RAL-93-026

Science and Engineering Research Council
Rutherford Appleton Laboratory
Chilton DIDCOT Oxon OX11 0QX

RAL-93-026

MERILL:
An Equational Reasoning System in Standard ML – A User Guide.

B Matthews

April 1993

MERILL :
An Equational Reasoning System in Standard ML.
A User Guide

(Prerelease version 0.4)

Brian Matthews[*]

Software Engineering Group,
Informatics Department
Rutherford Appleton Laboratory
Chilton
Didcot
OXON,
OX11 0QX.
bmm@inf.rl.ac.uk

March 25, 1993

[*]Currently at Dept of Computing Science, University of Glasgow, Glasgow, G12 8QQ, e-mail brian@dcs.glasgow.ac.uk

Contents

1 Introduction.

1.1 Motivation and History

This document is intended to be a guide to the features available in the MERILL equational reasoning system. We give an overview and then walk through of an example session. A more detailed description of all the commands and features of the MERILL system is then given. We give some pointers to the underlying theory involved and examples of the systems use.

MERILL is a general purpose order-sorted equational reasoning system. It is written in Standard ML and has been developed by the author at the S.E.R.C Rutherford Appleton Laboratory (RAL) and the Dept of Computing Science at Glasgow University. Its development has been sponsored by the IEATP project "Verification Techniques for LOTOS Specifications" between RAL, Glasgow University and Royal Holloway and Bedford New College. The development of MERILL was inspired and influenced by the ERIL (Equational Reasoning : an Interactive Laboratory) system developed by Jeremy Dick at RAL and Imperial College, London [3, 4, 5]. This system was a successful attempt to introduce order-sorted reasoning into a practical equational reasoning system. However, the design of ERIL became obsolete: it was considered too slow, and lacked significant features. It was decided to implement an entirely new system in a different language which would retain the major features of ERIL whilst being significantly faster and having new features. An important extension was to be the integration of order-sorted reasoning with reasoning modulo associative-commutative operators. Standard ML was chosen due to the ease of use associated with functional programming languages, coupled with the increasing availability of efficient implementation. The use of the module system also makes the system easy to modify to allow new extensions and experiments in equational reasoning techniques to be incorporated in the system.

A major part of the philosophy of MERILL is that the user is in control. Thus the system has few built in assumptions and does very little reasoning in the background. This also means that the user has to enter the object syntax themselves as the system has no built in assumptions about the form of for example variables. The user is given a great deal of freedom in the choice of object language. A general mixfix syntax is available for operators and the system will try to parse terms as unambiguously as possible.

Significant features of the MERILL system are:

- Equational reasoning over an Order-Sorted Signature.
- User defined mixfixed syntax for terms.
- Associative-Commutative operators.
- Equality Sets for data separation.
- Variety of term orderings.
- Several completion algorithms.
- Menu based interface.

The major feature missing in this release of MERILL is a tactic language. The ERIL system had a system of user definable "links" between equality sets which could be configured in many ways to produce a variety of different strategies for completion and other equational reasoning algorithms. This proved flexible as is demonstrated in [5, 21, 22]. However, there did prove to be major weaknesses in this method and work is currently underway to how best to generalise the link method into a tactic language. This will be incorporated into a future release of MERILL .

1.2 Order-Sorted Algebras

This user guide is not a suitable place to give a full description of the theory of order-sorted algebras and their associated rewriting theory. We refer the reader to [13, 10, 1, 11, 27, 18, 12, 9, 29]. However, there are two approaches to the semantics of order-sorted signatures given in the literature which differ slightly: the overloaded approach as explored in [12] and the non-overloaded given in [10, 27]. See [29] for a detailed comparison of the approaches. In the MERILL system, we follow the non-overloaded approach as we consider it more suitable for specification where it is not useful to confuse two semantically different functions by the same operator.

The MERILL system is also seen as an experimental work bench to experiment with new methods and semantics for order-sorted algebras. To overcome the problem of completion not allowing non-sort-decreasing rules to be used, a new theory of Dynamic Order-sorted rewriting is being developed [30, 24, 25] which allows the sorts of terms to be modified during the reasoning process. Experimental implementation work is being carried out and a full implementation will be incorporated into a future version of the MERILL system.

1.3 Other Term Rewriting Systems

ERIL is not the only current equational reasoning system available or indeed to have an influence on the development of MERILL . Others which should be mentioned are RRL [17], the Larch Prover (LP) [7], OBJ3 [14], ORME [20] and Elios-OBJ [8]. These all differ to a greater or lesser extent from MERILL .

RRL and LP are perhaps the most established equational reasoning systems. Powerful and well developed, they have been worked on for several years and have a wealth of diverse tools and practical experience associated with them. They differ from MERILL in that they are do not have the extra expressive power which derives from the order-sorted paradigm. Also they are essentially command driven on a current state of rules. They do not have the added flexibility which comes from separating the equation base into separate equality sets and using each equality set for a different purpose.

More similar to MERILL is the OBJ family of systems, of which the latest version is OBJ3. This is a system which has many features in common with MERILL such as order-sorted algebra, associative-commutative operators, and mixfix syntax. It also takes the idea of equality set to greater extreme with a module system, which can be parameterised. However, the OBJ3 system is seen as a very high-level programming language which uses term rewriting as the computational mechanism. It does not have unification, term orderings and the other basic tools associated with completion algorithms. A more recent adaptation of OBJ3 is Elios-OBJ which is a modification of OBJ3 to allow completion. This is an interesting approach which implements the completion algorithm in the OBJ language

itself in a structured style as exhibited in ORME (see below). However, this system seems to lack the generality and adaptability of MERILL .

The ORME system is a tool-box of routines for basic equational reasoning actions such as rewriting, unification, ordering and so on, written in the CAML language, together with some control routines which apply these basic rules in a variety of strategies for Knuth-Bendix Completion, Completion modulo Equational Theories, and Unfailing Completion. This system realises Lescanne's concept of completion being "Transition rule + Control" and the user can use the provided structural "glue" to build new routines for him or herself. The problem with this is that the user needs to learn the CAML language as the user explicitly writes programs in that language. A higher level tactic language would perhaps be a more friendly way of producing the same effect.

2 Overview.

This section gives an brief overview of the general features of MERILL , sketching the general features of the system. A reduced version of this chapter appears in [23]. Many points are repeated in the subsequent walk through section designed as a tutorial introduction to the system.

The overall organisation of MERILL is given in figure 1. The central database of the system is divided into three major components: the signature, the equality sets and the environment. These mutually dependent data sets combine to provide the raw material for the computational tools, and provide the storage for their results. Computational tools include rewriting, unification and completion.

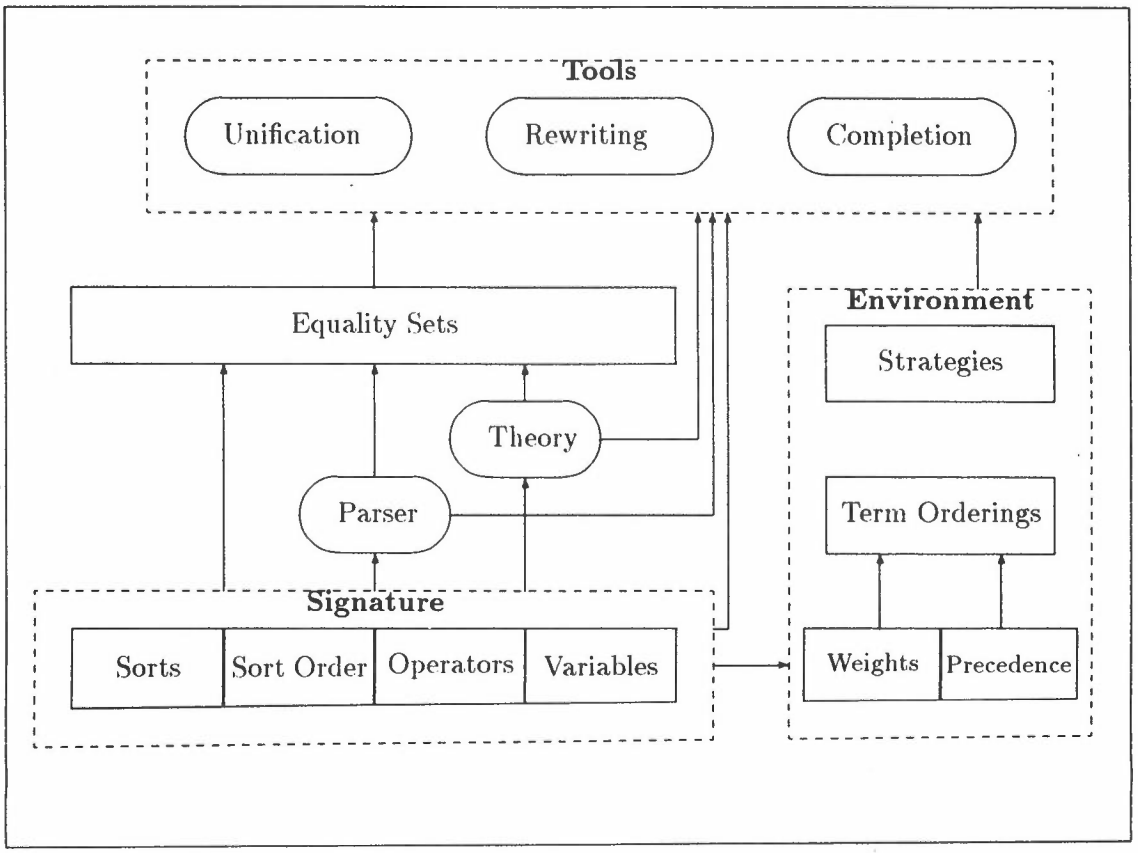


Figure 1: The Logical Organisation of MERILL .

We shall explore the features of MERILL by means of the example specification in figure 2, of basic arithmetic over the natural numbers.

This example shows the distinctive features of order-sorted equational logic as implemented in MERILL : a hierarchy of sorts; subsorts used to make partial functions total; multiple ranks for functions giving its domains for different argument sorts; the use of sorted variables define the behaviour of functions restricted to certain input domains. Further, the addition and multiplication functions are both declared to be associative and commutative and thus require special handling.

Sorts:	$Bool$, nat, $zero$, $posnat$	
Subsorts:	$zero < nat$, $posnat < nat$	
Operators:	$ff :> Bool$	$tt :> Bool$
	$0 :> zero$	$succ : nat > posnat$
	$\_ > \_ : nat\ nat > Bool$	$pred : posnat > nat$
	$\_ + \_ : nat\ nat > nat$	$\_ * \_ : nat\ nat > nat$
	$\_ + \_ : zero\ zero > zero$	$\_ * \_ : posnat\ posnat > posnat$
	$\_ + \_ : nat\ posnat > posnat$	$\_ * \_ : zero\ nat > zero$
	$\_ + \_ : posnat\ nat > posnat$	$\_ * \_ : nat\ zero > zero$
Variables	$n, n1, n2 : nat$, $p : posnat$	
Equations:	$pred(succ(n)) = n$	$n1 + n2 = n2 + n1$
	$n + 0 = n$	$(n + n1) + n2 = n + (n1 + n2)$
	$n + (succ(n1)) = succ(n + n1)$	$n1 * n2 = n2 * n1$
	$n * 0 = 0$	$(n * n1) * n2 = n * (n1 * n2)$
	$(succ(n)) * n1 = (n * n1) + n1$	
	$n * (n1 + n2) = (n * n1) + (n * n2)$	
	$0 > n = ff$	$p > 0 = tt$
	$succ(n) > succ(n1) = n > n1$	

Figure 2: An Example Signature.

