التحقق من أنواع البيانات المجردة (منهج هجين)

ناهد احمد على [1] و على الميلي[2]

[1] قسم علوم الحاسوب- جامعة الجزيرة – السودان

[2] المعهد التقني بنيو جيرسي- نيو جيرسي- الولايات المتحدة الامريكية

ملخص: *في الأونة الأخيرة، أخذت المنتجات البرمجية وظائف حرجة بصورة متزايدة، ونمت بسرعة فائقة وأصبحت كبيرة ومعقدة؛ مما جعل الحاجة إلى التأكد من جودتها تزيد بشكل ملحوظ. ويظل التحدي قائما في مجال التحقق من المنتجات البرمجية ذات الحجم والتعقيد؛ إذ حاولت بحوث عديدة حل هذه المشكلة، ولكن محاولاتها باءت بالفشل. وفي هذه الورقة نناقش الباحثين منهجاً هجيناً للتحقق من صحة أنواع البيانات المجردة في مقابل المواصفات البديهية المكتوبة بشكل منهجي مستخدمين ترميزاً يشبة مواصفات التتبع ، ويسمى بالمواصفات البديهية. وهذا المنهج يركز على استخدام تقنيات تحليلية وتجريبية بطريقة تكميلية تفادياً لعيوب استخدام كل تقنية بشكل منفرد.*

الكلمات المفتاحية : *أنواع البيانات المجردة، التحقق من البرامج، اختبار البرامج، منطق هوار، المواصفات البديهية.*

١. مقدمة

أخذت المنتجات البرمجية تؤدي بصورة متزايدة وظائف مهمة في عالمنا، وأصبح التحقق من صحتها وإعتماديتها ينمو بشكل متزايد. وكلما زاد حجمها وتعقيدها أصبح التحقق من صحتها أكثر تعقيداً. وقد سبب هذا الأمر فجوة تقنية واسعة تحاول بحوث البرمجيات الحالية سدها. وما زال التحدي قائماً - رغم مضي عقود عديدة من البحث - في مجال التحقق من المنتجات البرمجية ذات الحجم الصناعي؛ إذ لم تتمكن البحوث من إنجازه على مستوى كافٍ من الدقة والشمول لان الطرق المطورة في المعامل البحثية لا ترتقي إلى المشاكل في الحجم الصناعي، والتقنيات المستخدمة في الممارسات الصناعية لا تفي بالمعايير المتوقعة من صناعة البرمجيات.

تقترح هذه الورقة منهجاً انتقائياً يستخدم تشكيلة من الطرق، حيث تعطي كل طريقة عائداً جيداً في الاستثمار في بعض جوانب التحقق على حساب جوانب أخرى. واستناداً إلى قانون تناقص المنفعة، فإن تطبيق مثل هذا المنهج يمكننا من زيادة أثر جهدنا في التحقق إلى أعلى حد. وبالتحديد، نحن نستخدم طرقاً للتحقق من البرامج وطرقاً لاختبارها للتحقق من صحة تطبيق أنواع البيانات المجردة (ADT's) مقابل مواصفات أنواع البيانات المجردة المكتوبة بترميز يشبه مواصفات التتبع [17] يسمى بالمواصفات البديهية.

٢. تقنيات من أجل التحقق المنهجي من البرمجيات

يمكن تعريف التحقق المنهجي بأنه مجموعة من التقنيات يستخدمها مطورو البرامج لتصميم أنظمة ذات عمليات يُعتمد عليها. فهو يُكمل الاختبار ويجب استخدامه مقترناً معه لزيادة الاعتمادية عند تطوير نظام ما. من ناحية أخرى يعد التحقق المنهجي أداة أفضل من الاختبار لأنه شامل ويُحسن من معرفة النظام. ولكن، من السمات غير المفضلة للتحقق المنهجي أنه صعب ويستغرق وقتاً طويلاً [4].

يعزز التحقق المنهجي إعتمادية النظام المطور بالتأكد من أنه يلبي المتطلبات الوظيفية المعرفة، خاصة عند مراحل التصميم الأولى، ويعتمد التحقق المنهجي على طرق منهجية وهي عباره عن لغات و تقنيات وأدوات مبنية رياضياً لتوصيف والتحقق من الأنظمة [5].

هنالك أدوات متنوعة تساعد في عمليات كتابة المواصفات والتحقق من البرمجيات، فهي تمد مطوري الأنظمة بالترميزات والخوارزميات من أجل التوصيف و/ أو التحقق من الأنظمة منهجياً.

يعتمد التحقق المنهجي على بناء نموذج رياضي للنظام وعلى التوصيف المنهجي للمتطلبات التي يتم التحقق من صحتها بالنسبة للنظام. وأدوات التحقق التي تؤدي التحقق المنهجي تاخذ مدخلين: نموذج للنظام و مواصفاته ثم يتم التأكد من أن النظام يلبي المواصفات. وبناءً على كيفية تنفيذ النموذج والتحقق، تُقسم تقنيات التحقق المنهجي إلى تحليل ساكن و برهنة النظرية [4].

