Imperial College London

Department of Computing

Proof Searching & Proof Checking Software Validation Report

Computing Group Project

Supervisor: Dirk Pattinson

Autumn 2010

Jannis Bulian, jb1508@ic.ac.uk, 567339

Michal Parusinski, mgp08@ic.ac.uk, 566542

Ka Wai Cheng, kwc108@ic.ac.uk, 548464

Saguy Benaim, ssb08@ic.ac.uk, 552374

Contents

1. Summary 3

2. Unit Testing 3

3. General Validation 5

4. Managerial Documentation 7

A. Log-Book 20

1. Summary

This document is the third report for our Proof Searching and Proof Checking project, which deals about formal reasoning in Description Logic a formal tool for knowledge representation. In this report, we will detail the software validation methods used in the development of the project, as well as an account of the project management including a detailed log of our activities.

To enhance the correctness of our code, our software validation methods include various forms of testing (unit testing, stress testing, black-box, grey-box and white-box testing) and other tools such as *hlint*, a bug tracker and code reviews. To ensure good progress in our project, our management tools include tools provided by Google Code, most notably Mercurial, as well as frequent SCRUM-like meetings.

2. Unit Testing

As indicated in the inception report, a test-driven development method was adopted; hence unit testing was done at all stages of development. Unit tests were written before the implementation of functions or simultaneously; and test suites are run regularly, especially after any changes to code, to ensure the program still works as expected. All areas of code were tested in this way individually: model/proof construction; model checking; proof checking; parsing of user input and test files; and the outputting of the constructed proof and model.

Benefits

Although writing an extensive set of unit tests that cover all possible cases take almost as much time as implementing the code itself, there were many benefits from this approach.

By simultaneously writing the unit tests, it helped programmers think about and have a clear view of the expected/desired output of functions. This made coding of functions easier and members were able to identify mistakes quickly. Most of the time, bugs were found as the functions were implemented and were therefore corrected immediately.

When moving on to code other parts of the program, we have confidence that tested code is correct and produces expected results when given expected inputs. As a result, we were able to highlight differences in expectations between members writing different parts of the program for discussion in our meetings and come to an agreed solution.

Unit tests were written for code coverage, ensuring all input cases were considered and our code produces the expected results. Otherwise, unit testing exposed where cases were not handled correctly. This helped identify most bugs early on, before code became more complex and difficult to resolve.

In particular, having extensively tested proof checker and model checker by unit tests enabled these components of the program to be used with high confidence in other types of testing. The proof and model checkers are important in these other tests to check the results produced, for the automatically generated complex input cases, by the proof and model construction part of the program, which is very complicated and much harder to test simply by unit tests due to its exploratory nature.

Bugs

The bugs unit testing revealed include cases that were not covered properly, errors in the code, and bugs introduced due to changes in code.

Unit testing immediately revealed some forgotten cases in the initial code, such as:

- When falsity was in the set of inputs for the proof and model construction, a model was produced rather than the expected proof containing only falsity to show unsatisfibaility.

- In the case where top (truth) was among many concepts in the list of inputs, an

?

incorrect model ~~with~~ was returned without ensuring all other concepts in the list of inputs were also satisfiable.

- The possibility of cycles in the input global assumptions was not dealt with by the proof and model construction stage.

Some mistakes in code that may have been overlooked without our approach to unit testing:

- Some incorrect checks performed by the proof checker and model checker.
- Proofs should deal with set of concepts not containing any duplicates, however, the proof searcher produced a result containing duplicated concepts as a result of applying the proof rule for Exists since a specific case was not considered. This was where the inputs contained:
  - An Exists concept stating there exists a successor to a particular relation that satisfies a concept
  - A Forall concept stating all successors of the same relation must satisfy the same concept specified by the exists concept

e ?

After some changes to code, some bugs were introduced since the affect of the changes on all cases ~~were~~ was not considered. For example:

- After combining the separate parsers for user input and benchmark files to reuse functions, the parsing for one of the benchmark files was incorrect and needed to have further changes.

3. General Validation

We used several methods in addition to unit tests to validate the program. Although the unit tests were very useful and helped us find many bugs an early stage, they don't help with more complex bugs. This is why we made use of *hlint*, *code reviews* and *grey/black box testing* to complement them.