2.1 The Signature.

The signature is a database which defines the object language the user wishes to use. This comprises of four components: the sorts; a sort ordering over the declared sorts; operators together with their ranks; and sorted variable forms. Thus the user would move to the sort menu, add the above sorts, then move to the sort-ordering menu to declare the ordering.

Figure 3 gives an example screen from the MERILL system showing operators being added, This figure shows a typical example of the style of interface to the system. The top of the screen shows the operators already added to the system, together with their ranks. In the middle of the screen is a menu, from which the user selects an item, in this case "a" for adding new operators. Each rank is added individually; here we declare the greater-than operator, and two ranks of the multiplication operator. The addition function is annotated "`(ASSOC COMM)`" to denote that the AC equational theory has been declared for this operator.

Classes of variable names are similarly declared with their sort within the variables menu. Using the operator and variable declarations, the system generates a mixfix parser which is then used by the rest of the system to parse terms. In addition to declaring operators, the system allows the user to declare that certain operators are either commutative or associative-commutative. The system then generates a representation of this equational theory for use in completion.

2.2 Equality Sets.

The user can declare equalities which are associated together in equality sets. This allows separation of data into various sets which the user can handle separately. Equalities can

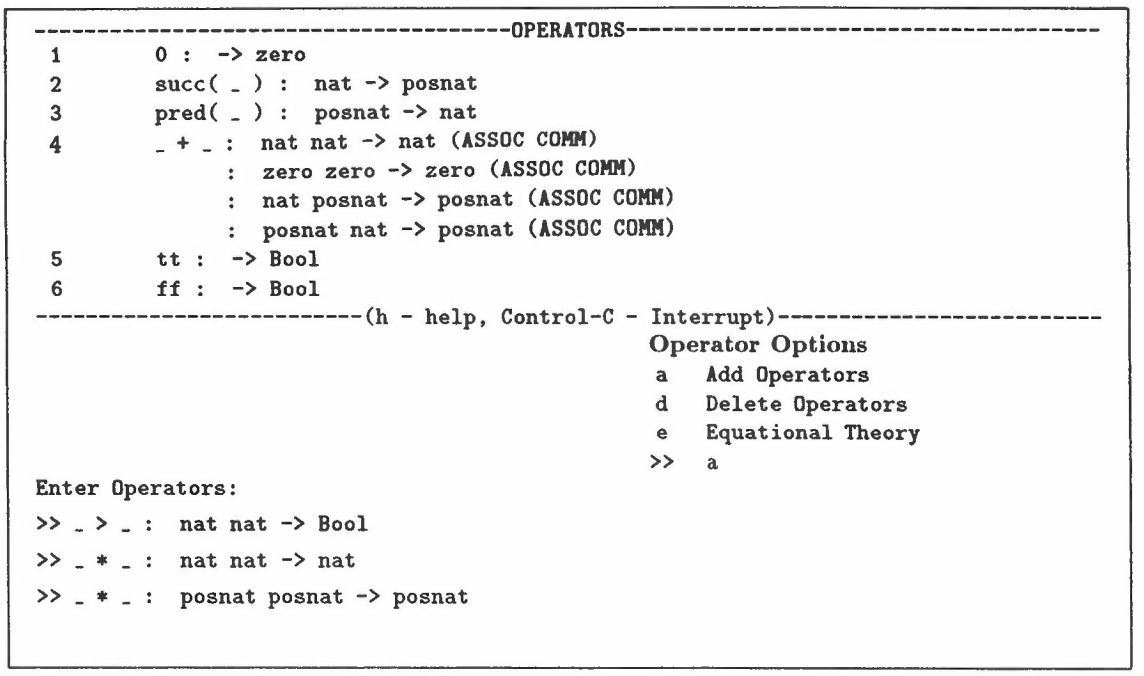


Figure 3: Adding operators to MERILL .

be of 4 varieties: equations, that is unoriented axioms; rewrite rules used for rewriting; conjectures, which are passive and used as a query, declared true when rewritten to identity; and conditional, which we can use for rewriting, but not for completion.

The advantage of using equation in sets is that the interaction between them can be strictly controlled. For example, an equality can be rewritten by one set of rules but not another, or critical pairs can be generated between two sets of rules and the resulting equalities placed in another. This allows the user to control the reasoning process more tightly than keeping all rules together.

2.3 Environment.

The environment allows the selection of term orderings and completion strategies which are independent of the signature used. Currently, the Recursive Path ordering, and Knuth-Bendix ordering are available, as well as an Associative Knuth-Bendix ordering suitable for use with AC operators. To use these orderings the user must set up appropriate precedences and weights on operators, also within the environment.

2.4 Tools in MERILL .

A series of tools are available which use this database of signature, equality sets and environment. Upon the signature alone, there are tests for regularity, monotonicity and inhabitedness. Note that these properties are not enforced by MERILL , but the signature can be checked for them.

Tools available for operating on terms and equalities include explicit unification of terms; rewriting of both terms and equations; the generation of critical pairs, and three order-sorted completion algorithms. The first is an ordinary completion algorithm which does not take into account AC-operators; the second takes into account AC-operators, but only

in when all rewrite rules are left linear [15]; and the third is a full AC-completion algorithm which generates the associative extensions to rules [26]. The second is faster than the third, but can only be used in restricted cases. The user can choose which equality sets to use with the completion algorithm.

In the example we wish to complete the given set of rules using the full AC-completion The organisation of the algorithm is given in figure 4. Boxes represent equality sets, solid arrows movement of equalities during completion, and dashed arrows rewriting of equalities in one set by another set (subsumption if equations are used rather than rules). The algorithm is data driven, in a similar style to ERIL and also to ORME [20]. We select three sets from the equality sets database, which we shall call A, T and R. A contains the unoriented equations; T is a set of temporary rewrite rules, and R is the final target set to hold the rewrite rules. In addition, another set H can be used which holds conjectures to be considered in parallel with the completion process.

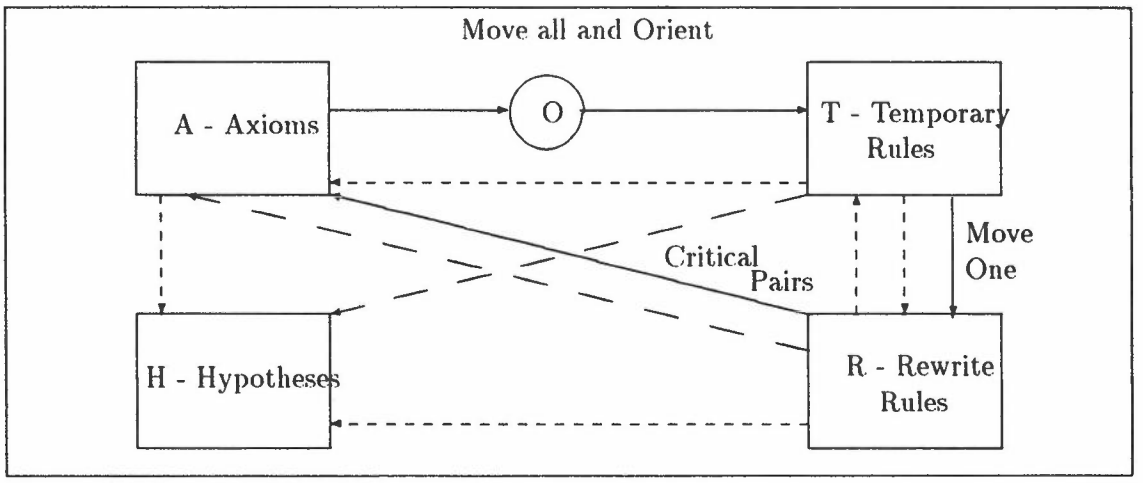


Figure 4: Running AC-Completion.

The strategy described in this diagram is to move all axioms from A, orienting them through the term ordering O and placing them in T. The ordering also checks whether the equation is sort-decreasing. All equalities in the four sets are then rewritten by T and R, (rewriting by T and R on themselves omitted in the diagram for clarity). Then, using the desired selection strategy from the environment, a single rule is moved from T to R, generating all critical pairs between that rule and existing members of R, including itself. Non-trivial critical pairs are then placed in A, and the cycle begins again. Termination occurs when the sets A and T are empty; the set R will then contain a confluent set.

To run this we first need to set up a term ordering. Only one of the implemented orderings accepts AC-operators; the Associative Knuth-Bendix ordering. To use this both weights and precedence must be declared in the environment. However, the rule set given is not suitable for this ordering and we shall use a manual ordering.

Figure 5 gives the results of running this strategy upon the given example from an actual MERILL session. Note that in this case no new non-trivial pairs were created. However, rule 10 is new, and is marked with an asterisk. This is because this is an extended rule, a necessary addition to ensure confluence of AC-rewriting. This rule is generated by superposing rules on the associative laws of the equational theory generated by the annotations on operators.

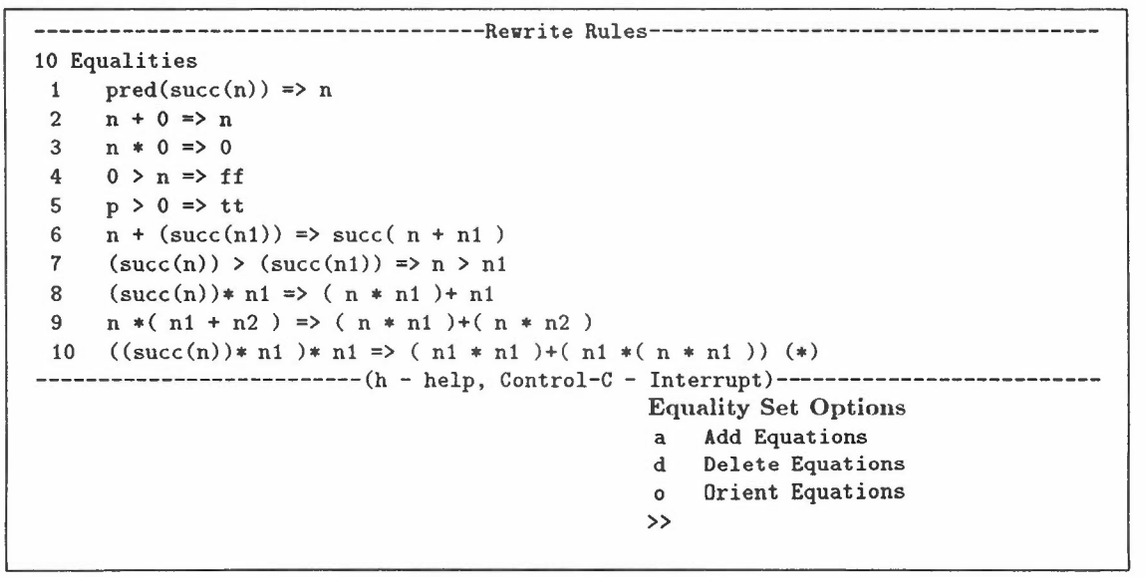


Figure 5: Complete Set of Rewrite Rules for Arithmetic on Naturals.