١.٢ التحليل الساكن

تضم التقنيات الساكنة مجموعة من التقنيات المرتبطة التي تحسب آلياً معلومات عن سلوك البرنامج دون تنفيذه [12]. معظم الأسئلة حول سلوك البرنامج لا يمكن اخذ قرار بشأنها أو لا يمكن إيجاد إجابتها بطريقة فعالة ومن ثم فإن جوهر التحليل الساكن هو الحساب بكفاءة ضمانات مقربة ولكنها سليمة (ضمانات غير مضللة). هنالك ثلاث تقنيات أساسية للتحليل الساكن :

١.١.٢ التحليل الساكن المجرد

هو تحليل يستخدم في أدوات تطوير البرمجيات مثل تحليل المؤشر في المترجمات الحديثة. والأساس المنهجي لمثل هذه التقنيات هو التفسير المجرد الذي أدخله العالمان رادهيه كوسوت (Radhia Cousot) و باتريك كوست (Patrick Cousot)[8].

التفسير المجرد: هو نظرية تقريب البنيات الرياضية، وتحديداً تلك المُضمنة في النماذج الدلالية لأنظمة الحاسوب [8].

والمجال المجرد هو تمثيل تقريبي لمجموعة من القيم الملموسة [12]، وتسمى الدالة التي تستخدم لتحويل القيم الملموسة للقيم المجردة دالة التجريد. والتفسير المجرد يفسر معنى البرنامج على المجال المجرد لايجاد حل تقريبي [12]. و التفسير المجرد يمكن اشتقاقه من التفسير الملموس من خلال تحديد نظرائه من العمليات الملموسة مثل الجمع أوالاتحاد، في المجال المجرد [12]. إذا تم الإيفاء بقيود رياضية محددة بين المجال المجرد والمجال الملموس، فإن النقاط الثابتة المحسوبة في المجال المجرد يمكن ضمانها على أنها تقريبات سليمة للنقاط الثابتة

الملموسة[10]. يمكن استخدام التفسير المجرد للبناء المنظم للطرق والخوارزميات الفعالة لتقريب المسائل المعقدة أو التي لايمكن أخذ قرار بشأنها في علم الحاسوب. وبالتحديد فإن التحليل الساكن بالتفسير المجرد، والذي يسنتتج آلياً الخصائص الديناميكية لأنظمة الحاسوب، كان مؤخراً ناجحاً جداً في التحقق آلياً من الخصائص المعقدة لأنظمة الوقت الحقيقي وأنظمة السلامة الحرجة [8].

المجالات الرقمية المجردة: عبر السنوات، تم تصميم مجالات مجردة عديدة ومختلفة لحساب الثوابت حول المتغيرات الرقمية حيث فئة الثوابت المحسوبة وفئة الخواص التي يمكن برهنتها، تختلف مع القوة المعبرة للمجال[12]. لذلك فان المحلل الساكن يتم التعبير عنه بواسطة مجال رقمي مجرد أي مجموعة من الخواص الرقمية الممثلة بالحاسوب مجتمعة مع الخوارزميات لتحسب دلاليات أوامر أو تعليمات البرنامج[4]. وهناك القليل من المجالات الرقمية المجردة، ومن الأمثلة الشائعة مجال الفترة [8] الذي يكتشف حدود المتغيرات والمجال متعدد السطوح[9] للتبيانات المرتبطة. كل مجال يحقق بعض التكلفة في مقابل توازن الدقة. حيث تعتبر المجالات غير الترابطية مثل مجالات الفترة أكثر سرعة ولكن أقل دقة من المجالات الترابطية (التي لها القدرة على اكتشاف علاقات المتغيرات) [4].

تحليل الشكل: تحليل الشكل هو تقنية تحليل ساكن للشفرة وهو يكتشف ويحلل خصائص هياكل البيانات المرتبطة والموزعة ديناميكياً في برامج الحاسوب[4]. وهو يستخدم في وقت الترجمة لاكتشاف أخطاء البرمجيات أو للتحقق من صحة الخصائص عالية المستوى في البرامج. في برامج جافا مثلاً،يمكن استخدامه للتأكد من أن طريقة الفرز تقوم بفرز القوائم بطريقة صحيحة. ولبرامج سي (C) يستخدم للبحث عن المواضع حيث كتلة من الذاكرة لم يتم تحريرها بشكل صحيح [4]. ورغم أن تحليلات الشكل قوية جداً، فهي عادة ما تستغرق وقتاً طويلاً في التشغيل، ولهذا فهي لم تكسب قبولاً واسع الانتشار في اماكن غير الجامعات ومعامل البحوث حيث تستخدم ولكن تجريبياً فقط [4].

