مجله محاسبات نرم، جلد 13، شماره 2 (پاییز و زمستان 1403)، صفحه 170_199
https://doi.org/10.22052/scj.2024.253499.1186

مجله محاسبات نرم
SOFT COMPUTING JOURNAL
تارنمای مجله: scj.kashanu.ac.ir

دانشگاه کاشان
University of Kashan

استفاده از یادگیری تقویتی جهت افزایش سرعت حل‌کننده‌های SMT با هدف تشخیص سریع‌تر مسیرهای غیرقابل اجرا در نرم‌افزار✧

مجید فیضی[1]، کارشناسی ارشد، محمد مهدی اثنی عشری[1*]، استادیار

[1] دانشکده مهندسی کامپیوتر، دانشگاه صنعتی خواجه نصیرالدین طوسی، تهران، ایران.

اطلاعات مقاله

تاریخچه مقاله:
دریافت 20 شهریور ماه 1402
پذیرش 26 دی ماه 1402

کلمات کلیدی:
آزمون خودکار نرم‌افزار
مسیرهای غیرقابل اجرا
حل‌کننده SMT
حل‌کننده Z3
یادگیری تقویتی
DQN

چکیده

آزمون نرم‌افزار یک موضوع مهم و لازم در فرآیند توسعه هر نرم‌افزاری می‌باشد. امروزه بخش عمده‌ای از آزمون نرم‌افزار به شیوه خودکار انجام می‌شود. در آزمون خودکار نرم‌افزار موانعی می‌توانند وجود داشته باشند که آزمون را با مشکل مواجه کنند. از این موانع، می‌توان به وجود مسیرهای غیرقابل اجرا در گراف جریان کنترلی حاصل از کدهای نرم‌افزار تحت آزمون اشاره کرد. مسیرهای غیرقابل اجرا، مسیرهایی هستند که با هیچ ورودی نمی‌توان از آنها عبور کرد. وجود چنین مسیرهایی، می‌تواند در حوزه‌های مختلف، مانند تولید داده آزمون، امنیت و مانند آن مشکلاتی را به وجود آورد و باعث هدر رفت منابع شود. برای پیشگیری از مشکلات ذکر شده، تشخیص مسیرهای غیرقابل اجرا بسیار مهم می‌باشد. اما تشخیص این مسیرها در نرم‌افزارهای بزرگ، به دلیل حجم زیاد کدها و زیاد بودن تعداد مسیرها در گراف جریان کنترلی، می‌تواند کاری زمانبر باشد. در روش ارائه شده در این مقاله، سرعت تشخیص مسیرهای غیرقابل اجرا با بکارگیری الگوریتم‌های یادگیری تقویتی بهبود داده شده است. در این روش سرعت حل‌کننده SMT به نام Z3 با بهره‌گیری از عامل DQN، افزایش داده می‌شود. Z3 ابزاری است که برای بررسی قابل حل بودن فرمول‌های ساخته شده از شرط‌های موجود در گراف جریان کنترلی در تشخیص مسیرهای غیرقابل اجرا استفاده می‌شود. در واقع در این مقاله تاکتیک‌های ارائه شده توسط Z3 با استفاده از یادگیری تقویتی یاد گرفته شده و سپس با انتخاب و اعمال تاکتیک‌های Z3 براساس یادگیری انجام شده، سرعت حل Z3 افزایش می‌یابد. با افزایش سرعت Z3، سرعت تمامی روش‌ها و الگوریتم‌هایی که از این ابزار برای تشخیص مسیرهای غیرقابل اجرا استفاده می‌کنند نیز افزایش خواهد یافت. همچنین ابزار Z3 در هر حوزه‌ای که بتوان مسائل را به مساله بررسی صدق‌پذیری یک فرمول کاهش داد نیز کاربرد دارد و بهبود عملکرد آن در چنین حوزه‌هایی نیز می‌تواند موثر باشد. روش ارائه شده در این مقاله در منطق‌های QF_NRA و QF_NIA آزمایش شده است. در آزمایش‌های انجام شده، نشان داده شده است که با استفاده از روش پیشنهادی، سرعت Z3 را می‌توان تا 4/7 برابر افزایش داد. همچنین در آزمایش‌های انجام شده با استفاده از این روش، سرعت Z3 در منطق QF_NRA، به طور میانگین 1/954 برابر و در منطق QF_NIA، به طور میانگین 1/687 برابر افزایش یافته است.

نحوه ارجاع به مقاله: مجید فیضی، محمد مهدی اثنی عشری، «استفاده از یادگیری تقویتی جهت افزایش سرعت حل‌کننده‌های SMT با هدف تشخیص سریع‌تر مسیرهای غیرقابل اجرا در نرم‌افزار»، مجله محاسبات نرم، جلد 13، شماره 2، ص 170_199، پاییز و زمستان 1403.

1. مقدمه

امروزه نرم‌افزار به بخشی از زندگی روزمره انسان‌ها مبدل شده است، و این موضوع باعث بزرگتر و پیچیده‌تر شدن نرم‌افزارها شده است. پیچیدگی نرم‌افزارها، فرایند توسعه آنها را دشوارتر و پرهزینه‌تر کرده است. یکی از مراحل مهم و تاثیرگذار در فرایند توسعه هر نرم‌افزاری، آزمون آن است. آزمون نرم‌افزار راهی برای اطمینان از درستی عملکرد نرم‌افزار می‌باشد. هدف در آزمون نرم‌افزار پیدا کردن بیشترین تعداد خطاها با کمترین تعداد آزمون است. البته باید به این نکته هم توجه کرد که دستیابی به نرم‌افزار کاملا عاری از خطا، عملا غیر ممکن است [1]. آزمون نرم‌افزار از مراحل مهم، زمانبر و پرهزینه در توسعه نرم‌افزار به شمار می‌آید. متاسفانه به دلیل وجود هزینه‌های مالی، نیاز به انجام انواع مختلف آزمون نرم افزار و نیز زمانبر و دشوار بودن این فرایند، در صنعت به خوبی به آن توجه نمی‌شود. این عدم توجه می‌تواند عواقب بدی مانند خاموشی بزرگ شمال شرقی آمریکا در سال 2003 و یا وجود نقص در دستگاه پرتو درمانی[1] که باعث مرگ افراد می‌شد [1] را به دنبال داشته باشد. بهترین راهکار برای برطرف کردن چنین مشکلاتی، خودکارسازی آزمون نرم‌افزار است. همانطور که در [2] بیان شده است، تا 50 درصد هزینه‌های توسعه نرم‌افزار برای آزمون نرم‌افزار است. از این رو محققان، تحقیقاتی برای خودکارسازی فرایند آزمون نرم‌افزار انجام داده‌اند، تا با این کار، هزینه‌ها و زمان مصرفی برای انجام آن را بکاهند.

اما در خودکارسازی آزمون نرم‌افزار موانع و مشکلاتی نیز وجود دارد. یکی از مهم‌ترین این موانع، وجود مسیرهای غیرقابل اجرا در واحدهای تحت آزمون، نظیر رویه‌ها یا کلاس‌ها، است. این مسیرها، آنهایی هستند که عملا با هیچ داده ورودی، نمی‌توان آنها را طی کرد. به عبارت بهتر، در هیچ اجرایی از واحد مورد نظر، این مسیرها طی نشده و تاثیری در عملکرد واحد ندارند. وجود چنین مسیرهایی، فرایند تولید خودکار آزمون را در یک حلقه بی انتها قرار می‌دهد. چون تولید کننده خودکار می‌خواهد به هر قیمتی که شده، ورودی مناسبی را بیابد که بتواند این مسیرها را هم پوشش دهد؛ چیزی که عملا امکان پذیر نیست. در [3] برخی از روش‌های تولید داده آزمون معرفی شده‌اند. برای جلوگیری از چنین وضعیتی، لازم است که ابتدا مسیرهای غیرقابل اجرا شناسایی شده و از گراف واحد تحت آزمون کنار گذاشته شوند.

تا کنون روش‌های مختلفی برای تشخیص مسیرهای غیرقابل اجرا ارائه شده است. برای مثال مقاله [4] براساس اجرای نمادین[2]، یک روش داوری ایستا[3] پیشنهاد می‌کند که با محاسبه دو بازه عددی برای هر نماد، بخشی از مسیرهای غیرقابل اجرا را به شکل دقیق تشخیص می‌دهد. در مقاله [5] نیز، روشی برای تشخیص مسیرهای غیرقابل اجرای بین‌رویه‌ای با استفاده از الگوهای محدودیت مسیر صدق‌ناپذیر، پیشنهاد شده است که در آن مسیرهای غیرقابل اجرای بین‌رویه‌ای بررسی شده و سپس محدودیت مسیر مربوط به هر مسیر غیرقابل اجرا استخراج شده است. سپس ویژگی‌های محدودیت مسیر غیرقابل حل مسیرهای غیرقابل اجرا را کاوش کرده و به 9 الگوی محدودیت مسیر صدق‌ناپذیر دست یافته‌اند. مقاله [6] نیز ابزار SMT-IPD[4] برای تشخیص مسیرهای غیرقابل اجرا را ارائه کرده است و از یک روش ایستا برای تشخیص آنها استفاده می‌کند که مبتنی بر حل‌کننده SMT است. در روش‌هایی که از حل‌کننده‌های SMT برای تشخیص مسیرهای غیرقابل اجرا استفاده می‌شود، سرعت حل‌کننده SMT تاثیر مستقیم در سرعت تشخیص مسیرهای غیرقابل اجرا دارد. بنابراین افزایش سرعت حل‌کننده SMT می‌تواند در تشخیص سریعتر مسیرهای غیرقابل اجرا موثر باشد. در این مقاله مسیرهای غیرقابل اجرا و چگونگی تشخیص آنها بررسی شده و تلاش شده است تا بهبودهایی برای تشخیص آنها صورت گیرد. هدف این مقاله افزایش سرعت فرایند

✧ نوع مقاله: پژوهشی
* نویسنده مسئول
پست(های) الکترونیک: majid.feyzi@email.kntu.ac.ir (فیضی)
esnaashari@kntu.ac.ir (اثنی عشری)

[1] Therac-25

[2] Symbolic Execution
[3] Static Judgment
[4] SMT solver based on Infeasible Path Detection

تشخیص مسیرهای غیرقابل اجرا با بهبود سرعت ابزارهای مورد استفاده برای تشخیص این مسیرها می‌باشد. ابزارهایی که در تشخیص مسیرهای غیرقابل اجرا کاربرد دارند، حل‌کننده‌های SMT هستند [6]. حل‌کننده SMT که در این مقاله مورد توجه و بررسی قرار گرفته است، حل‌کننده Z3 می‌باشد [7]. راهکار مورد استفاده در این مقاله بدین منظور استفاده از یادگیری تقویتی است. در واقع در این مقاله با استفاده یادگیری تقویتی تاکتیک‌های حل‌کننده Z3 انتخاب و اعمال می‌شوند تا سرعت حل فرمول‌های استخراج شده از مسیرها افزایش یابد.

آزمایش‌های انجام شده نشان داده‌اند که با به کارگیری روش پیشنهادی، می‌توان سرعت حل‌کننده Z3 را تا 4/7 برابر افزایش داد. بهبود سرعت عملکرد حل‌کننده Z3 نه تنها در تشخیص مسیرهای غیرقابل اجرا، بلکه در مسائل مختلف دیگر نیز می‌تواند مفید باشد، چرا که حل‌کننده Z3 در زمینه‌های مختلف دیگری، نظیر تایید نرم‌افزار و سخت افزار [8]، زمانبندی و آنالیز ایستا نیز کاربرد دارد [9].

در بخش 2 به معرفی و بررسی کارهای انجام شده در حیطه مورد بحث این مقاله می‌پردازیم. در بخش 3 به شکل خلاصه مفاهیم مقدماتی مورد نیاز را شرح خواهیم داد. در بخش 4 مساله موجود را تعریف خواهیم کرد. در بخش 50 روش پیشنهادی خود را به طور کامل شرح خواهیم داد. در بخش 6 آزمایش‌های صورت گرفته و نتایج آنها را بررسی خواهیم کرد و در بخش 7 نیز یک جمع‌بندی از مطالب ذکر شده خواهیم داشت.

2. کارهای مرتبط

در این بخش برخی از کارهای انجام شده برای تشخیص مسیرهای غیرقابل اجرا و نیز کارهای انجام شده به منظور تسریع حل‌کننده‌ها را بررسی می‌نماییم.

مقاله [4] یک روش داوری ایستا مبتنی بر اجرای نمادین ارائه می‌کند که بطور دقیق بخشی از مسیرهای غیرقابل اجرا و نیز بخشی از مسیرهای قابل اجرا تشخیص می‌دهد. این روش برای هر متغیر نمادین[5] به طور همزمان 2 مجموعه مقدار ممکن[6] و مجموعه مقدار ضروری[7] را محاسبه می‌کند و با استفاده از این اطلاعات بازه[8] به صورت قطعی تعیین می‌کند که مسیر قابل اجرا یا غیرقابل اجرا و یا نامعلوم است. به طور خلاصه روش ارائه شده در [4] برای هر نماد در هر مسیر که نشان دهنده یک متغیر می‌باشد، دو بازه عددی یافته و سپس با استفاده از آنها قابل اجرا بودن یا نبودن مسیر را تعیین می‌کند.

در اجرای نمادین به جای مقادیر واقعی، نمادها به ورودی‌ها اختصاص می‌یابند. بنابراین برای هر مسیر، یک عبارت از نمادهای ورودی به دست می‌آید. انفجار مسیر[9] یکی از محدودیت‌های اجرای نمادین می‌باشد. این به معنای افزایش نمایی تعداد مسیرهای قابل اجرا در برنامه با افزایش اندازه برنامه است که حتی در برنامه‌های دارای حلقه‌های نامحدود، تعداد مسیرها به بی‌نهایت می‌رسد [10]. اجرای نمادین نمی‌تواند همه مسیرهای قابل اجرا در برنامه‌های بزرگ را اجرا کند.

مجموعه مقدار ممکن یک تقریب کران بالا و مجموعه مقدار ضروری یک تقریب کران پایین از مقدار واقعی است. در مقاله [4] به ازای هر مسیر یک نماد برای هر یک از متغیرهای ورودی اختصاص یافته و سپس آن مسیر به صورت نمادین اجرا می‌شود و بازه هر نماد محاسبه می‌شود. حال اگر مجموعه مقدار ممکن نمادی خالی باشد، بنابراین تناقض‌هایی در این نماد وجود دارد و آن مسیر غیرقابل اجرا است. همچنین اگر حداقل مجموعه مقدار ضروری یکی از نمادها مقدار داشته باشد، بنابراین آن مسیر قابل اجرا است. ترکیب‌های مختلف مقادیر مجموعه‌های مقدار ضروری می‌تواند یک مورد آزمون[10] باشد که برنامه را به ازای یک مسیر اجرا می‌کند. به طور کلی در این روش قابل اجرا بودن یا نبودن هر مسیر براساس اطلاعات بازه‌های خروجی تعیین می‌شود.

مقاله [6] روشی ایستا برای تشخیص مسیرهای غیرقابل اجرا با به کارگیری حل‌کننده SMT ارائه می‌کند. در این مقاله ابزار

[5] Symbolic Variable

[6] Possible Value Set
[7] Necessary Value Set
[8] Range Information
[9] Path Explosion
[10] Test Case

SMT-IPD برای انجام این کار ارائه شده است. این روش در آغاز کار یک مجموعه زیرمسیر[11] ایجاد کرده و شرط‌های موجود در هر یک از آنها را استخراج می‌کند. پس از انجام این کار، محدودیت‌های[12] موجود در شرط‌های هر مسیر را به یک سری نامعادله تبدیل می‌کند. سپس، با استفاده از حل‌کننده SMT نامعادله‌ها را حل کرده و زیرمسیرها را به صورت زیرمسیرهای غیرقابل اجرا و نامشخص دسته‌بندی می‌کند. مسیرهایی که در دسته نامشخص قرار گرفته‌اند دوباره آزمون می‌شوند تا قابل اجرا بودن یا نبودنشان مشخص شود، و در پایان، قابل اجرا بودن یا نبودن همه مسیرها تعیین می‌شود. در روش پیشنهادی مقاله [6] با استفاده از زیرمسیر به جای مسیر سعی شده است تا جای ممکن مشکل انفجار مسیر مرتفع گردد. چرا که تعداد زیرمسیرها در برنامه‌های پیچیده به مراتب کمتر از تعداد مسیرها می‌باشد.

