اولیه آغاز می‌شود، درنظرگرفته‌می‌شود. از جمله رویکردهای بر پایهٔ GA می‌توان به پژوهش‌های گدفروید و همکاران [۵۱] و نیز آلبا و همکاران [۵۲] اشاره نمود. در پژوهش گدفروید و همکاران [۵۱] چارچوب جدیدی برای کاهش‌دادن فضای حالت در سامانه‌های واکنشی هم‌زمان معرفی‌شده که از GA استفاده می‌کند. این چارچوب از ابزاری برای پیمایش فضای حالت سامانه‌ها استفاده می‌کند که در پژوهش گدفروید [۵۳] از آن با عنوان Verisoft نام برده شده‌است. در پژوهش آلبا و همکاران، یک بررسی‌کنندهٔ متنی الگو بر پایهٔ GA با هدف کشف بن‌بست‌ها در برنامه‌های جاوا طراحی شده‌است. در پژوهش چیکانو و همکاران [۵۴] نویسندگان پنج الگوریتم فراابتکاری را مقایسه کرده‌اند که شامل PSO، ACO، SAو دو نسخه از GA بوده و برای حل مسئلهٔ یافتن بن‌بست‌ها در برنامه‌های جاوا استفاده می‌شوند. نتایج گزارش‌شده نشان می‌دهند که این الگوریتم‌های فراابتکاری کارایی بیشتری در یافتن خطاها نسبت‌به الگوریتم‌های جستجوی کلاسیک دارند.

در مقالهٔ ادلکامپ و همکاران [۵۵] چارچوبی با استفاده از جستجوی ابتکاری ارائه‌شده که می‌تواند خصوصیات ساختاری سامانه‌های الگوشده به‌وسیلهٔ GTS را تحلیل کند. به عقیدهٔ آن‌ها، جستجوی ابتکاری به کاهش زمان تحلیل و دستیابی به مثال‌های نقض کوتاه‌تر در GTS کمک می‌کند. این چارچوب از الگوریتم A* استفاده کرده و در HSF-SPIN[۵۶] اجرا می‌شود.

در پژوهش یوسفیان و همکاران [۵۷]، دو رویکرد جدید ارائه شده که می‌تواند بن‌بست‌ها را در سامانه‌های پیچیدهٔ توصیف‌شده توسط GTS کشف کند. در رویکرد اول به نام BAPSO، الگوریتم BAT (خفاش) جهت اصلاح تأثیرات PSO استفاده شده‌است. BAT یک الگوریتم فراابتکاری است که ایدهٔ آن از رفتارهای مکان‌یابی خفاش به‌وسیلهٔ اکوی صدا برای پیداکردن غذا گرفته شده‌است [۵۸]. همچنین، دومین رویکرد به نام BFA از ایدهٔ حریصانه استفاده می‌کند. نتایج تجربی نشان از سریع‌تر و دقیق‌تر بودن BAPSO و BFA نسبت‌به GA، PSO، A*، DFS و BFS در بسیاری از مطالعات موردی می‌دهند.

در پژوهش پیرا و همکاران [۵۹]، روشی جدید بر اساس روش‌های داده‌کاوی و با هدف مطالعهٔ اثر انفجار فضای حالت در بررسی الگوی سامانه‌های پیچیده و بزرگ ارائه شده‌است. این سامانه‌ها باید از طریق سبک معماری تعیین شوند و به‌وسیلهٔ سامانه تبدیل گراف الگوسازی شوند. سبک معماری یک فراالگوست که خانواده‌ای از معماری‌های مرتبط و ویژگی‌های مشترک آن‌ها را تبیین می‌کند [۶۰ و۶۱]. این روش که با نام اختصاری DMCDM شناخته‌می‌شود و هدفش بررسی یک الگوی بزرگ و پیچیده است، دانش ویژه‌ای را با استفاده از وارسی الگوی کوچک‌تری از همان سبک به دست می‌آورد. این روش برای ردکردن ویژگی‌های ایمنی و پیشروی و نیز تأیید ویژگی دسترسی‌پذیری در سامانه‌ها به‌کارمی‌رود. این روش جهت ارزیابی عملکردش در نرم‌افزار GROOVE اجرا شد. نتایج تجربی نشان از دقت و سرعت این رویکرد دارند. در مقابل این مزایا، این رویکرد محدودیت‌هایی هم دارد. نخست، EMCDM تنها در سامانه‌های بر پایهٔ سبک معماری کاربرد دارد. دوم، چون در یک فضای حالت نامتناهی، این روش نمی‌تواند فضای حالت الگوهای کوچک‌تر را به‌طور جامع بپیماید، در عمل با مشکلاتی مواجه می‌شود. سومین محدودیت این است که کارایی EMCDM بسیار تحت‌تأثیر الگوهای کوچک‌ترِ طراحی‌شده است. درواقع، عدم وجود یک الگوی

[DOI: 10.61186/jsdp.20.2.175]

کوچک‌تر برای یک سامانه باعث موفق‌نبودن EMSDM می‌شود. برای رفع چنین محدودیت‌هایی پژوهشگران رویکردی را پیشنهاد داده‌اند مبنی بر این‌که فقط قسمتی از فضای حالت، به‌جای‌این‌که بعضی از الگوهای کوچک‌تر به‌طور کامل پیموده شوند، پیمایش شود [۶۲]. در اولین رویکرد به نام LDM (یادگیری به‌وسیلهٔ داده‌کاوی)، از یک نسخهٔ اصلاح‌شدهٔ الگوریتم Apriori[۶۳] استفاده شده تا یک الگوی تکرارشونده را از فضای حالتی که به‌طور جزئی پیموده شده، کشف کند. یک الگوی تکرارشونده دنباله‌ای از قوانین است که در اثر تکرار استفاده از آن‌ها منجر به دسترسی به یک حالت هدف خواهدشد. درعین‌حال، دومین رویکرد تحت عنوان LBN از یک شبکهٔ بیزین برای یادگیری دانش از فضای حالت جزئی استفاده می‌کند. سپس، با استفاده از دانش کشف‌شده (الگوی تکرارشونده یا شبکهٔ بیزی)، باقیماندهٔ فضای حالت الگو به‌طور هوشمند پیمایش می‌شود تا زمانی که یک حالت هدف به‌دست‌آید. LDM و LBN مشابه با EMCDM، برای تحلیل ویژگی‌های دسترس‌پذیری، ایمنی و پیشروی می‌توانند استفاده شوند. در مقالهٔ [۶۴]، روشی با کمک الگوریتم بهینه‌سازی بیزین (BOA) ارائه شده‌است که هدفش کشف بن‌بست‌ها و وارسی ویژگی دسترس‌پذیری در سامانه‌های توصیف‌شده به‌وسیلهٔ سامانه تبدیل گراف است. در BOA یک شبکهٔ بیزین از طریق جمعیت، یادگیری انجام‌شده و سپس، نمونه‌سازی می‌شود تا بتواند جواب‌های جدیدی را به‌دست‌آورد[۶۵].

در پژوهش توماس [۶۶]، نویسنده از یک ساختار سلسله‌مراتبی بر اساس یک پروتکل مختصاتی برای الگوسازی یک سامانه هوشمند پیچیده استفاده کرده‌است. برای اثبات برخی ویژگی‌های مطلوب آن پروتکل مانند کامل و کران‌داربودن در این مقاله از سامانه بررسی الگوی نمادین کلارک-مک میلان[۲۲] برپایهٔ انشعاب منطق زمانی بهره برده شده‌است. محاسباتی که از مولکول‌های DNA استفاده می‌کنند، یک الگوی موازی عظیم را معرفی می‌کنند که می‌تواند بر محدودیت‌های موجود در مورد کارایی رایانه‌های سنتی فائق آید. به‌علاوه، بررسی الگو به‌عنوان یک روش رسمی صحت‌سنجی و یک مسئلهٔ پیچیده تاکنون در بسیاری از حوزه‌های محاسباتی کاربرد داشته‌است. ازاین‌رو، ژو و همکاران [۶۷]، از محاسبات DNA برای بررسی الگویی CTL استفاده کرده‌اند. بر طبق نتایج تجربی، بررسی الگو بر پایهٔ محاسبات DNA روش معتبری بوده و می‌تواند در زیست‌شناسی مولکولی به‌کارگرفته‌شود.

۳- پیش‌زمینه

۳-۱- سامانه تبدیل گراف

سامانه تبدیل گراف (GTS) یک زبان رسمی است که حالت‌ها و رفتارهای یک سامانه را از طریق گراف‌ها و تبدیلات میان آن‌ها الگوسازی می‌کند. سامانه تبدیل گراف یک زبان رسمی مبتنی‌بر نمودار برای الگوسازی سامانه‌هایی با ساختارهای پویاست [۶۸]. همچنین، GTS در بسیاری از فعالیت‌های توسعهٔ نرم‌افزار استفاده می‌شود؛ مانند تبدیلِ الگو [۶۹]، طراحی سبک‌های معماری [۷۰]، پالایش [۷۱]، آبَر الگوسازی [۷۲]، بازسازی [۷۳] و تحلیل کارایی [۷۴].