إن أول أداة تحليل ساكن معروفة لاكتشاف الأخطاء البسيطة في برامج سي (C) هي لينت (LINT) التي أطلقتها معامل بيل (Labs Bell) في عام 1979م. وأقتدت أدوات حديثة عديدة ب لينت حيث وسعت في خصائص تضمنت نوعية الأخطاء المكتشفة، التحذيرات المُقدمة وخبرة المستخدم. ومن الأمثلة البارزة لهذه الأدوات الحديثة فيندبوغس (FINDBUGS) للغة جافا. وقد أنتجت شركة غراماتتش (Grammatech) أداة كودسونار (CODESONAR) التي تستخدم التحليلات بين الاجرائية للتحقق من أخطاء النماذج في شفرة سي/ سي بلس بلس (C/C++).

ولدى أعمال كلوك (Kloc) أداة تسمى ك 7 (K7) لها نفس السمات وتدعم برامج جافا. أنتجت كوفريتي (Coverity) محللين الأول برفنت (PREVENT) له مقدرات شبهية بمقدرات كودسونار ولكنه أيضاً يدعم مايكروسوفت COM و Win32 APIs وتم تنفيذه تزامنياً باستخدام PThreads، Win32 أو WindRiver VxWorks. و الثاني اكستند (Extend) وهو عباره عن أداة لتعزيز معايير الشفرة.

المحلل الساكن استري (Astr'ee) هو مجالات مجردة تُنفذ لاكتشاف فيضان الذاكرة والنتائج المبهمة في العمليات الرقمية [3]. يستخدم استري في التحقق من برامج التحكم في طيران الأيربص. تسوق شركة بوليسباس للتقنيات (Polyspace Technologies) المحللات الساكنة سي(C) و أيدا (Ada).

يمكن مقارنة تجربة استخدام المحللات الساكنة مع تجربة استخدام المترجم. فمعظم المحللات تستخدم لتحليل أنظمة البرمجيات الكبيرة بتفاعل بسيط من المستخدم. هذه الأدوات قوية جداً حيث إنها فعالة ومناسبة للمدخلات الكبيرة و المتنوعة. وفي المقابل فإن الخصائص المراد إثباتها هي في الغالب بسيطة وعادة ما يصعب تشفيرها في هذه الأدوات [12]. وعلى عكس فحص النموذج، فإن توليد الأمثلة المضادة صعب إن لم يكن مستحيلاً بسبب فقدان الدقة في عمليات الضم والتوسع [15].

٢.١.٢ فحص النموذج

فحص النموذج هو تقنية تعتمد على بناء نموذج محدود للنظام والتحقق من أن المواصفات المطلوبة متضمنة في ذلك النموذج [4]. وعموماً فإن التحقق يُنفذ كبحث شامل في فضاء الحالة والذى نضمن أن ينتهي؛ لأن النموذج محدود ؛ وفي فحص النموذج يصبح من الصعب تقنياً اختراع خوارزميات وهياكل بيانات تتيح التعامل مع فضاءات بحث كبيرة.

فواحص النماذج: هي أدوات لفحص النموذج، لديها نوعان من المدخلات: نموذج نظام محدود الحالة وخاصية تم توصيفها منهجياً [4]. يعمل فاحص النموذج للتحقق مما إذا كان النظام يوفر الخاصية المطلوبة ويعطي إجابة بنعم أو لا. إذا كانت الإجابة لا (لا يوفر النظام الخاصية المطلوبة) ينتج فاحص النموذج مثالاً مضاداً (تشغيل للنظام ينتهك الخاصية) حيث يمكن أن نحلل هذا المثال المضاد لاكتشاف الأخطاء في تصميم النظام [4].

هنالك العديد من فواحص النماذج حيث نصف هنا أكثرها شهرة : هولزمان(Holzmann) كان رائداً في تطوير فاحص نماذج برمجيات الحالة الواضحة[18] [19]. وأستُخدم مشروعه سبين(SPIN) في البداية للتحقق من خصائص المنطق الزمني لبروتوكولات الاتصالات التى كتبت مواصفاتها في لغة بروموله (PROMELA). بعض برامج فحص النموذج مثل الإصدارة الأولى من جافا باذفيندر(Java Pathfinder - JPF) تترجم شفرة جافا إلى لغة بروموله وتستخدم سبين لفحص النموذج[25]. وبالإضافة إلى سبين و جافا باذفيندر هنالك ممثلان بارزان لفئة فاحص نماذج برمجيات الحالة الواضحة: الأول فاحص النموذج سي (C Model Checker- CMC) [22] و الثاني زينغ (ZING) الصادر من بحوث مايكروسوفت [1].