ابزار SMT-IPD برای تشخیص مسیرهای غیرقابل اجرا مراحل زیر را انجام می‌دهد: در مرحله اول، مجموعه‌ای از زیرمسیرها را ایجاد می‌کند. برای این کار ابتدا CFG برنامه تحت آزمون را بدست آورده و دورهای موجود در آن را حذف می‌کند. سپس وابستگی کنترلی برنامه آزمون را براساس روش محاسبه وابستگی کنترل بدست می‌آورد. سرانجام، CFG بدون دور حاصل از قسمت تحلیل ایستا را پیمایش کرده و یک مجموعه زیرمسیر ایجاد می‌کند. در مرحله دوم، قابل اجرا بودن هر زیرمسیر توسط این ابزار تعیین می‌شود. این ابزار برای انجام این کار، با الگوریتم ساخت نامعادله‌ها شرط‌های هر زیرمسیر را به نامعادله‌هایی تبدیل و سپس آنها را توسط حل‌کننده SMT حل می‌کند، و در آخر براساس نتایج بدست آمده از حل‌کننده SMT قابل اجرا بودن یا نبودن هر زیرمسیر را تعیین می‌کند. زیرمسیری که راه‌حلی برای نامعادله‌های آن یافت شده باشد قابل اجرا است، در غیر این صورت، غیرقابل اجرا یا تعیین نشده است. برای تعیین قابل اجرا بودن زیرمسیرهای تعیین نشده، این زیرمسیرها یک بار دیگر آزمون می‌شوند. در این مرحله، شرط‌های شاخه‌ای و نقاط تعریف متغیر وابستگی زیرمسیرها تحلیل می‌شوند. شرط‌های مسیر به صورت یک سیستم نامعادله تبدیل می‌شوند و سپس توسط حل‌کننده SMT حل می‌شود. اگر سیستم نامعادله‌ها قابل حل باشد، زیرمسیر قابل اجرا است ولی اگر قابل حل نباشد، غیرقابل اجرا است. اگر نتیجه سیستم نامعادله‌ها قابل تعیین نباشد، نمی‌تواند قابل اجرا بودن زیرمسیر را نیز تعیین کرد، که در این صورت زیرمسیر باید دوباره بررسی شود. در مرحله سوم، SMT-IPD زیرمسیرها را به مسیرها بسط می‌دهد و در نهایت، قابل اجرا بودن یا نبودن همه مسیرها را تعیین می‌کند. در این مرحله، SMT-IPD مجموعه زیرمسیر را بسط داده و نتایج تشخیص مسیرهای غیرقابل اجرا را بهبود می‌بخشد. SMT-IPD هر زیرمسیر را به مسیر بسط می‌دهد (تبدیل می‌کند) و قابل اجرا بودن یا نبودن مسیرهای بسط یافته را تعیین و نتایج تشخیص را تکمیل می‌کند.

مقاله [5]، روشی ارائه کرده است که در آن مسیرهای غیرقابل اجرای بین‌رویه‌ای با استفاده از الگوهای محدودیت مسیر صدق‌ناپذیر، تشخیص داده می‌شوند. در این روش، ابتدا با کاوش ویژگی‌های محدودیت مسیر، 9 الگوی محدودیت مسیر صدق‌ناپذیر که در مسیرهای غیرقابل اجرا رایج هستند شناسایی می‌شوند و سپس با استفاده از آنها مسیرهای غیرقابل اجرای بین‌رویه‌ای تشخیص داده می‌شوند. اگر شرط‌های محدودیت مسیری با یکی از 9 الگو یافت شده مطابقت داشته باشد بنابراین آن مسیر یک مسیر غیرقابل اجرا می‌باشد. هر ترکیبی از شرط‌ها می‌تواند به عنوان یک محدودیت مسیر تلقی شود. دو شرطی که متغیرهای مشترک دارند به یکدیگر وابسته هستند. اگر در مجموعه شرط‌های موجود در یک مسیر، هر شرط به شرط دیگری در همان مجموعه وابسته باشد، آنگاه به آن مجموعه شرط‌های وابسته می‌گویند. اگر بین شرط‌های وابسته یک مسیر تضادی وجود داشته باشد، آنگاه شرط‌های وابسته، یک محدودیت مسیر غیرقابل حل می‌باشند. الگوی محدودیت مسیر صدق‌ناپذیر، از محدودیت‌های غیرقابل حل موجود در یک مسیر بدست می‌آید. نویسندگان مقاله [5] برای کشف الگوی محدودیت غیرقابل حل 12695 مسیر غیرقابل اجرای

[11] Subpath
[12] Constraint

بین‌رویه‌ای را که در 6 برنامه به زبان C وجود داشت را به صورت دستی بررسی کرده‌اند. سپس محدودیت غیرقابل حل هر مسیر (شرط‌های محدودیت مسیری دارای تضاد) را با استخراج محدودیت مسیر هر یک از آنها مشخص کرده‌اند. در پایان، با کاوش ویژگی‌های محدودیت مسیر غیرقابل حل مسیرهای غیرقابل اجرا 9 الگوی محدودیت مسیر صدق‌ناپذیر کشف کرده‌اند.

مقاله [9] سعی در یادگیری حل فرمول‌های SMT دارد. حل‌کننده Z3 تاکتیک‌هایی دارد که به ترتیب (مشخص شده توسط کاربر) با هم ترکیب می‌شوند و یک استراتژی[13] را تشکیل می‌دهند. در یک فضای جستجوی گسترده، دست‌یابی به یک استراتژی مناسب که عملکرد خوبی داشته باشد، حتی برای متخصصان این حوزه نیز می‌تواند کار بسیار سختی باشد. مقاله [9] به مساله یافتن استراتژی مناسب می‌پردازد. مقاله [9] مساله حل فرمول‌های SMT را به صورت یک مساله جستجوی درخت بیان می‌کند که در هر مرحله آن یک تغییر شکل در فرمول ورودی صورت می‌پذیرد تا زمانی که فرمول حل شود. روش مقاله [9] از دو فاز تشکیل شده است. در فاز اول، یک سیاست را با استفاده از مجموعه‌ای از فرمول‌های حل نشده یاد می‌گیرد که با استفاده از آن یک تغییر مناسب را در هر گام به منظور حل هر فرمول انتخاب و اعمال کند. به بیان دیگر این روش سیاستی را که استراتژی‌های تسریع کننده در حل فرمول‌ها را می‌یابد، نتیجه می‌دهد. در فاز دوم هم یک استراتژی را به شکل یک برنامه بدون حلقه همراه با شاخه‌ها ترکیب[14] می‌کند. هدف مقاله [9] یافتن یک استراتژی است که زمان لازم برای حل فرمول‌های $F = \{f_i\}_{i=1}^{N}$ را به حداقل برساند. در مقاله [9] ابتدا سیاستی یاد گرفته می‌شود که مجموعه‌ای از استراتژی‌های کاندید را که تنها شامل توالی تاکتیک‌هایی است که در آن هر استراتژی در زیرمجموعه‌های مختلف مجموعه داده F عملکرد خوبی داشته است، پیدا می‌کند و سپس استراتژی‌های کاندید را با یکدیگر ترکیب می‌کند.

مقاله [11] روش‌های آزمایش کارآمد و اصولی را برای حل‌کننده‌های SMT توسعه داده است. برای این منظور، یک سیستم فازی[15] مبتنی بر یادگیری تقویتی به نام BanditFuzz را ارائه کرده است که ساختارهای دستوری ورودی‌های حل‌کننده را که علت اصلی مشکلات کارایی یا صحت در حل‌کننده‌های تحت آزمایش هستند، هدف قرار می‌دهد. در مقاله [11] با استفاده از BanditFuzz، 1700 ورودی منحصر به فرد از نظر نحوی کشف شده‌اند که منجر به پاسخ‌های ناسازگار در حل‌کننده‌های SMT پیشرفته Z3، CVC4، Colibri، MathSAT و Z3str3 در منطق‌های SMT ممیز شناور[16] و رشته‌ای شده است. نویسندگان مقاله [11] همچنین، با استفاده از BanditFuzz دو مجموعه معیار (با 400 نمونه برای منطق ممیز شناور و 300 نمونه برای رشته‌ها) ایجاد کرده‌اند که مشکلات کارایی را در همه حل‌کننده‌های در نظر گرفته آشکار می‌کند.

BanditFuzz از یادگیری تقویتی برای یادگیری اینکه کدام ساختارهای دستوری بیشترین احتمال ایجاد خطا یا مشکلات کارایی را دارند، استفاده می‌کند. فازر[17]های جهش سنتی یک عملگر جهش را به صورت تصادفی انتخاب یا اجرا می‌کنند و از رفتار برنامه‌های تحت آزمایش غافل هستند. فازر برنامه‌ای است که به طور خودکار ورودی‌های یک برنامه هدف تحت آزمایش را تولید می‌کند. در مقابل، BanditFuzz مشکل چگونگی جهش بهینه یک ورودی را به نمونه‌ای از مساله MAB[18] یادگیری تقویتی کاهش می‌دهد. مساله MAB یک مساله یادگیری تقویتی رایج است که براساس یک MDP با یک حالت واحد $S = \{s_0\}$ و مجموعه محدودی از عمل‌های A است. ایده اصلی در BanditFuzz، حفظ فهرستی از ساختارهای دستوری است که بر اساس احتمال اینکه علت اصلی یک خطا یا مشکل کارایی باشند، رتبه‌بندی می‌شوند. در ابتدا، تمام ساختارهای دستوری به طور یکنواخت در نظر گرفته می‌شوند که احتمال ایجاد خطا توسط عامل یادگیری تقویتی وجود دارد. BanditFuzz برای

[13] Strategy
[14] Synthesize

[15] Fuzzing System
[16] Floating Point
[17] Fuzzer
[18] Multi-Armed Bandits

شروع به طور تصادفی یک ورودی تولید می‌کند و همه برنامه‌ها در P را در ورودی I اجرا می‌کند. در هر یک از تکرارهای بعدی حلقه بازخورد خود، BanditFuzz ورودی I از تکرار قبلی خود را با استفاده از لیست رتبه‌بندی شده ساختارهای دستوری جهش می‌دهد (یعنی عامل یک عمل را انجام می‌دهد)، و همه حل‌کننده‌ها را در P بر روی نسخه جهش یافته I اجرا می‌کند. نتایج این اجراها را تجزیه و تحلیل می‌کند تا بازخورد (یعنی پاداش) را به عامل یادگیری تقویتی ارائه دهد که به احتمال زیاد باعث ایجاد تفاوت کارایی نسبی بین برنامه هدف T نسبت به برنامه مرجع R می‌شود. سپس لیست ساختارهای دستوری خود را با هدف به حداکثر رساندن پاداش (یعنی به حداکثر رساندن تفاوت کارایی نسبی بین حل‌کننده‌ها در P) به روز و دوباره رتبه‌بندی می‌کند. این فرآیند تا زمانی ادامه می‌یابد که عامل یادگیری تقویتی به یک رتبه‌بندی همگرا شود و منابع آن تمام شود.

مقاله [12] با استفاده از الگوریتم اجرای کانکالیک[19] و یادگیری تقویتی عمیق[20] و راه حل فازی ترکیبی[21] ابزاری برای یافتن باگ‌های عمیق به نام Dr.PathFinder ارائه می‌کند. در این مقاله، بر کاوش باگ‌های واقع در مسیر عمیق در مسیرهای اجرایی برنامه تمرکز شده است. برای دستیابی به این هدف، یک الگوریتم اجرای کانکالیک همراه با یادگیری تقویتی عمیق و راه حل فازی ترکیبی، برای یافتن باگ‌های عمیق پیشنهاد شده است. تجزیه و تحلیل برنامه، از جمله کشف آسیب‌پذیری، به دلیل فضای حالت نامتناهی آن به عنوان یک مساله غیرقابل تصمیم‌گیری شناخته می‌شود. بنابراین، برای ردیابی همه حالت‌ها، Dr.PathFinder حالت‌ها را با استفاده از یک شبکه عصبی عمیق تخمین می‌زند. سپس، بر اساس این تقریب، Dr.PathFinder شاخه‌ای را انتخاب می‌کند که انتظار می‌رود مسیر طولانی‌تری داشته باشد. برای این هدف، نویسندگان مقاله [12] یک الگوریتم یادگیری پیشنهاد داده‌اند که به عامل اجازه می‌دهد تا مسیرهایی را جستجو کند که انتظار می‌رود در طول اجرای کانکالیک عمیق‌تر باشند. عامل آموزش دیده به طور انتخابی شاخه‌هایی را که برای بهبود پوشش مفید نیستند هرس می‌کند. این کار در نهایت با محدود کردن مناسب تولید ورودی‌ها، فضای جستجوی اجرای کانکالیک را کاهش می‌دهد و در نتیجه فازر را قادر می‌سازد تا باگ‌های عمیق‌تری را پیدا کند. برخلاف بسیاری از فازرها که تقریبا به صورت متوالی از مسیرهای کم عمق شروع به جستجو می‌کنند، Dr.PathFinder بر روی مسیرهای عمیق‌تر تمرکز می‌کند.

روش عملکرد این روش به صورت زیر است که ابتدا فازر یک برنامه هدف و یک صف با موارد آزمون ساده را به عنوان آرگومان می‌گیرد. فازر یک مورد آزمون را از صف می‌گیرد و از آن به عنوان دانه‌ای برای الگوریتم جهش برای ایجاد یک مورد آزمون جدید استفاده می‌کند. برنامه هدف با این آزمون به عنوان ورودی اجرا می‌شود. هنگامی که فازر یک انتقال (پوشش) بلوک پایه جدید را پیدا می‌کند یا در حین اجرای برنامه هدف دچار خطا می‌شود، فازر مورد آزمون را جالب در نظر می‌گیرد و آنرا در صف قرار می‌دهد. پس از تکرار مراحل قبل اگر پوشش جدیدی پیدا نشد، فازر موتور اجرای کانکالیک تقویت عمیق را فراخوانی می‌کند. این موتور ابتدا موارد آزمون را از صف دریافت می‌کند و برنامه هدف را اجرا می‌کند. سپس موتور، محدودیت‌های مسیر شاخه‌ای را که در حین اجرای برنامه با آن مواجه می‌شود منفی می‌کند و اجرای نمادین را آغاز می‌کند. هنگامی که عامل DQN با یک حالت شرطی روبرو می‌شود، اجرای نمادین را به شاخه‌ای که در آن مسیر عمیق‌تری انتظار می‌رود هدایت می‌کند. پس از تکرار انتخاب شاخه و ردیابی مسیرها با عامل DQN، حل‌کننده SMT محدودیت‌های مسیر جمع‌آوری شده را حل می‌کند و موارد تست جدیدی را برای تحقق مسیرها ایجاد می‌کند. موارد آزمون تولید شده در صف قرار داده می‌شوند و دوباره استفاده می‌شوند.

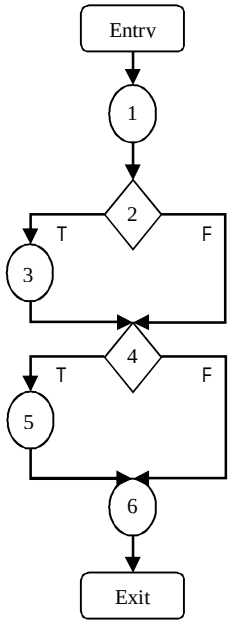
3. مفاهیم مقدماتی

در این بخش به معرفی و تعریف برخی از مفاهیم مورد استفاده در این مقاله پرداخته شده است.

[19] Concolic Execution
[20] Deep Reinforcement Learning
[21] Hybrid Fuzzing Solution

1.3. گراف جریان کنترلی

در آزمون نرم‌افزار می‌توان ساختارهای کمکی تهیه کرد که به آزمون کمک کنند. گراف‌ها رایج‌ترین ساختار برای کمک به آزمون نرم‌افزار هستند. گراف‌ها در حوزه های مختلف مانند [13] کاربرد دارند. گراف جریان کنترلی (با مخفف CFG)، مدل استاندارد نمایش جریان اجرای بین حالت‌های برنامه است [14]. در گراف جریان کنترلی گره‌ها نشان دهنده حالت‌ها و یال‌ها نشان دهنده جریان کنترلی بین حالت‌ها هستند. گراف جریان کنترلی در واقع یک مدل از نرم‌افزار ارائه می‌دهد. برای مثال شکل (1) یک تابع به نام abs و گراف جریان کنترلی متناظر با آن را نشان می‌دهد.




```
void int abs(int x) {
1:  int abs_value = 0;
2:  if (x >= 0)
3:     abs_value = x;
4:  if (x < 0)
5:     abs_value = -x;
6:  return abs_value;
}
```

شکل (1): تابع قدر مطلق در سمت راست شکل و گراف جریان کنترلی متناظر آن در سمت چپ شکل [14]

گراف جریان کنترلی بین‌رویه‌ای[22]، (با مخفف ICFG نیز شناخته می‌شود) نوع دیگری از گراف جریان کنترلی در علوم کامپیوتر است. در این نوع گراف‌ها، گراف جریان کنترلی رویه‌های فراخوانی شده در داخل رویه اصلی نیز با گراف جریان کنترلی رویه اصلی ترکیب می‌شوند. به عبارت دیگر، در گراف جریان کنترلی بین‌رویه‌ای، ارتباط بین رویه‌ها نیز در نظر گرفته می‌شود، این ویژگی باعث می‌شود تعداد مسیرها و گره‌ها در هر مسیر گراف جریان کنترلی بین‌رویه‌ای بیشتر از گراف جریان کنترلی ساده باشد.