3 A Walkthrough of the MERILL system

This chapter gives an example of a complete MERILL session, designed as a tutorial introduction to the system. We give a brief explanation of what is presented to the user and what actions are required in order to perform simple equational reasoning tasks. The example we give in detail is a standard example in term-rewriting; the completion of the free group. We then give a briefer indication on how to use the associative-commutative facilities of MERILL .

3.1 Top Level Interaction.

When the MERILL system is invoked the user is first given the top level menu. This menu allows the user to enter the major categories of the MERILL system.

```
--------------------------------------------------------------------------------
                              MERILL v.0.4
                  An Equational Reasoning System in Standard ML.
                              Brian Matthews
                  Glasgow University and Rutherford Appleton Laboratory 1993

--------------------------------------------------------------------------------
TOP LEVEL OPTIONS

a   Signature                              s   Save
n   Environment                           l   Load
e   Equalities                            c   Clear System
S   Statistics                            f   Finish
h   Help
--------------------------------------------------------------------------------
Select Option:  >> a
```

Interaction with the system is by selecting the (usually single character) keys for each item in the menu, which will then move the control to another menu, or to some other explicit action. For example, selecting "l" at the top level will move to the "load" menu which allows the user to load predefined scripts into the MERILL system. The user can "type ahead" by entering a series of commands (separated by spaces) and the system will not display the intervening menus.

3.2 Entering the Signature

MERILL has no built-in default language definitions and so the user must start by defining the syntactic object language which he or she intends to use. This can be done by loading a file, but at this stage we have none, so we must declare one by hand. This is done by moving to the signature menu. We select "a" at the top level to move to that menu.

```
---------------------------------Signature Options---------------------------------
SIGNATURE OPTIONS

s   Sort Options                                  i   Inhabitedness test
o   Sort Ordering Options                         m   Montonicity test
p   Operator Options                              r   Regularity test
v   Variable Options                             f   Finish
h   Help
-----------------------------------------------------------------------------------

Select Option:  >> s
```

On the left of the menu are various categories of symbols to be declared. On the right are various tests which can be run on the signature.

The first items which must be declared are the sorts of the object language. We select "s" and move to the sort menu. As we are entering new sorts we then type "a" and return.

```
---------------------------------------SORTS---------------------------------------
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                Sort Options
                                                 a   Add Sorts
                                                 d   Delete Sorts
                                                >>   a
Enter Sorts:
>> s
>>
```

We need only enter one sort needed to represent the group. A return on a blank line finishes the entering of sorts, and the sorts entered are displayed at the top of the screen.

```
---------------------------------------SORTS---------------------------------------
s
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                Sort Options
                                                 a   Add Sorts
                                                 d   Delete Sorts
                                                >>
```

Hitting a newline again will return you to the signature menu, where entering "p" will move you the menu for the next category of symbols which must be entered: the operators[1]. Again we enter "a" to add operators.

Three operators are needed for the group example, a unit constant "0", a prefix minus operator "- _ ", and an infix binary operator "_ + _". MERILL can handle general mixfix

[1] we use "p" for oPerators as the more obvious "o" is used for the sort-ordering option, which we don't need in this example

```
-------------------------------------OPERATORS-------------------------------------
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                  Operator Options
                                                   a   Add Operators
                                                   d   Delete Operators
                                                   e   Equational Theory
                                                  >>   a
Enter Operators:
>> 0 :  -> s
>> -_ :  s -> s
>> _ + _ :  s s -> s
>>
```

operators and uses the underscore character to represent the positions in the operator where arguments can appear. Note also that in addition to the operators, a rank is entered giving the signature of the operator in terms of previously entered sorts. The constant "0" has a nullary rank.

Entering a blank line will terminate the enter option and display all the current operators and their ranks together with a numbered key.

```
-------------------------------------OPERATORS-------------------------------------
 1       0 :  -> s
 2       - _ :  s -> s
 3       _ + _ :  s s -> s
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                  Operator Options
                                                   a   Add Operators
                                                   d   Delete Operators
                                                   e   Equational Theory
                                                  >>
```

Finally in the signature options we must declare the names and sorts of variables. Note that MERILL has no predefined conventions for variables and the user must declared them. Classes of variable names can be declared by using variable prefixes.

Returning to the signature menu, we choose the "v" option which takes us to the "variables" menu, and selecting "a" allows us to declared variable symbols.

Two kinds of variable names can be entered. Either complete variable names can be declared, or prefixes which can be extended in any way to form a variable. A variable prefix, declared by a string of any length followed by a "*", allows the system to recognise any string starting with this prefix as a variable of the declared sort[2]. The sort of a variable is also entered here, separated by a colon. The sort must have been previously declared in the sort options.

In this example, we enter three classes of variable: those starting with "x" or "y" or "z".

[2] unless the whole string is recognised as another operator symbol or whole variable

```
-------------------------------------VARIABLES-------------------------------------
 Declared Variables                                  Variable Prefixes
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                     Variable Options
                                                      a   Add Variables
                                                      d   Delete Variables
                                                     >>   a
Enter Variables:
>> x* :  s
>> y* :  s
>> z* :  s
>>
```

Entering a blank line displays the current variable declarations.

```
-------------------------------------VARIABLES-------------------------------------
 Declared Variables                                  Variable Prefixes
                                                     x* :  s
                                                     y* :  s
                                                     z* :  s
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                     Variable Options
                                                      a   Add Variables
                                                      d   Delete Variables
                                                     >>
```

We have now declared all the symbols that we need for the group example and can leave the signature menu by typing "`f`" on the "signature" menu which returns us to the top level.

3.3 Entering Equality Sets

Equalities within MERILL are stored in collections known as *Equality Sets*. These entities collect equations together for using as a unit. For example, when rewriting is performed, it is carried out using rules in a specified equality set and no others can interfere in the process. This separation of equations into separately manipulable sets is an important aspect of the control the user has over the reasoning carried out by MERILL .

To enter the axioms for Group Theory we need to go into the Equality Set menu. To move to this menu select "`e`" from the top level menu.

This menu gives information on the current equality sets at the top of the screen. Each equality set has a *label*, which is a string, usually a single letter, which gives a shorthand key to which to refer to the equality set, and a *title* which is a usually longer string giving some description of the set meaningful to the user. The numbers displayed on the right of each equality set label show the current number of equalities in the set together with

```
-----------------------------------Equality Sets-----------------------------------
 Label       Title              Current  Total
   A         Axioms                   0      0
   R      Rewrite Rules               0      0
-----------------------------------------------------------------------------------
 Equality Options
 g   Goto Equality Set                          n   Normalise a Term
 a   Add a New Equality Set                     r   Normalise an Equation
 d   Delete an Equality Set                     p   Critical Pairs
 m   Move Equations                             k   Knuth-Bendix Completion
 c   Copy Equations                            H   Huet's Left-Linear Completion
 P   Peterson & Stickel's AC-Completion         u   Unify Terms
 f   Finish                                    h   Help
-----------------------------------------------------------------------------------
Select Option:  >> g
Select Equality Set by Label.
Label>> A
```

the total number of equalities which have passed through the set. Beneath the display is a variety of actions which we shall come to as and when we need them. Initially, there are two default equality sets, `A` which will contain basic axioms and `R` which as we will see be used to hold rewrite rules.

To enter the group axioms we move into the `axioms` set by first selecting `g` followed by the label of the required set, `A`. This moves us to a screen displaying the selected equality set and actions which can be applied directly to that set.

```
------------------------------------Axioms-------------------------------------
0 Equalities
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                  Equality Set Options
                                                   a   Add Equations
                                                   d   Delete Equations
                                                   o   Orient Equations
                                                  >>   a
Enter >> 0 + x = x
Enter >> (- x) + x = 0
Enter >> ( x + y )+ z = x +( y + z )
Enter >>
```

We type a to add new equations and then enter the three axioms which define group theory. These are entered in a straightforward manner using the operators and variables declared earlier. Note that when the operators were declared, the system constructs a parser for the mixfixed operators. This parser is deterministic, and will resolve ambiguities in a unique way, not always being that expected by the user! Consequently, it is safe to add extra brackets when entering equalities. Thus the extra brackets in the first term of the second equation `(- x) + x` are essential to disambiguate from `- (x + x)` .

This results in:

```
------------------------------------Axioms-------------------------------------
3 Equalities
 1   0 + x = x
 2   (- x) + x = 0
 3   ( x + y )+ z = x +( y + z )
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                  Equality Set Options
                                                   a   Add Equations
                                                   d   Delete Equations
                                                   o   Orient Equations
                                                  >>
```

3.4 The Environment

Before going on to perform completion on the group axioms, we must first set up the term ordering which will be used to orient equations and hence ensure that rewriting will be terminating. The orderings options are controlled from the environment menu. This is accessed by returning to the top level menu and choosing the "n" option.

```
--------------------------------Current Environment--------------------------------
  Global Ordering    ..............  none
  Local Ordering     ..............  manual
Selection Strategy   ..............  by_size
  Display Level      ..............  1 Sorts :  OFF
  Help Directory     ..............  /users/peril/brian/LOTOS/src/help
    Help File        ..............  help.tex
-------------------------------------------------------------------------------
 Equality Options
 o   Global Orderings                          l   Local Orderings
 p   Precedence                                w   Weights
 s   Strategy                                 hd   Change Help Directory
 d   Display Level                            hf   Change Help File
 f   Finish                                    h   Help
-------------------------------------------------------------------------------
Select Option:  >> d
Set Display Level :
0 Full
1 Partial
2 Silent
s Toggle Sort display.
Enter Level Number, or toggle Sort display >> 0
```

The environment provides a miscellany of optional settings, the current status of which are displayed. For example, the display level determines how much information the system will display during completion, from silent, which displays very little, to full, which displays a good deal. We will run this example on full to show what is shown. This is altered by selecting "d" on the environment menu, followed by the appropriate level number.

The main purpose for entering the environment menu at this stage is to set the global term ordering. The default global ordering is set to "none" which means that no global term ordering is currently been set. To set the ordering, we select "o" and then subsequently choose an ordering from the menu.

```
AVAILABLE GLOBAL ORDERINGS
 1      User RPO - Left Status
 2      User RPO - Right Status
 3      User RPO - Multiset Status
 4      RPO - Left Status
 5      User KBO
 6      User AC-KBO
 none   none
Pick Number of Ordering (or "none") >> 1
```

In this example we choose to set the global to ordering to a Recursive Path Ordering with a left to right lexicographic status on all operators. This is described as "**User**" as the user has to set the precedences on operators in advance.

In order to set the precedence we choose the "p" option on the environment menu, which takes us to a menu for precedences. Note that this precedence on operators is used in term orderings alone and should not be confused with a precedence on operators which is used for parsing concrete syntax.

```
-------------------------------------PRECEDENCE-------------------------------------
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                 Precedence Options
                                                  a   Add Precedence Declarations
                                                  d   Delete Precedence Declarations
                                                 >>   a
Enter (f < g)
>> 0 < _ + _
>> _ + _ < - _
>>
```

Precedences are declared by typing operators in pairs separated by a "<". Note that the complete form of the operator must be entered, including all underscores. This includes the "prefix and brackets" default form for operators, so an operator "f" of arity 2 would have to be entered as "f(_,_)" [3]. Thus for the group example we set the precedences as shown above with the zero operator at the bottom and the inverse at the top of the precedence. A blank line again finishes the entering of the precedences and the current precedences are displayed at the top of the screen.