تحاول أداة التحقق من البرمجيات فريوسوفت (VERIOSOFT) تفادي إنفجار الحالة بالتخلص من الحالات التي تستعرضها[14]. ولأن الحالات لايمكن تخزينها، يجب زيارتها واستعراضها بصفة متكررة. وهذه الطريقة غير مجدية لأنظمة الإنتقال التي تحتوي على دوارات. بالإضافة لكونها لا حالة لها ويجب تحديد عمق البحث لتفادي عدم الانتهاء [12].

مقارنة ببرهنة النظرية فإن فحص النموذج يكون آلياً بالكامل وسريعاً حيث غالباً ما يعطي إجابة في ظرف دقائق. يمكن استخدام فحص النموذج في التحقق من المواصفات الجزئية وبذلك يوفر معلومات مفيدة حول صحة النظام حتى إذا لم يتم وصف ذلك النظام بالكامل. خلاصة الأمر، يتيح فحص النموذج أمثلة مضادة والتي عادة ما توضح أخطاء دقيقة في التصميم، وبذلك يمكن استخدامه في المساعدة على إزالة أخطاء البرامج. والعيب الرئيسي في فحص النموذج هو مشكلة انفجار الحالة [4].

٣.١.٢ فحص النموذج المحدود

إحدى أكثر تقنيات التحقق المنهجي شيوعاً في صناعة شبه الموصلات هي فحص النموذج المحدود (BMC). فقد نجحت هذه التقنية بسبب المقدرة المدهشة لمحللات سات (SAT Solvers) الاقتراحية. تم إبتكار فحص النموذج المحدود في العام 1999م بواسطة بير(Biere) وآخرون كتقنية قصد منها تكملة فواحص النماذج غير المحدودة والمؤسسة على مخططات القرار الثنائي(Binary Decision Diagrams-BDD)[2]. وقد سميت محدودة لأنها تستعرض فقط الحالات التي يمكن الوصول إليها ضمن عدد محدود من الخطوات.

في فحص النموذج المحدود، يتم حل التصميم قيد التحقق k مرة ثم يُربط مع الخاصية ليشكل صيغة اقتراحية يتم تمريرها إلى محللات سات[4]. يمكن الإيفاء بالصيغة إذا وفقط إذا كان هنالك تتبع لطول مقداره k ولا يثبت الخاصية. هذه التقنية لا تعطي خلاصة إذا لم يتم الإيفاء بالصيغة حيث سيكون هنالك أمثلة مضادة أطول من خطوات k.

رغم هذه العيوب، فإن التقنية ناجحة حيث أنها قادرة على التعرف على الأخطاء الغير قابلة للاكتشاف. والحالة المرضية للصيغة الأفتراضية ترتبط بالمسار من المرحلة الأولية إلى المرحلة التي يتم فيها انتهاك الخاصية.

يعتبر فحص النموذج المحدود أفضل تقنية في اكتشاف الأخطاء السطحية وتوفر تتبع كامل للأمثلة المضادة عندما تكتشف خطأ ما. وتدعم المدى الواسع من أوامر البرنامج، وهذا الدعم يتضمن هياكل البيانات الموزعة ديناميكياً. ولهذا فإن فحص النموذج المحدود لا يتطلب معرفة مسبقة عن هياكل البيانات التي يحتفظ بها البرنامج، ولكن يمكن تحقيق الكمال فقط في حالات البرامج السطحية جداً أو البرامج التي ليس بها حلقات عميقة [4].

هنالك تطبيقات كثيرة لفحص النموذج المحدود في التحقق من البرمجيات. كان التطبيق الأول في مجال البحث الرمزي محدود العمق في البرمجيات وتم تنفيذه بواسطة كوري (Currie) وآخرون[11]. أحد التطبيقات الأولى لفحص النموذج المحدود لبرامج سي هو سي بي إم سي (CBMC) [6] [7] الذي أُجري في جامعة كارنيغي ميلون (Carnegie Mellon University- CMU) حيث حاكى مدى واسع من البنيات المعمارية كبيئة للتصميم قيد الاختبار. والتطبيق الرئيسي للسي بي إم سي هو التحقق من إتساق نماذج الدوائر على مستوى النظام والمعطاة في سي أو نظام سي(SystemC) مع التطبيق المعطى في فريلوغ (Verilog). وهنالك إصدارة أخرى من سي بي إم سي طورت بواسطة آي بي إم (IBM) للبرامج المتزامنة [23].

طورت أبحاث إن إي سي (NEC) الأداة أف-سوفت (F-SOFT) وهي الأداة الوحيدة التي تُطبق تفكيك نظام انتقال كامل[20] وتتضمن سماتها محللات سات التى خصصت ل حل مشاكل فحص النموذج المحدود. ساتورن(SATURN) عباره عن تطبيق متخصص لفحص النموذج المحدود خصص للصفات التي يمكن التحقق منها [26] وصمم لتطبيق فك الحلقات. والأداة اكس(EXE) أيضاً ابتكرت لصيد الأخطاء[27] حيث تجمع التطبيق الصريح وطريقة مسار المحاكاة الرمزية لاكتشاف الأخطاء في البرمجيات على مستوى النظام مثل شفرة نظام الملف.