در آزمون نرم‌افزار هدف پوشش گرافی است که ساخته‌ایم. اما معیارهای پوشش[23] متفاوت هستند. پوشش گره‌ها، یال‌ها و یا پوشش مسیرها هر کدام می‌توانند یک معیار پوشش باشند [1]. برای مثال، وقتی می‌گوییم معیار مورد نظر ما پوشش گره است، بدین معنی است که در آزمون، هر کدام از گره‌های گراف را حداقل یکبار ملاقات کنیم و گره ملاقات شده‌ای نداشته باشیم. به طور مشابه، در معیار پوشش مسیر نیز هدف پیمایش یا اجرای همه مسیرهای موجود در گراف جریان کنترلی است. هر مسیر در گراف جریان کنترلی، در واقع یک مسیر از گره شروع تا گره پایان گراف جریان کنترلی است.

2.3. مسیرهای غیرقابل اجرا

در گراف جریان کنترلی مسیرهایی وجود دارند که به ازای هیچ مقدار ورودی، نمی‌توان آنها را اجرا کرد که به این مسیرها، مسیرهای غیرقابل اجرا می‌گویند [14]. وجود مسیرهای غیرقابل اجرا در برنامه، تولید خودکار داده آزمون برای آن را دچار مشکل می‌کند. تولیدکننده خودکار داده آزمون[24] تلاش می‌کند تا داده‌های آزمونی تولید کند که برنامه را، با توجه به معیار پوشش انتخاب شده، به صورت کامل تحت آزمون قرار دهد. اگر معیار پوشش مورد نظر، پوشش مسیر باشد، و مسیرهای غیرقابل اجرا در برنامه وجود داشته باشد، تولیدکننده داده آزمون در یک حلقه بی انتها تلاش خواهد کرد داده‌هایی برای آزمون این مسیرها بیابد و موفق نخواهد شد. این موضوع موجب هدر رفت منابع و زمان می‌شود [6]. با توجه به آنچه که بیان شد، با تشخیص و حذف این مسیرها می‌توان از این مشکل پیشگیری و در منابع و زمان صرفه‌جویی نمود.

یکی دیگر از دلایل اهمیت مسیرهای غیرقابل اجرا، مرتبط با امنیت است [15]. در مقاله مورد نظر، نویسندگان به این موضوع

[22] Interprocedural Control Flow Graph

[23] Coverage Criteria

[24] Test Data Generator

اشاره کرده‌اند که برخی از بدافزارها کدهای مخرب خود را در مسیرهای غیرقابل اجرا مخفی می‌کنند تا از شناسایی شدن آنها جلوگیری کنند. با تشخیص مسیرهای غیرقابل اجرا و حذف آنها، می‌توان کدهای مخرب بدافزارها را از بین برد. همچنین تشخیص مسیرهای غیرقابل اجرا در بهینه‌سازی کد نیز کاربرد دارد [14]. امنیت یکی از مسائل مهم و حیاتی است که در [16] مورد بررسی قرار گرفته است.

به طور کلی، مسیرهای غیرقابل اجرا در گراف جریان کنترلی به دلایل مختلفی وجود دارند. یکی از اصلی‌ترین دلایل، تضاد میان حالت‌های شرطی مسیر، یا به عبارت دیگر، وابستگی بین حالت‌های شرطی به واسطه یک متغیر مشترک است [17]. این وابستگی به طور معمول با نماد $X \rightarrow Y$ نشان داده می‌شود، که در آن X نتیجه حالت شرطی اول (صحیح یا غلط) و Y نتیجه حالت شرطی دوم را نمایش می‌دهد. به عنوان مثال، در تابع شکل (1) و مسیر $P4$، دو حالت شرطی وجود دارند: حالت‌های 2 و 4. هر دو حالت شرطی در متغیر x مشترک هستند و با توجه به آن تعریف شده‌اند، اما شرط هرکدام با دیگری در تضاد است؛ به عبارت دیگر، $x >= 0$ با $x < 0$ در تضاد است. وابستگی بین این دو حالت شرطی به صورت $true \rightarrow false$ و $false \rightarrow true$ نشان داده می‌شود، به این معنی که در مسیری که از این دو حالت شرطی عبور می‌کند، زمانی که نتیجه شرط اول صحیح باشد، باید نتیجه شرط دوم غلط باشد و یا بالعکس تا آن مسیر قابل اجرا باشد. بنابراین هر مسیری که وابستگی‌های $false \rightarrow false$ و $true \rightarrow true$ داشته باشد (مانند مسیرهای $P1$ و $P4$)، یک مسیر غیرقابل اجرا خواهد بود. همچنین، وابستگی بین حالت‌های شرطی و حالت‌های تخصیص[25] (حالت‌های غیرشرطی) در برنامه‌نویسی نیز می‌توانند منجر به مسیرهای غیرقابل اجرا شوند [6]. به حالتی که در آن یک مقدار به یک متغیر تخصیص می‌یابد، حالت تخصیص گفته می‌شود. به عنوان مثال فرض کنید یک حالت تخصیص مانند $d = 20$ داریم که بدون اینکه مقدار متغیر آن در حالت دیگری تغییر یابد، در یک حالت شرطی با شرط $d > 30$ به کار برده می‌شود. بنابراین هر مسیری که در آن ابتدا باید از حالت تخصیص گفته شده و سپس از شاخه صحیح حالت شرطی ذکر شده عبور شود یک مسیر غیرقابل اجرا می‌باشد.

علاوه بر موارد ذکر شده، وجود حالت‌های شرطی که هیچ وقت مقدار آنها صحیح نخواهد شد نیز می‌توانند باعث ایجاد کدهای مرده[26] شوند [17]. از آنجایی که حالت‌های کدهای مرده حالت‌هایی هستند که قابل دستیابی نیستند و هرگز نمی‌توانند اجرا شوند، می‌توانند باعث ایجاد مسیرهای غیرقابل اجرا شوند. در مقاله [18] گفته شده است که احتمال غیرقابل اجرا بودن یک مسیر با تعداد حالت‌های شرطی در آن مسیر رابطه مستقیم داد و هرچه تعداد حالت‌های شرطی بیشتر باشد احتمال غیرقابل اجرا بودن آن مسیر نیز بیشتر است. به عبارت دیگر شرط‌های کمتر در هر مسیر به معنی بالاتر بودن احتمال قابل اجرا بودن آن مسیر است [19].

3.3. صدق‌پذیری پیمانه نظریه‌ها

صدق‌پذیری پیمانه نظریه‌ها که به صورت مخفف SMT به کار برده می‌شود، مساله تصمیم‌گیری در مورد صدق‌پذیری فرمول‌های منطقی مرتبه اول، براساس برخی از نظریه‌های منطقی است [8]. نسخه ساده‌تر مساله SMT، مساله صدق‌پذیری دودویی است که به صورت مخفف SAT به کار برده می‌شود. هدف مساله صدق‌پذیری دودویی، بررسی صدق‌پذیری یک مساله بولی[27] است. منظور از صدق‌پذیری تعیین این موضوع است که آیا می‌توان برای متغیرهای یک رابطه یا فرمول، مقادیری تعیین کرد که در آن رابطه صدق کند یا نه؟ برای مثال عبارت $a + 1 = 0$ به ازای $a = -1$ صادق (صدق‌پذیر) می‌باشد اما به ازای $a = 1$ صادق نمی‌باشد. در حوزه‌های مختلف علوم کامپیوتر، بسیاری از مسائل را می‌توان به مساله بررسی صدق‌پذیری یک فرمول در یک منطق کاهش داد. حل‌کننده‌های مختلفی وجود دارند که مسائل SMT را حل می‌کنند. یکی از بهترین حل‌کننده‌ها، حل‌کننده Z3 می‌باشد که

[25] Assignment Statement

[26] Dead Code

[27] Boolean

توسط Microsoft Research به‌طور رایگان در اختیار عموم قرار داده شده است که در این مقاله نیز به کار گرفته شده است. Z3 برای تجزیه و تحلیل برنامه‌های کاربردی و تایید آنها استفاده می‌شود [7]. حل‌کننده Z3 از 3 قالب[28] ورودی SMT-LIB و Simplify و DIMACS پشتیبانی می‌کند. قالب ورودی پیش‌فرض Z3، قالب SMT-LIB است. SMT-LIB یک نوآوری بین‌المللی با هدف تسهیل تحقیق و توسعه در صدق‌پذیری پیمانه نظریه‌ها یا همان SMT است [20]. SMT-LIB یک زبان استاندارد برای بیان فرمول‌ها و رابطه‌ها ارائه می‌کند. در SMT-LIB فرمول‌ها دارای منطق‌های مختلفی مانند QF_NRA، QF_NIA و... می‌باشند [20]. دلیل وجود منطق‌های مختلف این است که بتوان در عمل از روش‌های کارآمدتری برای حل فرمول‌های آن منطق استفاده کرد. این منطق‌ها با دسته‌ای از حروف نام‌گذاری می‌شوند که تئوری‌های استفاده شده در آن منطق را نشان می‌دهند. برای مثال عبارت QF نشان‌دهنده فرمول‌های بدون سور[29]، IA نشان‌دهنده محاسبات اعداد صحیح[30]، RA نشان‌دهنده محاسبات اعداد حقیقی و N قبل از RA و یا IA به معنای غیرخطی بودن است. بنابراین QF_NRA نشان‌دهنده فرمول‌های اعداد حقیقی غیرخطی بدون سور است. برای اطلاعات بیشتر در مورد منطق‌ها می‌توان به مرجع [20] مراجعه کرد.

حل‌کننده Z3، حاوی تعداد زیادی ترکیب اکتشافی[31] دست‌ساز و کاملا یکپارچه از روش‌های اثبات الگوریتمی است که به آنها تاکتیک گفته می‌شود. در Z3 می‌توان این تاکتیک‌ها را روی فرمول ورودی اعمال کرد و هر تاکتیک می‌تواند فرمول ورودی را به یک فرمول دیگر که حل آن توسط حل‌کننده Z3 آسانتر و سریعتر است، تبدیل نماید. حل‌کننده Z3 دارای بیش از 100 عدد تاکتیک می‌باشد که هر کدام از آنها می‌توانند پارامترهای مخصوص خود را نیز داشته باشند. مانند تاکتیک simplify که بیش از 50 پارامتر متفاوت دارد. برای مثال تاکتیک simplify، ثابت‌ها و عبارات تکراری را حذف می‌کند (مانند حذف عدد ثابت صفر از $x - 0$ یا جایگزین کردن عبارت $x - x$ با عدد صفر).

علاوه بر تاکتیک‌ها، حل‌کننده Z3 قابلیت‌های دیگری را نیز ارائه می‌کند. یکی از این قابلیت‌ها امکان اندازه‌گیری و شمارش برخی مقادیر مانند تعداد اعداد ثابت فرمول و... در فرمول ورودی می‌باشد، که به آنها سنجه قاعده[32] گفته می‌شود. سنجه‌های قاعده با استفاده از فرمول ارزیابی می‌شوند. مانند سنجه قاعده num-consts که تعداد ثابت‌های تفسیر نشده[33] غیر بولی موجود در فرمول هدف را می‌شمارد. اعمال تاکتیک روی فرمول می‌تواند مقادیر سنجه‌های قاعده را تغییر دهد.

Z3 همچنین این امکان را می‌دهد تا بتوان آمار[34]هایی مانند مقدار حافظه مورد استفاده یا زمان لازم برای حل فرمول مورد نظر را بدست آورد. یکی از کاربردی‌ترین آمارها که در این مقاله نیز مورد استفاده قرار گرفته است، rlimit می‌باشد. به بیان ساده، rlimit تعداد عملیات اساسی انجام شده توسط حل‌کننده برای حل فرمول را نشان می‌دهد.

4.3. یادگیری تقویتی

به طور کلی، یادگیری تقویتی به معنای یادگیری راه‌هایی است که باعث بهبود عملکرد یک عامل در محیط خاص می‌شود. در این نوع یادگیری، عامل با تعامل با محیط، عمل[35]های مختلف را انجام می‌دهد و سعی می‌کند تا سیگنال پاداش عددی را به حداکثر برساند. یکی از ویژگی‌های مهم یادگیری تقویتی، عدم دانستن عمل‌های بهینه از قبل است. به عبارت دیگر، به یادگیرنده نمی‌گوییم که کدام عمل‌ها را انجام دهد؛ بلکه او باید با آزمون و خطا کشف کند که کدام عمل‌ها بیشترین پاداش را در پی خواهند داشت. جستجو با آزمون و خطا و پاداش با تاخیر، دو ویژگی برجسته یادگیری تقویتی هستند [21]. به طور کلی، یادگیری تقویتی ایده یادگیری از طریق تعامل با یک محیط

[28] Format
[29] Quantifier Free
[30] Integer Arithmetic
[31] Heuristic
[32] Probes
[33] Uninterpreted constants
[34] Statistics
[35] Action

است.

در یادگیری تقویتی یک محیط وجود دارد که عامل عمل‌هایی را در آن انجام می‌دهد و براساس عمل‌هایی که انجام داده است، پاداش‌ها و تنبیه‌هایی را دریافت می‌کند و براساس آنها یک سیاست بهینه را یاد می‌گیرد. در مراحل بعدی عامل برای انتخاب و انجام عمل‌های خود در محیط از سیاستی که یاد گرفته است استفاده می‌نماید. علاوه بر عامل و محیط که به آنها اشاره شد، سیاست[36]، سیگنال پاداش[37] و تابع ارزش[38] سه عنصر اصلی دیگر در یادگیری تقویتی می‌باشند. سیاست نحوه رفتار عامل در یک زمان معین را مشخص می‌کند. به طور کلی، یک سیاست نگاشت از حالت[39]های درک شده محیط به عمل‌هایی است که باید در آن حالت‌ها انجام شود. در واقع، سیاست‌ها احتمالات را برای هر عمل مشخص می‌کنند. سیگنال پاداش، هدف یک مساله یادگیری تقویتی را مشخص می‌کند. محیط در هر گام زمانی[40]، یک عدد واحد را به عنوان پاداش به عامل یادگیری تقویتی می‌فرستد. هدف عامل به حداکثر رساندن پاداش با تاخیر است. منظور از پاداش با تاخیر، کل پاداشی است که عامل در بلند مدت دریافت می‌کند. بنابراین سیگنال پاداش، مشخص کننده عمل‌های خوب و بد برای عامل است. مبنای اصلی برای تغییر سیاست، عنصر موثر برای تغییر سیاست سیگنال پاداش است که می‌تواند باعث تغییر سیاست در آینده شود. در حالی که سیگنال پاداش میزان خوب بودن عمل‌ها در کوتاه مدت را نشان می‌دهد، یک تابع ارزش مشخص می‌کند که چه عمل‌هایی در بلند مدت خوب هستند. به طور کلی، مقدار کل پاداشی که عامل می‌تواند با شروع از یک حالت در بلند مدت بدست آورد، ارزش آن حالت را تعیین می‌کند. در حالی که پاداش‌ها، مطلوبیت ذاتی و فوری حالات محیطی را تعیین می‌کنند، ارزش‌ها مطلوبیت بلند مدت حالت‌ها را نشان می‌دهند. در یادگیری تقویتی در ابتدای کار عامل در یک حالت اولیه از محیط قرار دارد. عامل در هر حالت با توجه به وضعیتی که در آن قرار دارد و نیز براساس تجربیاتی که یاد گرفته است یک عمل را انتخاب می‌کند و آنرا در محیط انجام دهد. پس از اعمال عمل انتخاب شده در محیط توسط عامل، محیط پاداشی به عامل می‌دهد و عامل به یک حالت یل وضعیت دیگر در محیط انتقال می‌یابد. عمل انتخاب و انجام عمل توسط عامل در حالت‌های مختلف محیط تا رسیدن به یکی از حالت‌های پایانی محیط ادامه می‌یابد. به حالت‌هایی که عامل با شروع از حالت اول تا رسیدن به حالت آخر (حالت پایان) مشاهده می‌کند، یک دوره[41] گفته می‌شود. عامل دوره‌های مختلف و حتی تکراری را در محیط سپری می‌کند. هدف عامل در یادگیری تقویتی، بهبود سیاست‌های خود است، به‌طوری که مجموع پاداش‌های دریافتی در طول زمان بیشینه[42] شود.

