Bill J. Ellis
July 7, 2005

Improved Term Simplification

Contents

*Hums are short notes intended for distribution between those involved with EPSRC grant GR/T12289/01. Hums describe ϵ-baked ideas, where $1 \geq \epsilon \geq 0$. (Hum refers to both the Praxis Humming bird and Winnie-the-Pooh's indirect approach to writing: "Poetry and Hums aren't things which you get, they're things which get you.").

1 Introduction

As noted in Hum 10 the term simplification seen in NuSPADE is not suitable for SPADEase. This note considers the problems and aims to explain a complicated but promising solution.

2 Shortcomings in NuSPADE

Simplification in NuSPADE was separated in to two distinct heuristics, implemented within a few methods:

- **Immediately less complex -** A single rewrite is applied to the goal, aiming to make it less complex. Complexity is measured by counting the number of operators in the goal. More sophisticated measures of complexity were considered, however these were difficult to formulate and difficult to guarantee that they described a reducing measure.

- **Search for immediately less complex -** A fixed number of rewrites are applied to the goal without any further constraints. The aim is to find a combination of rewrites that eventually lead to a less complex goal. The fixed number was generated based on the complexity of the goal, measured by finding the deepest term structure. The more complex the term the deeper the search required to find promising simplifications.

Unfortunately, the cost of the unconstrained search seen in the later heuristic depends on the number of visible rules. The number of visible rules in SPADEase is much larger than for NuSPADE, leading to an unacceptable search explosion. Thus this requires amending.

3 A new approach

Search can be reduced by focusing on a goal and adjusting the legal steps based on achieving this goal. A less complex term is produced where applying a rewrite that eliminates operators. Operators can be eliminated where bringing together related terms. This goal can be used constrain the search.

3.1 Trigger rule not required

Cancelling out related terms will simplify term structures. It was thought that this alone is not sufficient to achieved the simplifications desired as terms can only be cancelled if they are present in the goal. The introduction of the terms to be cancelled may require the invocation of a particular rule first. The controlled firing of this trigger rule will complicate matters. However, this suggests adding terms to the goal so that they can be eliminated. The real concern is adding structure to the goal so that existing terms can be eliminated. A trigger rule is not necessary.

3.2 Improved measure is required?

Nevertheless, there remains a significant caveat. The intended consequence of performing elimination is to produce a less complex goal. However, the simple measure of counting operators may not revel that this is the case. Consider the classic[1] scenario:

$$\begin{array}{ll} 100 * (i+1) & A * (B+C) \Rightarrow (A*B)+(A*C) \\ (100*i)+(100*1) & 100*1 \Rightarrow 100 \\ (100*i)+100 & \end{array}$$

The number of operators in the original goal ($100 * (i+1)$) is the same as for the final goal ($(100*i)+100$). In fact, exactly the same operators ($* \wedge +$) are seen in the final goal as in the input goal and (consequently) the same number of arguments. The change is that multiplication has been pushed down and plus has been pushed up the tree structure.

3.2.1 Just ignore does not work

A solution may be to ignore the need for a check on producing less complex terms, as cancelling terms will always create a less complex term. Unfortunately, this need not be the case. The rewrite rules used to cancel the terms may in turn create additional syntax making the goal worse, even taking into account the cancellation. For example, as seen above, the rule $A*(B+C) \Rightarrow (A*B)+(A*C)$ increases the number of terms, yet still leads to cancellation. There is a need for simplification to be moving towards some fundamental less complex form.

[1] A favourite example of this particular dilemma.

3.2.2 Considering a simplification measure...

During the development of NuSPADE various different forms for the simplification measure have been considered. All of these were found to reveal subtle non-reducing features, resulting in the adoption of the basic but clear operator counting measure.

If the number of operators is the same, the new goal might yet be considered to be less complex if the depth/occurrence of multiplication (and other heavy terms) is seen to reduce. This could form a reducing measure.