2.2 برهنة النظرية

برهنة النظرية هي تقنية يتم فيها التعبير عن النظام وخصائصه المطلوبة كمعادلات أو صيغ في المنطق الرياضي [5]. وهذا المنطق يُعطى بنظام منهجي يعرف مجموعة من البديهيات ومجموعة من قواعد الاستدلال. وبرهنة النظرية هي عملية الوصول إلى إثبات الخاصية من بديهيات النظام. ورغم أن البراهين يمكن أن يتم بناءها يدوياً ولكن اغلب التركيز على برهنة النظرية بمساعده الآلة. اليوم، تُستخدم مبرهنات النظرية بصورة متزايدة في التحقق الميكانيكي لمواصفات السلامة الحرجة لتصميمات العتاد والبرمجيات. ومبرهنات النظرية يمكن تقسيمها عموماً الي فئات تتراوح بين برامج عالية الأتمتة وذات أغرض عامة وبين أنظمة متفاعلة ذات أغراض خاصة.

وعلى عكس فحص النموذج، فإن برهنة النظرية قادرة على التعامل مباشرة مع فضاء الحالات اللامنتهية. وهي تعتمد على تقنيات مثل الحث التركيبي للإثبات عبر المجالات اللامنتهية. وبحكم تعريفها، فإن مبرهنات النظرية المتفاعلة تتطلب تفاعلاً مع الإنسان ولذلك فإن عملية برهنة النظرية بطيئة وعرضة للأخطاء. ولكن- كميزة - فإن المستخدم وفي سعيه لإيجاد البرهان يكسب نظرة متعمقة في النظام أو الخواص التى يسعى لبرهنتها [4].

٣. مبادىء المنهج

المنهج الذي نوصي به مبني على أربع ركائز هي:

- مواصفات أنواع البيانات المجردة تكتب بأسلوب سلوكي، مستخدمين ترميز يشبه مواصفات التتبع والذي يمثل سلوك أنواع البيانات المجردة (ADT's) بعلاقة من سلاسل المدخلات إلى المخرجات.
- تُعرف المواصفات بالحث عن طريق البديهيات والقواعد حيث تمثل البديهيات سلوك أنواع البيانات المجردة لسلاسل المدخلات الأولية بينما تربط القواعد بين سلوك أنواع البيانات المجردة لسلاسل المدخلات المعقدة وسلوكها لسلاسل المدخلات البسيطة [28].
- يتم التحقق من تطبيق أنواع البيانات المجردة مقابل بديهيات المواصفات باستخدام منطق هوار[16]. لأن البديهيات تمثل فقط جزءاً من سلوك أنواع البيانات المجردة فإن التحقق من تطبيقها مقابل البديهيات ليس كافياً.
- يتم اختبار تطبيق أنواع البيانات المجردة حسب مبدأ اختبار الغرفة النظيفة [21] مستخدمين قواعد المواصفات كحكيم اختبار.

سيكون التركيز في هذه الورقة على التحقق من تطبيق أنواع البيانات المجردة مقابل بديهيات المواصفات باستخدام منطق هوار، الذي يجعل منطق هوار صعب التطبيق على نطاق واسع، وبالأخص الذي يجعل أتمتته صعبة هي الحاجة إلى اختراع الخبر اللامتغير لتعليمات المدار في البرامج. ولكن هنالك سببين يجعلان منهجنا يتجنب هذا العائق:

- أولاً، أن البديهيات تمثل السلوك الأساسي البسيط لأنواع البيانات المجردة فهي لا تستدعي الحسابات المعقدة في شفرة المصدر، في الغالب قد تستدعي بعض الحلقات لتهيئة هيكل البيانات. ومثل هذه الشفرة إما أن تستخدم حلقات للمهام البسيطة التي يكون فيها اكتشاف الخبر اللامتغير سهلاً أو قد لا تستدعي الحلقات إطلاقاً.
- ثانياً، أن لدينا الوسائل لحساب الخبر اللامتغير آلياً في مجالات تطبيق محددة، مستخدمين النتائج من [13]. نحن نتصور أنه يمكننا وبسهولة تهيئة الأداة المقدمة في [13] لتوليد أخبار لامتغيرة بسيطة نحتاجها لأغراضنا (مثلاً للحلقات التي تقوم بتهيئ هيكل البيانات).