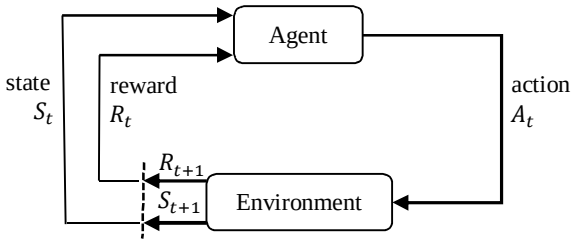
اکتشاف[43] و بهره‌برداری[44] دو مفهوم کلیدی در یادگیری تقویتی هستند. در بهره‌برداری، عامل از دانشی که تاکنون جمع‌آوری کرده است، برای انتخاب عمل‌ها استفاده می‌کند که انتظار دارد پاداش بیشتری داشته باشند. هدف از بهره‌برداری، بهبود پاداش در کوتاه‌مدت است. در اکتشاف، عامل عمل‌هایی را انتخاب می‌کند که نتایج آن‌ها نامعلوم است. اکتشاف به عامل امکان می‌دهد تا اطلاعات جدیدی از محیط جمع‌آوری کند و دانش خود را افزایش دهد. تعادل بین اکتشاف و بهره‌برداری، در مسائل یادگیری تقویتی بسیار مهم است. اگر عامل فقط از تجربیات گذشته استفاده کند، ممکن است در یک سیاست نامناسب گیر کند.

در یادگیری تقویتی، مسائل به صورت یک فرایند تصمیم‌گیری مارکوف[45] (MDP) مدل می‌شوند. فرایند تصمیم‌گیری مارکوف، رسمی‌سازی[46] تصمیم‌گیری ترتیبی است. در این فرایند، عامل در حالت s قرار دارد و یک عمل a را انتخاب می‌کند. پس از انجام هر عمل، عامل به حالت جدیدی منتقل می‌شود و پاداش

36 Policy
37 Reward Signal
38 Value Function
39 State
40 Time Step

41 Episode
42 Maximize
43 Exploration
44 Exploitation
45 Markov Decision Process
46 Formalization

مربوط به عمل a را دریافت می‌کند. شکل (2) نحوه تعامل یک عامل با محیط را در یک فرایند تصمیم‌گیری مارکوف نشان می‌دهد.



شکل (2): تعامل عامل با محیط در فرایند تصمیم‌گیری مارکوف [21]

در شکل (2)، t نشان‌دهنده گام زمانی، S_t حالتی است که عامل در گام زمانی t در آن حالت قرار دارد، A_t عملی است که عامل در حالت S_t انجام می‌دهد. عامل بعد از انجام عمل A_t از حالت S_t به حالت بعدی یعنی S_{t+1} منتقل می‌شود و پاداش R_{t+1} را دریافت می‌کند.

یادگیری تقویتی عمیق زیرمجموعه‌ای از یادگیری تقویتی است که اصول یادگیری عمیق[47] را با روش‌های یادگیری تقویتی ترکیب می‌کند. در این رویکرد، شبکه‌های عصبی عمیق به عنوان تقریب‌گرهای تابع ارزش یا سیاست در فرایند تصمیم‌گیری مارکوف (MDP) استفاده می‌شوند. این روش‌ها به خصوص در مسائلی که تعداد حالت‌ها زیاد است، عملکرد مناسبی دارند. به همین دلیل در این مقاله نیز از عامل یادگیری عمیق استفاده شده است.

4. تعریف مساله

همان‌طور که در بخش 3 گفته شد، مسیرهای غیرقابل اجرا مسیرهایی هستند که با هیچ داده ورودی نمی‌توان آنها را اجرا کرد. برای مثال، در گراف جریان کنترلی شکل (1)، چهار مسیر مختلف به شرح زیر از گره شروع تا گره پایان وجود دارد:

$P1 = (entry, 1, 2, 4, 6, exit)$
$P2 = (entry, 1, 2, 4, 5, 6, exit)$
$P3 = (entry, 1, 2, 3, 4, 6, exit)$
$P4 = (entry, 1, 2, 3, 4, 5, 6, exit)$

اگر با تخصیص مقادیر مختلف به پارامتر x، هر مسیر حداقل یکبار پیمایش یا به عبارت دیگر اجرا شود، در اینصورت گفته می‌شود که آزمون‌ها تابع را به طور کامل پوشش داده‌اند. در مورد مثال شکل (1)، هیچ مقداری برای پارامتر ورودی x وجود ندارد که بتواند دو مسیر $P1$ و $P4$ را پیمایش کند. لذا، این دو مسیر غیرقابل اجرا هستند.

تشخیص قابل اجرا بودن یا نبودن یک مسیر، یک مساله تصمیم‌ناپذیر است و این موضوع در مقاله [22] نشان داده شده است. بنابراین، یک روش کلی برای تشخیص مسیرهای غیرقابل اجرا وجود ندارد. اما با این حال روش‌های جزئی مختلفی برای تشخیص مسیرهای غیرقابل اجرا ارائه شده‌اند که تلاش می‌کنند تا جای ممکن، مسیرهای غیرقابل اجرا را تشخیص دهند. این روش‌ها به دو دسته روش‌های تحلیل ایستا[48] و روش‌های تحلیل پویا[49] تقسیم می‌شوند [6]، [23]، [24]. در روش تحلیل ایستا بدون اجرا کردن کدهای برنامه تحت آزمون، کدهای آن تجزیه و تحلیل شده و آزمون می‌شوند. در مقابل روش‌های تحلیل ایستا، روش‌های تحلیل پویا قرار دارند که در آنها کدهای برنامه تحت آزمون برای تجزیه و تحلیل عملکرد برنامه و آزمون آن اجرا می‌شوند. هر کدام از این روش‌ها می‌توانند برخی از مسیرهای غیرقابل اجرا را تشخیص دهند. ما در این مقاله بیشتر روی روش‌های تحلیل ایستا تمرکز داریم.

یکی از روش‌های ایستا برای تشخیص مسیرهای غیرقابل اجرا، استفاده از حل‌کننده‌های SMT برای بررسی قابل حل بودن یا نبودن شرط‌های موجود در هر مسیر است [6]. در این روش شرط‌های موجود در هر مسیر از گراف جریان کنترلی استخراج شده، سپس به صورت نامعادله‌هایی به حل‌کننده‌های SMT ارسال می‌شوند. پس از حل نامعادله‌ها، سازگاری و یا ناسازگاری بین شرط‌ها بررسی شده و براساس آن در مورد قابل اجرا بودن یا نبودن مسیر تصمیم‌گیری می‌شود. برای مثال نامعادله $x > 1 \text{^} x \leq 4$ به ازای $x = \{2\}$ قابل حل می‌باشد. حل‌کننده SMT قابل حل بودن و یا نبودن نامعادله‌ها را تعیین

[47] Deep Learning

[48] Static Analysis
[49] Dynamic Analysis

می‌کند. اگر نامعادله‌ها قابل حل باشند، پس مسیر متناظر آنها قابل اجرا می‌باشد، در غیر این‌صورت، آن مسیر غیرقابل اجرا است.

حل‌کننده‌های SMT برای حل نامعادله‌ها و فرمول‌های منطقی با قوانین خاص، استفاده می‌شوند. یکی از معروف‌ترین حل‌کننده‌های SMT حل‌کننده Z3 است [9]. این ابزار در مسائل مختلف کاربرد دارد، از جمله تحلیل ایستا و تولید موارد آزمون [7]. حل‌کننده Z3 دارای تاکتیک‌هایی می‌باشد. تاکتیک‌ها در حل‌کننده Z3 برای بهبود و سرعت‌بخشی به حل نامعادله‌ها استفاده می‌شوند. هر تاکتیک، نامعادله یا فرمول ورودی را به یک نامعادله یا فرمول دیگر تبدیل می‌کند تا حل آن ساده‌تر شود. انتخاب تاکتیک‌های مناسب برای هر ورودی مهم است و می‌تواند تاثیر زیادی در سرعت حل داشته باشد. به طور خلاصه، حل‌کننده Z3 با استفاده از تاکتیک‌های مختلف، نامعادله‌ها را به صورت سریع‌تر و دقیق‌تر حل می‌کنند. انتخاب تاکتیک‌های مناسب و تجربه در استفاده از آنها، کلید موفقیت در حل مسائل پیچیده است.

در صورتی که تعداد شرط‌های موجود در هر مسیر از گراف جریان کنترلی زیاد باشد، نامعادله/فرمول ایجاد شده نیز بزرگتر و پیچیده‌تر خواهد بود. در گراف جریان کنترلی بین رویه‌ای، تعداد شرط‌های موجود در هر مسیر حتی بیشتر هم می‌شود. در چنین شرایطی حل نامعادله/فرمول توسط Z3 زمانبر خواهد بود. بنابراین سرعت تشخیص مسیرهای غیرقابل اجرا به سرعت حل نامعادله‌ها/فرمول‌ها توسط Z3 وابسته است و با افزایش سرعت حل کننده Z3، سرعت تشخیص مسیرهای غیرقابل اجرا نیز افزایش می‌یابد. انتخاب و اعمال تاکتیک‌های مناسب روی نامعادله/فرمول می‌تواند در سرعت حل آن توسط Z3 تاثیر بگذارد. اما همانطور که گفته شد، انتخاب تاکتیک مناسب برای هر فرمول کار آسانی نیست. در روش پیشنهادی مقاله با استفاده از یادگیری تقویتی و انتخاب تاکتیک‌های مناسب برای فرمول‌های مختلف سرعت حل‌کننده Z3 افزایش می‌یابد.

5. روش پیشنهادی

از آنجایی که انتخاب تاکتیک‌های مناسب در حل‌کننده‌های SMT تاثیر قابل توجهی در سرعت و دقت این حل‌کننده‌ها دارد، روش پیشنهادی مقاله استفاده از یادگیری تقویتی برای انتخاب تاکتیک‌ها و بهبود عملکرد حل‌کننده‌های SMT است که این کار تشخیص هر چه بهتر و سریعتر مسیرهای غیرقابل اجرا را به دنبال دارد. نحوه عملکرد یادگیری تقویتی از طریق آزمون و خطا و تنبیه و پاداش می‌باشد، بنابراین نیازی به داشتن دانش قبلی نیست. با استفاده از یادگیری تقویتی می‌توان بدون داشتن دانش قبلی و اینکه یک تاکتیک روی یک فرمول چه تاثیری می‌گذارد، در مورد تاکتیک‌ها یاد گرفت. بدین شکل که عامل یادگیری تقویتی به ازای تاکتیک‌های موثر بر فرمول، پاداش دریافت می‌کند. تاکتیک‌های موثر تاکتیک‌هایی هستند که باعث سریعتر حل شدن فرمول می‌شوند و یا اینکه آنرا به طور صریح قابل حل می‌کنند. در واقع عامل یاد میگیرد که چه تاکتیک‌هایی برای کدام مجموعه فرمول‌ها مناسب هستند. با استفاده از یادگیری تقویتی می‌توان در مورد مسائل مختلف تصمیم‌گیری کرد و انتخاب تاکتیک مناسب یک عمل تصمیم‌گیری است. بنابراین استفاده از یادگیری تقویتی می‌تواند مفید واقع شود.

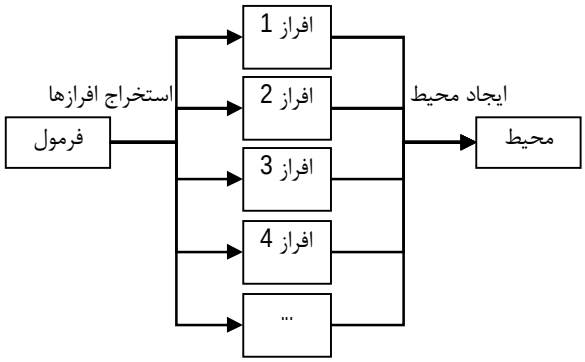
همانطور که در بخش 30 (مفاهیم مقدماتی) ذکر شد، یادگیری تقویتی از 5 عنصر اصلی تشکیل شده است، که ابتدا به معرفی آنها در روش پیشنهادی می‌پردازیم و سپس نحوه عملکرد روش پیشنهادی را شرح می‌دهیم. کدهای منبع روش پیشنهادی به صورت متن باز در دسترس می‌باشد[50].

1.5. محیط

در ابتدای کار یک محیط باید وجود داشته باشد تا عامل بتواند در آن آموزش ببیند. محیطی که برای آموزش عامل طراحی شده است، با استفاده از یک فرمول در قالب SMT که به صورت یک فایل با پسوند smt2. در دسترس است و توسط حل‌کننده Z3 قابل خواندن است، ایجاد می‌شود. بنابراین می‌توان با هر فایل

[50] https://github.com/majidfeyzi/Z3-tactics-learning

SMT یک محیط جداگانه ایجاد کرد. محیط باید به گونه‌ای طراحی شود که وابسته به یک فرمول خاص نباشد. همچنین نمی‌توان از همه فرمول‌ها برای ایجاد یک محیط استفاده کرد، چرا بی‌نهایت فرمول می‌تواند وجود داشته باشد. بنابراین محیط به جای اینکه وابسته به یک یا چند فرمول باشد، می‌تواند وابسته به افرازهایی باشد که بین فرمول‌های مختلف مشترک هستند. این افرازها می‌توانند گسترده و مختلف باشند و برخی از آنها باید انتخاب و برای طراحی محیط استفاده شوند. در واقع محیط می‌تواند با استفاده از افرازهایی که از یک فرمول بدست می‌آید ساخته شود. در این حالت، با اینکه محیط با استفاده از یک فرمول ایجاد می‌شود اما در واقع مجموعه‌ای از فرمول‌ها را شامل می‌شود. این افرازها می‌توانند ویژگی‌هایی مانند منطق فرمول، تعداد عملگرهای مختلف و... باشند که می‌توان از یک فرمول استخراج کرد. شکل (3) این فرآیند را نشان می‌دهد.



شکل (3): فرآیند ایجاد محیط با استفاده از افرازها یا ویژگی‌های فرمول

در این مقاله سعی شده است محیط به گونه‌ای طراحی شود که بتوان عامل‌های مختلف یادگیری تقویتی را در این محیط آموزش داد. از این رو تلاش شده است محیط طراحی شده، تا حد امکان به محیط‌های کتابخانه OpenAI Gym [25] تشابه داشته باشد. طراحی محیط به این شکل، این امکان را برای محققان فراهم می‌کند تا در صورت نیاز بتوانند به آسانی از این محیط در مطالعات خود بهره بگیرند.

1.1.5. حالت‌ها

با توجه به اینکه تعداد فرمول‌ها بسیار زیاد و بلکه نامتناهی است، در نظر گرفتن هر فرمول به عنوان یک حالت از محیط ایده چندان مناسبی به نظر نمی‌رسد. این روش در مقاله [9] مورد استفاده قرار گرفته و باعث شده که نتیجه قابل تعمیم روی فرمول‌های مختلف نباشد. به جای آن، در این مقاله سعی شده است ویژگی‌هایی برای فرمول‌ها در نظر گرفته شود که بر اساس این ویژگی‌ها و مقادیر آنها، بتوان فرمول‌ها را دسته بندی کرده و هر دسته را به عنوان یک حالت از محیط محسوب نمود. این ویژگی‌ها را، که تعداد آنها در این مقاله 27 عدد در نظر گرفته شده است، تحت عنوان «سنجه‌های قاعده» نامگذاری نموده‌ایم. به عبارت دیگر، هر حالت با مجموعه مشخص و متفاوتی از مقادیر برای این 27 سنجه بازنمایی می‌شود. این سنجه‌های قاعده را می‌توان در جدول (1) مشاهده نمود. مقادیر برخی از این سنجه‌های قاعده از حل‌کننده Z3 دریافت می‌شوند (با دادن فرمول به آن و دریافت مقدار سنجه به عنوان خروجی Z3) و مقادیر برخی نیز مستقل از Z3 تعیین می‌شوند. از 27 سنجه قاعده در نظر گرفته شده، 23 سنجه مقدار عددی و 4 سنجه مقدار بولی دارند. مقادیر 23 سنجه عددی و نیز مقدار یکی از سنجه‌های بولی، با انجام عمل‌ها توسط عامل می‌تواند تغییر می‌کند، که به دنبال آن، عامل وارد یک حالت جدید می‌شود. اما مقادیر 3 عدد از سنجه‌های قاعده بولی تنها به منظور ایجاد تمایز بین حالت‌ها در منطق‌های مختلف SMT، به حالت‌های محیط اضافه شده‌اند و لذا با انجام عمل‌های مختلف توسط عامل تغییر نمی‌کنند؛ مقادیر آنها تنها براساس فرمول اولیه سازنده محیط تعیین می‌شود.