3.2.3 ...Never quite works

The simplification measure becomes the key controlling feature, ensuring that the better goals are explored and ensuring that this exploration will terminate. As the heuristics desired and the terms seen become more esoteric the simplification measure will likely need extended. It is thought[2] that this centralisation of ideas into a single routine is too difficult and fundamentally flawed. It is possible to envisage a series of light weight terminating heuristics whose combination would lead to an extremely complex reducing simplification measure. Placing many heuristics in one routine is generally against the grain of component based proof planning. A single reducing measure is an attractive proposition, but rather flawed in practise.

3.2.4 Not improved measure - improved decomposition

The SPADE Simplifier does not have a universal reducing simplification measure. Instead a series of terminating phases are performed. Simplification is not a single strategy (as essentially seen in NuSPADE) but rather a number of strategies. Decomposing the problem to these individual strategies makes the task of simplification more manageable.

4 A newer approach

Inspired by the SPADE Simplifier a few simplification phases are considered. These are to be executed in serial with no backtracking. While backtracking may lead to additional proofs, it would also dilute termination properties and complicate the simplification model. Any deeper proof functionality required should be achieved by introducing a new phase.

- **Simple term elimination -** Much time was spent trying to ensure correct behaviour of an all encompassing simplification process where rules exist of the form: $A + 0 \Rightarrow A$. Such rules present problems as they are obviously good, yet have a quite different form to other good (but less obvious) rule applications. Such rules have a definite form (swapping an expression for one of its subterms) and are generally applicable directly.

- **Distributivity -** Time was also spent trying to interleave distributivity with a simplification cancellation process. No suitable controlling measure could be established[3]. Thus, instead, the pattern of distributivity can be established and all rules meeting this pattern exhaustively employed.

  This is particularly effective as distributivity is generally a good thing to perform before attempting the next step of cancellation. There is no advantage in interleaving this task - it is actually better for it to be performed upfront[4].

- **Targeted elimination -** Elimination occurs by brining together related terms. Rules are used to manipulate term structure to a point where elimination is applicable. Such rules may introduce additional terms (distributivity) or mearly manipulate the existing terms (commutativity, associativity). Restricting to only structural manipulation rules will ensure that the cancellations found are improvements. Any term level manipulation can be performed by a earlier phase. In practise the focus on only structural manipulation rules will offer a more constrained, and the generally desired, search.

Implementation details for the first two phases are fairly straight forward. The third phase is more involved and requires further consideration.

[2] After quite a lot of thought...

[3] This is probably possible, and an elegant solution may even exist. However the problems involved in finding a measure for such a simple operation tend to suggest this is a poor route to take regardless.

[4] Think: Doh!

5 Designing and coding the new approach

5.1 Identifying term pairs for cancellation

The simplification method will begin by identifying a pair of candidate terms in the goal for cancellation. The aim is to move two terms A and B in the goal G such that they become the arguments to a binary operator $A\ OP\ B$. Note that focus is placed on binary operators as they occur most often. However, any techniques considered should be naturally extendable to ternary operators and above. Valid pairs for consideration are:

- **Cancel variables -** Two instances of the same variable: $G(i, i)$. If successful this will be reduced to $G(operator(i, i))$. The cancellation of the two variables is possible if the operator is subtraction. Note that this need not be the case, thus cancellation is not always guaranteed even where successfully moving the targeted terms to the arguments of an operator.
- **Cancel numbers -** Two instances of numbers: $G(10, 20)$. If successful this will be reduced to $G(operator(10, 20))$. The cancellation of the two variables is possible by evaluating the resulting expression. Again, cancellation is not always guaranteed, as some expressions can not be evaluated, for example divide by zero.
- **Boolean evaluate -** Two instances of boolean values: $G(true, true)$. If successful A boolean expression can be evaluated to true or false.

5.1.1 Almost code

For every variable and number find:

- **Address (As a list of integers)**
- **Type of item**
- **Actual item**

Legal pairs will have the same type (and same item for variable) and have addresses that differ beyond the last element of the address list. Every pair can be found through backtracking. The address can be lexically sorted to prevent choosing the same pair twice but in a different selection order.