٤. توليد حالات التحقق

بناءا على نموذجنا المستخدم للمواصفات، توصف أنواع البيانات المجردة بعلاقة (غالباً داله) من سلاسل المدخلات إلى المخرجات، ولوصف سلوك كومة ما Stack مثلاً نجعل المدخلات هي عمليات الكومة التي قد يدخلها المستخدم (مثل عملية الافتتاح init، عملية الإيداع push،عملية النزع pop، عملية كشف القمة top، عملية الفراغ empty، وعملية كشف الحجم size) ونجعل المخرجات هي مجموعة الاستجابات التي قد تنتجها الكومة مثل عنصر القمة للكومة (للعملية top) وحجم الكومة (للعملية size) أو عملية منطقية Boolean(التي تشير إلى ما إذا كانت الكومة فارغة أو لا). أمثلة البديهيات لنوع البيانات كومة تتضمن [28] :

- **البديهي الأول :القمة الافتتاحية Init Top Axiom**

stack (init.top) = error.

- **البديهي الثاني : قمة الكومة Push Top Axiom**

stack(init.h.push(a).top) = a.

- **البديهي الثالث : الحجم الافتتاحي Size Axiom**

stack(init.size) = 0.

- **البديهي الرابع : الفراغ الإفتتاحي Init Empty Axiom**

stack(init.empty) = true.

- **البديهي الخامس : الفراغ العادي Push Empty Axiom**

stack(init.push(a).empty)= false.

ولكلٍ من هذه البديهيات، نضع حالة تحقق في شكل صيغة هوار التي يجب أن نثبتها بعد ذلك. ونفترض أن تطبيق نوع البيانات المجردة كومة يأخذ شكل الفئة في لغات البرمجة الموجهة للكائنات [24]، حيث أي رمز إدخال(*init, push, pop, top, size, empty*) يمثل عملية من عمليات الفئة، حيث تكون init و push و pop عمليات لا تعيد قيمة، وتعيد Top قيمة من نوع البيانات الخاص بالكومة، وتعيد size رقم صحيح وتعيد empty قيمة منطقية .ونوضح أدناه حالات التحقق التي تم توليدها لكلٍ من البديهيات الواردة أعلاه:

- **Init Top Axiom**

v: {**true**} init( ); y = top( ) {y = error}

حيث يتم الاعلان عن y كمتغير يحمل نوع البيانات الخاص بالكومة أو رسالة الخطأ.

- **Push Top Axiom**

v: {**true**} init( ); h ; push(a); y = top( ) {y = a}

لأى عنصر a ، حيث h عبارة عن سلسلة من العمليات الواردة و y متغير من نوع البيانات المخزنة في الكومة.

- **Size Axiom**

v: {**true**} init( ); y = size( ) {y = 0}

حيث y متغير من نوع الأرقام الصحيحة.

- **Init Empty Axiom**

v: {**true**} init( ); y = empty( ) {y = true}

حيث y متغير من نوع البيانات المنطقية.

- **Push Empty Axiom**

v: {**true**} init( ); push(a); y = empty( ) {y = false}

لأى عنصر a ، حيث y متغير من نوع البيانات المنطقية.

٥. التحقق من التطبيقات مقابل البديهيات

من أجل أن نثبت أن كل حالة تحقق في صيغة هوار مرتبطة مع البديهيات الواردة أعلاه، يجب أن نستبدل كل طلب استدعاء للدوال بمحتوى تلك الدوال. وهذا ينتج:

١. Push Top Axiom:

v: {true}
sindex:=0;
h;
sindex:=sindex+1; sarray[sindex]:=a;
if (sindex>0) {y := sarray[sindex];}
else {return error;}
{y = a}

وبتطبيق قاعدة التسلسل ثلاث مرات مستخدمين الخبر الوسيط *{sindex≥0}* بعد التسلسل *h* [24] . ينتج من هذا المعادلة التالية:

v0: {true} sindex:=0 {sindex=0}.

v1: {sindex=0} h {sindex≥0}.

v2: {sindex≥0} sindex := sindex+1 {sindex>0}.

v3: {sindex>0} sarray[sindex]:=a {sindex>0 ∧ sarray[sindex]=a}

v4: {sindex>0 ∧ sarray[sindex]=a}
if(sindex>0) {y:= sarray[sindex];} else {return error;} {y=a}

تطبيق قاعدة الإسناد على v0, v2, v3 ينتج على التوالي:

v00: true ⇒ 0=0,
v20: sindex ≥0 ⇒ sindex+1>0
v30: sindex >0 ⇒ sindex>0 ∧ a = a,

وكل واحدة من هذه تعتبر مسلمه من مسلمات نظامنا الاستنتاجي. حيث نهتم بجملة v1 التي نطبق عليها قاعدة التسلسل :

v10: {sindex≥0} h {sindex≥0}.