[3]This method for entering the precedence of terms is considered to be unsatisfactory and will be changed in future versions of MERILL

```
-----------------------------------PRECEDENCE-----------------------------------
_ + _ < - _
0 < _ + _
-------------------------(h - help, Control-C - Interrupt)-----------------------
                                                  Precedence Options
                                                   a   Add Precedence Declarations
                                                   d   Delete Precedence Declarations
                                                  >>
```

Returning to the Environment menu, we mention some of the other options available there which are relevant to the completion of the group.

The local ordering is a default strategy for ordering equations when the global ordering can order them either way. There are several of these local orderings available. The initial strategy is to appeal to the user and this will suffice for our example.

The selection strategy determines which order equations are chosen from a set of equations for consideration for converting into rewrite rules. These strategies should be fair. The initial one is "`by_size`" which picks the equation which has the "smallest" size by counting the number of operators which occurs in the terms of the equation. Again this is suitable for our example.

This menu also gives the option of displaying the sort of every term. As the sorts of all terms are the same in this example, this is irrelevant to this example and we leave the toggle off.

3.5 Completion.

We are now in the position to be able to perform Knuth-Bendix Completion on the group axioms. To do this, we return to the equality set menu (by selecting "`f`" once if we are at the environment menu followed by "`e`" at the top level menu).

```
-----------------------------------Equality Sets--------------------------------
 Label       Title              Current  Total
   A         Axioms                   3      3
   R     Rewrite Rules                0      0
--------------------------------------------------------------------------------
 Equality Options
 g   Goto Equality Set                          n   Normalise a Term
 a   Add a New Equality Set                     r   Normalise an Equation
 d   Delete an Equality Set                     p   Critical Pairs
 m   Move Equations                             k   Knuth-Bendix Completion
 c   Copy Equations                            H   Huet's Left-Linear Completion
 P   Peterson & Stickel's AC-Completion         u   Unify Terms
 f   Finish                                    h   Help
--------------------------------------------------------------------------------
Select Option:  >> k
```

By selecting "`k`" at this menu we enter the Knuth-Bendix completion routine. This requires

two equality sets with which to interact: a set from which to take the initial axioms and to place the generated critical pairs and a set into which to place newly oriented rewrite rules and by which to rewrite the equations. The first thing the Knuth-Bendix option does is enter into a brief dialogue.

```
-----------------------------Knuth Bendix Completion-----------------------------
Enter Set of Equations >> A
Enter Set of Rules >> R
By Steps (y/n) ?  y
Test Conjectures (y/n) ?  n
```

The first thing which we are asked for is a set of equations. We give it the set "A" which contains the group axioms and the currently empty set "R" to hold the rewrite rules as they are created. We are then asked whether the we would like to proceed by steps. This means that the process will halt, display the current results and ask if we wish to proceed after orienting every rule. As we wish to show the results in this demonstration, we respond with a "y". If we responded negatively, the completion would have ran without halting until it successfully completed, or failed. Finally, it asks whether we want to test conjectures. The user can set conjectures which are equalities he or she would like to prove as completion proceeds; that is using the completion as a partial decision procedure. As there are no conjectures to be considered in this example, we respond with a negative.

The completion then proceeds by selecting an axiom from the set "A", according to the selection strategy, and orienting it by the current global ordering.

```
3 Equations, 0 Rules
Scanning 0 + x = x
Rule No:1 Ordered as 0 + x => x

Normalising Rules
Generating Critical Pairs
Critical Pair :  (new/R1):  x = x
Reduced To :  x = x
Delete :  x = x
-------------------------------------Axioms-------------------------------------
2 Axioms
 2   (- x )+ x = 0
 3   ( x + y )+ z = x +( y + z )
-----------------------------------Rewrite Rules----------------------------------
1 Rewrite Rules
 1   0 + x => x
------------------------------------------------------------------------------

Do you wish to finish (y/n) ?
```

The newly oriented rule is placed in the set "R" and then used to rewrite existing rules and equations, which has no effect at this stage. It goes on to generate new critical pairs with rules in "R", and in this case generates one rule with itself, which is reduced and seen to be trivial and so is automatically deleted. The system halts with the system as shown above. By typing any key except "n", including return, we ask the system to proceed, and it selects the next rule for orientation.

```
2 Equations, 1 Rules
Scanning (- x )+ x = 0
Rule No:2 Ordered as (- x )+ x => 0

Normalising Rules
Generating Critical Pairs
Critical Pair :  (new/R2):  0 = 0
Reduced To :  0 = 0
Delete :  0 = 0
-------------------------------------Axioms-------------------------------------
1 Axioms
 3   ( x + y )+ z = x +( y + z )
-----------------------------------Rewrite Rules----------------------------------
2 Rewrite Rules
 1   0 + x => x
 2   (- x )+ x => 0
------------------------------------------------------------------------------
Do you wish to finish (y/n) ?
1 Equations, 2 Rules
Scanning ( x + y )+ z = x +( y + z )
Rule No:3 Ordered as ( x + y )+ z => x +( y + z )

Normalising Rules
Generating Critical Pairs
Critical Pair :  (new/R1):  0 +( y + z ) = y + z
Reduced To :  y + z = y + z
Delete :  y + z = y + z
Critical Pair :  (new/R2):  (- y )+( y + z ) = 0 + z
Reduced To :  (- y )+( y + z ) = z
Add Equation:  (- y )+( y + z ) = z
Critical Pair :  (new/R3):  x +( y + z ) = x +( y + z )
Reduced To :  x +( y + z ) = x +( y + z )
Delete :  x +( y + z ) = x +( y + z )
Critical Pair :  (new/R3):  ( x + y )+( y1 + z ) = ( x +( y + y1 ))+ z
Reduced To :  x +( y +( y1 + z )) = x +( y +( y1 + z ))
Delete :  x +( y +( y1 + z )) = x +( y +( y1 + z ))
-------------------------------------Axioms-------------------------------------
1 Axioms
 4   (- y )+( y + z ) = z
-----------------------------------Rewrite Rules----------------------------------
3 Rewrite Rules
 1   0 + x => x
 2   (- x )+ x => 0
 3   ( x + y )+ z => x +( y + z )
------------------------------------------------------------------------------
Do you wish to finish (y/n) ?
```

Orienting the second rule again does not do anything very interesting. However, the third, associativity axiom generates four critical pairs, one of which is non-trivial `(- y )+( y + z ) = z` and this is added to the equation set. This equation is then selected and the process continues. As the total amount of information is great at this stage, we omit most of the run through.

```
1 Equations, 3 Rules
Scanning (- y )+( y + z ) = z
Rule No:4 Ordered as (- y )+( y + z ) => z
......
```

Note also, that rules added to the rewrite rule set can be taken away. At a later stage in the completion we arrive at a point as shown in the next diagram. On orienting the new rule, it attempts to rewrite the rules. Two rules, numbers 5 and 7 are rewritable and as they rewrite to trivial equations are deleted. One of the other equations, number 13 also rewrites to a trivial equation and is deleted.

```
......
-------------------------------------Axioms-------------------------------------
5 Axioms
 12   - 0 = 0
 10   0 = (-(-(- z )))+ z
 13   z = (-(-(- 0 )))+ z
 7    y + z = (-(- y ))+ z
 8    z = (-( x + y ))+( x +( y + z ))
-----------------------------------Rewrite Rules--------------------------------
7 Rewrite Rules
 1   0 + x => x
 2   (- x )+ x => 0
 5   (- 0 )+ z => z
 6   (-(- z ))+ 0 => z
 7   (-(- 0 ))+ z => z
 4   (- y )+( y + z ) => z
 3   ( x + y )+ z => x +( y + z )
------------------------------------------------------------------------------
Do you wish to finish (y/n) ?
5 Equations, 7 Rules
Scanning - 0 = 0
Rule No:8 Ordered as - 0 => 0

Normalising Rules
Delete Rule:  0 + z = z
Delete Rule:  0 + z = z
Equation z = (-(-(- 0 )))+ z
Rewritten to z = z
Generating Critical Pairs
......
-------------------------------------Axioms-------------------------------------
3 Axioms
 10   0 = (-(-(- z )))+ z
 7    y + z = (-(- y ))+ z
 8    z = (-( x + y ))+( x +( y + z ))
-----------------------------------Rewrite Rules--------------------------------
6 Rewrite Rules
 8   - 0 => 0
 1   0 + x => x
 2   (- x )+ x => 0
 6   (-(- z ))+ 0 => z
 4   (- y )+( y + z ) => z
 3   ( x + y )+ z => x +( y + z )
------------------------------------------------------------------------------
Do you wish to finish (y/n) ?
```

Thus we continue the completion process until we get to the axiom which distributes the inverse operator over the binary operator.

```
11 Equations, 11 Rules
Scanning -( y + y1 ) = (- y1 )+(- y )
Rule No:17 Ordered as -( y + y1 ) => (- y1 )+(- y )

Normalising Rules
Delete Rule:  ((- y )+(- y1 ))+ y1 = - y
Delete Rule:  y +((- y )+(- y1 )) = - y1
Equation -( y +(- z )) = z +(- y )
Rewritten to z +(- y ) = z +(- y )
Equation -((- y )+ z ) = (- z )+ y
Rewritten to (- z )+ y = (- z )+ y
Equation z = x +( y +((-( x + y ))+ z ))
Rewritten to z = z
Equation z = (-( x + y ))+( x +( y + z ))
Rewritten to z = z
Equation - y = x +( y1 +(-( y +( x + y1 ))))
Rewritten to - y = - y
Equation -( x + y ) = y1 +(-( x +( y + y1 )))
Rewritten to (- y )+(- x ) = (- y )+(- x )
Equation x +((-( y + x ))+ z ) = (- y )+ z
Rewritten to (- y )+ z = (- y )+ z
Equation - y = (-( x +( y1 + y )))+( x + y1 )
Rewritten to - y = - y
Equation (-( y + y1 ))+( y + z ) = (- y1 )+ z
Rewritten to (- y )+ z = (- y )+ z
Equation 0 = x +( y +( y1 +(-( x +( y + y1 )))))
Rewritten to 0 = 0
Generating Critical Pairs
Critical Pair :  (new/R1):  (- y )+(- 0 ) = - y
Reduced To :  - y = - y
Delete :  - y = - y
......
Critical Pair :  (new/R3):  (- y )+(-( x + y1 )) = -( x +( y1 + y ))
Reduced To :  (- y )+((- y1 )+(- x )) = (- y )+((- y1 )+(- x ))
Delete :  (- y )+((- y1 )+(- x )) = (- y )+((- y1 )+(- x ))
-------------------------------------Axioms-------------------------------------
0 Axioms
-----------------------------------Rewrite Rules----------------------------------
10 Rewrite Rules
 8    - 0 => 0
 1    0 + x => x
 10   -(- z ) => z
 11   z + 0 => z
 2    (- x )+ x => 0
 12   z +(- z ) => 0
 4    (- y )+( y + z ) => z
 13   z +((- z )+ z1 ) => z1
 17   -( y + y1 ) => (- y1 )+(- y )
 3    ( x + y )+ z => x +( y + z )
------------------------------------------------------------------------------------
Do you wish to finish (y/n) ?
```

At this point there are no more equations to consider. The system recognises this and tells us the system is complete with 10 rewrite rules. It also gives us some statistics on how the completion algorithm has arrived at this complete set.

```
Complete.
Statistics:
Attempted Matches :  3324
Successful Matches :  226
Attempted Unifies :  556
Successful Unifies :  125
Critical Pairs :  88

Confluent Set of 10 Rewrite Rules

Execution Time:  3.640000
Garbage Collecting Time:  0.010000
Total Time:  3.650000
Press Enter/Return to continue.  >>
```