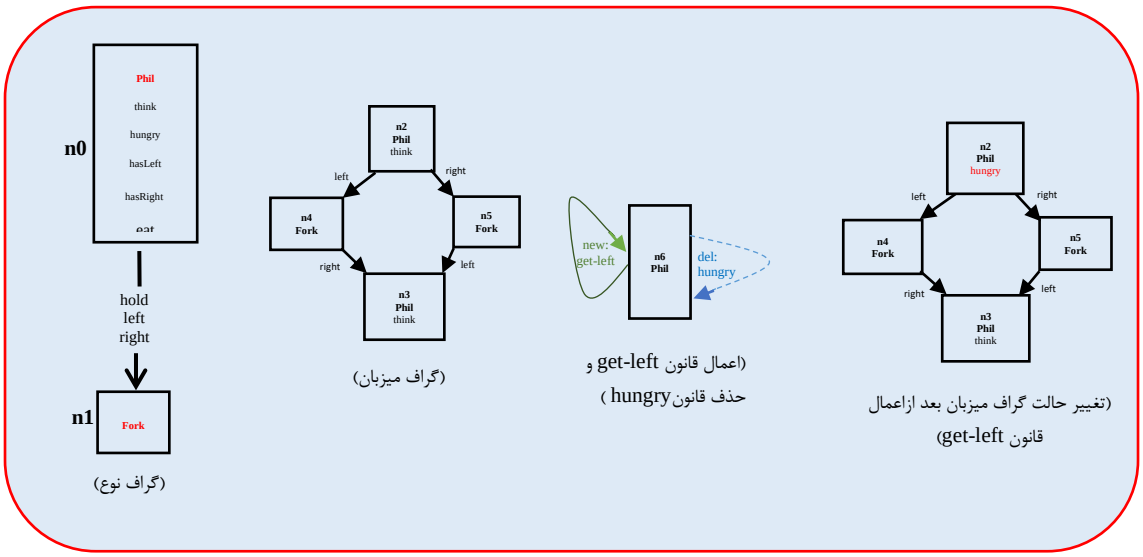
GTS به‌صورت سه‌گانه (TG، HG، R) تعریف می‌شود که TG نمودار نوع[۲۳]، HG نمودار میزبان[۲۴] و R مجموعه‌ای از قوانین تبدیل است [۷۵]. نمودار نوع TG به‌صورت (TGN, TGE, src, trg) مشخص می‌شود که در آن TGN مجموعه‌ای از انواع گره (رئوس) و TGE شامل مجموعه‌ای از لبه‌ها است. src و trg دوتابعی هستند که به هر یال، یک گره مبدأ و یک گره مقصد تخصیص می‌دهد. HG گراف میزبان است و باید نمونه‌ای از گراف نوع باشد. همچنین، گراف میزبان پیکربندی اولیهٔ یک سامانه را نشان می‌دهد. R مجموعه‌ای از قوانین تبدیل است. یک قانون تبدیل گراف R:LHS→RHS از یک نام R و یک جفت گراف نمونه LHS و RHS بر روی گراف نوع TG تشکیل شده‌است. LHS گراف نشان‌دهندهٔ پیش‌شرایط قانون و گراف RHS نشان‌دهندهٔ پس‌شرایط قانون است. همچنین، هر قانون ممکن است شامل یک یا چند NAC (شرایط کاربرد منفی) باشد که اگر این شرایط در گراف میزبان برقرار نباشد، آنگاه این قانون می‌تواند روی گراف میزبان اعمال شود. در یک قانون، گراف‌های LHS، RHS و NAC با هم ادغام شده و رمزگذاری رنگی برای تشخیص هرکدام از اجزای آن به‌کارمی‌رود. برای مثال، یال‌های نازک سیاه‌رنگ، متعلق به گراف‌های LHD و RHS هستند. بعد از اِعمال قانون جدید روی LHS نقطه‌چین آبی‌رنگ متعلق به گراف میزبان باید از آن حذف شوند. درحالی‌که یال سبزرنگ، بعد از اِعمال قانون جدید باید در گراف میزبان

[22] Clarke-McMillan

[23] Type graph
[24] Host graph

[DOI: 10.61186/jsdp.20.2.175]

RHSایجاد شود. در شکل (۱) جزئیاتی از الگوی طراحی‌شده برای مسئلهٔ غذا خوردن فیلسوفان در مجموعه ابزار GROOVE آورده شده‌است.



شکل- ۱: جزئیاتی از الگوی طراحی‌شده برای مسئلهٔ غذاخوردن فیلسوفان در مجموعه ابزار GROOVE

Figure-11: details of the designed model for the dining philosopher's problem in the GROOVE toolset

۲-۳- وارسی الگو

وارسی الگو یک روش خودکار برای تأیید درستی سامانه‌های نرم‌افزاری است. فرآیند وارسی الگو، اغلب در سطح الگو و قبل از پیاده‌سازی انجام می‌شود و قاعدهٔ اصلی به‌کارگیری آن توصیف سامانه و ویژگی‌های مهم آن، با استفاده از یک روش رسمی مانند زبان رسمیِ سامانه تبدیلِ گراف است [۷۶]. در این روش، الگو در قالب یک گراف ارائه می‌شود که حالت‌های گوناگون سامانه را به‌صورت گره دربرمی‌گیرد و جابه‌جایی میان آن‌ها به‌صورت یال و ویژگی‌هایی است که به‌طور معمول، در قالب فرمول‌های منطق زمانی می‌گنجند. وارسی‌کنندهٔ الگو به‌طور خودکار و با کاوش تمام حالت‌ها و سناریوهای شدنی سامانه، مشخص می‌کند آیا ویژگی تعیین‌شده در سامانه موردنظر برآورده می‌شود یا نه. اگر عمل وارسی، موفقیت‌آمیز باشد، (مشکل انفجار فضای حالت رخ ندهد)، یک مثال نقض/ شاهد تولید خواهدشد. مثال‌های نقض/ شاهدها بعضی رفتارهای مطلوب/ نامطلوب سامانه را گزارش می‌کنند و متخصصان می‌توانند از آن برای اصلاح معایب طراحی، بهره گیرند. مهم‌ترین ویژگی‌های قابل‌بررسی در فرآیند وارسی الگو، تأیید ایمنی، دسترس‌پذیری و زنده‌بودن (پیشروی) است. ازآنجاکه تأیید برخی ویژگی‌ها مانند ویژگی ایمنی در یک سامانه، مستلزم بررسی همه حالت‌های سامانه است، تلاش می‌شود این ویژگی اِنکار شود؛ به این مفهوم که یک حالت خطا (مانند بن‌بست) در فضای حالت سامانه پیدا شود. اگر چنین حالتی پیدا شود، آنگاه به مسیری که از حالت ابتدایی فضای حالت شروع‌شده و به یک حالت خطا ختم می‌شود، مثال نقض می‌گویند. ویژگی دسترس‌پذیری ادعا می‌کند که حالتی در فضای حالت وجود دارد که در آن حالت، ویژگی داده‌شده برقرار است (حالت هدف). اگر چنین حالتی پیدا شود، آنگاه به مسیری که از حالت ابتدایی فضای حالت شروع شده و به حالت هدف ختم می‌شود، شاهد می‌گویند. با وجود مزایای فراوان وارسی الگو، این روش نقاط ضعفی نیز دارد که به‌طور قطع انفجار فضای حالت، بزرگ‌ترین مشکل آن است.

۴- روش پیشنهادی

۴-۱- فاز اول

۴-۱-۱- الگویی از سامانه موردنظر را ایجاد می‌کنیم.

۴-۱-۲-گراف فضای حالت الگوی سامانه را ایجاد می‌کنیم.

۴-۱-۳- بخش کوچکی از گراف فضای حالت را به روش BFS پیمایش می‌کنیم. جستجوی BFS در الگوریتم (۱) آورده شده‌است.

[DOI: 10.61186/jsdp.20.2.175]

Algorithm 1: The BFS algorithm in GROOVE

```
1)  Input: M: a model described by GTS.
2)  Output: S: the state space of M.
3)  GraphState state = the initial state of M;
4)  LinkedList<GraphState> stateQueue=new LinkedList<GraphState> ();
5)  stateQueue.enqueue (state);
6)  S.nodes.add (state);
7)  while stateQueue.size () > 0 do
8)     state = stateQueue.dequeue ();
9)     for each  MatchResult match in state.getMatches () do
10)          GraphState next = state.applyMatch(match);
11)          if next != null then
12)                                   if !S.nodes.contains (next) then
13)                                        stateQueue.enqueue (next);
14)                                             S.nodes.add (state);
15)                                   end if
16)                                   S.edges.add (new Transition (state,match,next));
17)                              end if
18)               end for
19)  end while
20)  return S;
```

Algorithm 2: Generate the conditional probability table

```
1)   Input: training set, p: a set of all Rules.
2)   Output: conditional probability table.
3)   Table: conditional probability table.
4)   for i=1 to length of p do
5)          for j=1 length of p do
6)                Calculate the probability p(X1=ri | X0=rj) for each pair ri and rj in the
                  training set and set in the conditional probability table.
7)          end for;
8)   end for;
```

۴- ۲- فاز دوم

۴- ۲- ۱- یک مجموعهٔ آموزشی از دنباله‌های تولیدشده تشکیل می‌دهیم. دنبالهٔ قوانین هر مسیر به‌صورت یک تاپل مجزا در مجموعهٔ آموزشی قرار می‌گیرد. باید توجه داشت که ترتیب به‌کارگیری قوانین در پیمایش فضای حالت مهم هستند. درنتیجه، باید ترتیب نسبی قوانینِ موجود در مسیرها حفظ شوند.