بدیهی است که انتخاب سنجه‌های قاعده، تاثیر مستقیم و شدیدی روی نتیجه کار خواهد داشت و لذا انتخاب آنها باید با دقت انجام شود. همچنین، سنجه‌های قاعده باید به شکلی انتخاب شوند که به فرمول‌ها مرتبط باشند. به عنوان مثال، سنجه قاعده depth، که تعداد تاکتیک‌های اعمال شده روی فرمول را ارائه می‌دهد، ارتباطی با خود فرمول ندارد و لذا سنجه مناسبی محسوب نمی‌شود.

جدول (1): لیست سنجه‌های قاعده استفاده شده برای ایجاد حالت‌های محیط

	سنجه قاعده	توضیحات
1	size	تعداد ادعاها (Assertions) در فرمول داده شده
2	num-exprs	تعداد عبارات (Expressions)/اصطلاحات (Terms) در فرمول داده شده
3	num-consts	تعداد ثابت‌های غیر بولی در فرمول داده شده
4	num-bool-consts	تعداد ثابت‌های بولی در فرمول داده شده
5	num-arith-consts	تعداد ثابت‌های حسابی (Arithmetic) در فرمول داده شده
6	num-bv-consts	تعداد ثابت‌های بردار بیتی (Bit-Vector) در فرمول داده شده
7	+	تعداد عملگرهای جمع در فرمول داده شده
8	-	تعداد عملگرهای تفریق در فرمول داده شده
9	/	تعداد عملگرهای تقسیم در فرمول داده شده
10	*	تعداد عملگرهای ضرب در فرمول داده شده
11	==	تعداد عملگرهای برابری در فرمول داده شده
12	!=	تعداد عملگرهای مخالف در فرمول داده شده
13	>=	تعداد عملگرهای بزرگتر مساوی در فرمول داده شده
14	>	تعداد عملگرهای بزرگتر در فرمول داده شده
15	<=	تعداد عملگرهای کوچکتر مساوی در فرمول داده شده
16	<	تعداد عملگرهای کوچکتر در فرمول داده شده
17	and	تعداد عملگرهای and در فرمول داده شده
18	or	تعداد عملگرهای or در فرمول داده شده
19	bvand	تعداد عملگرهای bvand در فرمول داده شده
20	bvor	تعداد عملگرهای bvor در فرمول داده شده
21	bvxor	تعداد عملگرهای bvxor در فرمول داده شده
22	bvashr	تعداد عملگرهای bvashr در فرمول داده شده
23	bvshl	تعداد عملگرهای bvshl در فرمول داده شده
24	is-unbounded	تعیین می‌کند که آیا فرمول داده شده دارای ثابت‌های صحیح/واقعی که کران بالا/ پایین ندارند می‌باشد یا خیر
25	is-qfbv	تعیین می‌کند که آیا فرمول داده شده در منطق QF_BV است یا خیر
26	is-qfnia	تعیین می‌کند که آیا فرمول داده شده در منطق QF_NIA است یا خیر
27	is-qfnra	تعیین می‌کند که آیا فرمول داده شده در منطق QF_NRA است یا خیر

محیط باید دارای یک حالت شروع باشد تا عامل کار خود را از آن حالت شروع کند. حالت شروع حالتی است که در آن مقادیر سنجه‌های قاعده بدون استفاده از تاکتیک و با استفاده از فرمول اولیه ورودی تعیین می‌شوند. حالت پایانی مشخصی در این مقاله در نظر گرفته نشده است. اما در یکی از دو حالت زیر، دوره[51] را خاتمه یافته تلقی می‌نماییم: (1) فرمولی که به آن رسیده‌ایم به اندازه‌ای ساده باشد که به طور صریح قابل حل باشد. منظور از فرمولی که به طور صریح قابل حل باشد، فرمولی است که به اندازه‌ای ساده شده باشد که دیگر هیچ فرمول هدف فرعی نداشته باشد. (2) عامل از حداکثر تعداد عمل‌های قابل انتخاب در یک دوره عبور کرده باشد.

[51] Episode

2.5. عمل‌ها

هر عمل از یک تاکتیک و پارامترهای آن (در صورت وجود) تشکیل شده است. انجام هر یک از عمل‌ها توسط عامل در محیط به معنی اِعمال تاکتیک متناظر روی فرمول است. این کار باعث تغییر شکل فرمول شده و به دنبال آن، مقادیر سنجه‌های قاعده نیز تغییر می‌کنند و لذا، عامل به یک حالت جدید در محیط منتقل می‌شود. لیست عمل‌ها و تاکتیک و پارامتر سازنده هر عمل در جدول (2) نمایش داده شده‌اند. در کل 14 تاکتیک و 9 پارامتر در این مقاله در نظر گرفته شده است. نوع همه پارامترهای در نظر گرفته شده بولی می‌باشد. با لحاظ کردن پارامترهای مختلف برای هر تاکتیک، در مجموع عامل دارای 23 عمل خواهد بود. برای مثال برای تاکتیک aig دو عمل وجود دارد. عمل اول، تاکتیک aig بدون پارامتر است و عمل دوم، تاکتیک aig با پارامتر aig_per_assertion است که مقدار پارامتر true می‌باشد. تاکتیک aig با پارامتر aig_per_assertion و مقدار پارامتر false به عنوان عمل مجزا در نظر گرفته نمی‌شود. دلیل این موضوع این است که مقدار پیش‌فرض پارامترهای بولی، false است و تاکتیک aig با پارامتر aig_per_assertion و مقدار پارامتر false معادل همان تاکتیک aig بدون پارامتر است.

جدول (2): لیست عمل‌های ممکن در محیط و تاکتیک‌ها و پارامترهای متناظر با هر عمل

	عمل	تاکتیک	پارامترها
1	simplify	simplify	-
2	simplify(elim_and=true)		elim_and
3	simplify(blast_distinct=true)		blast_distinct
4	simplify(som=true)		som
5	simplify(pull_cheap_ite=true)		pull_cheap_ite
6	simplify(hoist_mul=true)		hoist_mul
7	simplify(local_ctx=true)		local_ctx
8	simplify(flat=true)		flat
9	smt	smt	-
10	bit-blast	bit-blast	-
11	bv1-blast	bv1-blast	-
12	solve-eqs	solve-eqs	-
13	aig(aig_per_assertion=true)	aig	aig_per_assertion
14	aig		-
15	qfnra-nlsat	qfnra-nlsat	-
16	sat	sat	-
17	max-bv-sharing	max-bv-sharing	-
18	reduce-bv-size	reduce-bv-size	-
19	purify-arith	purify-arith	-
20	propagate-values(push_ite_bv=true)	propagate-values	push_ite_bv
21	propagate-values		-
22	elim-uncnstr	elim-uncnstr	-
23	ackermannize_bv	ackermannize_bv	-

* برای اطلاعات بیشتر در مورد تاکتیک‌ها می‌توانید به صفحه راهنمای پروژه Z3 به پیوند https://microsoft.github.io/z3guide، بخش تاکتیک‌ها مراجعه نمایید.

3.5. سیگنال پاداش

به طور کلی، هدف یادگیری تاکتیک‌ها و پارامترهای موثر در هر حالت از محیط است. به عبارت دیگر، هدف تشخیص تاکتیک‌ها و پارامترها به ازای حالت‌های مختلف محیط است که سرعت حل فرمول توسط حل‌کننده را کمینه می‌کنند. بنابراین، سیگنال پاداش باید به گونه‌ای تعیین شود که با افزایش سرعت حل فرمول توسط حل‌کننده، رابطه مستقیم داشته باشد. در جدول (3)، پاداش‌ها برای هر عمل مشخص شده‌اند. عامل به طور پیش‌فرض، پاداش 2- برای هر عملی که در محیط انجام می‌دهد دریافت می‌کند. این پاداش به عنوان یک جریمه برای عامل عمل می‌کند و عامل را مجاب می‌کند تا کمترین تعداد عمل را در محیط انجام دهد. اگر عامل عملی انجام دهد که منجر به کاهش rlimit شود، جریمه 2- به 1- کاهش می‌یابد. همچنین، اگر عامل عملی انجام دهد که باعث شود تا فرمول به طور صریح قابل حل شود، بیشترین پاداش، یعنی مقدار 5 را دریافت می‌کند. پس از انجام عملی که منجر به فرمولی با قابلیت حل شدن صریح می‌شود، دوره خاتمه می‌یابد که معادل دریافت پاداش 5 می‌باشد. اما اگر عامل عملی انجام دهد که باعث بروز خطا در حل‌کننده شود، جریمه 5- دریافت می‌کند. با این جریمه، عامل به مرور دیگر چنین عمل‌هایی را در محیط انجام نمی‌دهد.

دریافت پاداش 1- به جای پاداش 2- در ازای انجام عمل‌هایی که باعث کاهش rlimit می‌شوند، باعث می‌شود تا این عمل‌ها نسبت به عمل‌هایی که تاثیری روی فرمول ندارند اولویت بیشتری برای عامل داشته باشند، چرا که عامل با انجام این عمل‌ها جریمه کمتری دریافت می‌کند. همچنین مقدار پاداش 1- برای عمل‌هایی که باعث کاهش rlimit می‌شوند، باعث می‌شود عامل تلاش کند تا با کمترین تعداد عمل به بیشترین پاداش دست پیدا کند. تخصیص مقدار پاداش 0 و یا مثبت (مثلا پاداش 1+) به چنین عمل‌هایی می‌تواند عامل را با خطا مواجه کند. در چنین حالتی ممکن است عامل عمل‌های اضافی و ناکارآمد انجام دهد و سراغ عمل‌هایی که منجر به فرمولی با قابلیت حل شدن صریح می‌شوند و بیشترین پاداش را به عامل می‌دهند نرود و یا آنها را اولویت‌های بعدی خود قرار دهد. دلیل این امر این است که عامل تلاش می‌کند تا پاداشی را که در طولانی مدت دریافت می‌کند بیشینه کند.

نکته‌ای که در اینجا باید بدان توجه کرد این است که اِعمال تاکتیک‌ها در Z3، خود باعث افزایش مقدار rlimit می‌شود. اما این افزایش نباید در rlimit حل فرمول لحاظ شود. به همین دلیل، محاسبه مقدار rlimit باید طبق روال زیر انجام شود:

(1) اِعمال تاکتیک روی فرمول مورد نظر
(2) اندازه‌گیری مقدار اولیه rlimit
(3) حل فرمول بعد از اِعمال تاکتیک
(4) اندازه‌گیری مقدار ثانویه rlimit
(5) محاسبه اختلاف rlimit اندازه‌گیری شده در گام 2 و گام 4 و در نظر گرفتن این مقدار به عنوان مقدار اصلی rlimit.

جدول (3): پاداش‌های محیط

مقدار پاداش	توضیحات
5	پاداش ساده‌سازی فرمول تا حدی که به طور صریح قابل حل باشد
-5	جریمه انجام عملی که مناسب فرمول نباشد و باعث بروز خطا شود
-1	پاداش عملی که بتواند rlimit را کاهش دهد
-2	پاداش پیش‌فرض هر گام زمانی

4.5. عامل

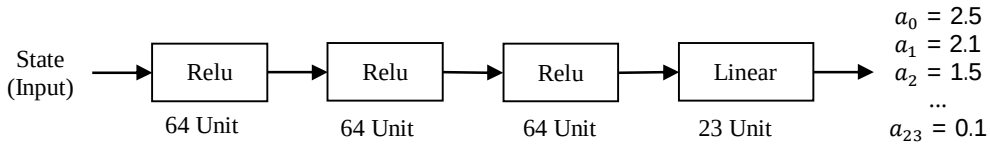
در این مقاله از عامل DQN [26] استفاده شده است که در ادامه همین بخش معرفی می‌شود. دلیل انتخاب عامل DQN آن است که عموما در محیط‌های با تعداد حالت‌ها و عمل‌های زیاد عملکرد مناسبی دارد.

عامل از یک شبکه عصبی که متشکل از سه لایه Dense (Relu) با 64 واحد در هر لایه، و یک لایه Dense (Linear) که تعداد واحدهای آن برابر با تعداد عمل‌های محیط است، بهره‌مند شده است. خروجی هر لایه ورودی لایه بعدی است. همچنین بهینه‌سازی که در شبکه عصبی عامل از آن استفاده شده است

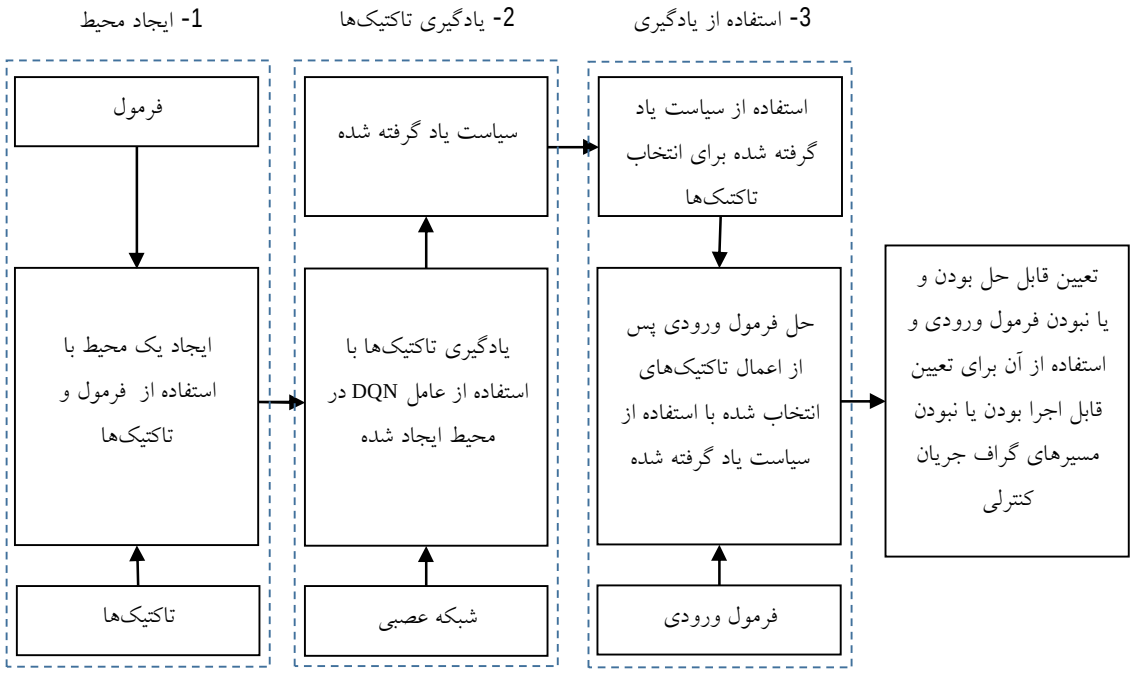
بهینه‌ساز Adam است. ورودی شبکه عصبی در حقیقت حالت محیط است و خروجی آن عملی است، که عامل باید طبق تجویز شبکه، در حالت فعلی انجام دهد. در واقع خروجی شبکه احتمال انتخاب هر عمل در حالت ورودی شبکه عصبی را تعیین می‌کند و عمل با احتمال بیشتر به معنای بهتر بودن آن عمل برای انجام در آن حالت است. شکل (4) این شبکه عصبی را نشان می‌دهد.

5.5. روش پیشنهادی در یک نگاه کلی

شکل (5) روند کلی روش پیشنهادی را نشان می‌دهد. همان گونه که در بخش قبل بیان شد، در این روش از الگوریتم DQN استفاده شده است. به منظور آموزش عامل DQN ابتدا یک فرمول ورودی در نظر گرفته می‌شود. با توجه به سنجه‌های قاعده آن فرمول، وضعیت محیط مشخص شده و به عنوان ورودی به شبکه عصبی داده می‌شود. عامل در هر گام زمانی یک عمل را (براساس مقدار اکتشاف و بهره‌برداری تعیین شده) انتخاب می‌کند و آن عمل را در محیط انجام می‌دهد. بدین ترتیب، فرمول با توجه به تاکتیک انتخابی بروزرسانی می‌شود. پاداش عملی که عامل انجام داده و مقدار تعیین‌کننده پایان دوره به عامل داده می‌شود. با توجه به اعمال تاکتیک انتخابی، بروز شدن فرمول و تغییر مقادیر سنجه‌های قاعده آن، عامل وارد حالت جدیدی از محیط می‌شود. چرخه فوق مجددا و تا زمان پایان یافتن دوره تکرار می‌شود. در صورتی که یک دوره به پایان برسد، محیط بازنشانی (Reset) می‌شود و عامل مجددا یک دوره جدید یادگیری را شروع می‌کند. عامل این کار را برای چندین دوره پیاپی تکرار می‌کند تا زمانی که یک سیاست بهینه را یاد بگیرد. بعد از اینکه عامل به یک سیاست بهینه دست یافت، می‌توان از این سیاست برای انتخاب تاکتیک‌ها برای فرمول‌های مختلف استفاده کرد.