We thus return to the equality set menu and find that the complete set of rules has been placed in the set "R".

```
-----------------------------------Equality Sets-----------------------------------
 Label        Title              Current  Total
   A          Axioms                   0     37
   R      Rewrite Rules               10     17
-----------------------------------------------------------------------------------
 Equality Options
 g   Goto Equality Set                        n   Normalise a Term
 a   Add a New Equality Set                   r   Normalise an Equation
 d   Delete an Equality Set                   p   Critical Pairs
 m   Move Equations                           k   Knuth-Bendix Completion
 c   Copy Equations                           H   Huet's Left-Linear Completion
 P   Peterson & Stickel's AC-Completion       u   Unify Terms
 f   Finish                                   h   Help
-----------------------------------------------------------------------------------
Select Option:  >> n
```

3.6 Rewriting.

We can then use this set of rules to perform rewriting for us. Thus to rewrite a term to normal form we select the "normalise" option, "n". This asks for a set of rules to normalise by and a term to normalise.

```
Rewriting by C/AC Rewriting.
Enter Set of Equalities to Reduce by >> R
Enter Term >> -(-(x + y) + (- ((- y ) ) ) + -(0 + -x))
1 ---> -(-(( x + y )+((-(- y ))+(-(- x )))))
2 ---> ( x + y )+((-(- y ))+(-(- x )))
3 ---> ( x + y )+( y +(-(- x )))
4 ---> ( x + y )+( y + x )
5 ---> x +( y +( y + x ))
Reduces To >> x +( y +( y + x ))
After 5 Rewrites.

Execution Time:  0.060000
Garbage Collecting Time:  0.0
Total Time:  0.060000

Rewriting by C/AC Rewriting.
Enter Set of Equalities to Reduce by >> R
Enter Term >> -((-(x + y))+ (- (-y)) + -(0 + -x))
1 ---> -((-( x + y ))+((-(- y ))+(-(- x ))))
2 ---> -((-( x + y ))+( y +(-(- x ))))
3 ---> -((-( x + y ))+( y + x ))
4 ---> (-( y + x ))+(-(-( x + y )))
5 ---> (-( y + x ))+( x + y )
6 ---> ((- x )+(- y ))+( x + y )
7 ---> (- x )+((- y )+( x + y ))
Reduces To >> (- x )+((- y )+( x + y ))
After 7 Rewrites.

Execution Time:  0.160000
Garbage Collecting Time:  0.0
Total Time:  0.160000
```

Alternatively, an equation can be tested to see if it is a theorem of the equational theory. To do this we select "r".

```
Select Option:  >> r
Enter Set of Equalities to Reduce by >> R
Equation Set or "user":  >> user
Enter Equation:  >> -((-(x + y))+ (-(0 + -x))+ (- (-y))) = 0
Enter Equation:  >>
Reducing by C/AC Rewriting:  -((-( x + y ))+((-( 0 +(- x )))+(-(- y )))) = 0

Execution Time:  0.0
Garbage Collecting Time:  0.0
Total Time:  0.0
Reduces to an Identity :
After 5 Rewrites.

Press Enter/Return to continue.  >>
```

3.7 An AC Example.

We now go on to give a brief example of using Associative-Commutative operators in MERILL . As many of the details are the same as above, we shall only give an indication of those details which differ.

The example we shall use is the completion of the Abelian Group. This uses most of the same object language as the free group example above. For example we use the same operators. However, we mark the _+_ operator with the annotation `(ASSOC COMM)`. This can be done in one of two ways. At the entry of the operator the annotation can be given after the rank of the operator:

```
>>  _ + _ : s s -> s (ASSOC COMM)
```

Alternatively, after an operator has already been added, the equational theory associated with it can be altered by selecting `e` from the operators menu. This starts a dialogue:

The user selects the operator to be annotated, and then the theory. `AC` sets the annotation to `(ASSOC COMM)`. This annotation will be displayed on all ranks of the operator.

The rest of the signature will be the same as previously. On leaving the signature menu the message

```
Calculating new equational theory
```

appears briefly. This tells us the system is checking the compatibility of the equational theory annotation to operators with the order-sorted signature, and if it is compatible, will generate an internal form for the theory. This is trivial in our example.

```
-------------------------------------OPERATORS-------------------------------------
 1        0 :  -> s
 2        - _ :  s -> s
 3        _ + _ :  s s -> s
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                  Operator Options
                                                   a    Add Operators
                                                   d    Delete Operators
                                                   e    Equational Theory
                                                  >>   e
Enter Number of Operator:
>> 1
_ + _
1 :  s s -> s
Theory?  (NONE | ASSOC | COMM | AC) :>> AC
```

We can now use the AC properties of the addition operator in our reasoning. For example, we can use the unification routine. This is entered in the equation menu by entering u:

```
Select Option: >>  u
Unifying Two Terms using Associative-Commutative and
Commutative Order-Sorted Unification.
Enter First Term: >>  x + 0
Enter Second Term: >>  z + y

Unifiers for  x + 0  and  z + y :
{ x -->  z1 + z
  y -->  z1 + 0  }
{ x -->  y + z1
  z -->  z1 + 0  }
{ z --> 0
  x --> y }
{ x --> z
  y --> 0 }
Number of Unifiers : 4

Execution Time: 0.020000
Garbage Collecting Time: 0.0
Total Time: 0.020000
Finished (y/n) ?  >>
Unifying Two Terms using Associative-Commutative and
Commutative Order-Sorted Unification.
Enter First Term: >>  x + x
Enter Second Term: >>  y + z

Unifiers for  x + x  and  y + z :
{ x -->  z1 +( z2 + z3 )
  y -->  z2 +( z3 + z3 )
  z -->  z1 +( z1 + z2 ) }
{ x -->  z1 + y
  z -->  z1 +( z1 + y ) }
{ x -->  z1 + z2
  y -->  z2 + z2
  z -->  z1 + z1  }
{ x -->  z + z1
  y -->  z +( z1 + z1 ) }
{ x --> z
  y --> z }
Number of Unifiers : 5

Execution Time: 0.040000
Garbage Collecting Time: 0.0
Total Time: 0.040000
Finished (y/n) ?  >>  y
```

We can also carry out Associative-Commutative Completion, using the algorithm first described by Peterson and Stickel [26]. However, to do this we again need to first set up the term ordering we are going to use. Thus we return to the environment menu. To be effectively used in the completion algorithm, the term ordering needs to be compatible with the equational theory. Only one of the term orderings defined in the MERILL system at present is guaranteed to be compatible with AC operators; the Associative Knuth-Bendix Ordering. Thus we select this ordering, number 6 on the global ordering menu. This is a variation on the Knuth-Bendix ordering as consequently needs both a precedence on operators and the weights of operators to be defined in advance, and they must conform to certain constraints outlined in the main reference manual. The Precedence on operators for the Abelian Group example is given as:

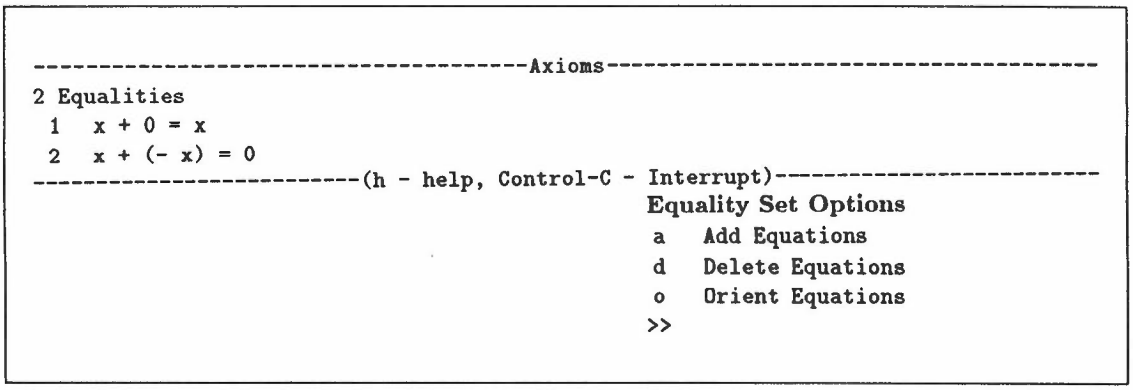
```
-----------------------------------PRECEDENCE-----------------------------------
0 < - _
_ + _ < - _
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                Precedence Options
                                                 a   Add Precedence Declarations
                                                 d   Delete Precedence Declarations
                                                >>
```

We have not used weights before, so we shall see how to enter them in. Selecting `w` from the environment menu will bring up the weights menu. Then selecting `a` will allow us to enter the weights. This is done by entering the full operator form (including underscores) and setting it equal to a number. The figure shows the weights suitable for this example; in many examples, choosing the correct weights is not so simple!

```
-------------------------------------WEIGHTS-------------------------------------
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                                Weight Options
                                                 a   Add Weight Declarations
                                                 d   Delete Weight Declarations
                                                >>  a
Enter (eg.  _ + _ = 1):
Enter Symbol & Weight:  >> _ + _ = 0
Enter Symbol & Weight:  >> - _ = 0
Enter Symbol & Weight:  >> 0 = 1
Enter Symbol & Weight:  >>
```

With the global term ordering set, we are ready to gives some axioms and perform completion. First the axioms defining an Abelian group. We enter into an equality set, `A` say, two axioms.

The axioms for associativity and commutativity are incorporated into the equational theory of the + operator and consequently are not needed here. We perform completion by selected `P` (for Peterson and Stickel) from the equality set menu. This puts us through a dialogue as with Knuth-Bendix completion: we choose to use set `A` for equations and `R` for rules again, and also to step through with no conjectures to try to prove. Completion now



proceeds rather differently from the Knuth-Bendix option. The algorithm sets up a new local equality set **Temporary Rewrites**. This set is used as an intermediate stage; axioms are moved into it, oriented into rewrite rules and used to reduce all the equality sets. Only when there are no more axioms to consider is a rule selected to consider its critical pairs with the other rules in the final rewrite set. Then the critical pairs are oriented before considering the generation of any more. By performing the algorithm in this fashion, unification, which is the most expensive operation in the AC-completion algorithm, is kept to a minimum. The system gives snapshots after it has moved the latest rewrite rule into the final set and generated it critical pairs.