۴-۲-۲- تمامی مسیرهایی که از حالت اولیه شروع و به حالت هدف ختم می‌شوند، استخراج می‌شوند. اگر هر مسیر را به‌صورت $s_0 r_0 s_1 r_1 \dots s_k r_k$ در نظر بگیریم، s_0 حالت اولیه و s_k حالت نهایی (حالت هدف) را نشان می‌دهند. ازآنجاکه حالت‌های روی مسیرهای موجود نیاز نیستند، نادیده گرفته‌می‌شوند. پس از حذف حالت‌ها از هر مسیر، دنباله‌ای از قوانین به‌صورت $r_0 r_1 \dots r_k$ حاصل می‌شود.

۴-۲-۳ وابستگی نسبی بین قوانین موجود در مجموعهٔ آموزشی استخراج و در جدول احتمالات شرطی قرار داده می‌شوند. شبه‌رمز تولید جدول احتمالات شرطی در الگوریتم (۲) آورده شده‌است.

۴-۳ - فاز سوم

۴-۳-۱ بقیهٔ فضای حالت الگوی بزرگ را سطح مشخصی (تا سطحی که حافظهٔ سامانه قادر به ذخیره‌سازی باشد.) پیمایش و مطابق ۴-۲-۱ پس از استخراج دنبالهٔ قوانین هر مسیر، در مجموعهٔ آزمایشی قرار می‌گیرند.

[DOI: 10.61186/jsdp.20.2.175]

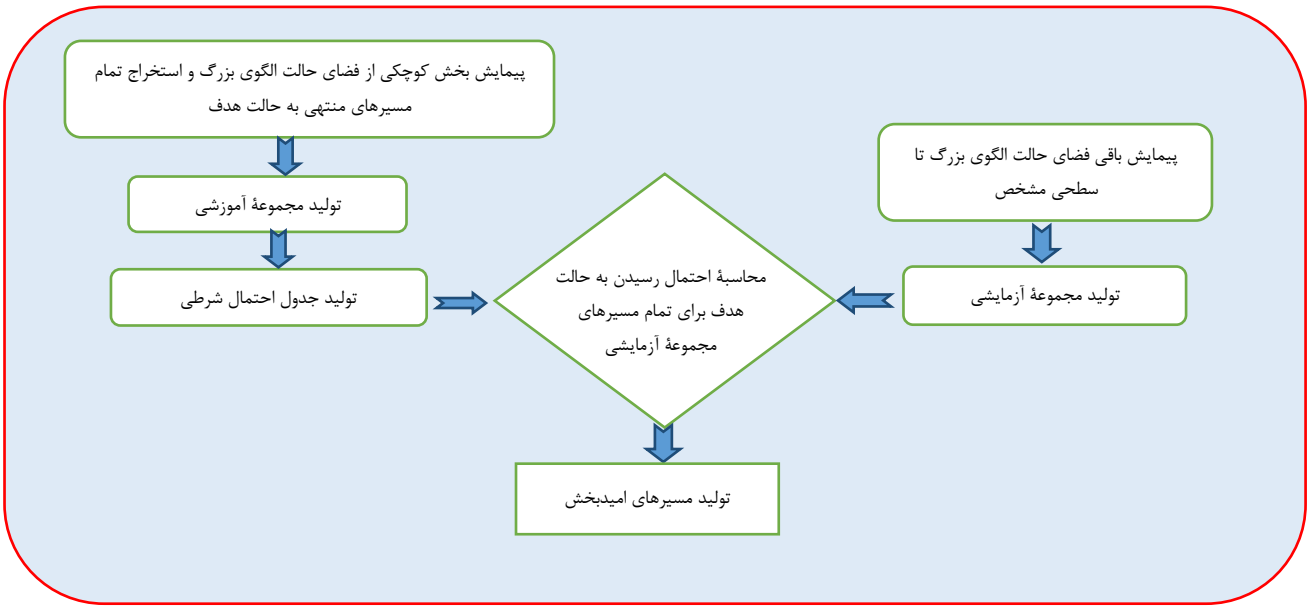
۴-۳-۲ با کمک جدول شرطی، احتمال رسیدن هر مسیر موجود در مجموعه آزمایشی ارزیابی و تعیین می‌شود. مسیرهای با احتمال بالاتر، به‌عنوان مسیرهای امیدبخش در نظر گرفته می‌شوند.

۴-۳-۳ مسیرهای امیدبخش در باقی فضای حالت الگوی بزرگ پیمایش می‌شوند. شبه‌رمز پیمایش مسیرهای امیدبخش در الگوریتم (۳) آورده شده‌است.

Algorithm 3: Intelligent checking of the large model.

1) ***Input:*** *test set, large model, conditional probability table.*
2) ***Output:*** *a witness for reachabiliy property*
3) ***foreach*** *path in the test set* ***do***
4) *Calculate the probability the path using conditional probability table.*
5) ***end for***
6) *Sort all paths of the test set according to the probability value.*
7) ***foreach*** *path in test set* ***do***
8) *apply path on the large model.*
9) ***If*** *current state is a goal state* ***then***
10) *return the path starting from initial state and leading to the current state as a witness*
11) ***else***
12) *path=next path in test set;*
13) ***end if***
14) ***end for***

نمودار روش پیشنهادی در شکل (۲) نشان داده شده‌است.



شکل- ۲: دیاگرام روش پیشنهادی

Figure-2: Diagram of the proposed approach

۴-۴ -تولید جدول احتمالات شرطی

جدول شرطی از مجموعهٔ آموزشی ایجاد می‌شود. در هر دنباله قوانین، قانونی که باید روی حالت فعلی اعمال شود، از قانون اعمال‌شده روی حالت پیشین، متأثر می‌شود؛ بنابراین، ساختار جدول شرطی به‌صورت یک زنجیر با دو گره X0 و X1 در نظر گرفته‌می‌شود؛ به‌طوری‌که گره X1، قوانین به‌کاررفته روی حالت‌های فعلی را نشان می‌دهد و گره X0 قوانین به‌کاررفته روی حالت‌های پیشین را نشان می‌دهد. جدول احتمال شرطی، توزیع احتمال شرطی $p(X1=rj|X0=ri)$ برای همه قوانین ri و rj را نشان می‌دهد.

برای محاسبهٔ توزیع‌های احتمال، مقدار $p(X0=ri)$ برابر با حاصل تقسیم تعداد تکرار قانون ri بر تعداد کل تکرار تمام قوانین در همهٔ مسیرهای استخراج‌شده خواهدبود. همچنین، مقدار $p(X1=rj|X0=ri)$ از تقسیم تعداد

[DOI: 10.61186/jsdp.20.2.175]

تکرار قوانین ri و rj به‌صورت پشت‌سرهم بر تعداد تکرار قانون ri محاسبه می‌شود.

بنابراین، ساختار موردنظر دارای دو گره (متغیر) X0 و X1 خواهدبود که مقادیر هرکدام، از مجموعه قوانین گرفته می‌شوند. نماد "#" در فرمول‌های زیر به معنای «تعداد» است. احتمال شرطی گره X1 با والد X0 با فرمول (۱) محاسبه می‌شود.

$$p(X1 = ri|X0 = rj) = \frac{p(X1=ri,X0=rj)}{p(X0=rj)} = \frac{\#(c1=r1,c0=rj)}{\#(c0=rj)} \quad i,j = 0,1,2 \quad (1)$$

۴- ۵ - مثال جامع

سامانه فرضی را با ۳ سه قانون r0، r1 و r2 در نظر می‌گیریم. ساختار جدول شرطی به‌صورت یک زنجیر با اندازهٔ برابر با دو فرض می‌کنیم. در این زنجیر، هر گره فقط به گره پیشین خود وابسته است. همچنین، فرض می‌کنیم که تعداد مسیرهای منتهی به حالت هدف در الگوی سامانه مفروض برابر با پنج و به‌صورت "c0c1c2c3" و ci={r0,r1,r2} هستند.