ومن أجل أن نثبت صحة هذه المعادلة، حيث h هي سلسلة من العمليات، يعتبر كافياً (عن طريق الإستقراء) أن نثبتها لكل عملية، من المؤكد فهي صحيحة لعمليات V (V- Operations) هي العمليات التي تنتج قيمة عن د استدعائها) حيث من الطبيعي أنها لا تعدل متغيرات الحالة [24]. أما بالنسبة لعمليات O (O- Operations) هي العمليات التي لا تنتج قيمة عن استدعائها ولكنها تؤثر في السلوك المستقبلي لأنوع البيانات المجردة):

١. Init: المعادلة {sindex≥0} init() {sindex≥0} هي صحيحة بمقتضى قاعدة التسلسل حيث *{true} init() {sindex=0}* صحيحة.

٢. Push: المعادلة {sindex≥0} push () {sindex≥0} صحيحة لأن عمليات الإضافة push تزيد قيمة *sindex* .

٣. Pop: المعادلة {sindex≥0} pop() {sindex≥0} صحيحة لأن عملية الحذف pop لا تقلل قيمة ال *sindex* إلا إذا كانت قيمة ال *Sindex* موجبة حيث تقل القيمة بمقدار1 فقط.

الأن يمكننا التركيز على الصيغة v4 التي نطبق عليها قاعدة التعليمة الإختيارية حيث نجد:

v40: {sindex>0 ∧ sarray[sindex]=a ∧ sindex>0}
y := sarray[sindex]; {y=a}

v41: {sindex>0 ∧ sarray[sindex]=a ∧ sindex≤0}
y := sarray[sindex]; {y=a}

المعادلة v41 بديهياً صحيحة لأن الشرط خطأ. نطبق قاعدة الإسناد على v40

v400: sindex>0 ∧ sarray[sindex] = a
⇒ sarray[sindex]=a.

هذه تعتبر مسلمة من مسلمات نظام هوار ، وهذا يلخص إثباتنا ويرسخ صحة v .

٢. Size Axiom:

v: {true}
sindex :=0;
y:={retrun sindex}
{y=0}

و لإثبات الصيغة v نطبق قانون التسلسل ونستنتج الصيغ التالية مستخدمين *{sindex=0}* كخبر وسيط:

v0: {true} sindex:=0 {sindex = 0}
v1: {sindex=0} y:={retrun sindex}{y = 0}

وبتطبيق قاعدة الإسناد على v0، v1 نجد:

v00: true ⇒0 = 0,
v10: sindex=0 ⇒ sindex

v0 و v1 مسلمتان من مسلمات نظام هوار. وهذا يلخص الإثبات ويرسخ صحة v.

٣. Init Empty Axiom:

v: {true}

sindex := 0;
y:=(sindex = 0)

{y=true}

ولإثبات صيغة v نطبق قاعدة التسلسل ونستنتج الصيغ التالية مستخدمين *{sindex=0}* كخبر وسيط:

v0: {true} sindex:=0 {sindex = 0}
v1: {sindex=0} y :=(sindex = 0) {y = true}

وبتطبيق قاعدة الإسناد على v0 و v1 نجد:

v00: true ⇒0 = 0,
v10: sindex=0 ⇒ (sindex = 0) = true

v0 و v1 مسلمتان من مسلمات نظام هوار ، وهذا يلخص إثباتنا ويرسخ صحة v .

٤. Push Empty Axiom:

v: {true}

sindex := 0;
sindex:=sindex+1; sarray[sindex]:= sitem;
y:=(sindex=0)

{y = false}

ولإثبات صيغة v نطبق قاعدة التسلسل ثلاث مرات ونستنتج الصيغ التالية [12].

v0: {true} sindex:=0 {sindex=0}
v1: {sindex=0} sindex:=sindex+1; {sindex=1}
v2: {sindex= 1} sarray[sindex]:=a; {sindex=1}
v3: {sindex=1} y:=(sindex=0) {y=false}

وقاعدة الإسناد المطبقة على v0 و v1 و v2 و v3 تنتج على التوالي:

v00: true ⇒0 = 0,
v10: sindex=0 ⇒sindex+1 = 1,
v20: sindex=1 ⇒sindex = 1,
v30: sindex=1 ⇒ (sindex = 0) = false,

كل واحدة من هذه تعتبرمسلمه من مسلمات نظام هوار وهذا يلخص الإثبات ويرسخ صحة v.

٦. خطة الأتمتة

نحن نهدف لبناء أداة آلية تستوعب حالات التحقق المستخدمة في القسم السابق وتثبتها في منطق هوار مستخدمين مبرهن نظرية و/أو نظام جبر حاسوبي مثل ماذماتيكة (Mathematica) من بحوث وولفرام (Wolfram Research). هذه الأداة يجب استخدامها مرتبطة مع أداة آلية أخرى تختبر نفس التطبيق مستخدمة قواعد المواصفات كحكيم اختبار. وهذا الأمر قيد البحث حالياً.

٧. خاتمة

ان المنهج المقترح في هذه الورقة سيمكننا من كتابة مواصفات شكلية تتميز بالبساطة والدقة والتجرد لوصف أنواع البيانات المجردة . كما يحاول استخدام منطق هوار لإثبات صحة حالات التحقق الموضوعه لكل بديهي من بديهيات المواصفات آلياً، حيث يوفر الوقت والجهد المبذولين في محاولة إثباتها يدوياً وهذا الأمر قيد البحث حالياً.