شکل (4): شبکه عصبی استفاده شده در عامل یادگیری تقویتی عمیق



شکل (5): روند کلی روش پیشنهادی

6.5. مثالی از روش پیشنهادی

در این بخش روش پیشنهادی با ذکر مثالی شرح داده می‌شود. دستورات SMT شکل (6) را در نظر بگیرید. این دستورات به صورت یک فایل با پسوند smt2 قابل ذخیره‌سازی و نگهداری است. حل‌کننده Z3 برای حل این فرمول به صورت پیش‌فرض باید 513 عدد عملیات پایه انجام دهد که این عدد در واقع همان مقدار rlimit می‌باشد. تاکتیک‌های smt و qfnra-nlsat بهترین تاثیر را روی این فرمول دارند و اعمال هر کدام از آنها روی این فرمول، مقدار rlimit را به 17 کاهش می‌دهد. بعد از اعمال هر کدام از این تاکتیک‌ها، دستورات SMT به صورت تنها یک مقدار صحیح ساده خواهند شد که به صورت صریح قابل حل می‌باشد.

در ابتدا با استفاده از فایل حاوی دستورات SMT ارائه شده در شکل (6) و نیز تاکتیک‌های موجود در جدول (2)، حالت ابتدایی محیط مشخص می‌شود. جدول (4) تمامی عمل‌های ممکن و همچنین مقادیر rlimit و پاداش متناظر با هر عمل را که براساس مقدار rlimit محاسبه می‌شود نشان می‌دهد.

در طول یادگیری، هر زمانی که عامل تاکتیک‌های smt و qfnra-nlsat را به کار گیرد پاداش 5+ را که بیشترین میزان پاداش است دریافت خواهد کرد، چرا که این تاکتیک‌ها باعث می‌شوند فرمول به طور صریح قابل حل شود.

```
(set-info :smt-lib-version 2.6)
(set-logic QF_NIA)
(set-info :source |AProve team, see http://aprove.informatik.rwth-aachen.de/, submitted for SMT-COMP 2014|)
(set-info :category "industrial")
(set-info :status sat)
(declare-fun a__4 () Int)
(declare-fun a__2 () Int)
(declare-fun a__3 () Int)
(declare-fun a__5 () Int)
(assert (and (>= (+ (* a__4 a__2) (* (- 0 1) (* a__4 a__3)) (* (- 0 1) (* a__4 a__5 a__3)) (- 0 1)) 0) (>= (+ (* a__5 a__2) (* (- 0 1) a__2)) 0) (>= (+ a__3 (* (- 0 1) (* a__5 a__5 a__3))) 0) (>= a__4 0) (>= a__2 0) (>= a__3 0) (>= a__5 0)))
(check-sat)
(exit)
```

شکل (6): نمونه‌ای از دستورات صدق‌پذیری پیمانه نظریه‌ها برای شرح مثال از نحوه عملکرد روش پیشنهادی

جدول (4): عمل‌های ممکن در حالت ایجاد شده با فرمول شکل (6) و پاداش‌های متناظر با هر عمل

	عمل	مقدار rlimit بعد از انجام عمل در محیط	آیا این عمل فرمول ورودی را به طور صریح قابل حل می‌کند؟	آیا این عمل مقدار rlimit را کاهش می‌دهد؟	پاداش عمل
1	ackermannize_bv	513	خیر	خیر	-2
2	purify-arith	467	خیر	بلی	-1
3	reduce-bv-size	473	خیر	بلی	-1
4	solve-eqs	467	خیر	بلی	-1
5	propagate-values	473	خیر	بلی	-1
6	simplify	467	خیر	بلی	-1
7	sat	473	خیر	بلی	-1
8	qfnra-nlsat	17	بلی	بلی	+5
9	simplify(hoist_mul=True)	509	خیر	بلی	-1
10	bv1-blast	473	خیر	بلی	-1
11	simplify(pull_cheap_ite=True)	467	خیر	بلی	-1
12	simplify(som=True)	467	خیر	بلی	-1
13	aig(aig_per_assertion=True)	467	خیر	بلی	-1

ادامه جدول (4): عمل‌های ممکن در حالت ایجاد شده با فرمول شکل (6) و پاداش‌های متناظر با هر عمل

	عمل	مقدار rlimit بعد از انجام عمل در محیط	آیا این عمل فرمول ورودی را به طور صریح قابل حل می‌کند؟	آیا این عمل مقدار rlimit را کاهش می‌دهد؟	پاداش عمل
14	smt	17	بلی	بلی	+5
15	simplify(flat=True)	467	خیر	بلی	-1
16	propagate-values(push_ite_bv=True)	473	خیر	بلی	-1
17	simplify(elim_and=True)	467	خیر	بلی	-1
18	simplify(local_ctx=True)	467	خیر	بلی	-1
19	simplify(blast_distinct=True)	467	خیر	بلی	-1
20	aig	473	خیر	بلی	-1
21	elim-uncnstr	473	خیر	بلی	-1
22	max-bv-sharing	473	خیر	بلی	-1
23	bit-blast	473	خیر	بلی	-1

جدول (5) نمونه‌ای از حالت‌های محیط را، که در یک دوره از عملکرد، عامل در آنها قرار گرفته است نمایش می‌دهد. هر حالت از لیستی از مقادیر سنجه‌های قاعده تشکیل شده است. هر سطر جدول (5) یک حالت را نشان می‌دهد. حالت شماره 1 در جدول (5) حالت شروع است. در این دوره عامل نتوانسته هیچ یک از عمل‌های شماره 8 و 14 جدول (4) را انجام دهد و دوره را به پایان برساند و دلیل خاتمه یافتن این دوره به حداکثر رسیدن تعداد عمل‌های انجام شده در محیط است. در این مثال، عامل در این دوره تنها سه حالت متمایز را مشاهده کرده است، اما حالت‌ها ممکن است به صورت تکراری توسط عامل مشاهده شوند.

جدول (5): نمونه‌ای از حالت‌هایی که عامل در یک دوره در آنها قرار گرفته است

شماره حالت	حالت (سنجه‌ها)										
	Other probes	is-qfnia	is-unbounded	≥	*	-	+	num-arith-consts	num-consts	num-exprs	size
1	0	1	1	4	1	1	1	2	2	11	4
2	0	1	1	4	2	1	1	2	2	12	4
3	0	1	1	4	2	2	2	2	2	13	4

جدول (6) نمونه‌ای دیگر از حالت‌های محیط که در یک دوره دیگر عامل در آن حالت‌ها قرار گرفته است را نمایش می‌دهد. هر سطر جدول (6) نیز یک حالت را نشان می‌دهد. حالت شماره 1 در جدول (6)، حالت شروع است که معادل همان حالت 1 در جدول (5) است. حالت شماره 2 در جدول (6) حالت پایان است. در این دوره عامل موفق شده یکی از عمل‌های شماره 8 و 14 جدول (4) را در محیط انجام دهد و دوره را به پایان برساند. عامل در این دوره تنها چهار حالت متمایز را مشاهده کرده است.

جدول (6): نمونه‌ای دیگر از حالت‌هایی که عامل در یک دوره در آنها قرار گرفته است

شماره حالت	حالت (سنجه‌ها)										
	Other probes	is-qfnia	Is-qfbv	is-unbounded	≥	*	-	+	num-arith-consta	num-consts	num-exprs	size
1	0	1	0	1	4	1	1	1	2	2	11	4
2	0	0	1	0	0	0	0	0	0	0	0	0
3	0	1	0	1	4	2	1	1	2	2	12	4
4	0	1	0	1	4	2	2	2	2	2	13	4

6. آزمایش‌ها

به منظور بررسی روش پیشنهادی، منطق QF_NIA با مجموعه داده‌های AProVE[52] و منطق QF_NRA با مجموعه داده‌های hycomp[53] مورد استفاده قرار گرفته است. همانطور که در بخش 5.4 گفته شد، در روش پیشنهادی از عامل DQN استفاده شده است. پارامترهای این عامل در جدول (7) نشان‌داده شده‌اند. نکته‌ای که در مورد پارامتر memory size باید به آن توجه کرد این است که هر خانه حافظه باید حالت فعلی، حالت بعدی، عمل، میزان پاداش دریافت شده و نیز مقدار تعیین‌کننده پایان دوره را در خود نگه دارد. به جهت مشخص شدن کیفیت رفتار عامل DQN، یک عامل تصادفی[54] نیز در نظر گرفته شده که در هر مرحله، یک اقدام را به صورت کاملا تصادفی انتخاب و در محیط اِعمال می‌کند. در آزمایش‌ها، نتایج حاصل از رفتار عامل DQN با نتایج حاصل از رفتار عامل تصادفی مورد مقایسه قرار گرفته است.

فاز آموزش در هر منطق به صورت جداگانه و روی 50 فرمول انجام شده است. پس از آموزش، به ازای هر منطق به صورت جداگانه و روی 100 فرمول آزمایش صورت پذیرفته است. فرمول‌های مربوط به آموزش و آزمایش از مقاله [9] انتخاب شده است.

با توجه به مقدار rlimit و زمان حل فرمول‌های ورودی فاز آزمایش (بدون اعمال هیچ‌گونه تاکتیکی)، فرمول‌ها به سه دسته آسان[55]، معمولی[56] و سخت[57] تقسیم بندی شدند. در جدول (8) مقادیر مرزی rlimit برای هر منطق نشان داده شده است. توجه نمایید که در دسته سخت منطق QF_NIA جدول (8)، مقدار حداکثر با مهلت زمانی 5 دقیقه‌ای بدست آمده است.

در فاز آموزش، یک مهلت زمانی 5 ثانیه‌ای برای حل‌کننده در نظر گرفته شده است. در فاز آزمایش نیز مهلت زمانی 5 دقیقه‌ای برای حل‌کننده در نظر گرفته شده است. این مهلت‌های زمانی براساس زمان حل فرمول‌ها بدون اِعمال تاکتیک، در نظر گرفته شده‌اند. در صورتی که حل‌کننده نتواند در مهلت زمانی تعیین شده فرمول ورودی را حل کند، مقدار rlimit بدست آمده در آن مهلت زمانی در نظر گرفته می‌شود. هر دوره به انجام 23 عمل توسط عامل محدود شده و پس از آن دوره خاتمه می‌یابد. در طی آموزش، هر فرمول 5000 بار به عامل داده شده و در هر بار، عامل یک دوره کامل روی آن اقدام نموده و سعی در ساده‌سازی آن نموده است.

معیارهای مورد استفاده برای ارزیابی نتایج به شرح زیر هستند: (1) میزان افزایش سرعت حل‌کننده در حل فرمول و (2) میزان کاهش rlimit. تمامی آزمایش‌ها روی یک سیستم Core i7 با 64 گیگابات حافظه RAM انجام شده است. هر نتیجه ذکر شده، میانگین 10 بار اجراست. در ادامه نتایج آزمایش‌ها انجام پذیرفته به صورت جداگانه برای هر منطق آورده شده است.

1.6. نتایج آزمایش‌ها در منطق QF_NRA

همانطور که در جدول (9) می‌توان مشاهده کرد، در منطق QF_NRA با استفاده از یادگیری انجام شده، سرعت حل حل‌کننده را می‌توان تا 2/136 برابر افزایش داد. همچنین می‌توان مشاهده کرد که مقدار rlimit تا 451 برابر کاهش می‌یابد. با مقایسه سرعت و rlimit اندازه‌گیری شده برای دو عامل آموزش دیده و عامل تصادفی، می‌توان نتیجه گرفت که عملکرد عامل آموزش دیده بسیار بهتر از عامل تصادفی بوده است. با بررسی جدول (9) متوجه می‌شویم که با پیچیده‌تر و سخت‌تر شدن فرمول‌ها، عامل آموزش دیده با انتخاب عمل‌های مناسب، عملکرد بهتری از خود نشان داده است.

در جدول (10) نیز می‌توان میزان افزایش سرعت به ازای هر فرمول را به تفکیک مشاهده کرد.

---

52 https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NIA/tree/master/AProVE

53 https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NRA/tree/master/hycomp

54 Pure Chance

55 Easy

56 Normal

57 Hard

جدول (7): پارامترهای عامل یادگیری تقویتی عمیق

پارامتر	مقدار	توضیحات
discount factor (gamma)	0/9	میزان اهمیت پاداش‌های جدید نسبت به پاداش‌های قدیمی‌تر برای عامل را تعیین می‌کند
learning rate	0/001	نرخ یادگیری بهینه‌ساز Adam را تعیین می‌کند
epsilon	1/0	مبادله بین اکتشاف و بهره‌برداری را تعیین می‌کند که در ابتدا بیشترین مقدار را دارد (اولویت با اکتشاف است)
epsilon minimum	0/1	حداقل مقدار ممکن برای epsilon را تعیین می‌کند
epsilon decay	0/999	میزان کاهش epsilon را در فاصله مشخص شده توسط پارامترtarget update interval تعیین می‌کند
batch size	64	اندازه دسته‌های نمونه‌برداری از حافظه عامل را تعیین می‌کند
memory size	500000	اندازه حافظه عامل را تعیین می‌کند
target update interval	10	فاصله بین به‌روزرسانی شبکه عصبی و مقدار epsilon (برحسب گام زمانی) را تعیین می‌کند
training interval	10	فاصله بین آموزش شبکه عصبی (برحسب گام زمانی) را تعیین می‌کند
learning start	1000	گام زمانی شروع یادگیری را تعیین می‌کند

جدول (8): مقادیر مرزی RLIMIT برای دسته‌های فرمول‌های آزمایش

منطق	آسان		معمولی		سخت	
QF_NRA (hycomp)	حداقل 85	حداکثر 828	حداقل 818	حداکثر 1067	حداقل 1071	حداکثر 1816
QF_NIA (AProVE)	حداقل 230	حداکثر 30103	حداقل 5106	حداکثر 5932657	حداقل 1813636	حداکثر 145264443

جدول (9): نتایج آزمایش‌ها در منطق QF_NRA

دسته	با عامل آموزش دیده (DQN)			با عامل تصادفی			تعداد فایل در هر دسته
	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	
آسان	1/695 برابر	2/136 برابر	21 برابر	203 برابر	1/104 برابر	1/901 برابر	1/186 برابر	2/403 برابر	33
معمولی	1/743 برابر	2/07 برابر	201 برابر	266 برابر	1/045 برابر	1/642 برابر	1/185 برابر	2/579 برابر	34
سخت	1/883 برابر	2/067 برابر	263 برابر	451 برابر	1/097 برابر	1/795 برابر	1/186 برابر	2/403 برابر	33

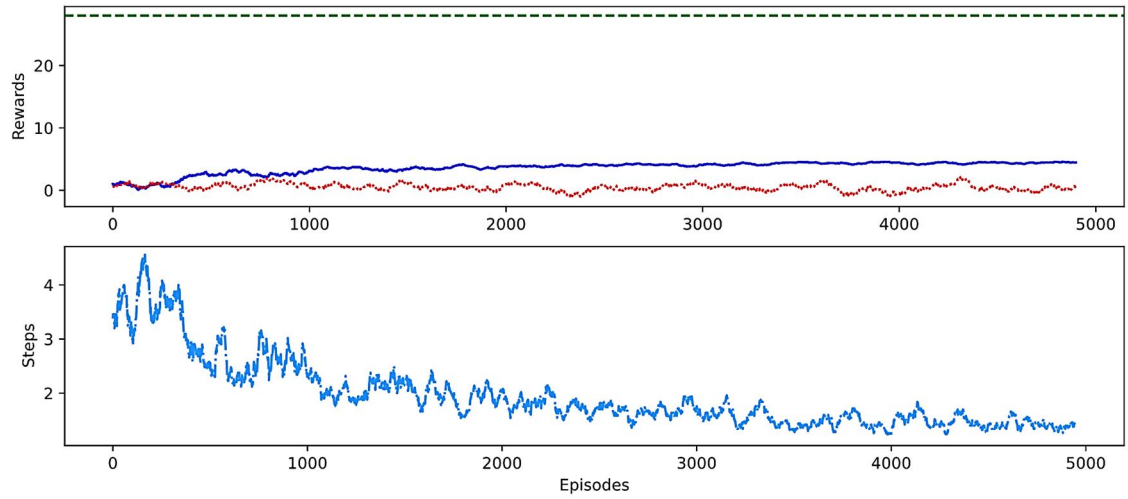
جدول (10): میزان افزایش سرعت در منطق QF_NRA به تفکیک فرمول

ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
1	ball_count_1d_plain.01.qfree_global_0.smt2	2/017 برابر	51	ball_count_2d_plain.02.qfree_global_0.smt2	1/893 برابر
2	ball_count_1d_plain.02.qfree_global_0.smt2	1/980 برابر	52	ball_count_1d_plain.01.redlog_global_0.smt2	1/861 برابر
3	ball_count_1d_plain.03.qfree_global_0.smt2	1/873 برابر	53	ball_count_2d_plain.03.qfree_global_0.smt2	1/953 برابر
4	ball_count_2d_hill_simple.01.qfree_global_0.smt2	1/865 برابر	54	ball_count_1d_plain.02.seq_lazy_lemmas_global_0.smt2	1/988 برابر
5	ball_count_2d_hill_simple.02.qfree_global_0.smt2	2/136 برابر	55	ball_count_1d_plain.04.redlog_global_0.smt2	1/993 برابر
6	ball_count_1d_plain.01.seq_lazy_global_0.smt2	1/879 برابر	56	ball_count_1d_plain.03.seq_lazy_linear_enc_lemmas_global_0.smt2	2/060 برابر
7	ball_count_1d_plain.02.seq_lazy_global_0.smt2	1/767 برابر	57	ball_count_2d_hill_simple.01.redlog_global_0.smt2	1/961 برابر
8	ball_count_1d_plain.01.seq_lazy_linear_enc_global_0.smt2	1/793 برابر	58	ball_count_2d_hill_simple.02.redlog_global_0.smt2	2 برابر
9	ball_count_1d_plain.02.seq_lazy_linear_enc_global_0.smt2	1/828 برابر	59	ball_count_1d_plain.04.seq_lazy_linear_enc_lemmas_global_0.smt2	1/968 برابر
10	ball_count_2d_hill_simple.10.qfree_global_0.smt2	1/971 برابر	60	ball_count_2d_hill_simple.03.redlog_global_0.smt2	2/009 برابر
11	ball_count_1d_plain.10.seq_lazy_global_0.smt2	1/695 برابر	61	ball_count_2d_hill.01.seq_lazy_linear_enc_global_0.smt2	1/987 برابر
12	ball_count_1d_plain.10.seq_lazy_linear_enc_global_0.smt2	1/804 برابر	62	ball_count_2d_hill.02.seq_lazy_linear_enc_global_0.smt2	1/994 برابر
13	ball_count_2d_plain.02.seq_lazy_linear_enc_global_0.smt2	1/786 برابر	63	ball_count_2d_hill_simple.05.redlog_global_0.smt2	1/970 برابر
14	ball_count_1d_plain.04.seq_lazy_global_0.smt2	1/810 برابر	64	ball_count_2d_hill.04.seq_lazy_linear_enc_global_0.smt2	2/001 برابر
15	ball_count_2d_hill_simple.10.seq_lazy_linear_enc_global_0.smt2	1/901 برابر	65	ball_count_2d_hill.01.seq_lazy_global_0.smt2	2/035 برابر
16	ball_count_2d_plain.01.seq_lazy_global_0.smt2	1/926 برابر	66	ball_count_2d_hill.03.seq_lazy_global_0.smt2	1/989 برابر
17	ball_count_2d_plain.02.seq_lazy_global_0.smt2	1/875 برابر	67	ball_count_2d_hill.05.seq_lazy_linear_enc_global_0.smt2	1/994 برابر
18	ball_count_2d_hill_simple.10.seq_lazy_global_0.smt2	1/918 برابر	68	ball_count_1d_plain.01.seq_lazy_lemmas_global_0.smt2	2/012 برابر
19	ball_count_2d_plain.10.seq_lazy_global_0.smt2	1/797 برابر	69	ball_count_2d_plain.04.qfree_global_0.smt2	1/883 برابر
20	ball_count_1d_plain.02.redlog_global_0.smt2	1/986 برابر	70	ball_count_1d_plain.03.seq_lazy_lemmas_global_0.smt2	1/989 برابر
21	ball_count_1d_plain.10.redlog_global_0.smt2	1/926 برابر	71	ball_count_2d_plain.05.qfree_global_0.smt2	2/056 برابر
22	ball_count_1d_plain.02.seq_lazy_linear_enc_lemmas_global_0.smt2	1/986 برابر	72	ball_count_1d_plain.04.seq_lazy_lemmas_global_0.smt2	1/971 برابر
23	ball_count_1d_plain.10.seq_lazy_linear_enc_lemmas_global_0.smt2	1/908 برابر	73	ball_count_1d_plain.05.seq_lazy_lemmas_global_0.smt2	1/948 برابر
24	ball_count_1d_plain.10.seq_lazy_lemmas_global_0.smt2	1/970 برابر	74	ball_count_2d_hill_simple.01.seq_lazy_linear_enc_lemmas_global_0.smt2	1/973 برابر
25	ball_count_2d_hill_simple.10.redlog_global_0.smt2	1/960 برابر	75	ball_count_2d_plain.01.seq_lazy_linear_enc_lemmas_global_0.smt2	1/942 برابر

ادامه جدول (10): میزان افزایش سرعت در منطق QF_NRA به تفکیک فرمول

ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
26	ball_count_2d_hill.10.seq_lazy_linear_enc_global_0.smt2	2/001 برابر	76	ball_count_2d_hill_simple.03.seq_lazy_linear_enc_lemmas_global_0.smt2	2/016 برابر
27	ball_count_2d_hill_simple.10.seq_lazy_linear_enc_lemmas_global_0.smt2	1/980 برابر	77	ball_count_2d_plain.03.seq_lazy_linear_enc_lemmas_global_0.smt2	2/045 برابر
28	ball_count_2d_plain.10.seq_lazy_linear_enc_lemmas_global_0.smt2	1/983 برابر	78	ball_count_2d_hill_simple.04.seq_lazy_linear_enc_lemmas_global_0.smt2	1/949 برابر
29	ball_count_2d_hill.10.seq_lazy_global_0.smt2	1/995 برابر	79	ball_count_2d_plain.05.seq_lazy_linear_enc_lemmas_global_0.smt2	1/955 برابر
30	ball_count_2d_hill_simple.10.seq_lazy_lemmas_global_0.smt2	1/976 برابر	80	ball_count_2d_hill.05.seq_lazy_global_0.smt2	2/023 برابر
31	ball_count_2d_plain.10.seq_lazy_lemmas_global_0.smt2	2/022 برابر	81	ball_count_2d_hill_simple.01.seq_lazy_lemmas_global_0.smt2	2/022 برابر
32	ball_count_2d_hill.10.seq_lazy_linear_enc_lemmas_global_0.smt2	1/980 برابر	82	ball_count_2d_hill_simple.02.seq_lazy_lemmas_global_0.smt2	2/001 برابر
33	ball_count_2d_plain.10.redlog_global_0.smt2	1/959 برابر	83	ball_count_2d_plain.01.seq_lazy_lemmas_global_0.smt2	1/949 برابر
34	ball_count_1d_plain.04.qfree_global_0.smt2	2/049 برابر	84	ball_count_2d_hill_simple.03.seq_lazy_lemmas_global_0.smt2	1/887 برابر
35	ball_count_2d_hill_simple.03.qfree_global_0.smt2	1/824 برابر	85	ball_count_2d_hill_simple.04.seq_lazy_lemmas_global_0.smt2	1/991 برابر
36	ball_count_2d_hill_simple.01.seq_lazy_linear_enc_global_0.smt2	1/972 برابر	86	ball_count_2d_plain.04.seq_lazy_lemmas_global_0.smt2	1/970 برابر
37	ball_count_2d_hill_simple.04.qfree_global_0.smt2	1/987 برابر	87	ball_count_2d_slope.02.seq_lazy_linear_enc_global_0.smt2	2/056 برابر
38	ball_count_1d_plain.05.qfree_global_0.smt2	2/070 برابر	88	ball_count_2d_slope.03.seq_lazy_linear_enc_global_0.smt2	2/030 برابر
39	ball_count_2d_plain.01.seq_lazy_linear_enc_global_0.smt2	1/864 برابر	89	ball_count_2d_hill.02.seq_lazy_linear_enc_lemmas_global_0.smt2	1/988 برابر
40	ball_count_1d_plain.04.seq_lazy_linear_enc_global_0.smt2	1/855 برابر	90	ball_count_2d_slope.04.seq_lazy_linear_enc_global_0.smt2	2/009 برابر
41	ball_count_2d_plain.03.seq_lazy_linear_enc_global_0.smt2	1/765 برابر	91	ball_count_2d_hill.03.seq_lazy_linear_enc_lemmas_global_0.smt2	1/984 برابر
42	ball_count_2d_hill_simple.03.seq_lazy_global_0.smt2	1/956 برابر	92	ball_count_2d_hill.04.seq_lazy_linear_enc_lemmas_global_0.smt2	2/023 برابر
43	ball_count_2d_hill_simple.05.qfree_global_0.smt2	1/892 برابر	93	ball_count_2d_slope.05.seq_lazy_linear_enc_global_0.smt2	1/977 برابر
44	ball_count_1d_plain.05.seq_lazy_global_0.smt2	1/764 برابر	94	ball_count_2d_plain.01.redlog_global_0.smt2	2/067 برابر
45	ball_count_2d_hill_simple.04.seq_lazy_linear_enc_global_0.smt2	2/007 برابر	95	ball_count_2d_hill.05.seq_lazy_linear_enc_lemmas_global_0.smt2	2/028 برابر
46	ball_count_1d_plain.05.seq_lazy_linear_enc_global_0.smt2	1/743 برابر	96	ball_count_2d_plain.04.redlog_global_0.smt2	2/007 برابر
47	ball_count_2d_hill_simple.04.seq_lazy_global_0.smt2	1/994 برابر	97	ball_count_2d_plain.05.redlog_global_0.smt2	2/007 برابر
48	ball_count_2d_hill_simple.05.seq_lazy_linear_enc_global_0.smt2	1/877 برابر	98	ball_count_2d_hill.01.seq_lazy_lemmas_global_0.smt2	1/981 برابر
49	ball_count_2d_plain.05.seq_lazy_global_0.smt2	2/008 برابر	99	ball_count_2d_hill.02.seq_lazy_lemmas_global_0.smt2	1/925 برابر
50	ball_count_2d_plain.01.qfree_global_0.smt2	1/967 برابر	100	ball_count_2d_hill.03.seq_lazy_lemmas_global_0.smt2	2/011 برابر

شکل (7) به صورت نمونه روند یادگیری عامل DQN در محیط ایجاد شده در منطق QF_NRA را نشان می‌دهد. نمودار بالای شکل میزان پاداش دریافتی در هر دوره از یادگیری عامل DQN (خط صاف آبی رنگ)، میزان پاداش دریافتی در هر دوره برای عامل تصادفی (خط نقطه چین قرمز رنگ) و میزان پاداش حداکثری که عامل می‌تواند دریافت کند (خط حاشور سبز رنگ) را نشان می‌دهد. میزان پاداش حداکثری از طریق محاسبه پاداش برای ترکیب‌های مختلف عمل‌ها به دست آمده است. نمودار پایین شکل (7) تعداد گام‌های انجام شده توسط عامل DQN در هر دوره را نشان می‌دهد. همانطور که در شکل (7) مشخص است برخلاف عامل تصادفی که نمی‌تواند پاداش دریافتی خود را افزایش دهد، عامل DQN در طول یادگیری توانسته است پاداش دریافتی خود را افزایش دهد. با توجه به نمودار پایین شکل (7)، عامل DQN توانسته است همزمان با افزایش پاداش دریافتی، تعداد عمل‌های انجام شده در هر دوره را نیز کاهش دهد و از آنجایی که هر عمل معادل یک تاکتیک (و پارامترهای آن) است، بنابراین عامل به درستی آموزش دیده است و عمل‌ها (تاکتیک‌های) مناسب را انتخاب کرده و انجام می‌دهد و عمل اضافی انجام نمی‌دهد.



شکل (7): نمودار روند یادگیری عامل یادگیری تقویتی عمیق در منطق QF_NRA

2.6. نتایج آزمایش‌ها در منطق QF_NIA

همان‌طور که در جدول (11) می‌توان مشاهده کرد، در منطق QF_NIA با استفاده از یادگیری انجام شده، سرعت حل حل‌کننده را می‌توان تا 4/744 برابر افزایش داد. همچنین می‌توان مشاهده کرد که مقدار rlimit تا 459060 برابر کاهش می‌یابد. در برخی موارد عامل آموزش دیده نتوانسته مقدار rlimit را کاهش و یا سرعت حل را افزایش دهد. دلیل این امر این است که عامل به اندازه کافی آموزش ندیده است و نیاز به آموزش بیشتر دارد. همچنین محدودیت زمانی در نظر گرفته شده نیز در این امر تاثیرگذار است. در جدول (12) نیز می‌توان میزان افزایش سرعت به ازای هر فرمول را به تفکیک مشاهده کرد.

شکل (8) نیز به صورت نمونه روند یادگیری عامل DQN در محیط ایجاد شده در منطق QF_NIA را نشان می‌دهد. نمودار بالای شکل میزان پاداش دریافتی در هر دوره از یادگیری عامل DQN (خط صاف آبی رنگ)، میزان پاداش دریافتی در هر دوره برای عامل تصادفی (خط نقطه چین قرمز رنگ) و میزان پاداش حداکثری که عامل می‌تواند دریافت کند (خط حاشور سبز رنگ) را نشان می‌دهد. در این منطق نیز، میزان پاداش حداکثری از طریق محاسبه پاداش برای ترکیب‌های مختلف عمل‌ها به دست آمده است. نمودار پایین شکل (8) نیز تعداد گام‌های انجام شده توسط عامل DQN در هر دوره را نشان می‌دهد. در این منطق نیز عامل DQN بر خلاف عامل تصادفی، در طول یادگیری توانسته است پاداش دریافتی را بیشینه کند. همچنین

عامل DQN توانسته است تعداد عمل‌های انجام شده در هر اپیزود را کاهش دهد و بنابراین عامل به درستی آموزش دیده است و عمل‌ها (تاکتیک‌های) مناسب را انجام می‌دهد و عمل اضافی انجام نمی‌دهد.

