University of Minho

Informatics Department

Master in Computing Engineering

(http://mei.di.uminho.pt)

Formal Methods in Software Engineering (FMSE – 30 ECTS)

2012/2013

(http://mei.di.uminho.pt/?q=pt-pt/1213/mfes)

Cohesive Project (CP) Workshop : Presentation of Project proposals

Room DI-1.08

15th November, 2012

With Partnership by

IAE
Instituto de Aeronáutica e Espaço

Schedule:

- 12h00 – Presentation of project proposals by the **Brazilian Aeronautics and Space Institute** (IAE/ITA) and QMUL (on behalf: J.C. Campos, UM, PT)
- 13h00 – Lunch break
- 14h00 - Presentation of project proposals by **Software Improvement Group** (M. Ferreira, SIG, NL)
- 14h40 – Presentation of project proposals by **CSAIL** (Eunsuk Kang, CSAIL, MIT)
- 15h00 - Presentation of project proposals by **Primavera Software Factory** (Carlos Argainha, PSF, PT)
- 15h20 – Coffee break
- 16h00 – Presentation of project proposals by **OutSystems** (Tiago Alves, OutSystems, PT)
- 16h40 – Presentation of projects proposals by **Galois** (Sally Browning, Galois, USA).
- 17h40 – Comments. Schedule (milestones). Projects and tutors. Assessment.
- 18h00 – Closing.

IMPORTANT:

The information contained in the current document is confidential and for exclusive use by FMSE sponsor companies, students and project tutors inside the classroom. NDAs will be signed wherever requested by industrial sponsors.

Safety Critical Interactive Computing Systems' Modelling

MFES INTEGRATED PROJECT PROPOSAL

HASLAB / INESC TEC & DEPT. INFORMÂ·TICA / UNIVERSIDADE DO MINHO

2012/13

This research theme follows from an ongoing effort to develop tools and techniques for the systematics analysis of interactive computing systems. A modelling and verification tool has been developed (the IVY workbench), that supports a modelling, verification, and analysis cycle. Modelling is done using Model Action Logic. Verification is achieved through model checking. A number of case studies have been carried out [1, 2, 3]. As a result, the Brazilian Aeronautics and Space Institute (IAE) has expressed interest in exploring the applicability of IVY to safety critical space software systems.

The goal of the current proposal is to further the applicaton of IVY to IAE's current satellite launcher preparation and control system (BCVLS). BCVLS is composed of a number of stations, enabling control of the several aspects of the preparation and launch of satellite launch rockets. Each preparation phase enables interaction with the launch rocket via a graphical user interface. Components of BCVLS have already be modeled. The goal now is to study the scalability of the models, and the best abstractions to promote this scalability. One of the aspects to consider will be the inclusion of the timing dimension in the models.

The expected result will be a collection of models together with data on the performance of the verification tool for each model. It is expected that the process will lead to improvements in the modelling language and supporting tools.

Proponents: José Creissac Campos (DI/UMinho), Miriam Alves (IAE)

Contact: `jose.campos@di.uminho.pt`

References

[1] J. C. Campos and M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In *Interactive Systems: Design, Specification and Verification*, volume 5136 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, July 2008.

[2] J. C. Campos and M. D. Harrison. Interaction engineering using the IVY tool. In *ACM Symposium on Engineering Interactive Computing Systems (EICS 2009)*, pages 35–44. ACM, 2009.

[3] J.C. Campos and M.D. Harrison. Modelling and analysing the interactive behaviour of an infusion pump. *Electronic Communications of the EASST*, 45: Formal Methods for Interactive Systems, 2011.

Formalization of the ECSS-E-70-41A standard

MFES INTEGRATED PROJECT PROPOSAL

HASLAB / INESC TEC & DEPT. INFORMÁTICA / UNIVERSIDADE DO MINHO

2012/13

The ECSS-E-70-41A standard [1] (Ground systems and operations – Telemetry and telecommand packet utilization), prepared by the ECSS Working Group for Ground Systems and Operations, is one of the series of ECSS Standards for the management, engineering and product assurance in space projects and applications. ECSS (European Cooperation for Space Standadization) is a cooperative effort of the European Space Agency, national space agencies and European industry associations with the purpose of developing and maintaining common standards.

