<name>
<College>
<CRSID>

Diploma in Computer Science Project Proposal

Exact Real Arithmetic

<date>

Project Originator: This is a Model Project Proposal

Resources Required: See attached Project Resource Form

Project Supervisor: *<name>*

Signature:

Director of Studies: *<name>*

Signature:

Overseers: *<name>* and *<name>*

Signatures: *<no need to obtain Overseers' signatures yourself>*

< *This Model Proposal refers to the language ML and to data structures known as lazy lists. These are used only as examples. It is perfectly possible to use a different language and to use different data structures. Likewise any specific representations of numbers are also given simply as examples. At various points in this document* alternatives *are suggested or implied, and the submitted proposal is to be selected from the options given.*>

Introduction and Description of the Work

Floating-point calculations do not, in general, lead to exact results. Moreover, there are circumstances when the results of very few operations can be wildly in error even when using 64-bit precision. A well-known example is the following iteration which takes the golden ratio as the starting point:

$$\gamma_0 = \frac{1+\sqrt{5}}{2} \qquad \text{and} \qquad \gamma_n = \frac{1}{\gamma_{n-1}-1} \qquad \text{for } n > 0$$

It is trivial to demonstrate that $\gamma_0 = \gamma_1 = \gamma_2 \ldots$ but when this iteration is carried out using any finite precision arithmetic the values diverge from γ_0 after very few iterations.

People concerned with theorem proving sometimes want to perform arithmetic calculations but cannot tolerate any possibility of the end results being incorrect. In general, the value to be computed will be a boolean obtained by comparing the results of various numeric calculations. For example, it may be important to know whether $\gamma_{20}<4$ even though the exact value of γ_{20} is not too important.

One way of enabling a theorem prover to evaluate such a boolean reliably is to provide it with so-called "exact" real arithmetic. Where normal floating-point arithmetic uses fixed precision, exact arithmetic will work in variable precision. Furthermore, at all stages of the calculation, it will keep track of the extent to which rounding errors could have accumulated. When it comes to display a result, a comparison is performed to check that all is well.

A key idea is that when an exact real arithmetic system finds itself close to being compromised by the effects of round-off it must be able to go back and (automatically) extend the precision of all its previous working until eventually it has a result that is as accurate as is needed.

The effect is that when the system computes a value and is asked to display it to (say) five significant figures the user can be certain that the figures displayed are correct, however unstable the numerical calculation involved.

Tests involving equality necessarily give trouble. For example, evaluating the boolean formula $\sin(\pi)=0.0$ might reasonably loop, evaluating $\sin(\pi)$ to ever higher precision trying to detect that the answer was non-zero. This problem is inherent in the idea of exact real arithmetic (exactness comes at the cost of guaranteed termination) but its implications should be regarded as outside the scope of any project based on this Model Proposal.

Resources Required

<The most suitable language may turn out to be non-standard and its availability needs to be established. This would be the case if Poly/ML is chosen as the language.>

Starting Point

<This is the place to declare any prior knowledge relevant to the project. For example any relevant courses taken prior to the start of the Diploma year.>

Substance and Structure of the Project

<You should choose between options I and II below after discussions with your Supervisor and Overseers.>

I The aim of the project is to investigate alternative representations of real numbers and to produce a suite of procedures for carrying out arithmetic operations. It will implement methods <X> and <Y> as described below and compare them in terms of implementation complexity and performance.

II The aim of the project is to implement the <X> style of exact real arithmetic as described below.

<If just one method is implemented it will be expected that more work will be done in that single implementation than if two or more different schemes are being compared.>

1. A real number can be represented as a lazy list, with each successive value in the list giving a better approximation to the number concerned. The values stored are numerators of fractions which have implicit denominators, so that item i in the list is treated as having denominator B^i for some pre-chosen base B.

   Lazy lists (which it will be reasonable to implement using the language ML) enable later, higher precision, representations to be calculated only when needed. Poly/ML supports arbitrary precision integer arithmetic and its use means that having huge multi-precision integers as later terms in a lazy list does not present a problem.

2. A real number can be stored as a pair of rational numbers, each with a common denominator which is a power of 2. The two numbers represent the lower and higher end-points of an interval within which the exact value is known to lie.

   In addition to the interval, a number needs to have associated data structures that enable the source of the number to be identified and extra calculations to be performed which update the number in such a way that the width of the interval is reduced.

   The rational numbers involved frequently grow to be multiple-precision integers, so either the code has to be written in a language that supports them or time has to be allowed for implementing support for them. <*You have to decide which approach to adopt before handing in your Project Proposal.*>

3. A real number can be stored as the first few partial quotients of its continued fraction. The continued fraction for a number x is a sequence of integers a_i defined by letting $x_0 = x$ and then

$$a_i = \text{floor}(x_i) \qquad \text{and} \qquad x_{i+1} = \frac{1}{(x_i - a_i)}$$

   Arithmetic on continued fractions has been explored by Gosper.