One other point to be noted in this example is the use of extended rules. The algorithm will generated "associative extensions" to rules. These are marked with an asterisk in the output, which also marks that they are protected from being rewritten in certain circumstances. For more details, see Peterson and Stickel's paper.

```
Scanning 0 + x = x
Rule No:1 Ordered as 0 + x => x

Normalising Rules on Right
Normalising Rules on Left
Generating Extensions.

Normalising Rules on Right by Extensions
Normalising Rules on Left by Extensions
Normalising Equations
Scanning x +(- x ) = 0
Rule No:2 Ordered as x +(- x ) => 0

Normalising Rules on Right
Normalising Rules on Left
Generating Extensions.
Extended Rule:  ( x +(- x ))+ z => z (*)

Normalising Rules on Right by Extensions
Normalising Rules on Left by Extensions
Normalising Equations
Considering Critical Pairs of Rule:  x + 0 => x
Critical Pair :  (new/R1):  x = x
Reduced To :  x = x
Delete :  x = x
-------------------------------------Axioms-------------------------------------
0 Axioms
-----------------------------Temporary Rewrites-----------------------------
2 Temporary Rewrites   2   x +(- x ) => 0
                       3   ( x +(- x ))+ z => z (*)
-------------------------------Rewrite Rules--------------------------------
1 Rewrite Rules
 1   x + 0 => x
----------------------------------------------------------------------------

Do you wish to finish (y/n) ?
```

Thus completion will continue until finally we result in the complete set of 6 axioms for the Abelian Group, including the one extended rule. This could be regenerated but we choose to keep it for simplicity.

```
------------------------------------Axioms------------------------------------
0 Axioms
---------------------------------Temporary Rewrites---------------------------------
0 Temporary Rewrites
-----------------------------------Rewrite Rules-----------------------------------
6 Rewrite Rules
 3   - 0 => 0
 1   x + 0 => x
 5   -(- x ) => x
 2   x +(- x ) => 0
 4   ( x +(- x ))+ z => z (*)
 7   -( z + z1 ) => (- z )+(- z1 )
-------------------------------------------------------------------------------
Do you wish to finish (y/n) ?
```

And we can see the statistics of this completion. Note that the number of successful unifications exceeds the total number of attempts to unify! This is because a successful unification may result in more than one unifier.

```
Complete.
Statistics:
Attempted Matches :  2944
Successful Matches :  247
Attempted Unifies :  124
Successful Unifies :  177
Critical Pairs :  58

Confluent Set of 6 Rewrite Rules


Execution Time:  13.140000
Garbage Collecting Time:  0.080000
Total Time:  13.220000
Press Enter/Return to continue.  >>
```

We can then use this complete set to perform AC rewriting, which proceeds as before.

4 A Reference Manual

In this section of this user guide, we give a detailed account of the actions and commands of the MERILL system.

4.1 The Top Level Options.

The top level menu subdivides the system into several working areas. The user uses the system by selecting a character from the menu and pressing return. This brings up the next display and a new menu of options. A series of options can be selected by typing ahead. In this case, the intervening menus are not displayed.

Two options, by default, appear on every menu. These are:

h typing "h" calls up the help information on that menu. Giving an argument gives more help on a specific topic. More information on help is given below.

f leaves this level of menu and returns to its parent. If the current menu is the top level, it leaves the system, asking for confirmation first.

The top level menu gives the following options.

a enters the signature menu for manipulating and checking the signature.

n enters the environment menu for manipulating and checking the environment.

e enters the equalities menu for manipulating and checking the equalities.

S display the current system statistics.

s enter the menu for saving parts of the system into files.

l enter the menu for loading parts of the system from files.

c clears the current system, asking for confirmation first. This removes all signature entries, sets the environment to default and empties the equation sets.

More detailed descriptions of these options are given in the appropriate sections.

4.2 Signatures.

The MERILL system is based around the concept of an order-sorted signature, denoted by a order-sorted algebra.

Unlike other term rewriting systems, which have predefined conventions about which symbols are variables and which are operator symbols, the MERILL system insists that the user declares all the sort, operator symbols and variable symbols which are used in the reasoning part of the system. To this end, the user is supplied with an interface which allows him or her to enter and edit a signature, creating a database of sorts, operators and

variables. In addition to the symbolic declarations, the user can declare the ordering on the sorts used, and also a series of useful checks on the system are supplied, although not insisted upon.

The following options are available.

s enters a menu for manipulating sort names.

o enters a menu for manipulating the partial ordering on sorts.

p enters a menu for manipulating operator names and their associated ranks.

v enters a menu for manipulating variable names and sorts.

i invokes the inhabitedness test on the sorts of the signature.

m invokes the monotonicity test on the operators of the signature.

r invokes the regularity test on the operators of the signature.

g enters a menu for manipulating sort generators. This is not as yet properly implemented, and can be ignored for the time being.

4.2.1 Sorts.

The Sort Options Menu displays those sorts currently declared. Sort identifiers are formed by a string of characters from one syntactic category. Note that there are also two predefined sorts in the system, Top and Bottom, giving an upper and lower bound on sorts. These are not displayed.

Two options are available:

a Add new Sorts to the database. The system continues to ask for Sorts until a blank line is entered. Multiple sort entries can be given on one line. Repeated declarations of sorts will be silently ignored.

d Delete Sorts from the database. The system continues to ask for Sorts until a blank line is entered. If the sort to be deleted has not been declared, it is silently ignored.

4.2.2 Sort Orderings.

The Sort-Ordering Menu gives the current ordering on Sorts. The user declares a set of pairs of sorts as subsort declarations, and the subsort relation is calculated as the least transitive reflexive closure containing the declarations. This ordering is stored and in an efficient manner; no repeated or reflexive pairs are stored and no pair is stored which could be deduced using the transitive closure of the relation. Circularities are also prevented. Note that there are also two predefined sorts available, Top and Bottom, the first of which is larger than any other sort, the second smaller. These are not displayed in the ordering.

a Add new subsort declarations to the database. The system continues to ask for pairs of sorts until a blank line is entered. Pairs can be entered in several formats. They can

be separated by either a "<" or a ">" symbol in which case the obvious ordering is taken. If there is no separating symbol, then the first is taken to be less than the second sort. If only one sort is entered, the system will prompt for a second sort. Again the first is taken to be less than the second. Reflexive declarations will be ignored with a warning. Repeated declarations of sort-order pairs will be silently ignored. Declarations which would lead to circularity generate an error message and are then ignored. No new Sorts can be introduced at this stage.

d Any of the displayed subsort declarations can be deleted, using the same input conventions as for entering subsort declarations. As the sort ordering is defined to be the least transitive reflexive closure containing the current declarations, the sort-ordering is recalculated on the basis of the remaining declarations.

4.2.3 Operators.

The operators of the users signature are maintained using the operator options.

Operators can be declared in a mixfix format. That is they can be declared to have a sequence of separate symbols with the arguments interspersed between them. In declaring the operators symbolic form, the positions which arguments should occupy are denoted by an individual underscore. Also in the declaration of the operator, its signature or rank should also be declared, in the same line. This is given as a sequence of predeclared sort names for the arity of the operator, and a sort name for the coarity. The arity is separated from the symbolic form by a colon ":": the arity from the coarity by an arrow "->". If no arrow is given and only one sort name, the operator is assumed to be a constant. The machine will check whether the arity of the operator corresponds with the number of underscores given in the symbolic form: if it does not correspond an error is reported, unless no underscores appear in which case it assumed that the standard "prefix with brackets and commas" notation. Some examples may help to clarify here.

```
_ + _ : int int -> int              (* infix, binary operator + *)
_ + _ : nat nat -> nat              (* overloaded *)
a : int                             (* constant *)
b : -> int                          (* constant *)
f : int int -> int                  (* function of form f( _ , _ ) *)
f(_) : int -> int
       (* due to arity this is a different operator to the above *)
if _ then _ else _ : bool s s -> s          (* a mixfix operator *)
_ _ : s s -> s                    (* Juxtaposition as an operator *)
```

From the declarations of operators, the system builds its own parser of terms, handling the mixfix forms to try to produce an unambiguous parse. Note as it is currently implemented, the parser chooses one particular parse - which may not be the desired parse. This feature will be altered at a later date to allow the user to choose between a list of possible parses. Note also that brackets can always be used to disambiguate terms.

The user can also declare that operators are associative, commutative, or both, or that they are generators of a particular sort. This is done by declaring one or more of the following attributes in parentheses after the operator declaration:

```
COMM ASSOC GEN
```

Thus we may have

```
0 : -> nat   (GEN)            (* 0 and succ are generators of nats *)
succ : nat -> nat   (GEN)
_ + _ : nat nat -> nat      (COMM ASSOC)                (* + is  AC *)
_ * _ : matrix matrix -> matrix   (ASSOC) (* matrix mult assoc only *)
```

The COMM and ASSOC are used by the system for rewriting and completion and will generate the respective equational theories. Note that currently a declaration will apply over the whole of the sort structure, where ever the terms are well-defined. The system will automatically check the compatibility of the declaration and will delete the declaration if it is incompatible with the signature.

There are three options available on the operator menu.

a This allows the user to repeatedly add new operator forms and their arities and attributes, subject to the conditions mentioned above. Entering a blank line terminates the addition mode and displays the newly added operators at the top of the screen.

d This allows the deletion of operators. Operators are displayed with an accompanying index number. To delete operators, the user is invited to pick a number. Subsequently, the declared arities for that number are displayed with indices and the user is invited to choose an index number to delete a particular arity, or "all" to delete all arities. If all arities are deleted (by either mechanism) the operator form is deleted. Note that if a form is deleted, then the parser is adjusted to prevent further use of the operator. Note also that this does not check whether there are any terms currently in the system using this operator. If there are, then it will not be able to be displayed or manipulated and will cause error messages.

e The user can change the equational theory associated with an operator by choosing this option. The user is prompted for the number of an operator and then a keyword, from {NONE, ASSOC, COMM, AC}, for no equational theory, the associative theory, commutative theory and the associative-commutative theory respectively. This declaration overides all others previously declared and applies to all ranks of the operator. Errors occur if the operator chosen is not binary, or the keyword is not one of the above.

4.2.4 Variables.

There are two types of variable symbol declarations which can be entered. These are called "Whole Variables" and "Variable Prefixes". The first are variable names in their own right. The second form a class of variable names, each with a symbolic form which has the declared prefix. The prefixes cannot overlap - i.e. one prefix declaration cannot be a prefix of another prefix declaration. The system will check for such overlap. However, a Whole variable can be contained in or contain a Variable Prefix declaration. The Whole variables have priority.

Both types of variable name declaration must have a previously declared sort allocated to them. This is done in a variable declaration with the variable declaration (any lexical string) followed by a colon ":" and then the sort name. A variable prefix is distinguished from a whole variable by the use of a asterisk "*" after the variable declaration.

```
x* : int      (* all strings beginning with an x are sort int vars *)
fre* : S       (* all strings beginning with a fre are sort S vars *)
fred : int                  (* but fred is interpreted as a int var *)
joe : S                                  (* and joe is a sort S var *)
```

However, note that in the above example "freda" would be a sort S variable name.

Variable names can be the same as sort names. They can overlap with operator forms, but operator forms take precedence in the parsing mechanism.

The Variable menu gives two options.

a add new variable symbol declarations. They are added in the above format. The system will prompt for new declarations until a blank line is entered. Note that Whole and Prefix declarations can be interspersed.

d delete variable declarations. This is carried out by entering the variable symbol declaration.

4.2.5 Inhabitedness.

The inhabited test whether all the declared sorts have ground terms within them declared in or inferred from the current signature. It then displays two lists: those sorts which are inhabited and those which are not. Note that this function does not enforce inhabitedness, but merely points out those sorts which are not inhabited.

4.2.6 Monotonicity.

The monotonic option will each signature of the given operators in a pairwise fashion to see if any pair break the monotonicity requirement:

Pairwise condition on symbols. If:
$f : s_1\ s_2\ ...\ s_n \rightarrow s$ and
$f : t_1\ t_2\ ...\ t_n \rightarrow t$
then
if $\forall i.s_1 \leq t_i \Rightarrow s \leq t$

then the signatures are pairwise monotonic.

Note that this option does not enforce monotonicity. It displays those pairs of operator arities which break the condition. It is up to the user to manipulate the signature until is has the monotonic property.