The ECSS-E-70-41A standard addresses the utilization of telecommand and telemetry packets for the remote monitoring and control of subsystems and payloads. The standard has been adopted by the Brazilien Aeronautical Institute of Technology (ITA) in its ITASAT project[1], which aims to design, build, launch and operate an university satellite. ITA researchers have developed a formal model of parts of the ECSS-E-70-41A standard in Uppaal (a model-checker for real-time systems modeled as networks of timed automata), and have manually derived a software system from the model. This system will run embedded in the ITASAT satellite.

The current proposal's goal is to independently develop an alternative model of the standard (also in Uppaal), and explore automatic code generation from that model. This proposal further develops the work initially done at ITA by providing: a) a comparison of the impact of differences in modelling approaches in both the verification and code generation results; b) pointers to the automation of the code generation process.

Proponents: José Creissac Campos (DI/UMinho), José Machado (DEM/UMinho); Emilia Villani (ITA)

Contact: `jose.campos@di.uminho.pt`

References

[1] ECSS Working Group for Ground Systems and Operations. Space engineering: Ground systems and operations – telemetry and telecommand packet utilization. Standard ECSS-E-70-41A, European Cooperation for Space Standardization (ECSS), January 2003.

[1] `http://www.itasat.ita.br/`

Integration of Model Checking and Theorem Proving in the IVY workbench

MFES INTEGRATED PROJECT PROPOSAL

HASLAB / INESC TEC & DEPT. INFORMÁTICA / UNIVERSIDADE DO MINHO

2012/13

This research theme follows from an ongoing effort to develop tools and techniques for the systematics analysis of interactive computing systems. A user interfaces' modelling and verification tool has been developed (the IVY workbench), that supports a modelling, verification, and analysis cycle. Modelling is done using Model Action Logic. The tool features a plugin architecture and currently verification is achieved through model checking. A number of case studies have been carried out [1, 2, 3], and as a result of this a number of limitations with the model checking based approach have been identified. To address these limitations preliminary work on translating MAL models and CTL properties into PVS (a theorem prover) has been done at Queen Mary, University of London (QMUL).

The goal of the current proposal is to further the development of IVY by developing a plugin for theorem proving based verification. The work will follow from the work done at QMUL. The expected result will be an integration of the PVS theorem prover into the IVY workbench tool. This will encompass translating both the models and the properties, and also identify relevant strategies for performing verification.

Proponents: José Creissac Campos (DI/UMinho), Michael Harrison & Paolo Masci (QMUL)

Contact: `jose.campos@di.uminho.pt`

References

[1] J. C. Campos and M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In *Interactive Systems: Design, Specification and Verification*, volume 5136 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, July 2008.

[2] J. C. Campos and M. D. Harrison. Interaction engineering using the IVY tool. In *ACM Symposium on Engineering Interactive Computing Systems (EICS 2009)*, pages 35–44. ACM, 2009.

[3] J.C. Campos and M.D. Harrison. Modelling and analysing the interactive behaviour of an infusion pump. *Electronic Communications of the EASST*, 45: Formal Methods for Interactive Systems, 2011.

Fault containment in component-based software systems

Abstract

Larger and larger software systems are being build every day. The sheer volume of such systems poses a great challenge for software engineers that have to build and maintain them. To alleviate the problem, software engineers apply the concepts of divide and conquer and separation of concerns to split large systems into self-contained components. These components interact with each other via well-defined interfaces. Although this approach helps to deal with large code volume and complexity, it introduces a problem of gracefully propagate errors between components of the system and, if possible, at some point supress them by providing alternatives. This does not mean that a component-based system cannot display error messages to its users, on the contrary. This means that the system should not crash or deliver random error messages to its users because one of its components crashed; instead it should be prepared to gracefully deal with seemingly random component failures.

Background and project description

The Software Improvement Group (SIG) is a fast-growing management advisory firm that bases its advise on factual software analysis. To this end, SIG has a software analysis and evaluation laboratory that employs state-of-the-art analyses for source code, architecture, deployment schemes and production environments. One of these analyses is rooted in a reliability model of which fault containment is a key driver.

