Automatically Discharging VDM Proof Obligations using HOL4

Research Proposal, Final version

March 6, 2007

Author: Sander Vermolen

This proposal will give the setup of the research to automatically discharging the VDM proof obligations. In section 1 I will first give a short domain description to give a basic understanding of the topic. In section 2, a problem description will explain what the project is supposed to solve. A goal description will then give an answer to the question how this project will solve the problem, followed by an approach section, which will explain what the project will look like. And finally, a practical issues section will explain the last implementation details of the project.

1 Domain description

Several techniques will be used during the project. In this section I will mention the most important ones, with a brief description and if possible a reference to where more information can be found.

1.1 Vienna Development Method

The Vienna Development Method[1] (VDM) is a model based formal method, which has its origins in the 1970s at the IBM Research Lab in Vienna. In 1996 the formal specification language, VDM-SL, became the ISO standard language (ISO/IEC 13817-1). It is used in model-driven design. Because of its formal basis, it is very suitable for formal proofs.

VDM-SL as well as VDM++ (an object-oriented extension of VDM) is supported by the VDMTools[2]. This collection of programs supports, among others, syntax and type checking of the formal language, integrity examining, interpretation of the language and various conversions to other known programming languages and modeling formats. VDMTools is currently the main set of VDM-SL supporting tools used in industry. The Overture project[3] is trying to develop a modern open-source set of support tools for VDM++.

1.2 Higher Order Logic

Higher Order Logic (HOL) is an automated proof system. “A programming environment in which theorems can be proved and proof tools implemented.”
The current version of HOL is HOL4[4], which is the successor of HOL98. The program itself, including its libraries and proven theorems are open source. HOL can be controlled using the meta-language Moscow-ML.

2 Problem description

As most high-level languages, VDM++ has a rich typing system. Although this allows for a higher degree of abstraction in VDM models, it is also undecidable at compile time. The type checker of VDMTools is able to detect the obvious type errors, but absence of runtime errors during execution of the model cannot be guaranteed. When it has encountered an undecidable issue, it will generate a proof obligation, which has to be valid to ensure the consistency of the model.

Proof obligations currently have to be proven manually. If all proof obligations have been proven to

1 www.vdmportal.org
2 www.vdmtools.jp/en
3 www.overturetool.org
4 hol.sourceforge.net

be valid, the model is consistent and will therefore not result in runtime errors. This is a correct approach, considering that it will result in a consistent model. However, proving all obligations manually is a tedious task. Especially when dealing with larger and more complex models.

3 Goal description

To solve the problem of manual proof, the goal of this project is to automate this task: Automatically discharging the proof obligations generated by the integrity examiner of VDMTools (and hopefully eventually by an integrity examiner of the Overture project).

This has already been attempted in the PROSPER[5] project, which was (according to its documents) able to discharge 90% of the obligations in a larger case study. The tool set developed in this project used the HOL98 proof tool in combination with the PROSPER environment. Despite its promising results, the tool set was never further developed. And the sources of the conversion have all been lost. Furthermore, there were many parts of the VDM-SL language, which were not or only partially implemented.

Considering the increased power of theorem provers (HOL4 instead of HOL98), there is also good reason to believe that a higher level of automated proof can be reached. This project might not be able to use this yet, but will be able to develop a basis to support it. Hopefully, the ideas of PROSPER which have not been lost can be used along the way. We can say beforehand, that the project is unlikely to reach the level of automation that has been achieved in PROSPER. Even though their ideas and some libraries may be used, this project will only run for half a year and will only be supported by one person, while the PROSPER has run for three years, being supported by multiple developers. Nevertheless, there should be enough time to get a reasonable body of proof support operational. We will use case studies during development, to keep track of what has been achieved. But more about those in section 4 and section 3.1.

The gap between a functional subset of VDM++ and HOL4 that needs to be bridged is quite significant but certainly not impossible. Some of the major issues are:

- Many of the constructs defined in VDM are defined differently or just do not exist in HOL
- Partial functions in VDM have to be translated to total functions in HOL, since HOL only allows for total functions
- HOL does not support an Object Oriented approach, whereas VDM++ does
- VDM is based on a three valued logic system, while HOL is based on two valued logic
- VDM supports sub typing and union types, while HOL does not directly support sub typing and non-disjoint union types. Also, type equivalence in VDM is structural whereas HOL uses a name-based equivalence.
- Many types used in VDM cannot be translated to HOL directly. They will need a conversion and artificially added proof obligations and operators.

Once a conversion is available, the next step will be to actually prove the obligations at hand and thereby prove the consistency of the model. This will involve writing tactics to prove useful theorems beforehand and prove the obligations once available. A great deal of time will have to be put into this, as it is the part where ‘the real thing’ happens.

In short, the project will mainly consist of the following tasks:

- Developing a VDM++ to HOL4 converter, using the abstract syntax of the model as input and HOL4 libraries (hopefully including many of the ideas of PROSPER) to support the output. The proof obligations produced by the integrity examiner will be specified as logical VDM++ expressions (predicates) and can therefore be converted using the same program. This

5 www.dcs.gla.ac.uk/prosper

converter will be specified in VDM++ and code to execute it will be generated automatically or it might also be interpreted.

- Writing the appropriate HOL4 libraries and proving the theorems in them (hopefully using the existing HOL98 libraries).
- Writing the tactics to guide the automated proof of the obligations.

The main challenge of the project lies of course in the proof of the obligations. This does however not restrict to any single task. Each of the tasks has a crucial influence on the final proof. The consequences of decisions made in each of the tasks on the final proof will therefore have to be well thought through.