4.2.7 Regularity.

This option runs a test of the Regularity of the current signature. Note that this option does not enforce regularity. It displays those pairs of operator arities which break the condition. It is up to the user to manipulate the signature until is has the regularity property.

This again is done pairwise. A finite Order-Sorted signature is Regular if and only if for all operators f which have ranks $w_1 \rightarrow s_1$ and $w_2 \rightarrow s_2$ and if there is some w_0 such that $w_0 \leq w_1$ and also $w_0 \leq w_2$ then there is some $w \leq w_1, w_2$ such that f has rank $w \rightarrow s$ and $w_0 \leq w$.

We can extend this definition to a non-monotonic signature by adding the extra condition that if there is another rank $w' \rightarrow s'$ such that $w_0 \leq w' \leq w_1, w_2$, then $s \leq s'$.

4.3 Equations.

The system is based around a notion of equality sets - collections of equations which can be handled in isolation. By keeping them in this manner, the interaction between the various equations can be controlled tightly by the user.

Each equality set is identified by a *label*, which is usually a single character string, and a *title* which is a longer more descriptive name.

4.3.1 Equality Sets.

Once an equality set has been selected, there are several options available to manipulate them.

The equality set menu provides functions which manipulate equality sets together with a series of operations to perform actions on the equality sets and the equations within them. General information on equality sets is displayed above the equality set menu. Each equality set is identified by a *label*, which is usually a single character string, and a *title* which is a longer more descriptive name, which are displayed together with a the number of equalities currently in the set and the total number which have been in that set over the session.

To manipulate a particular equality set, select **g** and then the label of the desired equality set. A variety of other options are available here: see the entries for each option.

The Equalities menu allow the user to interact with a particular equality set. At the top of the screen, the name of the current equality set is displayed, followed by the current equalities in the set.

The options currently available are:

a add equations to the equality set, in the form s = t . The user will be repeatedly prompted until a blank line is entered. Only operator symbols from the currently declared operators can be used. The parser will currently choose a sensible parse:

WARNING this may not be the one desired. The equation can be quickly deleted if necessary and brackets used to disambiguate.

d delete equations from the equality set. The equalities are displayed with an index number. The user will be repeatedly asked to enter the indices of the equalities to delete until a blank line is entered. Alternatively the command "all" can be used to delete all the current equalities.

o orient equations in the set into rewrite rules. The user will be repeatedly prompted for the indices of rules to orient, until a blank line is entered. Alternatively the command "all" will orient all the equalities in the set. The orientation is carried with respect to the current global and local term orderings.

4.3.2 Equality Types.

Four types of equality are currently possible within the MERILL system

$=$	**Equations**
$=>$	**Rewrite Rules**
$=?=$	**Conjectures**
$e_1, \ldots, e_n ==> e$	**Conditionals**

Equations are unoriented axioms. They can in some circumstances be applied to other equations through subsumption.

Rewrite Rules are oriented rules that can be used to rewrite other equalities.

Conjectures are equalities which are purely passive in nature. They represent a query and are kept in their most reduced form. In addition a record of their original form is kept and will be displayed in braces along side the current form.

Conditional equations can be entered in the form

```
e1 , e2 , ... , en ==> e
```

where the equation e (which can be any variety of equality) will be conditional on the equations e_i. This must be all entered on the one line.

At the moment conditional equations can be rewritten and also be used for rewriting. There is no handling of conditional equations in the completion process.

4.3.3 Unification

Selecting the "u" command on the equality menu allows the user to directly unify two terms which are input from the terminal. Any variables appearing in both terms are renamed to be different from each other. The system will then display the resulting unifiers.

The Unification algorithm used is the full Commutative and Associative/Commutative unification algorithm. If only one arity is declared to be commutative or AC, then the system takes the operator to be commutative or AC everywhere; different theories are not

allowed in different parts of the signature. Care must be taken here as problems can occur if the signature is not sort-preserving or regular.

4.3.4 Rewriting

Two commands on the equality set menu allow normalising of terms and equalities.

n Normalises a term by rewrite rules in a nominated equality set. The system will display the rewriting steps in turn.

r Normalises a set of equations, by rewrite rules in a nominated equality set. The user can choose whether to enter these equations directly or pick them from an existing equality set.

4.3.5 Critical Pairs

Selecting the "p" option from the equality set menu allows the user to directly generate critical pairs between selected equalities. The user enters a dialogue where first one set and then the second set of equalities are selected and associative-commutative critical pairs are generated between members of the two sets. These pairs are not reduced by any set of rules. The user then has the option of saving the generated pairs in a nominated set of rules.

4.3.6 Moving and Copying Equations.

Within in the Equality Set Menu, the user is allowed to Move and Copy equations between equality sets.

The move option `m` allows the user to move equations from one set to another.

The system will prompt for the label of an equality set. It will then display that equality set and prompt for the user to pick equalities by number. The user can pick several equalities this way. After entering a blank line, the system will prompt for the label of the set the equations should be moved to and the transfer will then take place.

The copy option `c` allows the user to copy equations from one set into another.

The system will prompt for the label of an equality set. It will then display that equality set and prompt for the user to pick equalities by number. The user can pick several equalities this way. After entering a blank line, the system will prompt for the label of the set the equations should be copied to and the copying will then take place.

4.4 The Environment.

The Environment is a collection of the sundry extra options which control the operation of the MERILL system.

4.4.1 Global Orderings.

The global term ordering is a noetherian ordering on terms, generic on operators, which controls the ordering of rewrite rules during completion. Several global term orderings have been implemented.

1 **User RPO - Left Status.** A version of the Recursive Path Ordering [2] which uses the precedences on operators given previously by the user and all operators are of left lexicographic status.

2 **User RPO - Right Status.** A version of the Recursive Path Ordering which uses the precedences on operators given previously by the user and all operators are of right lexicographic status.

3 **User RPO - Multiset Status.** A version of the Recursive Path Ordering which uses the precedences on operators given previously by the user and all operators are of multiset lexicographic status.

4 **RPO - Left Status.** A version of the Recursive Path Ordering which attempts to infer precedences from the users promptings and all operators are of left lexicographic status. However, this is currently not correctly implemented and shouldn't be used.

5 **User KBO.** A version of the Knuth-Bendix Ordering [19], where all operators have their weights and precedences predefined by the user before use.

6 **User AKBO.** A version of the Knuth-Bendix Ordering, where all operators have their weights and precedences predefined, which in addition can handle AC-operators. This method is subject to 3 conditions.

   1 C and AC-Operators have a multiset status, others left-lexicographic.

   2 Weight of all AC-operators is zero.

   3 AC-Operators are minimal in the precedence ordering.

   This ordering is based on: "AC-Termination of Rewrite Systems: A Modified Knuth-Bendix Ordering" by Joachim Steinbach [28].

In addition, by not selecting any of the above, no global term ordering will be enforced, and the local ordering will be used for ordering rules. This is particularly useful for manual ordering of rules in completion when no ordering is applicable or easy to set up.

4.4.2 Local Orderings.

The Local Ordering is invoked when the Global Term ordering has insufficient information to determine the orientation of an equation. Note that this is not the same as Unorientability: in that case the Global Ordering cannot order the equation in either direction. In this case the Global Ordering can order the equation in both directions and consequently appeals to the local ordering for a judgement.

Currently the available Local Orderings are:

by-size Puts the term in the equation with the largest number of operators on the left-hand side.

manual Asks the user which way she or he would like to see it oriented

as-is Orients the equation in the left-to-right direction as it finds it.

4.4.3 Local Strategies.

The Local Strategy determines the order in which rules are chosen for consideration in the Knuth-Bendix Algorithm. The Strategy should be "fair". That is, all possible critical pairs are generated and considered for orientation. This is particularly important when the algorithm is being used as a semi-decision procedure to some set of conjectures.

Currently available strategies are:

by-size Picks equations according to the number of operators occurring in both sides - the smallest first. Fair.

by-age Picks equations according to the length of time that they have been in the system - the oldest first. Fair.

manual Prompts the user to select which equation to consider next. Not Fair.

4.4.4 Function Precedences.

The precedence menu allows the user to enter the precedence of operators. This is the precedence used in the standard orderings, and should not be confused with precedence on operators for parsing.

To enter and delete precedences, the user must use the full syntactic description of the operator, with underbars for placeholders. This includes prefix/bracket operators.

Some examples.

```
_ . _   <   _ -
 e  <   _ . _
f(_ , _) <  g(_)
```

Two menu options are given

a Add a precedence operation. The system continues to ask for precedences until a blank line is entered.

d Delete precedences. The system continues to ask for precedences until a blank line is entered.

The current method for manipulating precedences is very unsatisfactory and will be re-designed in a future version of MERILL .

4.4.5 Weights.

Weights can be assigned to operators for use by the Knuth-Bendix Ordering to orient equations into rewrite rules. The Weights option in the environment menu provides a means of assigning weights to operators.

Two options are provided.

a add weights. The user is repeatedly prompted for weights to be assigned to operators. They are added in the form "operator = weight". Note that the whole operator form (as it appears in the operator display) must be entered. If a form is entered twice, the second weight assignment will overwrite the first.

d delete weights. The user may repeatedly enter operator form and the weight is removed from the operator. Note again that the complete form must be used.

4.5 Completion Algorithms.

Three completion algorithms are currently available in the MERILL system.

4.5.1 Knuth-Bendix Completion.

This is an implementation of the Knuth-Bendix completion algorithm [19] adapted to work over an order-sorted signature.

The system will ask for a set of axioms to convert into rules, and a set of rules in which to place new rules. New axioms are placed into the axiom set and the rewrite rule set keep all equalities involved interreduced. Axioms are selected according to the current strategy, and ordered according to the current global and local orderings. For details, see the section on the environment. The user is asked whether completion should proceed by steps, where the system will display and pause after the processing of each new rule. The user is also asked for any conjectures to be proved. These are equalities to be reduced in parallel with the completion process: they are proven when reduced to an identity. When all conjectures are proven, the completion stops and asks whether completion should continue.

Completion proceeds by selecting a rule, orienting it, interreducing all rules and equations, and then generating all critical pairs. When a rule is not orientable, by either the global term ordering, or by the rule not being sort-decreasing, the system will halt and ask whether the user wants to continue. Is so, it will delay consideration of that equation for the time being. When all equations have been considered and all critical pairs have been generated, the completion will halt with success and display statistics on the completion run.

4.5.2 Huet's Completion Algorithm.

This is an implementation of the Huet's Left-linear completion algorithm [16] adapted to work over an order-sorted signature. It handles associative- commutative operators

without using special unification algorithms, but is restricted to left-linear rewrite rules only.

Huet's completion algorithm is initialised and runs in a similar manner as the Knuth-Bendix completion algorithm, with the exception that it will generate an error and halt if a non left-linear rule is generated.

4.5.3 Peterson and Stickel's Completion Algorithm.

An combined implementation of Peterson and Stickel's commutative and associative-commutative completion algorithms [26]. This works by both generating critical pairs by using AC unification and also generating extended rules for the associative law.

The algorithm is initialised in the same way as the Knuth-Bendix algorithm, but proceeds in a different manner. An temporary intermediate equality set is set up and all current equations are moved to this, oriented and interreduced before selecting one of these rules for generating critical pairs. This different completion strategy is used as the AC-unification algorithm is computationally much more expensive. By completing in this manner, the algorithm is more efficient.

Otherwise, proceeds as Knuth-Bendix Algorithm.