فرض کنید همهٔ دنباله قوانین منتهی به حالت هدف در الگو به‌صورت {r0r1r1r2,r1r0r2r1,r1r2r0r1,r0r1r2r0} باشند. احتمالات شرطی گره‌ها با استفاده از فرمول (۱) به‌صورت زیر محاسبه می‌شود:

$$p(X1 = r0|X0 = r0) = \frac{0}{6} = 0, \quad p(X1 = r1|X0 = r0) = \frac{4}{6} = 0.66, \quad p(X1 = r2|X0 = r0) = \frac{1}{6} = 0.16$$

$$p(X1 = r0|X0 = r1) = \frac{1}{8} = 0.125, \quad p(X1 = r1|X0 = r1) = \frac{1}{8} = 0.125, \quad p(X1 = r2|X0 = r1) = \frac{3}{8} = 0.375$$

$$p(X1 = r0|X0 = r2) = \frac{3}{6} = 0.5, \quad p(X1 = r1|X0 = r2) = \frac{1}{6} = 0.16, \quad p(X1 = r2|X0 = r2) = \frac{1}{6} = 0.16$$

احتمالات به‌دست‌آمده می‌تواند برای پیمایش باقی فضای حالت الگو استفاده شود. درواقع، رویکرد پیشنهادی به‌طور تقریبی تعیین می‌کند چه قانونی باید روی حالت فعلی به‌کارگرفته‌شود تا حالت هدف زودتر پیدا شود. فرض کنید دنبالهٔ قوانین متناظر با تعدادی از مسیرها در الگوی بزرگ به‌صورت {p1=r0r1r2r0r1,p2=r0r2r2r1r0,p3=r1r1r2r1r0, p4=r1r2r0r1r2} باشند، بر اساس مقادیر جدول احتمال شرطی، احتمال رسیدن هر مسیر به‌صورت زیر محاسبه می‌شود.

احتمال رسیدن مسیر p1 به هدف = 0.66×0.375×0.5×0.66=0.0816

احتمال رسیدن مسیر p2 به هدف = 0.16×0.16×0.16×0.125=0.0005

احتمال رسیدن مسیر p3 به هدف = 0.125×0.375×0.16×0.125=0.0009

احتمال رسیدن مسیر p4 به هدف = 0.375×0.5×0.66×0.375=0.0464

مسیرهای متناظر با دنبالهٔ قوانین p1، p4، p3 و p2 به ترتیب دارای بیشترین شانس رسیدن به حالت هدف هستند. مسیرهای با احتمال بالاتر به‌عنوان مسیرهای امیدبخش درنظرگرفته‌می‌شوند.

۵- نتایج تجربی

برای ارزیابی راه‌حل ارائه‌شده و مقایسهٔ آن با روش‌های دیگر، آن را با زبان برنامه‌نویسی جاوا و در ابزار متن‌باز GROOVE پیاده‌سازی کرده‌ایم. به این منظور، تعدادی از رده‌های موجود از جمله Simulator، Exploration و Closing Strategy را تغییر داده، همچنین، تعدادی ردهٔ جدید اضافه کرده‌ایم.

راه‌حل ارائه‌شده به‌همراه الگوریتم‌های فرامکاشفه‌ای و تکاملی از جمله ژنتیک (GA)[۷۷]، پرندگان (PSO)[۱۵]، ترکیب پرندگان و جستجوی گرانشی (PSO-GSA)[۱۵] و الگوریتم‌های فرامکاشفه‌ای، از جمله BFS، DFS و *IDA[۷۸] و جستجوی پرتوی (BS)[۳۶] و الگوریتم بهینه‌سازی بیزین (BOActp) [۶۴] و الگوریتم‌های مبتنی‌بر یادگیری نظارت‌شده از جمله Ada boost [۱۷] و Random forest روی چندین مسئلهٔ شناخته‌شده که وارسی الگوهای بزرگی از آن‌ها، به دلیل بزرگ‌بودن فضای حالت الگو، امکان‌پذیر نیست، اجرا کرده و سپس، نتایج آن‌ها در بخش‌های مرتبط نشان خواهیم‌داد. برای داشتن یک مقایسهٔ منصفانه، همه روش‌ها در ابزار GROOVE پیاده‌سازی شده‌اند؛ زیرا ابزارهای مختلف ساختارهای متفاوتی دارند

[DOI: 10.61186/jsdp.20.2.175]

که عملکرد را تحت‌تأثیر قرار می‌دهند. ازآنجاکه الگوریتم‌های DFS و BFS در الگوهای بزرگ با مشکل انفجار فضای حالت روبرو می‌شوند، مقداری برای آن‌ها در جداول نتایج نشان داده‌نمی‌شود. همچنین، چنانچه رویکردی پس از مصرف همهٔ حافظه موجود یا رسیدن به بیشنیهٔ تعداد تکرار نتواند هیچ حالت هدف را پیدا کند، در جدول‌های نتایج از اصطلاح "Not found" استفاده می‌شود. همهٔ نتایج بر روی سیستمی با پردازندهٔ Intel Core i7 (۲.۲ گیگاهرتز) و سه گیگابایت حافظهٔ RAM تولید و در جدول‌ها گزارش شده‌است.

شاخص‌های اولیه همراه با مقادیر مناسب آن‌ها برای اجرای همهٔ رویکردها در جدول (۱) نشان داده شده‌است.

جدول-۱: مقادیر اولیهٔ شاخص‌ها

Table-1: The initial values of parameters

Benchmark		Value	GA		PSO	
Beam width	Dining philosophers, process life cycle	10	Crossover rate	50%	C1	2
	Travel agency	30	Mutation rate	30%	C2	2
	Electronic control Units	60	Position of crossover	Middle of chromosomes	W	0.8
	Shopping	40				

۵-۱- مسائل موردبررسی

مسائل استفاده‌شده در مقاله شامل الگوهایی با اندازه‌های مختلف از مسئلهٔ غذاخوردن فیلسوفان [۷۹]، مسئلهٔ چرخهٔ حیات پردازه [۸۰]، مسئلهٔ آژانس مسافرتی [۸۱]، مسئلهٔ خرید [۸۲] و مسئلهٔ واحدهای کنترل الکترونیکی [۸۳] است.

۵-۱-۱ مسئلهٔ غذاخوری فیلسوفان:

در این مسئله، چندین فیلسوف دور یک میز نشسته‌اند و بین هردوی آن‌ها یک چنگال قرار دارد. هر فیلسوف می‌تواند در یکی از سه حالتِ فکرکردن، گرسنگی و خوردن باشد. همهٔ فیلسوفان نخست، در حالت فکرکردن هستند و می‌توانند بعد از مدتی احساس گرسنگی کرده و به حالت گرسنگی بروند. هر فیلسوف گرسنه می‌تواند چنگال چپ و راستش را در صورت آزادبودن، برداشته و به حالت خوردن وارد شود. فیلسوفی که در حال خوردن است، بعد از مدتی چنگال‌های در دسترس را رها کرده و دوباره به حالت فکرکردن می‌رود. این روند به تعداد نامحدودی تکرار می‌شود.

۵-۱-۲ مسألهٔ چرخهٔ حیات فرایند:

این مسئله، چرخهٔ زندگی یک فرایند را از مرحلهٔ ایجاد تا اتمام آن در یک سامانه‌عامل الگوسازی می‌کند. اگر حافظهٔ کافی در دسترس باشد، I/O یا CPU پردازهٔ ایجادشده به حافظه بارگذاری شده و منتظر وسایل می‌ماند. بعد از اجرای پردازه و اتمام آن، تمام منابعِ در دسترس آزاد می‌شود.

۵- ۱- ۳ مسئلهٔ آژانس مسافرتی:

این مسئله، سامانه‌ی را توصیف می‌کند که برای مشتریان خود خدماتی از قبیل رزرو پرواز و هتل ارائه می‌کند. در این سامانه تعدادی عامل هوشمند وجود دارند که درخواست مشتریان را دریافت می‌کنند و سپس، با توجه به محدودیت‌های مالی و شرایط سفر، راه‌حل‌هایی پیشنهاد می‌دهند. قابل‌ذکر است که با توجه به ویژگی دسترس‌پذیری در این مسئله، قوانین غیرضروری را در نظر نگرفته‌ایم.

۵-۱-۳ مسئلهٔ خرید:

این مسئله، فرایند خرید توسط مشتری‌ها را در یک فروشگاه توصیف می‌کند. در الگوهای مختلف این مسئله، زمانی که همهٔ مشتری‌ها خریدهایشان را تمام کرده‌باشند، فرایند خرید به اتمام می‌رسد.

۵-۱-۴ واحدهای کنترل الکترونیکی (ECU):

این مسئله مربوط به پیکربندی مجدد واحدهای کنترل الکترونیکی در سامانه‌های اتومبیل است. درواقع، این مسئلهٔ پیکربندی اجزای نرم‌افزار در حال اجرا بر روی میان‌افزار RTE را توضیح می‌دهد. هدف از این مسئله یافتن حالتی است که در آن برخی از سخت‌افزارها خراب

[DOI: 10.61186/jsdp.20.2.175]

شوند؛ مانند خاموش‌کردن ECU یا راه‌اندازی مجدد RTEپس از به‌روزرسانی نرم‌افزار.

الگوی همهٔ مسائل بالا با استفاده از GTS و در ابزار GROOVE الگوسازی شده‌اند. در جدول (۲) جزئیات هر مسئله از قبیل تعداد قوانین، تعداد گره‌ها و لبه‌ها برای هر نمونه الگو نمایش داده شده‌است.