5.2 Marking up the goal

The targeted terms can be identified with annotations.

$$(i + i) - (i + i)$$
$$(m(i) + i) - (i + m(i))$$

Note there is no need to have different markings for the different terms, as it does not matter which term is which, only that they are to be brought together.

5.3 Avoid term attraction measure

The original plan was to apply rules to the annotated goal and ensure that the annotated terms became closer together. Formulating such a term attraction measure is difficult as the goal can take any form and the terms any position. The overall heuristic is to try and push terms to opposite ends of extreme branches so that they become adjacent. However, this involves reasoning about the effects of rule applications rather than the rules themselves. The task becomes trying to work out what the rule did then decide if this is worthwhile. For both efficiency and practical reasons it is better to select the correct rule in the first place.

Note that the focus on annotating the rules came as a consequence of not being able to find a reducing term attraction measure. Perhaps an elegant measure does exist. Nevertheless, it seems better to pre-process the rules and hard code this measure rather than apply all rules and dynamically test this measure.

5.4 Steal ideas form Rippling

The rippling style of annotated rules is copied. It is reasonable to ask if a rippling like approach is taken because rippling is familiar or because this is actually a good solution. Rippling is about manipulating term structure in a goal directed manner to eliminate syntactic differences. This simplification is also about manipulating term structure in a goal directed manner to reduce the amount of syntax. Although these similarities are at a high level they tend to suggest that an approach in the style of rippling might reasonably be applicable. Further, this style was sought after alternatives were found to be too complex. Stealing ideas from rippling seems to work.

5.4.1 Almost code

Find rules where every variable and operator on the left occurs once on the right. The rule is manipulating structure alone. For each of these rules determine its distinct variables and the address of these variables on both the left and right hand sides:

$$A + (B + C) \Rightarrow (A + B) + C$$

Variable	LHS	RHS
A	$[1]$ $([1])$	$[1, 1]$ $([1, 1])$
B	$[2, 1]$ $([1, 2])$	$[1, 2]$ $([2, 1])$
C	$[2, 2]$ $([2, 2])$	$[2]$ $([2])$

Look for any collection of terms that are together on the right hand side and not together on the left hand side. This involves looking for multiple terms that have matching address pattern up to the final element of the address list on the right hand side and the same terms having an address different by at least two numbers at the end. Such rules should be annotated.

5.4.2 Caveat

Consider the perfectly legal rule:

$$(1 + 2) + 3 \Rightarrow 1 + (2 + 3)$$

This rule will not be selected and annotated as, in the above, only variables are considered. All terms should be accepted as being brought together.

5.5 Do not build upon Prolog matching

A technique has yet to be found that implements the desired application of these rules through Prolog matching alone. The problem is that all subterms containing a single marked term (as well as the marked term itself) are deemed to represent the marked term. This reflects multiple combinations of annotations which does not naturally lend itself to a pure pattern matching based algorithm. Essentially to implement the rule application via pattern matching will require creating multiple annotations of the rule or goal.

5.5.1 Almost code

The annotations take a form to assist the processing of rule application:

$$A + (B + C) \Rightarrow (A + B) + C \qquad [y(A), y(B), n(C)]$$

- **y(VAR)** - Variable VAR must match against a subterm that contains a single marked term.
- **n(VAR)** - Variable VAR must match against a subterm that contains zero marked terms.

Thus the application of these annotated rules is to bring together marked terms. Further, this is thought to be measure reducing as it only ever moves marked terms closer together[5].