المصادر و المراجع

[1]Andrews T. and et al., "Zing: Exploiting program structure for model checking concurrent software," *In Concurrency Theory (CONCUR)*, pp. 1–15, Springer, 2004.

[2]Biere A. and et al., "Symbolic model checking without BDDs," *In Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 1579 of LNCS, pp. 193–207, Springer, 1999.

[3]Blanchet B. and et al., "Design and implementation of a special purpose static program analyzer for safety-critical real-time embedded software," *In The Essence of Computation: Complexity, Analysis, Transformation.* Essays Dedicated to Neil D. Jones, vol. 2566 of LNCS, pp. 85–108, Springer, 2002.

[4]Campetelli A., "Analysis Techniques: State of the Art in Industry and Research," *Technische Universität München Fakultät Für Informatik*, Tech. Rep. 1.0, 2009.

[5]Clarke M. E. and et al, "Formal methods: State of the art and future directions," *Report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research, ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.

[6]Clarke E., Kroening D. and Yorav K., "Behavioral consistency of C and Verilog programs using bounded model checking," *In Design Automation Conference (DAC)*, pp. 368–371, ACM, 2003.

[7]Clarke E., Kroening D., and Lerda F., "A tool for checking ANSI-C programs," *In Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 2988 of LNCS, pp. 168–176, Springer, 2004.

[8]Cousot P. and Cousot R., "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," *In POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 238–252, ACM Press, 1977.

[9]Cousot P. and Halbwachs N., "Automatic discovery of linear restraints among variables of a program," *In Principles of Programming Languages (POPL)*, pp. 84–96, ACM, 1978.

Operating Systems Review, vol. 36, no. SI, pp.75–88, 2002.

[23]Rabinovitz I. and Grumberg O., "Bounded model checking of concurrent programs," *In Computer Aided Verification (CAV)*, vol. 3576 of LNCS, pp. 82–97, Springer, 2005.

[24]Tchier F. and Mili A., "On the Verification and Validation of Software Modules : Applications in Teaching and Practice," *Tech. Rep. of NJIT*, 2013.

[25]Visser W. and et al., "Model checking programs," *Automated Software Engineering (ASE)*, vol. 10, no. 2, pp. 203–232, 2003.

[26]Xie Y. and Aiken A., "Scalable error detection using Boolean satisfiability," *In Principles of Programming Languages (POPL), ACM*, pp. 351–363. 2005.

[27]Yang J. and et al., "Automatically generating malicious disks using symbolic execution," *In IEEE Symposium on Security and Privacy (S&P), IEEE*, pp. 243–257, 2006.

[28]هاني عمار و علي الميلي، *هندسة البرمجيات: الأوجه التقنية و التنظيمية و الإقتصادية*، *مؤسسة فيليبس للنشر*، فيليبسبرج، نيوجيرسي الطبعة الأولى، مارس2006 .

[10]Cousot P. and Cousot R., "Systematic design of program analysis frameworks," *In Principles of Programming Languages (POPL)*, pp. 269–282, ACM, 1979.

[11] Currie W. D., Hu J. A., and Rajan P. S., "Automatic formal verification of DSP software," *In Design Automation Conference (DAC)*, pp. 130–135, ACM, 2000.

[12]D'Silva V., Kroening D., and Weissenbacher G., "A Survey of Automated Techniques for Formal Software Verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.

[13]Ghardallou W. and et al., "A Versatile Concept for the Analysis of Loops," *Journal of Algebraic and Logic Programming*, vol. 81, no. 5, pp. 606-622, 2012.

[14]Godefroid P., "Model checking for programming languages using VeriSoft," *In Principles of Programming Languages (POPL)*, pp. 174–186, ACM, 1997.

[15]Gulavani B. S. and Rajamani S. K., "Counterexample driven refinement for abstract interpretation," *In Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 3920 of LNCS, pp. 474–488, Springer, 2006.

[16]Hoare C.A.R., "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12 , no. 10, pp. 576–580, 1969.

[17]Hoffman D. and Snodgrass R., "Trace Specifications: Methodology and Models," *IEEE-TSE.* , vol. 14, no. 9, 1988.

[18]Holzmann J. G., "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[19]Holzmann J. G., "Software model checking with SPIN," *Advances in Computers*, vol. 65, pp. 78–109, 2005.

[20]Ivancic F. and et al., "Model checking C programs using F-SOFT," *In International Conference on Computer Design (ICCD), IEEE*, pp. 297–308, 2005.

[21]Mills H., Dyer M. and Linger R., "Cleanroom Software Engineering," *IEEE Software*, vol. 4, no. 5, pp. 19-25, 1987.

[22]Musuvathi M. and et al., "CMC: a pragmatic approach to model checking real code," *SIGOPS*