The user interface we chose allows to input concepts in a familiar way for anyone who has worked with formal logic before (e.g. '$A \& B \to C$'). We have not systematically tested this on users, but the users of our program were always able to learn the interface quickly.

We use *hlint*, a program similar to lint, that analyses Haskell programs and suggests changes, to ensure a good and consistent style throughout the program. This not only ensured that the source code is easier to read and understand, but also made programming mistakes less likely.

As mentioned in previous repo we make extensive use of *code reviews*. These helped improve the quality of the code and made sure that we would understand each others code. It also lead to the discovery of some small programming mistakes in the reviews.

After each significant change we used *grey-box testing* to validate these changes, i.e. we tried, using our knowledge of the inner workings of the program, to find possible problems. This sometimes lead to the discovery of small issues, for example a forgotten case.

Furthermore, we used two different ways of *black-box testing* in addition to the unit testing described above. We wrote a small program that automatically generates large test cases, runs the proof/model searcher on them, and then either the proof or the model checker on the result. We also wrote a parser, so we could read complex 'benchmark concepts' that are available from the proof searcher community, and test these in the same way. Both these approaches turned out to be very useful and helped us discover some rather obscure bugs (e.g. bugs that would only occure if concepts were nested in a very particular and complex way). These kind of tests can also be understood as *stress testing* of our program, because the cases are larger than we expect normal user input to be and are very likely to test concepts that go beyond what users would normally input.

We integrated these tools into a validation process. We usually ran *hlint* and unit tests before submitting the code to the repository. Then we gave code reviews and used grey-box testing to improve the quality of the code and find bugs that weren't found before. On a regular basis we ran the black-box tests on the whole program, discovering issues that were missed in the previous steps. This whole setup gave us high confidence that the program is working correctly.

4. Managerial Documentation

Management Tools

We hosted our project on Google Code which offered us various development and management tools. As part of the Google Code environment, we used Mercurial Version Control and reported bugs and other issues on the Issue Tracker. We also used the Wiki and Code review options in Google which helped us to provide documentation and communicate our changes efficiently and effectively. The use of Google's environment helped us centralise all the management tools in one place which is easy to access and control.

We also used cabal, a system for building and packaging Haskell libraries and programs, to manage different packages and libraries in our project.

Management Policies

The most efficient management policy we used, was the frequent SCRUM like meetings which we held three times a week. Two meetings would be held with the groups members only. In these, we would describe the progress with respect the the set tasks, look at any problems or issues that occurred, and set new tasks for the next meeting. The other meeting would be held with our supervisor in which we discussed next iterations and any issues regarding the code.

These meeting were key in the way code was changed. Using Mercurial we ensured code changes were coordinated. Code reviews and issues in the Issue tracker were addressed and code was changed accordingly.

Before submitting code we run the unit tests (as a test suite) to see if any tests broke as a result of our code changes. We built additional unit tests and added them to the test suite. The tests suite was managed using cabal, which allowed an easy way of running the test suite. We also run *hlint* to improve the code quality before committing the changes using Mercurial.

Knowledge Transfer

Most knowledge transfer was done through the meetings described above. In these informal meeting we discuss technical and theoretical issues and scheduled meetings. During these meetings, we solved any problems that have risen from a gap in our technical or theoretical knowledge. Outside of meetings, we used code reviews through Google Code to help other group members improve their code and share our knowledge. The most significant knowledge transfer was made during the meetings with our supervisor, were the more difficult concepts and later iterations were discussed.

Group Meetings and Tasks

All group meetings, their dates and people attended are recorded in the log book. The log book itself contains detailed description of these tasks and is included in the appendix. The following table gives the distribution of tasks between group members. Additional tasks of code reviews, peer reviews and attending meetings or discussions were not included as tasks.