When developing a software system, there are expected and unexpected errors, being that it is easier to develop error-handling strategies for the expected errors. In component-based systems, each component should be considered and independent entity (although most components in a system are typically developed by the same team) and, therefore, only the errors explicitly declared in the component's interface could be expected. All other errors should be regarded as unexpected errors, without prejudice of these also having to be dealt with. Handling an error is basically a choice between not doing anything and letting the error "bubble up", or trapping the error to do something with it. In the latter case, there is another choice that is to simply supress the errors, thus ignoring it, or to take some mitigating action.

Due to the nature of unexpected errors, the models used to express them are typically probabilistic, where a probability is assigned to the likelihood of one of those errors being generated. Furthermore, fault containment models extend the models for unexpected errors by introducing the notion of components where the errors can be generated and a notion of propagation of errors between components. Because each component has the option of handling errors internally or to let them "bubble up", another probability is associated with to the notion of propagation, expressing the likelihood of an error being propagated between two components. Finally, the components need to be coupled to each other depending on whether one relied on the other. The propagation of errors can only happen from a component to other components that rely on it. An example of such model is explained in [1].

For SIG it is relevant to explore such probabilistic fault containment models to determine their applicability and suitability for the management advisory practice. A suitable model according to these criteria needs to be reasonably easy and fast to build, and based on sound and simple concepts that can be used as a communication device between technical and non-technical people. One final desirable quality of such a model is some calculation power that should enable answering questions such as:

- Which component should be selected as a replacement for an existing component?
- Which component has the greatest potential for improvement based on the impact on the reliability in the entire system?

Other very important issue revolving around the probabilistic models for fault containment are:

- How to assign the internal failure probabilities to each component?
- How to assign the error propagation probabilities to each component dependency?

Tasks

The project will involve the following tasks:

- Get acquainted with the probabilistic model described in [1];
- Identify the properties of error propagating and non-error propagating dependencies:
  - Types of dependencies;
  - Relevant characteristics;
  - Relation to the components they "connect";
  - Places in the system's architecture where they typically occur;
  - Places in the system's architecture where such connections increase/decrease overall system reliability;
- Develop a set of real-world case studies to test and refine the model and the identified dependency characteristics.

References

[1] *A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems*, Vittorio Cortellessa and Vincenzo Grassi, Component-Based Software Engineering, 2007, http://link.springer.com/chapter/10.1007%2F978-3-540-73551-9_10?LI=true

Extending transaction handling to application logic

Abstract

The concept of a transaction is very clear in the database domain. There, the technology provides straightforward mechanisms that turn a set of independent operations into one undividable unit, called transaction. A transaction must succeed or fail as a whole. That means that intermediate inconsistent states are not possible. However well established the concept of transaction is in the database realm that is insufficient to assure a constantly consistent state in a software system. The reason is quite obvious: software systems are built-up from more components than just databases, and there is plenty room to store state elsewhere (e.g. caches, file systems). As a result of the lack of transactional support in general-purpose languages (e.g. Java, C#) developers are required to implement their own mechanisms to assure state consistency in their applications. This results in increasing complexity of the development task and of the resulting source code. The problem here is the lack of an abstraction for general-purpose languages that relieves the developers from having to implement ad-hoc solutions for transaction handling.

Background and project description

The Software Improvement Group (SIG) is a fast-growing management advisory firm that bases its advise on factual software analysis. To this end, SIG has a software analysis and evaluation laboratory that employs state-of-the-art analyses for source code, architecture, deployment schemes and production environments. One of these analyses is rooted in a reliability model of which fault containment is a key driver.

When developing a software system with complex operations that are implemented via chaining of other operations, one needs to consider what happens to the state of the system if one of the operations in a chain fails? How should the intermediate results be handled? Where there side effects? Answering this questions is not always straightforward as developers of the same system often have a local view over what they develop and do not always know how the APIs they create will be used. Therefore, transaction handling is more of an architectural concern.

Some frameworks (such as JTA and Spring for Java; the IDisposable interface for C#; the Command design pattern) already provide mechanisms that allow for some degree of transactionality, however implementations using these frameworks are very much targeted at data persistence (the case of JTA and Spring) or to user interfaces (the case of the IDisposable interface). In this project we propose to study existing transaction handling mechanisms available for general-purpose languages to assess their adequacy for use in scenarios that are not only focused on data persistence. From this study we also expect to learn what are the properties of a transactional operation and how can those be extended to operations outside the data persistence realm (e.g. object interactions, service calls).