4. A fractional number with decimal expansion of (say) $0.14159\ldots$ can be represented as a lazy list $[1, 4, 1, 5, 9, \ldots]$. Arithmetic on such lazy decimals can run into severe trouble with the confusion between $[1, 9, 9, 9, 9, \ldots]$ and $[2, 0, 0, 0, 0, \ldots]$, and it may prove necessary to adjust the representation slightly to soften the impact of this problem.

   It would also be natural to implement a scheme which uses a radix larger than 10, but nevertheless all calculations will be performed in terms of single digits so the representation does not need to rely on any underlying big-integer arithmetic.

5. Other representations for exact real numbers can also be found in the literature, for instance storing a number by giving a polynomial equation that it satisfies together with a bounding interval. For example, the square root of 2 could be represented by $[q^2 - 2 = 0, 1 < q < 2]$.

<*The sketches above are to be taken as starting points for investigation of the literature and discussion with your Supervisor. A look at the 1996 Diploma Dissertation,* Exact Real Arithmetic, using ML, *by Chetan Radia, is likely to prove useful since it provides some pointers into the literature, and discusses terms such as B-adic and B-approximable.*>

<*Note also that it may make sense to store some numbers in your package in one format and others in another. For example, it might be sensible to represent numbers that happen to be rational as exact fractions but reserve a more general scheme for numbers like* $\sqrt{2}$.>

The project has the following main sections:

1. A study of the selected representation of real numbers and the algorithms for handling numbers so represented. This will involve study of both general background material on error propagation in numerical calculation, and a detailed study of previous work both in Cambridge and elsewhere of the selected style of exact arithmetic.
2. Familiarisation <*if need be*> with the programming language to be used and the surrounding software tools. Detailed design of data structures to represent real numbers. Planning implementation and test strategies for basic arithmetic.
3. Developing and testing the code for the principal arithmetic operations (addition, subtraction, multiplication, division and square root).
4. Evaluation using the golden ratio example and other illustrative cases. Collating the evidence that will be included in the Dissertation to demonstrate that the code behaves as required.
5. Writing the Dissertation.

<If two or more representations are to be implemented and compared, the above is probably sufficiently ambitious. If just one representation is to be used it would be appropriate to augment (3) with further procedures such as logarithms and trigonometrical functions. Before simply adding these to the text of your proposal you need to check the literature and discuss details quite carefully with your Supervisor since not all representations support the elementary functions at all comfortably.>

<It would also be possible to adjust the style and emphasis of the project either towards the best possible computational capability for your code, or towards the most careful and formally justified certainty that the answers delivered are correct. It is unlikely that this project would call for a substantial or elaborate user-interface but that side of things could also figure as a modest aspect of the entire project.>

Success Criterion

To demonstrate, through well-chosen examples, that the project is capable, with reasonable efficiency, of performing exact real calculations.

Timetable and Milestones

<In the following scheme, weeks are numbered so that the week starting on the day on which Project Proposals are handed in is Week 1. The year's timetable means that the deadline for submitting dissertations is in Week 34.>

<*In the Project Proposal that you hand in,* actual dates *should be used instead of week numbers and you should show how these dates relate to the periods in which lectures take place. Week 1 starts immediately after submission of the Project Proposal.*>

<*The timetable and milestones given below refer to just one particular interpretation of this document: this uses ML and a lazy-list data-structure. Even if you select exactly this interpretation you will need to review the suggested timetable and adjust the dates to allow as precisely as you can for the amount of programming and ML experience you have at the start of the year. Take account of the dates you and your Supervisor will be working in Cambridge outside Lecture Term. Note that some candidates write the Introduction and Preparation chapters of their dissertations quite early in the year, while others will do all their writing in one burst near the end.*>

<*If Java were to be used there would be no need to allow time to learn ML but time would have to be budgeted for gaining experience of lazy lists in Java and for writing procedures to handle arbitrary precision integers.*>

Before Proposal submission

<*This section will not appear in your Project Proposal.*>

Discussion with Overseers and Director of Studies. Allocation of and discussion with Project Supervisor, preliminary reading, choice of the variant on the project <*a lazy list representation in this example*> and language <*ML in this example*>, writing Project Proposal. Discussion with Supervisor to arrange a schedule of regular meetings for obtaining support during the course of the year.

Milestones: Phase 1 Report Form (on the Monday immediately following the main Briefing Lecture), then a Project Proposal complete with as realistic a timetable as possible, approval from Overseers and confirmed availability of any special resources needed. Signatures from Supervisor and Director of Studies.