Week:	11 October 2010 - 17 October 2010
J. Bulian:	• Investigated different possibilities for collaboration platforms, tried out github, and set up a Google Code account, including some initial setting (3 hours) • Read papers and thought about design (6 hours)
S. Benaim:	• Read papers and thought about design (6 hours)
M.G. Parusinski:	• Read papers and thought about design (6 hours)
K.W. Cheng:	• Read papers and thought about design (6 hours)

Week:	18 October 2010 - 24 October 2010
J. Bulian:	• Set up a template for the first report and started writing own part, proofreading and discussing other parts (4 hours) • Read more papers, improved and elaborated design (5 hours)

S. Benaim:	• Wrote own part of report, proofread and discussed other parts (4 hours) • Read more papers, improved and elaborated design (5 hours)
M.G. Parusinski:	• Wrote own part of report, proofread and discussed other parts (4 hours) • Read more papers, improved and elaborated design (5 hours)
K.W. Cheng:	• Wrote own part of report, proofread and discussed other parts (4 hours) • Read more papers, improved and elaborated design (5 hours) • Set up some additional software (1 hour)

Week:	25 October 2010 - 31 October 2010

J. Bulian:	• Finalised report, proofread the other parts and rewrote the introduction (4 hours) • Read additional papers (2 hours) • Started work on the proof/model searcher (4 hours)
S. Benaim:	• Finalised report and proofread (4 hours) • Started work on the proof/model searcher, jointly with Jannis (4 hours) • Wrote unit tests for proof/model searcher (8 hours)
M.G. Parusinski:	• Finalised report and proofread (4 hours) • Created Signature.hs/Proof.hs/Model.hs (4 hours) • Wrote unit tests (2 hours)

K.W. Cheng:	• Finalised report and proofread (4 hours) • Wrote proof checker (4 hours) • Wrote unit tests for proof checker (2 hours)

Week:	1 November 2010 - 7 November 2010
J. Bulian:	• Finished first implementation of proof/model searcher (6 hours) • Fixed some issues with the proof/model searcher (2 hours) • Wrote a main file (1 hour)
S. Benaim:	• Added more unit tests for proof/model searcher (4 hours)
M.G. Parusinski:	• Added precedence rule for Description Logic (2 hours) • Added a function sending concept to its negation normal form (1.5 hours) • Model checker for atomic concepts (2 hours) • Wrote tests for the above (1 hour)

K.W. Cheng:	• Wrote proof checker (4 hours) • Wrote unit tests for proof checker (3 hours) • Tested and fixed bugs (1 hour)

Week:	8 November 2010 - 14 November 2010
J. Bulian:	• Started work on progress report, setting up a template (3 hours) • Made the interfaces for proof searcher / proof checker match (1 hour) • Fixed bugs in the proof/model searcher and improve style (4 hours) • Factored out some useful functions and added several new ones to a dedicated utils file (2 hours)
S. Benaim:	• Started work of progress report (3 hours) • Started work on cycles within proof/model searcher (6 hours) • Added additional tests for proof/model searcher (1 hour)

M.G. Parusinski:	• Basic testing for model checker (3 hours) • Model checker without reports for failure (4 hours)
K.W. Cheng:	• Added additional checks on inputs in Proof Checker (3 hours) • Added more unit tests (2 hours) • Improved code and made changes to code due to changes in proof tree data structure (1 hour) • Testing and fixing bugs (1 hour)

Week:	15 November 2010 - 21 November 2010
J. Bulian:	• Fixed more bugs in the proof/model searcher (3 hours) • Wrote a parser for modal K formulas into ACL (8 hours) • Worked on improving some of the new utilities in ProofUtils (2 hours) • Wrote parser for user input (2 hours)

S. Benaim:	• Work on cycles within proof/model searcher (5 hours) • Added additional tests for proof/model searcher to test cycles (3 hours)
M.G. Parusinski:	• Extensive testing for the model checker (2 hours) • Model checker with reports for failure (3 hours) • Cabal package management system setup (1 hours) • Function to perform all tests (1 hour)
K.W. Cheng:	• Improved code and messages produced by Proof Checker and adapting unit tests (2 hours) • Testing and fixing bugs after changes to Proof Checker (2 hours) • Wrote separate parsers for benchmark files using Happy (5 hours) • Combined parsers to use reuse grammar rules, tokens, functions (2 hours) • Testing parser and fixing bugs (1 hour)

Week:	22 November 2010 - 28 November 2010
J. Bulian:	• Further improved user input language (2 hours) • Wrote tests for the user input language (3 hours) • Fixed some of the proof search tests (1 hour) • Cleaned up of the parser and improved performance (2 hours)
S. Benaim:	• More tests for cycles (3 hours) • Additional work on cycles implementation (7 hours)
M.G. Parusinski:	• Creation of a test generating function (3 hours) • Improved clarity of the units tests combined them together (15 hours)