Tasks

The project will involve the following tasks:

- Get acquainted with existing transactional or otherwise supporting technologies available for general-purpose languages.
- Clearly define what are the properties expected from a transaction.
- Define how such properties can be consistently and systematically implemented in an OO language (preferably Java) in such a way that it makes the task of the developer easier.
- Implement a proof of concept based on real-world case studies.

- Validate the developed implementation with respect to impact in system reliability and developer productivity.

Model validation through test case generation

One of the most challenging tasks in formal methods is validating a model to ensure that it accurately reflects the behavior of the system. This project explores the problem of model validation using test case generation. An exhaustive set of test cases can be automatically generated from a model, and executed on the implementation. The output from the implementation is then compared against that of the model; any discrepancy between the two would indicate a potential problem in the model, or a bug in the implementation. As a case study, you will apply this idea to validate a formal Alloy model of Git, a popular revision control system.

Contact: eskang@csail.mit.edu

Primavera

Empresa: Technology – PRIMAVERA Software Factory

Local: Braga

Contactos: Carlos Argainha: carlos.argainha@primaverabss.com

Tema do projecto

Especificação, implementação e validação formal do processo de cálculo do preço de custo médio de artigos.

Descrição

A questão da correcta valorização dos stocks de uma empresa, é um assunto muito critica em múltiplos sectores de actividade, como por exemplo empresas de logística. As empresas precisam de saber em cada momento, por exemplo, qual é o valor actual dos seus stocks e qual é o valor médio do custo de uma unidade de artigo que têm para venda, o designado preço de custo médio (PCM).

Apesar de numa primeira análise parecer uma questão simples, a correta gestão deste processo é complexa, e como foi referido anteriormente crítica em muitas organizações.

O objectivo proposto para este projecto, é a especificação de um sistema que faça a correta valorização dos stocks e cálculo do PCM, a implementação desse sistema em C# e a validação formal do mesmo.

De seguida descreve-se os requisitos deste do processo de gestão do preço de custo médio de um artigo.

Um artigo pode ter movimentos de stock de dois tipos, entrada de stock e saída de stock. Em cada um destes tipo de movimentos para um artigo, é indicada a quantidade movimentada, o preço unitário do movimento, se esse movimento actualiza ou não o PCM, e data desse movimento (NOTA – a partir deste ponto sempre que se referir data significa data e hora ao segundo desse movimento).

A cada movimento de stock, tendo em conta o referido anteriormente, se esse movimento actualiza o PCM, o PCM deve ser calculado da seguinte forma:

> O valor do Preço de Custo Médio (PCM) é o calculo da média de custo dos movimentos de stock, à data.
>
> **Novo PCM** = (Valor em Stock + Valor movimento) / (Quantidade em Stock + Quantidade movimento) em que:
> **Valor em Stock** = Stock actual * PCM actual
> **Valor movimento** = Qt. movimento * Preço Unitário de movimento
> **Qt. em Stock** = Stock actual

Os movimentos de stock, não são geridos no sistema de forma sequencial, isto é a qualquer momento, pode ser realizado um movimento de entrada ou saída, em qualquer data. Tendo em conta o descrito anteriormente, isto significa que neste cenário, será necessário recalcular o valor do PCM nos movimentos posteriores.

FMES Project Proposals

Tiago L. Alves (tiago.alves@outsystems.com)
Lúcio Ferrão (lucio.ferrao@outsystems.com)

BACKGROUND:

OutSystems is a multi-awarded Portuguese company offering an innovative product Agile Platform. Key to this unique product is to allow the development of scalable web enterprise applications using visual model-driven engineering. This approach is powered by a series of innovations allowing the Agile Platform to cover the full lifecycle of building and management of applications.
The innovation areas where the OutSystems' R&D Groups is responsible for is show in Figure 1.