جدول- ۲: جزئیات مسائل در نظر گرفته‌شده

Table- 2: Details of the considered problems

مسئله	تعداد قوانین	مدل	تعداد گره‌ها	تعداد یال‌ها
غذاخوردن فیلسوف‌ها	۵	۳۰ فیلسوف	۶۰	۶۰
چرخهٔ حیات پردازه‌ها	۸	۱۵ پردازه و ۱۵ قطعه حافظه	۳۲	۱۵
آژانس مسافرتی	۱۴	آژانس با ۴ مشتری	۱۳	۷
خرید	۸	۱۰ مشتری و ۲۰ کالا	۴۱	۴۲
واحد کنترل الکترونیکی	۴	ECU-4	۱۲	۱۲

۵-۲- ارزیابی روش پیشنهادی برای تأیید ویژگی دسترس‌پذیری EFq

به‌منظور ارزیابی اثربخشی رویکرد پیشنهادی در بررسی ویژگی دسترس‌پذیری EFq، در ادامه، خصوصیت حالت q را به‌عنوان حالت هدف و به این صورت درنظرمی‌گیریم: در مسئلهٔ غذاخوردن فیلسوفان «همهٔ فیلسوفان گرسنه هستند.»، در مسئلهٔ چرخهٔ حیات فرایند «همهٔ فرایندها اجرای خود را به پایان رسانیده‌اند.»، در مسئلهٔ خرید «یک مشتری نیمی از کالاهای موجود را خریداری کرده‌است.»، در مسئلهٔ آژانس مسافرتی «نیمی از مشتریان موجود پروازها و هتل‌های خود را رزرو کرده‌اند.» و در مسئلهٔ واحدهای کنترل الکترونیکی «نیمی از ECUهای موجود باید خاموش شوند و تمام اجزای نرم‌افزاری باید وجود داشته باشند».

مقایسهٔ عملکرد روش پیشنهادی با دیگر روش‌ها در مسئلهٔ غذاخوردن فیلسوفان در جدول (۳) نشان داده شده‌است. بر اساس نتایج گزارش‌شده، کارایی این روش در الگوهای با اندازه متوسط (بالاتر از چهل فیلسوف) از نظر متوسط زمان اجرا خیلی بهتر از دیگر روش‌ها به‌جز روش‌های BFA و BS است. اگرچه الگوریتم‌های BFA و BS در الگوهای با اندازهٔ متوسط قادر به یافتن حالت هدف هستند؛ در الگوهای بسیار بزرگ با مشکل انفجار فضای حالت روبه‌رو می‌شوند. همان‌طورکه نشان داده‌شده، رویکرد پیشنهادی حتی در الگوهای بسیار بزرگ مسئله (به‌عنوان‌مثال، هشتاد فیلسوف) می‌تواند حالت هدف را پیدا کند.

در این مسئله فقط یک حالت هدف وجود دارد و در الگوهای بزرگ، در سطوح عمیق‌تر فضای حالت یافت می‌شود. BFS فضای حالت گسترده‌ای را جستجو می‌کند و باعث مصرف تمام فضای حافظهٔ موجود قبل از کاوش بخش‌های عمیق‌تر می‌شود. DFS و *IDA برخلاف BFS، فضای حالت را به‌طور عمیق جستجو می‌کنند و به احتمال زیاد تا هنگامی‌که به‌طور تقریبی، تمام فضای حالت را کاوش نکنند، قادر به شناسایی و کشف حالت هدف نیستند. ازاین‌رو، این رویکردها نمی‌توانند حالت هدف را تشخیص دهند. ازآنجاکه جستجوی پرتو، فضای جستجو را در عمق و عرض کاوش می‌کند، به نظر می‌رسد شانس بالایی برای پیداکردن حالت هدف داشته‌باشد. در الگوهای بزرگ این مسئله، فضای جستجو در بخش‌های عمیق‌تر بسیار گسترده است؛ بنابراین، برای افزایش شانس شناسایی حالت هدف، باید مقدار عرض پرتو افزایش یابد. با افزایش عرض پرتو، کارایی این روش، مشابه روش BFS خواهدبود. این بدان معنا است که روش BS در الگوهای بزرگ مسئله با شکست مواجه خواهدشد.

رویکرد پیشنهادی را برای تشخیص حالت هدف در مسائل چرخهٔ حیات فرایند، آژانس مسافرتی و واحدهای کنترل الکترونیکی به‌کارمی‌گیریم. جدول‌های (۴) تا (۷) مقایسهٔ نتایج روش پیشنهادی با روش‌های دیگر را نشان می‌دهند. همان‌طورکه در جدول‌ها نشان داده شده‌است، همهٔ روش‌ها به‌جز روش پیشنهادی با مشکل انفجار فضای حالت مواجه می‌شوند؛ اما روش پیشنهادی در الگوهای بسیار بزرگ حالت هدف را تشخیص می‌دهد. به دلایل ذکرشده جستجوی پرتو، DFS و BFS نمی‌توانند هیچ حالت هدفی را در الگوهای بزرگ این مسائل پیدا کنند.

[DOI: 10.61186/jsdp.20.2.175]

جدول- ۳: مقایسهٔ زمان اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسئلهٔ غذاخوردن فیلسوفان

Table- 3: Comparison of running times of all approaches to solving the dining philosopher's problem to verify the reachability property

روش / مدل	۲۰ فیلسوف (ثانیه)	۲۵ فیلسوف (ثانیه)	۳۰ فیلسوف (ثانیه)
IDA*	Not found		
BS	۱۱۶.۶۳	۳۴۲.۵۴	۸۷۳.۲۳
GA	۱۲.۳۸	۲۷.۳۱	۷۵.۶۲
PSO	۶۲.۳۲	۸۷.۶۴	۱۲۳.۷۹
PSO-GSA	۵۵.۲۸	۸۴.۸۲	۱۰۲.۹۴
Model checking+ AdaBoost	۵.۷	۶.۸۷	۷.۰۸
Model checking+ Random forest	۵.۵۶	۹.۱۸	۱۰.۱۵
BOActp	۲۰.۴۵	۳۶.۹۷	۵۷.۳۸
Proposed method	۴.۵۶	۶.۱۲	۶.۹۵

جدول- ۴: مقایسه زمان اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسئلهٔ چرخهٔ حیات فرایند

Tab-le 4: Comparison of running times of all approaches to solving the Process life cycle problem to verify the reachability property

روش / مدل	۱۰ فرایند- ۱۰ قطعه حافظه (ثانیه)	۱۲ فرایند- ۱۲ قطعه حافظه (ثانیه)	۱۵ فرایند- ۱۵ قطعه حافظه (ثانیه)
IDA*	Not found		
BS	Not found		
GA	۷.۴۹	۱۴.۲۵	Not found
PSO	۶.۱۷	۲۶.۴۳	Not found
PSO-GSA	۴۶.۳۲	۳۱۲.۳۴	Not found
Model checking+ AdaBoost	۱۱.۳۵	۱۷.۷	۲۵.۶۸
Model checking+ Random forest	۱۳.۵۸	۲۰.۱۲	۲۸.۳۱
BOActp	۷.۹۹	۱۰.۸۴	Not found
Proposed method	۱۰.۴۶	۱۵.۷۲	۲۲.۹

جدول- ۵: مقایسه زمان اجرای همهٔ روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسئلهٔ آژانس مسافرتی

Tab-le 5: Comparison of running times of all approaches to solving the travel agency problem to verify the reachability property

روش / مدل	آژانس با چهار مشتری (ثانیه)	آژانس با شش مشتری (ثانیه)	آژانس با هشت مشتری (ثانیه)
IDA*	۲۴.۸۵	Not found	Not found
BS	۱۲.۴۳	Not found	Not found
GA	۵.۷۶	۴۳.۱۱	۶۵.۳۱
PSO	۶.۶۹	۴۱.۵۲	Not found
PSO-GSA	۶.۱۲	۴۱.۵۲	Not found
Model checking+ AdaBoost	۳.۹	۴.۸۳	۱۰.۸۴
Model checking+ Random forest	۴.۷۱	۶.۱۱	۹.۱۷
BOActp(2019)	۴.۱۸	۷.۱۳	۱۰.۴۷
Proposed method	۳.۴	۵.۴	۷.۸۹

[DOI: 10.61186/jsdp.20.2.175]

جدول- ۶: مقایسهٔ زمان اجرای همهٔ روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسئلهٔ خرید

Table- 6: Comparison of running times of all approaches to solving the shopping problem to verify the reachability property

روش / مدل	۶ مشتری- ۱۶ کالا (ثانیه)	۸ مشتری- ۱۸ کالا (ثانیه)	۱۰ مشتری- ۲۰ کالا (ثانیه)
IDA*	۱.۵۴	۳.۷۶	۱۰.۲۴
BS	Not found		
GA	۳.۲۳	۵.۰۶	۴۰.۴۶
PSO	۲.۶۹	۴.۸۱	۳۷.۴۳
PSO-GSA	۲.۴۳	۳.۷۶	۳۵.۳۱
Model checking_+AdaBoost	۲.۵۸	۵.۷۶	۷.۱۷
Model checking+ Random forest	۵.۷۳	۶.۵۷	۸.۹۳
BOActp	۳.۳۵	۴.۹۱	۱۷.۷۲
Proposed method	۲.۱۶	۵.۰۹	۶.۹۳