جدول (11): نتایج آزمایش‌ها در منطق QF_NIA

دسته	با عامل آموزش دیده (DQN)				با عامل تصادفی				تعداد فایل در هر دسته
	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	
آسان	1 برابر	2/099 برابر	13 برابر	6098 برابر	0/78 برابر	1/263 برابر	0/99 برابر	1/434 برابر	39
معمولی	1/114 برابر	4/744 برابر	306 برابر	459060 برابر	0/969 برابر	1/216 برابر	0/949 برابر	1/429 برابر	38
سخت	1/947 برابر	1/43 برابر	1 برابر	138553 برابر	0/17 برابر	1/145 برابر	0/984 برابر	1/111 برابر	23

جدول (12): میزان افزایش سرعت در منطق QF_NIA به تفکیک فرمول

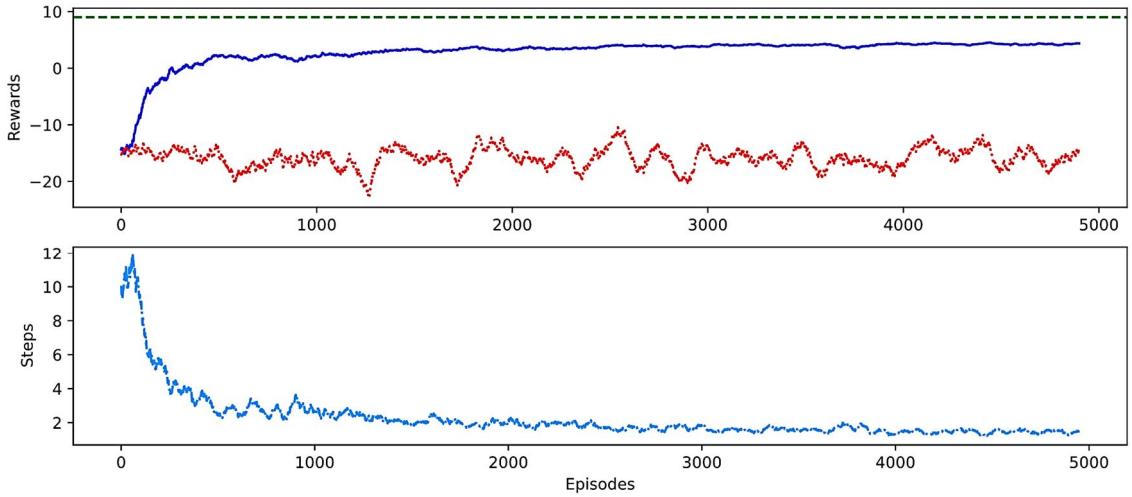
ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
1	aproveSMT106352422603502658.smt2	1/5 برابر	51	aproveSMT201351602224939005.smt2	1/631 برابر
2	aproveSMT110406245392905194.smt2	1 برابر	52	aproveSMT243075587948004402.smt2	2 برابر
3	aproveSMT164281113689408140.smt2	1/632 برابر	53	aproveSMT752462605509726678.smt2	1/267 برابر
4	aproveSMT237454731236054743.smt2	1/667 برابر	54	aproveSMT125187419356719603.smt2	1/374 برابر
5	aproveSMT263445627778943920.smt2	1/333 برابر	55	aproveSMT233295163504864559.smt2	1/260 برابر
6	aproveSMT4726660327701427.smt2	1/531 برابر	56	aproveSMT991944918242929447.smt2	1/968 برابر
7	aproveSMT191275437130012394.smt2	1/964 برابر	57	aproveSMT358912782933805782.smt2	2/033 برابر
8	aproveSMT172200405223484420.smt2	1/488 برابر	58	aproveSMT206305505307433843.smt2	1/726 برابر
9	aproveSMT32025504445261213.smt2	1/636 برابر	59	aproveSMT265925057088562932.smt2	1/470 برابر
10	aproveSMT106628954746449373.smt2	2/023 برابر	60	aproveSMT257699921050028985.smt2	1/463 برابر
11	aproveSMT141510234371783052.smt2	1/919 برابر	61	aproveSMT474375145462252696.smt2	1/990 برابر

ادامه جدول (12): میزان افزایش سرعت در منطق QF_NIA به تفکیک فرمول

ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
12	aproveSMT335045203561853343.smt2	1/786 برابر	62	aproveSMT3146499174646083 53.smt2	2/029 برابر
13	aproveSMT199924319808952062.smt2	2/089 برابر	63	aproveSMT2692833200093845 99.smt2	1/114 برابر
14	aproveSMT465387089974364589.smt2	1/709 برابر	64	aproveSMT2086179981585928 34.smt2	1/940 برابر
15	aproveSMT475511664945327223.smt2	1/930 برابر	65	aproveSMT4258100025895060 85.smt2	1/254 برابر
16	aproveSMT338922425342401784.smt2	1/802 برابر	66	aproveSMT2057119268416200 90.smt2	1/818 برابر
17	aproveSMT18817844084034737.smt2	1/917 برابر	67	aproveSMT2268940209867039 71.smt2	1/776 برابر
18	aproveSMT426987210793294180.smt2	1/989 برابر	68	aproveSMT2834872983082843 66.smt2	1/336 برابر
19	aproveSMT299708961479900596.smt2	2/080 برابر	69	aproveSMT2011322022945986 18.smt2	1/745 برابر
20	aproveSMT216462801678736982.smt2	2/019 برابر	70	aproveSMT3429881946768792 81.smt2	1/982 برابر
21	aproveSMT374020975285891691.smt2	2/047 برابر	71	aproveSMT2777950454192646 94.smt2	1/770 برابر
22	aproveSMT233357233309295005.smt2	2/073 برابر	72	aproveSMT1309119525253885 58.smt2	1/585 برابر
23	aproveSMT236580308612717708.smt2	1/889 برابر	73	aproveSMT4583004905830627 27.smt2	1/483 برابر
24	aproveSMT11071903920252280.smt2	1/942 برابر	74	aproveSMT3904318256168740 15.smt2	2/034 برابر
25	aproveSMT297536522076030542.smt2	2/081 برابر	75	aproveSMT2719239440667886 66.smt2	1/533 برابر
26	aproveSMT359820136277819026.smt2	1/884 برابر	76	aproveSMT2335122113415398 5.smt2	1/797 برابر
27	aproveSMT405002793410787217.smt2	1/949 برابر	77	aproveSMT3253307178636335 4.smt2	1/412 برابر
28	aproveSMT388188197435784909.smt2	1/983 برابر	78	aproveSMT2056637576938305 08.smt2	1 برابر
29	aproveSMT97032048972614298.smt2	1/937 برابر	79	aproveSMT3539224193204171 26.smt2	1 برابر
30	aproveSMT309344887399351087.smt2	1/920 برابر	80	aproveSMT1918662598883852.smt2	1 برابر
31	aproveSMT7903591660074047.smt2	2/099 برابر	81	aproveSMT4096503054497268 32.smt2	1 برابر
32	aproveSMT221654206070531325.smt2	1/910 برابر	82	aproveSMT4419963961808927 64.smt2	1/278 برابر
33	aproveSMT185813161521293895.smt2	2/082 برابر	83	aproveSMT2655564333808112 08.smt2	1/430 برابر
34	aproveSMT395451549965794310.smt2	2/087 برابر	84	aproveSMT2936141580541588 32.smt2	1 برابر
35	aproveSMT166207192320126878.smt2	1/972 برابر	85	aproveSMT3341809707980873 84.smt2	1/101 برابر
36	aproveSMT479758788885008279.smt2	2/074 برابر	86	aproveSMT4631923779606371 55.smt2	1 برابر

ادامه جدول (12): میزان افزایش سرعت در منطق QF_NIA به تفکیک فرمول

ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
37	aproveSMT346224908798084830.smt2	2/006 برابر	87	aproveSMT451682996481631551.smt2	1 برابر
38	aproveSMT414771145475661582.smt2	1/959 برابر	88	aproveSMT463905008853206218.smt2	1 برابر
39	aproveSMT256227000274928233.smt2	1/926 برابر	89	aproveSMT868126669190659 88.smt2	1 برابر
40	aproveSMT101417910736011502.smt2	4/038 برابر	90	aproveSMT143564062931235708.smt2	1 برابر
41	aproveSMT379723609808799032.smt2	1/401 برابر	91	aproveSMT174156337543845992.smt2	1 برابر
42	aproveSMT324761792014523705.smt2	1/227 برابر	92	aproveSMT82980353335522103.smt2	1 برابر
43	aproveSMT201343392445663280.smt2	3/446 برابر	93	aproveSMT433457179563078955.smt2	1 برابر
44	aproveSMT370632052300982227.smt2	2/877 برابر	94	aproveSMT397893769027547 99.smt2	1 برابر
45	aproveSMT258682010928380630.smt2	1/704 برابر	95	aproveSMT365922853089376869.smt2	1 برابر
46	aproveSMT474292193369835 50.smt2	1/655 برابر	96	aproveSMT109680382490054156.smt2	1 برابر
47	aproveSMT341225430958194141.smt2	1/865 برابر	97	aproveSMT388283012372597473.smt2	1 برابر
48	aproveSMT14682641666560919.smt2	1/543 برابر	98	aproveSMT453198363063263721.smt2	1 برابر
49	aproveSMT337115653666620019.smt2	1/971 برابر	99	aproveSMT185857169594433069.smt2	1 برابر
50	aproveSMT448542283443020288.smt2	4/744 برابر	100	aproveSMT407976301189716929.smt2	1 برابر



شکل (8): نمودار روند یادگیری عامل یادگیری تقویتی عمیق در منطق QF_NIA

3.6. مقایسه نتایج آزمایش‌ها

با استفاده از جدول (13) می‌توان نتایج آزمایش‌ها را مقایسه کرد. با بررسی جدول (13) می‌توان متوجه شد که در منطق QF_NRA عامل تصادفی به این علت که تعداد تاکتیک‌های موثر بر فرمول‌های این منطق نسبت به منطق‌های دیگر بیشتر است و هر چه تعداد تاکتیک‌های موثر بیشتر شود، احتمال انتخاب تاکتیک موثر توسط عامل تصادفی نیز بیشتر می‌شود، عملکرد نسبتا خوبی از خود نشان داده است. اما با این وجود عامل آموزش دیده در هر دو منطق خیلی بهتر از عامل تصادفی عمل کرده است.

جدول (13): مقایسه نتایج آزمایش‌ها

منطق	با عامل آموزش دیده (DQN)				با عامل تصادفی			
	حداقل افزایش سرعت	حداکثر افزایش سرعت	حداقل میزان کاهش rlimit	حداکثر میزان کاهش rlimit	حداقل افزایش سرعت	حداکثر افزایش سرعت	حداقل میزان کاهش rlimit	حداکثر میزان کاهش rlimit
QF_NRA	1/695 برابر	2/136 برابر	21 برابر	451 برابر	1/045 برابر	1/901 برابر	1/185 برابر	2/579 برابر
QF_NIA	1 برابر	4/744 برابر	1 برابر	459060 برابر	0/17 برابر	1/263 برابر	0/984برابر	1/434 برابر

7. نتیجه‌گیری

Z3 ابزاری است که در کاربردهای متفاوتی در حوزه‌های مختلف دارد که یکی از آنها تشخیص مسیرهای غیرقابل اجرا است. بنابراین افزایش سرعت عملکرد این ابزار می‌تواند موجب بهبودهایی در حوزه‌های کاربرد آن نیز شود. برای مثال افزایش سرعت حل‌کننده‌های SMT می‌تواند کار را برای استفاده از گراف جریان کنترلی بین‌رویه‌ای آسان‌تر کند که در نتیجه آن می‌توان مسیرهای غیرقابل اجرای بشیتری را تشخیص داد. در این مقاله نشان داده شد که می‌توان با استفاده از یادگیری تقویتی، به ازای هر دسته از فرمول‌ها، تاکتیک‌های مناسبی را در حل کننده Z3 یاد گرفت، که استفاده از آنها منجر به حل سریع‌تر فرمول‌ها بشود. در روش پیشنهادی مقاله، با استفاده از عامل DQN تاکتیک‌های مختلف حل‌کننده Z3 یاد گرفته می‌شوند و سپس براساس یادگیری انجام شده تاکتیک‌ها انتخاب و روی فرمول‌های مختلف اعمال می‌شوند تا سرعت حل‌کننده Z3 افزایش یابد. در آزمایش‌هایی که در منطق‌های QF_NRA و QF_NIA انجام شد، نشان داده شد که با این روش سرعت حل‌کننده Z3 تا 4/7 برابر افزایش می‌یابد. در این مقاله یک محیط برای تاکتیک‌های Z3 طراحی و ارائه شد که محققان می‌توانند از آن استفاده کنند[58]. در طراحی محیط تلاش بر این بود که به گونه‌ای طراحی شود که قابلیت به کارگیری در الگوریتم‌های مختلف یادگیری تقویتی را داشته باشد. به همین دلیل این محیط همانند محیط‌های کتابخانه OpenAI Gym [25] طراحی و ارائه گردید.

تعارض منافع: نویسندگان اعلام می‌کنند که هیچ تعارض منافعی ندارند.

[58] https://github.com/majidfeyzi/Z3-tactics-learning

مراجع

[1] P. Ammann and J. Offutt, Introduction to Software Testing. Cambridge, U.K.: Cambridge Univ. Press, 2016, doi: 10.1017/9781316771273.

[2] B. Beizer, Software Testing Techniques. New York, NY, USA: Van Nostrand Reinhold, 1990.

[3] H. Farzaneh, S. Bakhshayeshi, R. Ebrahimi Atani, and A. Shahbahrami, "A survey on test data generation techniques based on Mutation Testing," Soft Comput. J., vol. 2, no. 1, pp. 72-85, 2013, dor: 20.1001.1.23223707.1392.2.1.61.5 [In Persian].

[4] Y.-W. Wang, Y. Xing, and X.-Z. Zhang, "A method of path feasibility judgment based on symbolic execution and range analysis," Int. J. Future Gener. Commun. Netw., vol. 7, no. 3, pp. 205-212, 2014, doi: 10.14257/ijfgcn.2014.7.3.19.

[5] H. Zhu, D. Jin, Y. Gong, Y. Xing, and M. Zhou, "Detecting interprocedural infeasible paths based on unsatisfiable path constraint patterns," IEEE Access, vol. 7, pp. 15040-15055, 2019, doi: 10.1109/ACCESS.2019.2894593.

[6] S. Jiang et al., "An approach for detecting infeasible paths based on a SMT solver," IEEE Access, vol. 7, pp. 69058-69069, 2019, doi: 10.1109/ACCESS.2019.2918558.

[7] L. de Moura and N. Bjorner, "Z3: An efficient SMT solver," in Proc. Int. Conf. Tools Algorithms Constr. Anal. Syst., 2008, pp. 337-340, doi: 10.1007/978-3-540-78800-3_24.

[8] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in Handbook of Model Checking, Springer, 2018, pp. 305-343, doi: 10.1007/978-3-319-10575-8_11.

[9] M. Balunovic, P. Bielik, and M. Vechev, "Learning to solve SMT formulas," Adv. Neural Inf. Process. Syst., vol. 31, 2018.

[10] S. Anand, P. Godefroid, and N. Tillmann, "Demand-driven compositional symbolic execution," in Proc. Int. Conf. Tools Algorithms Constr. Anal. Syst., 2008, pp. 367-381, doi: 10.1007/978-3-540-78800-3_28.

[11] J. Scott, F. Mora, and V. Ganesh, "BanditFuzz: Fuzzing SMT solvers with reinforcement learning," University of Waterloo, Waterloo Research, 2020.

[12] S. Jeon and J. Moon, "Dr. PathFinder: Hybrid fuzzing with deep reinforcement concolic execution toward deeper path-first search," Neural Comput. Appl., vol. 34, pp. 10731-10750, 2022, doi: 10.1007/s00521-022-07008-8.

[13] M. Hajibaba and S. Parsa, "Software Fault Localization using Cross Entropy and N-gram Models," Soft Comput. J., vol. 2, no. 1, pp. 44-59, 2013, dor: 20.1001.1.23223707.1392.2.1.59.3 [In Persian].

[14] S. Ding, H.B.K. Tan, and K.P. Liu, "A survey of infeasible path detection," in Proc. 7th Int. Conf. Eval. Novel Approaches Softw. Eng. (ENASE), 2012, pp. 43-52.

[15] S. Jang, H.-Y. Kim, Y.-H. Choi, and T.-M. Chung, "A study of advanced hybrid execution using reverse traversal," in Proc. 2011 Int. Conf. Inf. Manag. Innov. Manag. Ind. Eng., vol. 2, 2011, pp. 557-559, doi: 10.1109/ICIII.2011.278.

[16] H. Ghorbani Moghadam, B. Jamasb, and H. Dehdashti Jahromi, "Review on security requirements in the software production process," Soft Comput. J., vol. 10, no. 2, pp. 72-83, 2022, doi: 10.22052/scj.2022.242857.0 [In Persian]

[17] B. Barhoush and I. Alsmadi, "Infeasible paths detection using static analysis," Res. Bull. Jordan ACM, vol. 2, no. 3, pp. 120-126, 2013.

[18] N. Malevris, D. Yates, and A. Veevers, "Predictive metric for likely feasibility of program paths," Inf. Softw. Technol., vol. 32, no. 2, pp. 115-118, 1990, doi: 10.1016/0950-5849(90)90110-D.

[19] D. Yates and N. Malevris, "Reducing the effects of infeasible paths in branch testing," ACM SIGSOFT Softw. Eng. Notes, vol. 14, no. 8, pp. 48-54, 1989, doi: 10.1145/75309.75315.

[20] SMT-LIB, Accessed: Feb. 6, 2022, [Online]. Available: https://smtlib.cs.uiowa.edu/.

[21] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction. London, U.K.: The MIT Press, 2018.

[22] E.J. Weyuker, "The applicability of program schema results to programs," Int. J. Comput. Inf. Sci., vol. 8, no. 5, pp. 387-403, 1979, doi: 10.1007/BF00995175.

[23] D. Gong and X. Yao, "Automatic detection of infeasible paths in software testing," IET Softw., vol.

4, no. 5, pp. 361-370, 2010, doi: 10.1049/iet-sen.2009.0092.

[24] D. Kundu, M. Sarma, and D. Samanta, “A UML model-based approach to detect infeasible paths,” J. Syst. Softw., vol. 107, pp. 71-92, 2015, doi: 10.1016/j.jss.2015.05.007.

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, “OpenAI Gym,” arXiv preprint, arXiv:1606.01540, 2016.

[26] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, pp. 529-533, 2015, doi: 10.1038/nature14236.