K.W. Cheng:	• Improved parser to reuse code and remove redundant code, grammar rules, tokens (2 hours) • Improved proof checker messages (2 hours) • Adapted unit tests to use the clearer function written by Michal (2 hours) • Added unit tests for lexical analysis of benchmark files and fixed bugs revealed (3 hours)

Week:	29 November 2010 - 5 December 2010
J. Bulian:	• Improved the test setup and including the parser tests (2 hours • Fixed more bugs and factored out some code as well as further improvements in the proof search (5 hours)
S. Benaim:	• General improvements in proof search (3 hours) • Additional work on cycles implementation (5 hours)

M.G. Parusinski:	• Parser test for simple cases (2 hours) • Helped Saguy with debugging (2 hours)
K.W. Cheng:	• Worked on locating the exists bug and finding the smallest case which produces the bug (5 hours)

Week:	6 November 2010 - 11 December 2010
J. Bulian:	• Helped fix problems with caching (part of proof/model searcher), studying code (3 hours) • Writing validation report, settings up templates (4 hours)
S. Benaim:	• Fixed problem with caching (5 hours) • Wrote own part of validation report (4 hours)

M.G. Parusinski:	• Wrote own part of validation report (4 hours) • Function to create pdf's containing the proof (3 hours)
K.W. Cheng:	• Wrote own part of validation report (4 hours) • Model graphical output into DOT language (5 hours) • Unit tests for model output (3 hours) • Testing and fixed bugs for model output (2 hours)

A. Log-Book

Date:	12th October 2010
People present:	Dr. Pattinson, S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Statement of the problem and its implications 2. Possible challenges of the problem 3. Reports and deadlines to meet 4. Goals of the project
Achievements:	N \ A
Agreements:	N \ A
Tasks assigned:	Read literature about Description Logic and Knowledge Based system

Date:	13th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Statement of the problem and its implications 2. Organisation of the project 3. Assignment of potential roles in the group 4. Goals of the project 5. Design of the software
Achievements:	N \ A

Agreements:

1. Assignments: Model Checker (M.G. Parusinski), Proof Checker (K.W. Cheng), Proof and Model Searcher (J.Bulian and S.Benaim)
2. Critical Goals (for our project):
   - Implementing an algorithm that performs proof searching
   - Implementing an algorithm that performs model construction
   - Implementing an algorithm that performs proof checking
   - Implementing an algorithm that performs model checking
   - Ensuring our algorithms are correct if terminates
3. Secondary goals
   - Generating human readable format for models and proofs
   - Implement a parser for reading formulas
   - Testing the code with tests unit
   - Extend to non-standard logics
4. Programming language used will be Haskell
5. M. G. Parusinski becomes the Secretary of the project, J. Bulian the Leader
6. We will use tools to enhance programming: Code review, Version Control, Automated testing

Tasks assigned:	1. Learn about Automated Testing 2. Think further about the design 3. Think about which Version Control to choose 4. Provide any relevant documentation to other team members

Date:	14th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Statement of the problem and its implications 2. Organisaton of the project and agree on role assignment discussed in previous meetings 3. Theoretical issues relating to Description Logic
Achievements:	N \ A
Agreements:	1. HUnit will be used for automated testing 2. Code review and version control will be done through Google code.
Tasks assigned:	N \ A

Date:	15th October 2010
People present:	Dr. Pattinson, S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Inception report 2. Division of the work amongst team members 3. Implementation issues (type definitions) 4. Whether to make the project open-source
Achievements:	N \ A
Agreements:	Make the project open-source
Tasks assigned:	1. Read literature about DL, tableau calculus 2. Check CoLoss (a software already performing proof search) 3. Correct the type definitions

Date:	18th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng

Points discussed:	1. Inception report 2. Timetable and schedule for the project 3. Should the language definition be minimal or not 4. Setting up regular meetings
Achievements:	Setting-up data types necessary for the project
Agreements:	1. Monday meetings at 12.00-13.00, everybody agreed this as their first choice 2. Thursday meeting at 11.00-12.00, everybody agreed this as their first choice 3. Use the minimal Haskell definition for the Logic representation