جدول- ۷: مقایسهٔ زمان اجرای روش‌های مختلف برای تأیید ویژگی دسترس‌پذیری در مسئلهٔ واحدهای کنترل الکترونیکی

Table- 7: Comparison of running times of all approaches to solving the ECU problem to verify the reachability property

روش / مدل	چهار واحد الکترونیکی (ثانیه)	شش واحد الکترونیکی (ثانیه)	هشت واحد الکترونیکی (ثانیه)
IDA*	۲۵۱	Not found	Not found
BS	۵.۶۹	۱۱۴۴	Not found
GA	۴.۸۲	۱۶۳.۹۲	Not found
PSO	۴.۷۲	۱۸۴.۷۳	Not found
PSO-GSA	۴.۸۳	۱۹۳.۵۱	Not found
Model checking+ AdaBoost	۴.۱	۵.۴۶	۲۷.۲۵
Model checking+ Random forest	۴.۱۲	۷.۸۱	۳۷.۶۳
BOActp	۳.۶۵	۵۶.۸۲	۳۱.۴۱
Proposed method	۳.۵۹	۵.۲۵	۲۵.۰۳

۶- بحث

مزایای رویکرد پیشنهادی در مقایسه با روش‌های دیگر را می‌توان به شرح زیر خلاصه کرد:

- سرعت اجرای بالاتر: نتایج ارائه‌شده در بخش ۵. ۲ نشان می‌دهند روش پیشنهادی، سرعت اجرای بالاتری بر روی مسائل موردبررسی در مقایسه با روش‌های دیگر دارد.
- کاوش تعداد کمتری از حالت‌ها در فضای حالت مسئله: رویکرد پیشنهادی تعداد حالات کمتری برای رسیدن به یک حالت هدف در فضای حالت اغلب مسائل بررسی‌شده جستجو می‌کند. به‌عنوان‌مثال، جدول (۸) تعداد حالت-های کاوش‌شده به‌وسیلهٔ همهٔ روش‌ها در یک الگوی نمونه از مسائل مختلف برای تأیید ویژگی دسترس‌پذیری را نشان می‌دهد. همان‌طورکه در جدول (۸) نشان داده شده‌است، روش پیشنهادی نیاز به کاوش تعداد کمتری حالت دارد.

[DOI: 10.61186/jsdp.20.2.175]

جدول- ۸: مقایسهٔ تعداد حالات کاوش‌شده توسط همهٔ روش‌ها برای تأیید ویژگی دسترس‌پذیری

Table- 8: Comparison of the number of the explored states by all approaches to verify the reachability property

روش / مدل	GA	PSO	PSO-GSA	BS	IDA*	BOActp	Model checking+ AdaBoost	Model checking+ Random forest	Proposed method
Dining philosophers (۳۰ فیلسوف)	۳۴۵۴۲	۴۳۲۱۲	۳۹۸۵۴	۸۶۷۰	Not found	۲۹۶۸۵	۲۸۲۴	۲۶۷۹	۲۱۳۵
Process life cycle (۲۰ فرایند - ۸ حافظه)	۶۵۴۳	۲۶۵۴۳	۴۳۲۳۴	۲۱۱۷۵	۱۲۱۷	۱۱۴۱	۱۵۳۲	۱۳۸۰	۱۲۰۹
Shopping (۲۰ مشتری- ۳۰ کالا)	۷۸۱۹	۳۴۳۲	۲۲۶۴	Not found	۴۹۸	Not found	۱۴۹۴	۱۱۰۰	۴۵۶
Travel agency (۴ مشتری)	۱۵۸	۱۳۰	۱۱۹	Not found	Not found	۳۵۸	۸۴	۷۳	۶۹
ECU-4	۱۷۴	۱۲۸	۱۱۵	۹۵	۸۶	۹۰	۱۰۸	۸۱	۷۵

- تولید مثال‌های نقص با طول کوتاه‌تر: مثال‌های نقص/ شاهدها بعضی رفتارهای مطلوب/ نامطلوب سامانه را نشان می‌دهند و می‌توانند توسط متخصصان برای اصلاح معایبِ طراحی، استفاده شوند. ازاین‌رو، تولید مثال نقص/ شاهد کوتاه‌تر می‌تواند معیار مهمی در ارزیابی اثربخشی روش‌ها باشد. روش پیشنهادی، مثال نقص/ شاهد را با طول کوتاه‌تری در اغلب مسائل بررسی‌شده تولید می‌کند. به‌عنوان‌مثال، جدول (۹) طول شاهد در روش‌های مختلف را نشان می‌دهد. همان‌طورکه در جدول نشان داده شده‌است، روش پیشنهادی، شاهدهای با طول کوتاه‌تری در مقایسه با دیگر روش‌ها تولید می‌کند.

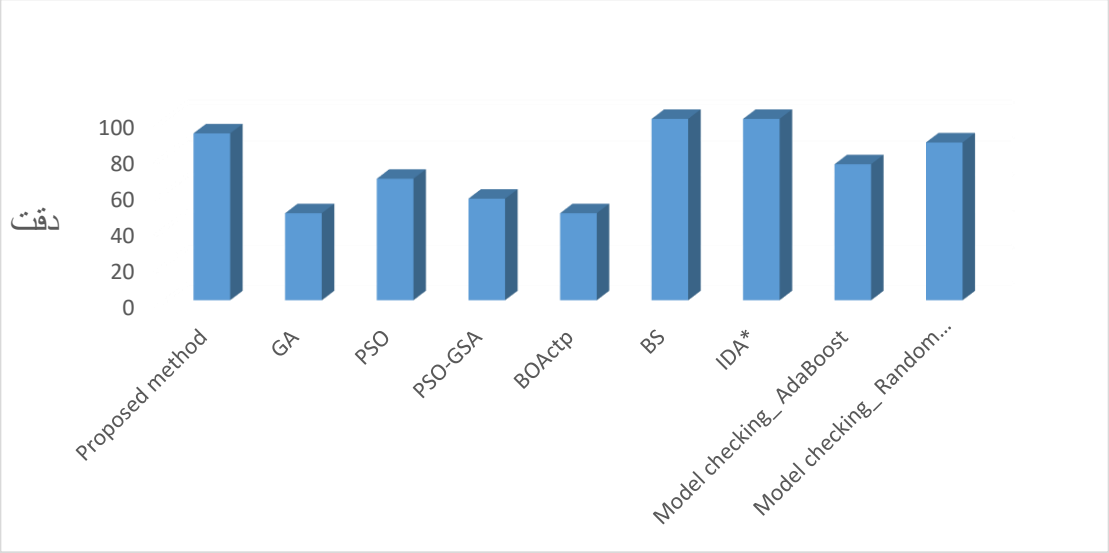
جدول- ۹: مقایسه طول شاهد تولیدشده توسط همهٔ روش‌ها در تأیید ویژگی دسترس‌پذیری

Table- 9: Comparison of the length of the generated witnesses by all approaches to verify the reachability property

روش / مدل	GA	PSO	PSO-GSA	IDA*	BS	Model checking+ AdaBoost	BOActp	Model checking+ Random forest	Proposed method
Dining philosophers (۳۰ فیلسوف)	۹۱	۱۱۵	۹۸	Not found	۷۵	۶۴	۷۰	۶۶	۶۰
Process life cycle (۱۰ پردازه- ۱۰ حافظه)	۳۸،۲۲	۳۹،۳۶	۳۸،۲	۳۰	Not found	۱۷،۵۴	۴۱،۳۹	۱۲	۱۱
Shopping (۲۰ مشتری- ۳۰ کالا)	۱۷۰	۱۶۸	۱۷۳	۱۰۰		۱۵۶	۱۴۳	۱۲۵	۱۱۰
Travel agency (۴ مشتری)	۲۱	۲۳	۱۹	Not found		۲۱	۲۰	۱۸	۱۶
ECU-4	۲۵	۲۶	۲۵	۲۵	۲۴	۲۴	۲۶	۲۷	۲۲

[DOI: 10.61186/jsdp.20.2.175]

- دقت بالاتر: روش پیشنهادی در مقایسه با روش‌های دیگر از دقت بالاتری برخوردار است. در این پژوهش، هدف از دقت، نسبت تعداد اجراهای موفق به کل اجراهاست (همچنین، hit Rate گفته‌می‌شود). به‌منظور مقایسهٔ دقت روش پیشنهادی با دقت روش‌های دیگر، یک الگوی نمونه از هر مسئله در نظر می‌گیریم، به‌طوری‌که همهٔ روش‌ها بتوانند دست‌کم یک بار به حالت هدف برسند. شکل (۳) دقت همهٔ روش‌ها در تأیید ویژگی دسترس‌پذیری را نشان می‌دهد. در یک الگوی داده‌شده، BS، BFA و IDA* همواره به یک حالت هدف می‌رسند. ازاین‌رو، در الگوهای درنظرگرفته‌شده، دقت آن‌ها ۱۰۰٪ خواهد بود. همچنین، دقت روش پیشنهادی بسیار بالاتر از دقت رویکردهای PSO، GA، PSO-GSA و BAPSO است.