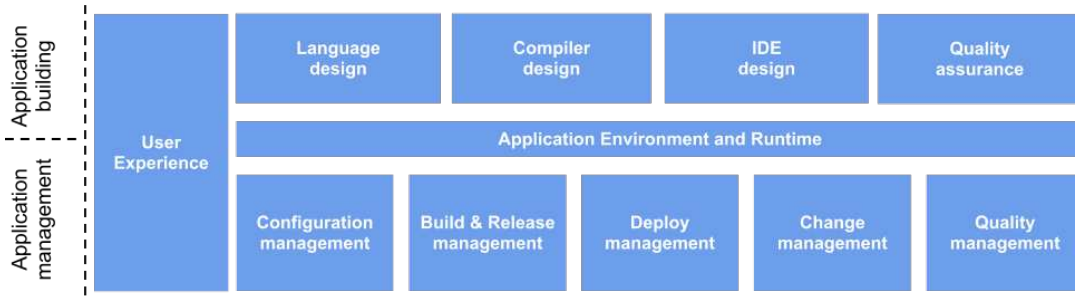

Figure 1. Application Lifecycle Management areas covered by OutSystems' Agile Platform.

At the core of the Agile Platform lays the OutSystems Visual Language. This language offers high-level abstractions over visual components and interactions common in web applications. Using this simple language developers can describe full-fledge models representing web applications which are then generated natively in C# or in Java. The language is made available using the Service Studio IDE building an application model from components with drag and drop operations. In addition to the building process, the True Change engine continuously ensures the model quality by automatically reporting warnings/errors and automatically heals the model performing self-activated refactorings.
The Agile Platform, in addition to application building, supports provides advanced and automated support for application management, in particular: Configuration, Build & Release, Deploy, Change and Quality.

For each area of Language design and Quality assurance a project is proposed: "Formalization of the OutSystems Visual Language" and "Quality Indicators for Agile Platform's Applications".
For both projects the students will have opportunity to be part of a challenging and dynamic industrial environment by interacting with the OutSystems R&D group and the Agile Platform.
From the OutSystems side we are looking forward to the opportunity to collaborate with students and to jointly coming up with new insight and future directions on how to further improve the Agile Platform.

PROJECT 1: FORMALIZATION OF THE OUTSYSTEMS VISUAL LANGUAGE

Over 10 years the OutSystems Visual Language has evolved to support present and future requirements of our customers. Looking forward for the next 10 years it is important to assure a steady foundation that supports the continuous sustainable growth of the Agile Platform. Part of this assurance process includes revisiting core parts of the product such as the language.
In this project we aim at the formalization of the OutSystems Visual Language and the reverse engineering of its semantic rules using Alloy. The purpose is of this effort is to guide future language changes. From this formalization alternative directions can be undertaken: i) identify shared abstractions and inconsistencies to improve the language; ii) finding missing or validating language semantic rules implemented in the Agile Platform IDE; iii) further extend the language and its semantic rules for higher-level concepts (e.g. application and application portfolio).
In this project the students are expected to accomplish the following tasks:

- Formalization of the OutSystems Visual Language into Alloy
- Formalization of the language semantic rules
- Reverse engineering of semantic rules throughout the generation of counter examples and validation with the Agile Platform
- Proposal of language improvements (subject to the choice of direction)
- Validation of the results

OutSystems will provide the students with a specification that defines the language and generates the persistence model (subject to acceptance of NDA), and support with the installation and usage of the Agile Platform.

PROJECT 2: QUALITY INDICATORS FOR AGILE PLATFORM APPLICATIONS

The Agile Platform has helped our clients to develop, deploy and maintain hundreds of enterprise applications. Key to this success is the simplicity and easiness on how to create and change a web application using a visual language, a model that represents the web applications and generators for native C# or Java. Despite this easiness we have observed, for some clients, applications with high complexity and declining quality - observations that corroborates Lehman's laws of software evolution #1 and #7.
In this project we aim at providing objective quality indicators for applications developed with the Agile Platform using metrics. More specifically, we aim at the definition of metrics that quantify properties of the models and criteria for metric acceptance. The purpose is twofold: first to help OutSystems to objectively discuss quality; second to provide mechanisms to guide OutSystems clients in avoiding quality issues. As starting point of this work the students will use well-known source code metrics (e.g. McCabe and Coupling) and techniques to aggregate measurements into ratings (as in "Benchmark-based Aggregation of Metrics to Ratings" by T.L. Alves et al.).
In this project the students are expected to accomplish the following tasks:

- Literature review on source code metrics (e.g. size, complexity and coupling)
- Analysis of the OutSystems Visual Language
- Selection and adaptation of source code metrics to the OutSystems model
- Formalization and implementation of selected metrics
- Metrics collection and analysis of OutSystems applications
- Proposal of metric-based quality indicators
- Validation of the approach

OutSystems will provide support with the Agile Platform installation and support and the definition of the language and models used the generate enterprise applications (subject to NDA).