Tasks assigned:	1. Set up a Google Account for code.google.com if not already available 2. Install Mercurial on favourite O.S. 3. Read about DL and tableau calculus 4. Compare Extreme Programming and Scrum 5. Think about first iteration plan 6. Think about schedule 7. Think about development method (see above)

Date:	21th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Agile development method used for the project 2. Algorithms to implement
Achievements:	Everybody has access to the code.google.com services Mercurial, Code Review, etc.
Agreements:	N \ A

Tasks assigned:	1. Read the literature more carefully and understand the algorithms 2. Look at all the Agile Software Development methods and choose one 3. Describe the first iteration plan 4. Finish the draft timetable

Date:	22th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Testing practices 2. Agile software development 3. Inception report
Achievements:	N \ A

Agreements:	1. Procedure: write tests, write code, do testing, and then submit code 2. Review others' code whenever there are new submissions and if possible 3. Comment the code with clear comments 4. Respect haskell standards
Tasks assigned:	1. Read all relevant documentation 2. Read about agile software development 3. Agree on goals and agile software development method 4. Each member choose features for the software development pratice 5. Write the report 6. Deadline spreadsheet and tasks spreadsheet

Date:	26th October 2010
People present:	S. Benaim, J. Bulian, K.W. Cheng
Points discussed:	Inception Report
Achievements:	N \ A

Agreements:	1. Software development method Agile development: mixture of XP, Crystal Clear, Agile Unified Process 2. Agreement on design and early assignments of tasks 3. Agreement on a draft schedule 4. Agreement on a first iteration plan
Tasks assigned:	N \ A

Date:	27th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	Finalise the inception report and for check any mistakes
Achievements:	Inception Report
Agreements:	N \ A
Tasks assigned:	N \ A

Date:	28th October 2010
People present:	Dr. Pattinson, S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Ask the supervisor to check the inception report and ask questions about the literature 2. Additional corrections for inception report

Achievements:	N \ A
Agreements:	N \ A
Tasks assigned:	N \ A

Date:	29th October 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. First iteration plan
Achievements:	N \ A
Agreements:	N \ A
Tasks assigned:	1. Write constructor functions of Description Logic to allow modular design 2. Give precedence rule for operators (like in logic) 3. Implement rules as an abstract data type 4. Add knowledge base to the proof system 5. Use 'either' type for outputs

Date:	1st November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	Deal with unusual cases inside the algorithms

Achievements:	1. Precedence rule for operators 2. Constructor functions and abstract data type representing rules implemented
Agreements:	1. Write code for ALC instead of AL 2. Model checker and proof checker will provide a report in case of failure
Tasks assigned:	1. Model checker which outputs a report 2. Proof checker which outputs a report 3. Proof/Model searcher

Date:	5th November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Check progress and set new tasks

Achievements:	1. Early implementation of the proof checker 2. Pseudo-code and design for proof/model searcher 3. Achieved model checker for simple concepts without reports for failure
Agreements:	N \ A
Tasks assigned:	1. Create a concept sorter, to improve Proof or Model Searcher 2. Output report for model checker 3. Finish the proof/model checker

Date:	8th November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Check progress and set new tasks

Achievements:	1. Working basis for the model checker 2. Proof checker done with basic unit tests 3. Proof/Model searcher with basis unit tests
Agreements:	Agreements for the progress report: 1. Decide to write the code for ALC instead of AL since it was agreed as it is more pratical as a consequence we combined two iterations 2. Two new features has been added which returns why proofs or models fail, and users can check their own models and proofs 3. No new extension planned at the moment 4. Same agile development method including progress mesure 5. Code review for managing different levels of haskell programming skills 6. Discussing theory in groups for managing different levels of logic skills 7. Fairly divide tasks amongst group member, plus discussing issues to ensure members contribute fairly 8. The Haskell Programming language has no license restrictions regarding written code

Tasks assigned:	1. Output message if model checker return False, and provide interface for Proof/Model searcher 2. More extensive testing for the proof/model searcher 3. More extensive testing for the model checker 4. More extensive testing for the proof checker 5. Agree on other logics which we want to extend our software to 6. Write draft progress (part 1 of report 2) : Saguy 7. Write draft revisions (part 2 of report 2) : Jannis 8. Write draft people management (part 3 of report 3) : Michal 9. Write draft Ethical and Environmental impact (part 4 of report 4) : Ka Wai