شکل- ۳: مقایسهٔ دقت روش‌ها برای تأیید ویژگی دسترس‌پذیری

Figure- 3: Comparison of the accuracy by all approaches to verify the reachability property

۷- نتیجه‌گیری و کارهای آینده

وارسی الگو روشی شناخته‌شده و به‌طور کامل، خودکار در تأیید رسمی سامانه‌های نرم‌افزاری است. بااین‌حال، استفاده از این روش برای تأیید ویژگی‌های داده‌شده ایجاب می‌کند که تمام حالت‌های ممکن سامانه ایجاد شود. در سامانه‌های بزرگ، ممکن است بررسی خصوصیات سامانه به‌وسیلهٔ روش‌های کلاسیک موجود، امکان‌پذیر نباشد؛ زیرا به دلیل نیاز به کاوش تمام فضای حالت، ممکن است منجر به انفجار فضای حالت شود. در این پژوهش، یک رویکرد مبتنی‌بر کشف وابستگی قوانین به کمک جدول احتمالات شرطی ارائه شده‌است. این رویکرد، برای جلوگیری از انفجار فضای حالت به‌منظور بررسی ویژگی دسترس‌پذیری در سامانه‌های الگوشده با روش رسمی GTS استفاده می‌شود. روش پیشنهادی، برای کشف مسیرهای امیدبخش که درنهایت منجر به شناسایی حالت هدف می‌شوند، طراحی شده‌است.

برای تأیید ویژگی ایمنی باید تمام فضای حالت سامانه موردنظر را ایجاد و سپس، با بررسی تمام حالاتِ ممکن، ویژگی ایمنی را تأیید کرد. اما همان‌طورکه پیشتر گفته‌شد، تولیدِ تمامِ فضایِ حالت در سامانه‌های پیچیده و بزرگ با مشکل انفجار فضای حالت روبرو می‌شود و در عمل، فرایند وارسی الگو را مختل می‌کند. ازآنجاکه ویژگی دسترس‌پذیری دوگان ویژگی ایمنی است، با تأیید ویژگی دسترس‌پذیری می‌توان به‌جای تأیید ویژگی ایمنی آن را رَد کرد.

نتایج تجربی بر روی چند مورد مطالعه‌شده، نشان می‌دهند که روش پیشنهادی دارای سرعت بالاتر، توانایی تولید شاهدهای کوتاه‌تر و نیاز به کاوش تعداد کمتری از حالت‌ها در مقایسه با روش‌های دیگر است.

روش پیشنهادی برخی محدودیت‌ها نیز دارد. نگهداری جدول‌های احتمالات شرطی در سامانه‌هایی که تعداد قوانین زیادی دارند، به حافظهٔ زیادی نیاز دارد.

برای پژوهش‌های آینده، پیشنهاد می‌شود برای شناسایی مؤثرتر مسیرهای امیدبخش از الگوریتم‌های مارکوف و یادگیری عمیق استفاده کرد.

[DOI: 10.61186/jsdp.20.2.175]

8-Refrence ۸- مراجع

[1] Baier C, Katoen JP, Larsen KG (2008) Principles of model checking, vol 2620264. MIT press, Cambridge

[2] Chicano F, Francisco M, Ferreira M, Alba E (2011) Comparing metaheuristic algorithms for error detection in java programs. In: International symposium on Search based software engineering, vol 6956. Springer, Berlin, Heidelberg, pp 82–96

[3] Taentzer G, Ehrig K, Guerra E, Lara JD, Lengyel L, Levendovszky T, Prange U, Varro D, Varro-Gyapay S (2005) Model transformation by graph transformation: a comparative study. In: Proceedings of model transformations in practice workshop, Models conference, Montego Bay, Jamaica

[4] Rafe V, Hajvali M (2013) Designing an architectural style for pervasive healthcare systems. J Med Syst 37(2):1–13

[5] Heckel R, Thöne S (2005) Behavioral refinement of graph transformation-based models. Electron Notes the or Comput Sci 127(3):101–111

[6] Engels G, Hausmann JH, Heckel R, Sauer S (2000) Dynamic meta modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. In: International conference on the unified modeling language. Springer, Berlin Heidelberg, pp 323–337

[7] Mens T (2006) On the use of graph transformations for model refactoring. In: Generative and transformational techniques in software engineering. Springer, Berlin Heidelberg, pp 219–257

[8] Rafe V, Rahmani AT (2008) Formal analysis of workflows using UML 2.0 activities and graph transformation systems. In: International colloquium on theoretical aspects of computing. Springer Berlin, Heidelberg, pp 305–318

[9] Naddaf M, Rafe V (2014) Performance modeling and analysis of software architectures specified through graph transformations. Comput Inform 32(4):797–826

[10] Clarke E, McMillan K, Campos S, Hartonas-Garmhausen V (1996) Symbolic model checking. In: International conference on computer aided verification, vol 1102. Springer, Berlin, Heidelberg, pp 419–422

[11] Godefroid P, Van Leeuwen J, Hartmanis J, Goos G, Wolper P (1996) Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem, vol 1032. Springer, Heidelberg

[12] Lafuente AL (2003) Symmetry reduction and heuristic search for error detection in model checking. In: Workshop on model checking and artificial intelligence

[13] Rafe V (2013) Scenario-driven analysis of systems specified through graph transformations. J Vis Lang Comput 24(2):136–145

[14] Alba E, Chicano F (2008) Searching for liveness property violations in concurrent systems with ACO. In: Proceedings of the 10th annual conference on genetic and evolutionary computation, pp 1727–1734

[15] Rafe V, Moradi M, Yousefian R, Nikanjam A (2015) A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. Appl Soft Comput 33:136–149

[16] Pira E, Rafe V, Nikanjam A (2019) Using evolutionary algorithms for reachability analysis of complex software

systems specified through graph transformation, Reliability Engineering and System Safety, vol 191

[17] Partabian J, Rafe V, Parvin H, Nejatian S (2019), An Approach Based on Knowledge Exploration for State Space Management in Checking Reachability of Complex Software Systems, soft computing, vol. 24, pp.1-16.

[18] Yasrebi M, Rafe V, Parvin H, Nejatian S (2020), An efficient approach to state space management in model checking of complex software system using machine learning technique, journal of intelligent & Fuzzy system, vol.38, no. 2, pp. 1761-1773.

[19] Rensink A, Boneva I, Kastenberg H, Staijen T (2010) User manual for the GROOVE tool set. Department of Computer Science, University of Twente, Enschede

[20] Peng H, Tahar S (1998) A survey on compositional verification. Technical Report, Department of Electrical and Computer Engineering, Concordia University

[21] Roever WP (1998) The need for compositional proof systems: a survey. In: Compositionality: the significant difference. Springer, Berlin, Heidelberg, pp 1–22

[22] Godefroid P (1990) Using partial orders to improve automatic verification methods. In: International conference on computer aided verification. Springer, Berlin, Heidelberg, pp 176–185

[23] Valmari A (1988) Error detection by reduced reachability graph generation. In: Proceedings of the 9th European workshop on application and theory of petri nets, pp 95–112

[24] Godefroid P, Wolper P (1993) Using partial orders for the efficient verification of deadlock freedom and safety properties. Formal Methods Syst Des 2(2):149–164

[25] Godefroid P, Van Leeuwen J, Hartmanis J, Goos G, Wolper P (1996) Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem, vol 1032. Springer, Heidelberg

[26] Lafuente AL (2003) Symmetry reduction and heuristic search for error detection in model checking. In: Workshop on model checking and artificial intelligence

[DOI: 10.61186/jsdp.20.2.175]