Projects by galois, INC

Contact: Sally A Browning, PhD (sally@galois.com)

Project 1: Painless I/O and communications subsystems for parallel simulations

In computational science, modeling and simulation codes typically can be split into three major subcomponents: computational logic in which the physics or domain-oriented logic is performed, data management and I/O, and control and data flow abstractions that expose opportunities for parallel and concurrent execution. These are all inter-related: the computational logic is applied to data stored within application data structures; iteration over these data structures is implemented either sequentially or in parallel via some abstraction over an iteration space; storage and in-core management of data is managed by a subsystem that talks to storage and network message passing layers.

Your average computational scientist would ideally like to focus the majority of their attention on the application logic, such as implementing the physics that defines operations over data structures in the code. Details related to building data structures, their memory management, and corresponding I/O for storage and network communication (e.g., marshalling and unmarshalling) is necessary but distracting from the goals of the developer. An opportunity exists to provide both domain specific languages in support of automatic code generation of these parts of the program, as well as static analysis of code that developers write natively in C or C++ to automatically infer what code is necessary to implement these I/O and communication layers.

In the modern world of hybrid systems, in which a computer likely has processors of different classes (e.g., multicore CPU and GPU accelerator), as well as memory hierarchies associated with each type of processor, there exists an opportunity to treat the traditional I/O problem (memory to disk) as an instance of a broader problem of mapping data from one part of the memory hierarchy to another. Static analysis tools to analyze the data usage of a program as well as its traversal over this data are very relevant today in helping computational scientists to use modern machines without forcing them to spend more time than necessary on computational infrastructure instead of the computational logic that is most closely tied to the domain specific problem that they wish to solve.

Project 2: Verification of a C library with Frama-C

Frama-C (`http://www.frama-c.com`) is a tool for compositional verification of C code. It starts with C programs annotated with specifications (such as pre-conditions, post-conditions, loop invariants, and so on) in a language called ACSL (Axiomatic C Specification Language). It then generates a collection of verification conditions — theorems that must be valid for the program to meet its specification — and can use a variety of back-end provers to attempt to discharge these verification conditions. The back-end provers include both automated provers (mostly SMT) and interactive provers.

Most of the existing work with Frama-C has been on fairly restricted programs designed to be verified. However, some initial attempts have been made to apply it to a general-purpose C library [1]. Further work of this sort, perhaps on the

[1] `http://www.fokus.fraunhofer.de/de/quest/_download_quest/_projekte/acsl_by_example.pdf`

standard C library, some of the UNIX APIs, or some other small but widely-used library could be very valuable for getting verification of more real-world top-level programs to be practical.

A potential project could include the following:

- Select a relatively small but influential C library.
- Identify some relevant properties that it should satisfy. A possible starting point could just be memory safety: prove that all pointers that are dereferenced point to
- properly-allocated (and not yet freed) memory.
- Insert these properties into the code as ACSL annotations.
- Attempt to use Frama-C to prove that the properties hold, using either automated or interactive theorem provers.

Project 3: A Verification Condition Generator for LLVM

The goal of this topic is to write a verification condition generator (VCG) for LLVM.

LLVM has emerged as an intermediate language for a number of compilers, such as Apple's C compiler Clang and the Haskell compiler GHC. LLVM has a number of characteristics which make it ideal as a target for verification: the language is reasonably simple and well-defined (as opposed to C), and it is in a form, single static assignment (SSA) which makes it simple to construct linear sized verification conditions (VCs). Finally, a LLVM program is usually the result of some compilation step, and so is closer to the actual code — we do not need to trust that the compiler front-end is working correctly.

There are a number of challenges involved in writing such a VCG: firstly, LLVM is not structured in the same way that a traditional programming language is, so the VCG needs to discover any loops in the program; secondly, there is a gap between the language which the programmer wishes to reason about (for example, C), and LLVM — the programmer will assert that properties hold of the C function, but the VCs will be generated for the intermediate LLVM code, and so assertions will need to be translated. VCGs have been extensively studied in the literature, so answers to these (and other) problems may exist: see, for example, "A New Algorithm for Identifying Loops in Decompilation" by Tao Wei, Jian Mao, Wei Zou, and Yu Chen.