5.5.2 Caveat

Recall the perfectly legal rule:

$$(1 + 2) + 3 \Rightarrow 1 + (2 + 3)$$

Note that the annotations above do not apply as their is no variable to refer to using $y(VAR)$ or $n(VAR)$. However, the ground form can only match with another ground form. It is not possible to match a subterm containing the marked term against a ground form. Thus Prolog matching can be used to describe the application of ground parts of rules:

$$m(1) + (m(2) + 3) \Rightarrow (m(1) + m(2)) + 3 \qquad []$$

Or in mixed forms:

$$A + (m(2) + 3) \Rightarrow (A + m(2)) + 3 \qquad [y(A)]$$

$$A + (m(2) + C) \Rightarrow (A + m(2)) + C \qquad [y(A), n(C)]$$

[5]This is far from definite, but it seems to be appear to be the case.

5.6 Additional annotated rules

Unfortunately, in practise, the rewrite rules above are unlikely to offer the simplifications desired as some term manipulation may be required for the rewrite rules to become applicable. In particular, there is the rule:

$$A + B \Rightarrow B + A$$

Applications of such commutative operations may be key in manipulating the term structure to bring terms together. Note that rules other than commutativity may be applied to move a single marked term rather than bring a pair of marked terms together. Catering for these cases in a terminating manner is more involved.

5.6.1 Almost code

As for the rules above, these rules should manipulate structure only. Thus every variable and operator on the left should occur once on the right. Find the address of every term on the left and right hand side.

$$A + (B + C) \Rightarrow (A + B) + C$$

Variable	LHS	RHS
A	$[1]$ $([1])$	$[1,1]$ $([1,1])$
B	$[2,1]$ $([1,2])$	$[1,2]$ $([2,1])$
C	$[2,2]$ $([2,2])$	$[2]$ $([2])$

Number each element in the address from 1 upwards.

Variable	LHS	RHS
A	$[(1,1)]$ $([1])$	$[(1,1),(2,1)]$ $([1,1])$
B	$[(1,2),(2,1)]$ $([1,2])$	$[(1,1),(2,2)]$ $([2,1])$
C	$[(1,2),(2,2)]$ $([2,2])$	$[(1,2)]$ $([2])$

The closer to left (1) the better. Score all 1 with 1, all 2 with a worse 101, and 3 by 1001 etc.

Variable	LHS	RHS
A	$[(1,1)]$ $([1])$	$[(1,1),(2,1)]$ $([1,1])$
B	$[(1,101),(2,1)]$ $([1,2])$	$[(1,1),(2,101)]$ $([2,1])$
C	$[(1,101),(2,101)]$ $([2,2])$	$[(1,101)]$ $([2])$

The deeper the item and the more to the right the worse it is. Multiply each score by its depth.

Variable	LHS	RHS
A	$[(1)]$ $([1])$	$[(1),(2)]$ $([1,1])$
B	$[(101),(2)]$ $([1,2])$	$[(1),(202)]$ $([2,1])$
C	$[(101),(202)]$ $([2,2])$	$[(101)]$ $([2])$

Sum the scores.

Variable	LHS	RHS
A	1 $([1])$	3 $([1,1])$
B	103 $([1,2])$	203 $([2,1])$
C	203 $([2,2])$	101 $([2])$

Want to move variables to lower scores. So the rules are:

$$(A + B) + C \Rightarrow A + (B + C) \qquad [y(A), n(B), n(C)]$$

$$(A + B) + C \Rightarrow A + (B + C) \qquad [y(B), n(A), n(C)]$$
$$A + (B + C) \Rightarrow (A + B) + C \qquad [y(C), n(A), n(B)]$$

5.7 Do not constrain based on operators

As the aim is to move marked terms to a point they can be cancelled, the operators being moved towards should be potentially cancelable. Unfortunately, most operators meet this requirement, minimising the constraint this offers. Further, it may be that the key to simplification involves first moving marked terms across un-cancelable operators. Thus it is possibly best to ignore this restriction as not giving many gains and being dubiously useful.

6 Executive summary

The problems of term simplification are considered. It was found to be too complex to specify a single, powerful, simplification component. Thus instead the simplification task will be decomposed into discrete components performed in sequence. Some of these components have been considered. Deeper discussion is given to a new simplification technique based on manipulating the goal to bring together related terms, aiming to evaluate these to a simpler form.