3.1 Case studies

As a means of evaluating what has been achieved during the project and as a guideline in development, I will use cases as goals of single iterations, or parts of iterations. There will be a number of ad hoc case studies of increasing complexity to be used for development mainly, as these should be defined during development. There are four case studies that have been defined beforehand, which can be used as goals during development. These cases have also been used in the PROSPER project. Although they are written in VDM-SL, a translation to VDM++ is straight forward. The cases are called: Alarm, Autopilot, Memory, Tracker and can be found in the PROSPER code.

If enough time is available, some more sophisticated case studies might also be included in the project, using these to evaluate the quality of the code on a real-life situation. These more sophisticated case studies include a case study of Bluetooth, or a case study supplied by Chess. These will only be useful if the code will be of sufficient quality at the end of the development.

4 Approach

4.1 Starting point

The project will not start from scratch. A lot of documentation from PROSPER is available and some of their Hol98 libraries can be used. This will allow the project to make a quick start, but will also increase the risk of it to fail. All ingredients for a successful project need to have been arranged beforehand. Therefore I will give a list of all prerequisites of the project, explaining why these are available along the way and if necessary what fall-back scenario is available.

- *Understanding of VDM++*

  Since VDM++ is the source language of the translation, a thorough understanding of it will most likely be the first requirement to be able to start the project. Currently I have studied enough documents about VDM, to be convinced that this will not be a risk to the project. There are undoubtedly a lot of topics I have not read about yet, but the ones required to the project can easily be caught up along the way.

- *Understanding of HOL and Moscow-ML*

  Understanding the HOL theorem prover is slightly trickier. Although tutorials on the tool are available on the Internet, these do not all prove to be as useful as they claim to be. To get a good feeling of the theorem prover, most people advise to take about a month of learning. Obviously this would mean a loss of too much time and will therefore have to be done in less.

  Fortunately I have already been able to take some time to get to know the software and its ideas and have tried it. Furthermore, some time can be arranged to learn the tool even further. Apart from all this, it is important to keep in mind that I will not have to start from scratch. I will be able to use the available HOL theorems to get a better understanding of the tool, thus learning to use it along the way.

All together, using the knowledge I currently have about the tool, combined with some extra time to study on the tool and the available knowledge in the existing libraries, I am convinced that using the tool will not be a problem, but certainly an issue not to forget.

- *Knowledge of VDM proofs*

  I already have some knowledge about program proofs, but obviously not about VDM proofs yet. A visit to Newcastle in the first month of the project should solve this issue, where they do have a lot of knowledge on this topic.

- *PROSPER documents*

  These are available, since the project has been finished a long time ago (although the sources of the conversion unfortunately did not survive this long).

- *Proof obligations*

  Eventually, the proof obligations have to be produced by the open-source (Overture) variant of the VDM-SL type checker. But this tool will not be finished by the time the project starts. I will therefore use the closed-source type checker of the VDMTools set during the project.

- *HOL4 theorem prover*

  Apart from the knowledge of HOL, also a running copy has to be obtained. I have already acquired the copy and compiled the predefined theorems, resulting in a running version.

- *VDM++ interpreter or code generator to test the product*

  Although an open-source version of these will not be available during the project, a closed source version will do too (at least during development). VDMTools will provide a closed-source version of both.

- *Eclipse plugin development*

  As the project should finally fit in an Eclipse plugin, knowledge on developing these plugins may be useful. Just as VDM proofs, this knowledge might also be gained in Newcastle.

4.2 Development methods

We know beforehand that I will not be able to automate all proof obligations and will not be able to convert the entire functional subset of VDM++. Furthermore, since it is a project that will take half a year, results during the run, will be required in order to asses the progress made. Therefore, I will be using an incremental approach. First developing a basic version of the final product and I will step by step augment this with more sophisticated parts. One could think of adding supported constructs and adding tactics to support more types of obligations.

The project will therefore consist of iterations. Each iteration will provide and require its own literature study, its own code development, its own case studies and its own additions to the final document. Therefore I will make a schedule at the beginning of each of the phases, to plan these actions and will be able to asses the results afterwards. The case studies at each of the phases will be a goal to work to. I will define these at the beginning and will try to reach them during the phase, or of course document the problems if I am not able to reach the goal.

Each of the phases should result in a deliverable. At least a document resulting from the activities, but most likely in most phases also some source code.

5 Practical issues

5.1 Schedule

Since the exact schedule depends on the time that each increment will take and since the content of the increments cannot exactly be decided upon in advance, giving a schedule of the content of the project here would not be useful. It would change over time anyway. Instead of that, I can give a

schedule of the locations and some very rough ideas of the activities:

When	What	Where
February – March	Developing a basic version (and possibly incrementing it) and literature study on HOL, PROSPER and VDM.	Mainly Århus and part of it in Newcastle
April – First half of May	Incrementing the basic version	Mainly Århus and possibly part of it in Newcastle
Second half of May - June	Incrementing the previous version and if enough time is available, doing some more sophisticated case studies.	Part of it in Århus, part of it in Nijmegen and possibly part of it in close cooperation with Chess
July	Finishing the project[6]	Nijmegen

Note that although these activities seem to be primitive tasks in this schedule, they will not be in practice. The iterative approach will be used to tackle these larger problems. Each iteration consisting of one or more case studies and its own additions to the final product. This explicitly includes writing the final document during the project, not in the end.

5.2 Contact, deliverables & deadlines

Preferably there should be a progress report once every week and regular contact to discuss the deliverables. There should at least be a deliverable at the end of each of the phases, but most likely, more will reduce the amount of work to be done at once and will therefore be preferable. The timing of the deadlines that have not been fixed here, should be fixed when the detailed schedule for each of the phases is defined.

6 This will not be writing the final document. It will be a spare time, that can be used for tasks that should, but have not been done during the project, but also time for the final activities, such as giving a final presentation of the project. If no spare time is required, it can be filled with an extra iteration.