[27] Wolper P, Leroy D (1993) Reliable hashing without collision detection. In: International conference on computer aided verification. Springer, Berlin Heidelberg, pp 59–70
[28] Stern U, Dill D (1995) Improved probabilistic verification by hash compaction. In: Advanced research working conference on correct hardware design and verification methods, vol 987. Springer, Berlin, Heidelberg, pp 206–224
[29] Stern U, Dill DL (1996) A new scheme for memory-efficient probabilistic verification. In: Formal description techniques IX. Springer, US, pp 333–348
[30] Courcoubetis C, Vardi M, Wolper P, Yannakakis M (1992) Memory efficient algorithms for the verification of temporal properties. In: Computer-aided verification, vol 1, issue 2–3. Springer, US, pp 129–142
[31] Isenberg T, Steenken D, Wehrheim H (2013) Bounded model checking of graph transformation systems via SMT solving. In: Formal techniques for distributed systems, vol 7892. Springer, Berlin, Heidelberg, pp 178–192
[32] Sivaraja H, Gopalakrishnan G (2003) Random walk based heuristic algorithms for distributed memory model checking. Electron Notes Theory Comput Sci 89(1):51–67
[33] Yang CH (1999) Prioritized model checking. Ph.D. thesis, Stanford Universit
[34] Edelkamp S, Lafuente AL, Leue S (2001) Directed explicit model checking with HSF-SPIN. In: Proceedings of the 8th international SPIN workshop on Model checking of software. Springer, New York, pp 57–79
[35] Yang CH, Dill DL (1998) Validation with guided search of the state space. In: Proceedings of the 35th annual design automation conference, pp 599–604
[36] Groce A, Visser W (2004) Heuristics for model checking Java programs. Int J Software Tools Technology Transf (STTT) 6(4):260–276
[37] Edelkamp S, Reffel F (1998) OBDDs in heuristic search. In: Annual conference on artificial intelligence. Springer, Berlin Heidelberg, pp 81–92
[38] Maeoka J, Tanabe Y, Ishikawa F (2016) Depth-first heuristic search for software model checking. In: Computer and information science, vol 614. Springer International Publishing, pp 75–96
[39] Edelkamp S, Leue S, Lafuente AL (2004) Directed explicit-state model checking in the validation of communication protocols. Int J Softw Tools Technol (STTT) 5(2–3):247–267
[40] Estler HC, Wehrheim H (2011) Heuristic search-based planning for graph transformation systems. KEPS 2011:54
[41] Snippe E (2011) Using heuristic search to solve planning problems in GROOVE. In: 14th Twente student conference on IT. University of Twente
[42] Ziegert S (2014) Graph transformation planning via abstraction. arXiv preprint arXiv: 1407.7933
[43] Elsinga JW (2016) On a framework for domain independent heuristics in graph transformation planning. Master's thesis, University of Twente
[44] Duarte LM, Foss L, Wagner FR, Heimfarth T (2010) Model checking the ant colony optimization. In: Hinchey M, Kleinjohann B, Kleinjohann L, Lindsay P, Rammig FJ, Timmis J, Wolf M (eds) Distributed, parallel and biologically inspired systems, vol 329. Springer, Berlin, Heidelberg, pp 221–232
[45] Alba E, Chicano F (2007a) Ant colony optimization for model checking. In: International conference on computer aided systems theory. Springer, Berlin, Heidelberg, pp 523–530
[46] Francesca G, Santone A, Vaglini G, Villani ML (2011) Ant colony optimization for deadlock detection in concurrent systems. In: 2011 IEEE 35th annual computer software and applications conference. IEEE, pp 108–117
[47] Alba E, Chicano F (2008) Searching for liveness property violations in concurrent systems with ACO. In: Proceedings of the 10th annual conference on genetic and evolutionary computation, pp 1727–1734
[48] Kumazawa T, Yokoyama C, Takimoto M, Kambayashi Y (2016) Ant colony Optimization based model checking extended by smell like pheromone. In: Proceedings of the 9th EAI international conference on bio-inspired information and communications technologies (formerly BIONETICS), pp 214–220
[49] Takada K, Takimoto M, Kumazawa T, Kambayashi Y (2017) ACO based model checking extended by smell-like pheromone with hop counts. In: Harmony search algorithm: proceedings of the 3rd international conference on harmony search algorithm, vol 514 (ICHSA 2017). Springer, p 52
[50] Foster H, Uchitel S, Magee J, Kramer J (2006) LTSA-WS: a tool for model-based verification of web service compositions and choreography. In: Proceedings of the 28th international conference on software engineering, Shanghai, China, pp 771–774
[51] Godefroid P, Khurshid S (2002) Exploring very large state spaces using genetic algorithms. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin Heidelberg, pp 266–280
[52] Alba E, Chicano F, Ferreira M, Gomez-Pulido J (2008) Finding deadlocks in large concurrent java programs using genetic algorithms. In: Proceedings of the 10th annual conference on genetic and evolutionary computation, pp 1735–1742
[53] Godefroid P (1997) Model checking for programming languages using Verisoft. In:

[DOI: 10.61186/jsdp.20.2.175]

Soft Comput. https://doi.org/10.1007/s00500-018-3314-7
[68] Godefroid P (1997) Model checking for programming languages using Verisoft. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages. ACM, pp 174–186
[69] Taentzer G, Ehrig K, Guerra E, Lara JD, Lengyel L, Levendovszky T, Prange U, Varro D, Varro-Gyapay S (2005) Model transformation by graph transformation: a comparative study. In: Proceedings of model transformations in practice workshop, Models conference, Montego Bay, Jamaica
[70] Rafe V, Hajvali M (2013) Designing an architectural style for pervasive healthcare systems. J Med Syst 37(2):1–13
[71] Heckel R, Thöne S (2005) Behavioral refinement of graph transformation-based models. Electron Notes Theory Comput Sci 127(3):101–111
[72] Engels G, Hausmann JH, Heckel R, Sauer S (2000) Dynamic meta modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. In: International conference on the unified modeling language. Springer, Berlin Heidelberg, pp 323–337
[73] Mens T (2006) On the use of graph transformations for model refactoring. In: Generative and transformational techniques in software engineering. Springer, Berlin Heidelberg, pp 219–257
[74] Naddaf M, Rafe V (2014) Performance modeling and analysis of software architectures specified through graph transformations. Comput Inform 32(4):797–826
[75] Rafe V, Rahmani AT (2009) Towards automated software model checking using graph transformation systems and Bogor. J Zhejiang Univ Sci A 10(8):1093–1105
[76] Moradi M. Rafe V. Yousefian R. and Nikanjam A. (2015). A Meta-Heuristic Solution for Automated Refutation of Complex Software Systems Specified through Graph Transformations. Applied Soft Computing. vol. 33, 136-149.
[77] Yousefian R, Rafe V, Rahmani M (2014) A heuristic solution for model checking graph transformation systems. Appl Soft Comput 24:169–180
[78] Edelkamp S, Lafuente AL, Leue S (2001) Protocol verification with heuristic search. In: AAAI symposium on model-based validation of intelligence, pp 75–83
[79] Schmidt A. Model checking of visual modeling languages Master's Thesis Hungary: Budapest University of Technology; 2004
[80] Gaschnig J (1979) Performance measurement and analysis of certain search algorithms. Technical Report. CMU-CS-79-124, Carnegie-Mellon University
[81] Thöne S (2005) Dynamic software architectures: a Style based modeling and Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages. ACM, pp 174–186
[54] Chicano F, Francisco M, Ferreira M, Alba E (2011) Comparing metaheuristic algorithms for error detection in java programs. In: International symposium on Search based software engineering, vol 6956. Springer, Berlin, Heidelberg, pp 82–96
[55] Edelkamp S, Jabbar S, Lafuente AL (2006) Heuristic search for the analysis of graph transition systems. In: International conference on graph transformation. Springer, Berlin, Heidelberg, pp 414–429
[56] Holzmann GJ (1997) The model checker SPIN. IEEE Trans Softw Eng 23(5):279–295
[57] Yousefian R, Aboutorabi S, Rafe V (2016) A greedy algorithm versus metaheuristic solutions to deadlock detection in graph transformation systems. J Intell Fuzzy Syst 31(1):137–149
[58] Yang XS (2010) A new meta heuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010), vol 284, pp 65–74
[59] Pira E, Rafe V, Nikanjam A (2016) EMCDM: efficient model checking by data mining for verification of complex software systems specified through architectural styles. Appl Soft Comput 49:1185–1201
[60] Abowd G, Allen R, Garlan D (1993) Using style to give meaning to software architecture. In: ACM SIGSOFT software engineering notes, vol 18, no. 5. ACM, pp 9–20
[61] Garlan D, Allen R, Ockerbloom J (1994) Exploiting style in architectural design environments. In: ACMSIGSOFT software engineering notes, vol. 19, no. 5. ACM, pp 175–188
[62] Pira E, Rafe V, Nikanjam A (2018) Searching for violation of safety and liveness properties using knowledge discovery in complex systems specified through graph transformations. Inf Softw Technol 97:110–134
[63] Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of 20th international conferences on very large data bases, VLDB, vol 1215, pp 487–499
[64] Pira E, Rafe V, Nikanjam A (2017) Deadlock detection in complex software systems specified through graph transformation using bayesian optimization algorithm. J Syst Softw 131:181–200
[65] Pelikan M, Goldberg DE, Tsutsui S (2003) Hierarchical Bayesian optimization algorithm: toward a new generation of evolutionary algorithms, vol 3. Springer, Berlin, Heidelberg, pp 2738–2743
[66] Thomas JP (2000) Design and verification of a coordination protocol for cooperating systems. Soft Comput 4(2):130–140
[67] Zhu W, Han Y, Zhou Q (2018) Performing CTL model checking via DNA computing.

[DOI: 10.61186/jsdp.20.2.175]

refinement technique with graph transformations. Ph.D. Thesis, Faculty of Computer Science, Electrical Engineering, and Mathematics, University of Paderborn
[82] Hausmann JH (2005) Dynamic meta modeling: a semantics description technique for visual modeling techniques. Ph.D. Thesis, Universität Paderborn
[83] S. Ziegert. "Graph transformation planning via abstraction", arXiv preprint arXiv: 1407.7933. 2014.

جعفر پرتابیان دکتری خود را در رشته سامانه‌های نرم‌افزاری در دانشگاه آزاد اسلامی واحد یاسوج گذرانده و هم‌اکنون استادیار دانشگاه آزاد اسلامی واحد لامرد است.

نشانی رایانامه ایشان عبارت است از:
jaafar_partabian@yahoo.com

[DOI: 10.61186/jsdp.20.2.175]