Date:	11th November 2010
People present:	Dr. Pattinson, S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Theory about Tableau Calculus 2. Extensions to other logics 3. Progress report

Achievements:	Early draft of report except for precise schedule
Agreements:	A possible extension: Counting successors, ABox (validity at specific points)
Tasks assigned:	N \ A

Date:	15th November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	N \ A
Achievements:	N \ A
Agreements:	N \ A
Tasks assigned:	1. Extensive testing for each part of the software 2. Set up cabal for package management 3. Set up cabal for running all the tests together

Date:	19th November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	Testing and running HLint
Achievements:	N \ A
Agreements:	M. G. Parusinski becomes the Test Master ensuring that the tests are consistent and can be all run together

Tasks assigned:	1. Clean up the tests (Michal) 2. Write test generation functions (Michal) 3. Turn inputs into sets (Saguy) 4. Resolve global testing (Michal) 5. Deal with cycles/caching (Saguy) 6. Output testing result to file (Michal) 7. Implement the parser (Jannis + Ka Wai)

Date:	22nd November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	N \ A
Achievements:	1. Early implementation of the parser 2. Clearer and more consistent messages provided by cabal testing system 3. Several issues has been corrected
Agreements:	N \ A

Tasks assigned:	1. Improve clarity of the testing for Proof and Model Searcher (Michal) 2. Improve clarity of the testing for Proof Checker (Ka Wai) 3. Create a test generating function (Michal) 4. Add parser to the Cabal package (Michal) 5. Improve Model Checker (Michal) 6. Improve parser implementation (Jannis + Ka Wai) 7. Write tests for the parser (Jannis + Ka Wai)

Date:	25th November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	Dealing with caching how to avoid infinite loops in proofs.
Achievements:	1. Model Checker improved 2. Tests for the parser has been implemented
Agreements:	N \ A
Tasks assigned:	Same tasks at hand

Date:	26th November 2010

People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	Fixing several issues that has arised
Achievements:	Early implementation for test generation
Agreements:	N \ A
Tasks assigned:	1. Think about how to implement *ABox* extension (individual) (All) 2. All together provide help in solving issue with cycles 3. Implement tests for the Parser (Michal) 4. Choose between *ABox* and/or satisfaction operators and/or counting restrictions

Date:	29th November 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. *ABox* (Individuals) (Logic) 2. How to output models and proofs to a visual format
Achievements:	1. Significant improvement in implementing cycles 2. Early testing implementation for the parser
Agreements:	N \ A

Tasks assigned:	1. Find a proof system for individuals

Date:	2^{nd} December 2010
People present:	Dr. Pattinson, S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	How to adapt the proof searcher for *ABox* (Individuals) also known as ALCO • Labelled tableaux using Labelled sequences: always some individual considered and new rules $\Longrightarrow$ equality rule, substitution rule • Blocking strategy to avoid non terminating cases, since extension rule always add a new point for non empty *TBox* in case above
Achievements:	N \ A
Agreements:	N \ A
Tasks assigned:	N \ A

Date:	3^{rd} December 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Validation report 2. Graphical outputs for models and proofs

Achievements:	1. Several test has been implemented for parsers 2. Bug fixes and improvement for cycles
Agreements:	N \ A
Tasks assigned:	1. Implement model output (Ka Wai) 2. Implement proof output (Michal) 3. Fixes all bugs discovered 4. Read ALCO papers

Date:	3rd December 2010
People present:	S. Benaim, J. Bulian, M.G. Parusinski, K.W. Cheng
Points discussed:	1. Third report (Software Validation) content 2. Impact of testing on our project

Achievements:	1. Some informations about testing and its benefits has been sent to Ka Wai 2. Model output partially implemented 3. Bug fixed about applyExists 4. Partial proof output
Agreements:	N \ A
Tasks assigned:	1. Improve proof output and model output 2. Finish the report 3. Information about validation techniques if relevant 4. Fix cycles and/or caching 5. Think about division of work for the final report 6. Write the main function (Michal)