Weeks 1 to 5

<*Real work on the project starts here (as distinct from just work on the proposal). A significant problem for Diploma candidates is that this critical period largely coincides with the Christmas vacation. There is no guarantee that supervisors will be available outside Lecture Term, but Diploma students take much less of a Christmas break than undergraduates do, and so have some opportunity for uninterrupted reading and initial practical work at this stage. It is important to have completed some serious work on the project before the pressures of the Lent Term become all too apparent.*>

Study ML and the particular implementation of it to be used. Practise writing various small programs, including ones that use lazy lists.

Milestones: Some working example ML programs including cases that create and use lazy lists.

Weeks 6 and 7

Further literature study and discussion with Supervisor to ensure that the chosen representation of exact real numbers is understood. Implementation in ML of data types which accord with the representation, and code that makes it possible to construct data structures representing simple constants. Write code that can be used to display the data structures in human-readable form so that it is possible to check that they are as expected.

Milestones: Ability to construct and display data structures that represent simple constants such as 0, 1 and 2.

Weeks 8 to 10

Implement code to add, subtract and negate numbers. Also implement either code that displays a number to a specified guaranteed precision or which performs a “less-than” comparison between two numbers. This will involve being able to make the precision of intermediate results grow on demand.

Start to plan the Dissertation, thinking ahead especially to the collection of examples and tests that will be used to demonstrate that the project has been a success.

Milestones: Ability to run some very simple calculations in such a way that precision has to be increased (requiring a lazy list has to be extended) during the course of the calculation.

Weeks 11 and 12

Complete code for the multiplication and division. Refine the code for addition, subtraction etc. Prepare further test cases. Review timetable for the remainder of the project and adjust in the light of experience thus far. Write the Progress Report drawing attention to the code already written, incorporating some examples, and recording any augmentations which at this stage seem reasonably likely to be incorporated.

Milestones: Basic arithmetic operations now working, but probably with some serious inefficiencies in the code, Progress Report submitted and entire project reviewed both personally and with Overseers.

Weeks 13 to 19 (including Easter vacation)

Implement square roots <*and any other elementary functions selected at proposal time to bulk out the single-implementation project*>. Write initial chapters of the Dissertation.

<*The Easter break from lectures can provide a time to work on a substantial challenge such as the computation of logarithms, where an uninterrupted week can allow you to get to grips with a fairly complicated algorithm. This is a good time to put in some quiet work (while your Supervisor is busy on other things) writing the Preparation and Implementation chapters of the Dissertation. By this stage the form of the final implementation should be sufficiently clear that most of that chapter can be written, even if the code is incomplete. Describing clearly what the code will do can often be a way of sharpening your own understanding of how to implement it.*>

Milestones: Preparation chapter of Dissertation complete, Implementation chapter at least half complete, code can perform a variety of interesting tasks and should be in a state that in the worst case it would satisfy the examiners with at most cosmetic adjustment.

Weeks 20 to 26

<*Since your project is, by now, in fairly good shape there is a chance to use the immediate run-up to exams to attend to small rationalisations and to implement things that are useful but fairly straightforward. It is generally not a good idea to drop all project work over the revision season; if you do, the code will feel amazingly unfamiliar when you return to it. Equally, first priority has to go to the exams, so do not schedule anything too demanding on the project front here. The fact that the Implementation chapter of the Dissertation is in draft will mean that you should have a very clear view of the work that remains, and so can schedule it rationally.*>

Work on the project will be kept ticking over during this period but undoubtedly the Easter Term lectures and examination revision will take priority.

Weeks 27 to 31

<*Getting back to work after the examinations and May Week calls for discipline. Setting a timetable can help stiffen your resolve!*>

Evaluation and testing. Finish off otherwise ragged parts of the code. Write the Introduction chapter and draft the Evaluation and Conclusions chapters of the Dissertation, complete the Implementation chapter.

Milestones: Examples and test cases run and results collected, Dissertation essentially complete, with large sections of it proof-read by friends and/or Supervisor.

Weeks 32 and 33

Finish Dissertation, preparing diagrams for insertion. Review whole project, check the Dissertation, and spend a final few days on whatever is in greatest need of attention.

<*In many cases, once a Dissertation is complete (but not before) it will become clear where the biggest weakness in the entire work is. In some cases this will be that some feature of the code has not been completed or debugged, in other cases it will be that more sample output is needed to show the project's capabilities on larger test cases. In yet other cases it will be that the Dissertation is not as neatly laid out or well written as would be ideal. There is much to be said for reserving a small amount of time right at the end of the project (when your skills are most developed) to put in a short but intense burst of work to try to improve matters. Doing this when the Dissertation is already complete is good: you have a clearly limited amount of time to work, and if your efforts fail you still have something to hand in! If you succeed you may be able to replace that paragraph where you apologise for not getting feature X working into a brief note observing that you can indeed do X as well as all the other things you have talked about.*>

Week 34

<*Aim to submit the dissertation at least a week before the deadline. Be ready to check whether you will be needed for a* viva voce *examination.*>

Milestone: Submission of Dissertation.