4.6 Saving out of the System.

The save options generate files containing parts of the current state of the system. The files are generated in a special ASCII format, which can be edited by the user.

There are three different options available.

a save the current signature only.

e save the current equality sets only.

s save the whole state of the system, including the current environment information such as global ordering, precedence and weights.

On selecting any of these, the system will then prompt for a file name, and then a title line. The title can be any line of text, which the system will put as a comment at the top of the file.

An example of a saved file is given below.

```
# a specification of integers with addition.
signature
sorts
int
pos
neg
zero
```

```
.
sort_ordering
pos < int
neg < int
zero < pos
zero < neg
.
opns
0 :  -> zero
succ : int -> int
succ : pos -> pos
succ : zero -> pos
pred : int -> int
pred : neg -> neg
pred : zero -> neg
 _ + _  : int int -> int     (ASSOC COMM)
 _ + _  : pos pos -> pos
 _ + _  : zero zero -> zero
.
vars
n*  :  neg
p*  :  pos
x*  :  int
y*  :  int
.
.
env
global  user-KBO
local  manual
precedence
  0  <   succ ( _ )
  0   <   pred ( _ )
  succ ( _ ) <  _ + _
  pred ( _ ) <  _ + _
.
weights
   _ + _   =   0
  pred ( _ )   =   0
  succ ( _ )   =   0
 0  =   1
.
strategy  by_size
.
eqns
R   Rewrite Rules
.
A   Axioms
pred(succ(x)) = x
succ(pred(x)) = x
x + 0 = x
x + succ(y) = succ (x + y)
x + pred(y) = pred (x + y)
```

4.7 Loading into the System.

The load options read files generated by the save option.

There are three different options available.

a load signature only.

e load equality sets only.

s load a whole state of the system, including the current environment information such as global ordering, precedence and weights.

On selecting any of these, the system will then prompt for a file name. If syntax errors appear in the files, this will be reported as will any clashes with the existing system declarations. If the latter happens, the new declaration is ignored, as in typing from the terminal.

4.8 Running the System.

4.9 System Statistics.

The MERILL system keeps track of the number of times basic operations are called. Currently, the system counts the number of matches, and unifications attempted, together with the number that succeed, and a count of the number of critical pairs generated. There are two figures for each operation. The total figures keeps a count of the total number of these operations called since the invocation of the system, the partial ones since the last time the figure has been reset, by, for instance, calling the completion procedure.

Also given by the system statistics are some information on the total running times used by the system. Note that the total times given here include timings for the execution of the user interface handling functions, which may absorb a disproportionate amount of the runtime of the system.

4.10 Levels of Display

Currently, three levels of display are available: Full, Partial and Silent. They determine how much information is printed during, for example, a completion process. They are set via entering a number through the environment menu, where the current display level is shown.

Additionally, the user can toggle the Sort display option. If this is toggled to ON, then the sort terms is displayed with the terms.

4.11 The Help System

The MERILL on-line help system is available from every menu. The menu choice "**h <help-key>**" will seek help on the subject given by the help key. The help-key must be a single word (with underscores if necessary), but may be upper or lower case or a mixture.

e.g. this message is given by calling

```
h Help
```

If no help-key is given, then the default help message is given for the current level.

If the help-key "?" ¡question-mark¿ is given, then a list of the current valid keys is given.

The current Help File and the directory where it is to be found is displayed in the environment display and also can be changed from the Environment menu.

5 Installing MERILL

The MERILL system has been written on Standard ML of New Jersey, and is compatible with at least version 0.87. It can run on other implementations of Standard ML, but uses some of the more specific features of this implementation and would require (relatively minor) modification. We release the source code of the system and thus the user must install the system him or her self. Standard ML is notoriously space hungry and we recommend at least a 16Mb machine to install MERILL , although a smaller machine will suffice.

The source is contained within two directories below the source directory, "`sigs`" which has the files containing ML signatures and "`modules`" for the ML modules. In the source directory there is a Standard ML load file. Before installation, edit the file "`modules/help.sml`" changing the line:

```
val Help_Dir = ref "/users/peril/brian/LOTOS/src/help"
```

to be the help directory in your source directory. Also edit the load file "`load`" by changing the line:

```
val SrcDir = "/users/peril/brian/LOTOS/src/"
```

to be your source directory.

Then start up Standard ML of New Jersey and enter:

```
use "load";
```

The system should then load; this may take a few minutes.

On completion of the loading process, then entering:

```
save_eril_image ();
```

will save the whole SML system in one image (some 5Mb). Alternatively, a much smaller image can be created by:

```
make_eril_exec ();
```

which produces an executable without the underlying system. This can be made even smaller by using the "`noshare`" version of the SML system. Caution when using this second command: this will close the current SML session.

6 Future Developments

The MERILL system is still under development and at present only a basic system for largely completion based equational reasoning exists. The system is not as robust as it could be; as a base of experience using the system grows, many changes in adaptability and use will no doubt arise.

Also the system is designed to be a flexible and easy to change system for experimenting with new tools in equational reasoning. Future developments planned for the MERILL system include.

- **A Tactic Language.** Several existing provers, notably ERIL and ORME mentioned earlier have a flexible system of tactics and configurations to allow the user to adapt the use of the system to their own requirements. It is planned to give MERILL some kind of tactic language to allow this. This is a subject for research.
- **Dynamic Order-Sorting.** The order-sorted paradigm at the heart of the MERILL system is well-known to have some problems, notably with respect to the use of sort-decreasing rules only in the completion procedure. New work on Dynamic order-sorts based on [30] is progressing and these new results are being implemented in a new version of the MERILL system [24, 25].
- **Orderings.** The orderings available in the MERILL system are inflexible at present. They are tedious to use as they demand the user to set up the precedence and weights in advance, and only the limited Associative Knuth-Bendix ordering is applicable in the presence of AC operators. To overcome this several pieces of work are planned.
  1. The implementation of a Incremental Knuth-Bendix Ordering, after [6]. An initial implementation has been produced by Nick Cropper and will be integrated into the system.
  2. Implementation of an automatic Recursive Path Ordering.
  3. Investigation and implementation of further AC compatible orderings.
- **Induction.** Inductive proof is a central method of reasoning with equations. It is planned to allow for structural induction in the style of LP.
- **The User Interface.** The teletype user interface, while it is claimed in preferable to LP and also to a raw Standard ML interface, is still unsophisticated and can be tedious to use. It would be desirable to design and implement a windows based interface for the system.
- **Module Support.** A longer term plan for the system is to allow for module and theory store to be integrated into the system for larger scale program specification and refinement.

References

[1] R. J. Cunningham and A. J. J. Dick. Rewrite systems on a lattice of types. *Acta Informatica*, 22:149–169, 1985.

[2] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[3] A J J Dick. ERIL. Equational reasoning: an interactive laboratory. In B. Buchberger, editor, *Proceedings of the EUROCAL conference*. Springer-Verlag, 1985.

[4] A J J Dick. *Order-Sorted Equational Reasoning and Rewrite Systems.* PhD thesis, Imperial College, University of London, 1987.

[5] A. J. J. Dick and J. R. Kalmus. ERIL (Equational Reasoning: an Interactive Laboratory) user's manual. Technical Report RAL-88-055, Rutherford Appleton Laboratory, 1988.

[6] A. J. J. Dick, J R Kalmus, and U H Martin. Automating the Knuth-Bendix ordering. *Acta Informatica*, 28:95–119, 1990.

[7] Stephen J Garland and John V Guttag. An overview of LP, the Larch Prover. In N Dershowitz, editor, *Proc. 3rd Conference on Rewriting Techniques and Applications.*, volume 355 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 1989.

[8] I Gnaedig. ELIOS-OBJ. Theorem proving in a specification language. In B. Kreig-Bruckner, editor, *Proc. of ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 182–199. Springer-Verlag, 1992.

[9] I Gnaedig, C Kirchner, and H Kirchner. Equational completion in order-sorted algebras. *Theoretical Computer Science*, 72:169–202, 1990.

[10] Martin Gogolla. Partially ordered sorts in algebraic specifications. In *Proc. 9th CAAP.*, pages 139–153. Cambridge University Press, 1984.

[11] J A Goguen, J-P Jouannaud, and J Meseguer. Operational semantics for order-sorted algebra. In *Proc. of Int. Conf. on Automata, Languages and Programming.*, volume 194 of *Lecture Notes in Computer Science*, pages 221–231. Springer-Verlag, 1985.

[12] J A Goguen and J Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritence overloading, exceptions and partial operations. Technical Report Tech Monograph no. PRG-80, University of Oxford, 1989.

[13] Joseph Goguen. Order-sorted algebra. Technical Report 14, UCLA Computer Science Dept., 1978.

[14] Joseph A. Goguen and Timothy Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, SRI International Computer Science Laboratory, 1988.

[15] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980.

[16] G. Huet. A complete proof of the correctness of the knuth-bendix completion algorithm. *Journal of Computer Systems and Sciences*, 23(1):11–21, 1981.

[17] Deepak Kapur and Hantao Zhang. An overview of the Rewrite Rule Laboratory (RRL). In N Dershowitz, editor, *Proc. 3rd Conference on Rewriting Techniques and Applications.*, volume 355 of *Lecture Notes in Computer Science*, pages 559–563. Springer-Verlag, 1989.

[18] C Kirchner, Kirchner H, and Meseguer J. Operational semantics of OBJ-3. In *Proc. 9th International Conference on Automata Languages and Programming.*, volume 241 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[19] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J Leech, editor, *Computational Problems in Abstract Algebra*, pages 263 – 297. Pergamon Press, 1970.

[20] Pierre Lescanne. Implementation of completion by transition rules + control: ORME. In *Proc. of 10th ACM Symp on Principles of Programming Languages*, volume 463 of *Lecture Notes in Computer Science*, pages 262–269. Springer-Verlag, 1990.

[21] Brian Matthews. Strategies for theorem proving in an equational reasoning system. Master's thesis, Imperial College, University of London, 1988.

[22] Brian Matthews. Experiments with strategies for equational reasoning. Technical Report RAL-92-025, Rutherford Appleton Laboratory, 1992.

[23] Brian Matthews. MERILL: An equational reasoning system in standard ML. to appear, 5th Int Conf on Rewriting Techniques and Applications, Montreal, Canada, June, 1993.

[24] Brian Matthews. A semantics for dynamic order-sorted equational logic. University of Glasgow, in Preparation, 1993.

[25] Brian Matthews and Phil Watson. Dynamic order-sorted rewriting. University of Glasgow, in Preparation., 1993.

[26] Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *Journal of the Association for Computing Machinery*, 28(2):233–264, 1981.

[27] G Smolka, W Nutt, J Goguen, and J Meseguer. Order-sorted equational completion. In Nivat M. and Aït-Kaci H., editors, *Resolution of Equations in Algebraic Structures*. Academic Press, 1988.

[28] Joachim Steinbach. Ac-termination of rewrite systems: A modified knuth-bendix ordering. In H. Kirchner and W. Wechler, editors, *Proceedings of 2nd Int Conf on Algebraic and Logic Programming*, volume 463 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, 1990.

[29] Uwe Waldmann. Semantics of order-sorted specifiations. Technical Report 297, Universität Dortmund, 1989. To appear, Theoretical Computing Science, Dec 1992.

[30] P. Watson and A. J. J. Dick. Least sorts in order-sorted term rewriting. Technical Report CSD-TR-606, Royal Holloway and Bedford New College, University of London., 1989.