SETS: A Basic Set Theory Package

Francis J. Wright
School of Mathematical Sciences
Queen Mary and Westfield College
University of London
Mile End Road, London E1 4NS, UK.
Email: F.J.Wright@QMW.ac.uk

March 20, 2004

Abstract

The SETS package for REDUCE3.5 and later versions provides algebraic-mode support for set operations on lists regarded as sets (or representing explicit sets) and on implicit sets represented by identifiers. It provides the set-valued infix operators (with synonyms) `union`, `intersection` (`intersect`) and `setdiff` (\, `minus`) and the Boolean-valued infix operators (predicates) `member`, `subset_eq`, `subset`, `set_eq`. The union and intersection operators are n-ary and the rest are binary. A list can be explicitly converted to the canonical set representation by applying the operator `mkset`. (The package also provides an operator not specifically related to set theory called `evalb` that allows the value of any Boolean-valued expression to be displayed in algebraic mode.)

1 Introduction

REDUCE has no specific representation for a set, neither in algebraic mode nor internally, and any object that is mathematically a set is represented in REDUCE as a list. The difference between a set and a list is that in a set the ordering of elements is not significant and duplicate elements are not allowed (or are ignored). Hence a list provides a perfectly natural and satisfactory representation for a set (but not vice versa). Some languages, such as Maple, provide different internal representations for sets and lists, which may allow sets to be processed more efficiently, but this is not *necessary*.

This package supports set theoretic operations on lists and represents the results as normal algebraic-mode lists, so that all other REDUCE facilities that apply to lists can still be applied to lists that have been constructed by explicit set operations. The algebraic-mode set operations provided by this package have all been available in symbolic mode for a long time, and indeed are used internally by the rest of REDUCE, so in that sense set theory facilities in REDUCE are far from new. What this package does is make them available in algebraic mode, generalize their operation by extending the arity of union and intersection, and allow their arguments to be implicit sets represented by unbound identifiers. It performs some simplifications on such symbolic set-valued expressions, but this is currently rather *ad hoc* and is probably incomplete.

For examples of the operation of the SETS package see (or run) the test file `sets.tst`. This package is experimental and developments are under consideration; if you have suggestions for improvements (or corrections) then please send them to me (FJW), preferably by email. The package is intended to be run under REDUCE3.5 and later versions; it may well run correctly under earlier versions although I cannot provide support for such use.

2 Infix operator precedence

The set operators are currently inserted into the standard REDUCE precedence list (see page 28, §2.7, of the REDUCE 3.6 manual) as follows:

```
or and not member memq = set_eq neq eq >= > <= < subset_eq
subset freeof + - setdiff union intersection * / ^ .
```

3 Explicit set representation and `mkset`

Explicit sets are represented by lists, and this package does not require any restrictions at all on the forms of lists that are regarded as sets. Nevertheless, duplicate elements in a set correspond by definition to the same element and it is conventional and convenient to represent them by a single element, i.e. to remove any duplicate elements. I will call this a normal representation. Since the order of elements in a set is irrelevant it is also conventional and may be convenient to sort them into some standard order, and an appro-

priate ordering of a normal representation gives a canonical representation. This means that two identical sets have identical representations, and therefore the standard REDUCE equality predicate (=) correctly determines set equality; without a canonical representation this is not the case.

Pre-processing of explicit set-valued arguments of the set-valued operators to remove duplicates is always done because of the obvious efficiency advantage if there were any duplicates, and hence explicit sets appearing in the values of such operators will never contain any duplicate elements. Such sets are also currently sorted, mainly because the result looks better. The ordering used satisfies the `ordp` predicate used for most sorting within REDUCE, except that explicit integers are sorted into increasing numerical order rather than the decreasing order that satisfies `ordp`.

Hence explicit sets appearing in the result of any set operator are currently returned in a canonical form. Any explicit set can also be put into this form by applying the operator `mkset` to the list representing it. For example

```
mkset {1,2,y,x*y,x+y};
```

```
{x + y,x*y,y,1,2}
```

The empty set is represented by the empty list `{}`.

4 Union and intersection

The operator `intersection` (the name used internally) has the shorter synonym `intersect`. These operators will probably most commonly be used as binary infix operators applied to explicit sets, e.g.

```
{1,2,3} union {2,3,4};
```

```
{1,2,3,4}
```

```
{1,2,3} intersect {2,3,4};
```

```
{2,3}
```

They can also be used as n-ary operators with any number of arguments, in which case it saves typing to use them as prefix operators (which is possible with all REDUCE infix operators), e.g.

```
{1,2,3} union {2,3,4} union {3,4,5};

{1,2,3,4,5}

intersect({1,2,3}, {2,3,4}, {3,4,5});

{3}
```

For completeness, they can currently also be used as unary operators, in which case they just return their arguments (in canonical form), and so act as slightly less efficient versions of `mkset` (but this may change), e.g.

```
union {1,5,3,5,1};

{1,3,5}
```

5 Symbolic set expressions

If one or more of the arguments evaluates to an unbound identifier then it is regarded as representing a symbolic implicit set, and the union or intersection will evaluate to an expression that still contains the union or intersection operator. These two operators are symmetric, and so if they remain symbolic their arguments will be sorted as for any symmetric operator. Such symbolic set expressions are simplified, but the simplification may not be complete in non-trivial cases. For example:

```
a union b union {} union b union {7,3};

{3,7} union a union b

a intersect {};

{}
```

In implementations of REDUCE that provide fancy display using mathematical notation, such as PSL-REDUCE 3.6 for MS-Windows, the empty set, union, intersection and set difference are all displayed using their conventional mathematical symbols, namely $\emptyset$, $\cup$, $\cap$, $\setminus$.

A symbolic set expression is a valid argument for any other set operator, e.g.

```
a union (b intersect c);

b intersection c union a
```

Intersection distributes over union, which is not applied by default but is implemented as a rule list assigned to the variable `set_distribution_rule`, e.g.

```
a intersect (b union c);

(b union c) intersection a

a intersect (b union c) where set_distribution_rule;

a intersection b union a intersection c
```

6 Set difference

The set difference operator is represented by the symbol `\` and is always output using this symbol, although it can also be input using either of the two names `setdiff` (the name used internally) or `minus` (as used in Maple). It is a binary operator, its operands may be any combination of explicit or implicit sets, and it may be used in an argument of any other set operator. Here are some examples:

```
{1,2,3} \ {2,4};

{1,3}

{1,2,3} \ {};

{1,2,3}

a \ {1,2};

a\{1,2}
```

```
a \ a;

{}

a \ {};

a

{} \ a;

{}
```

7 Predicates on sets

These are all binary infix operators. Currently, like all REDUCE predicates, they can only be used within conditional statements (`if`, `while`, `repeat`) or within the argument of the `evalb` operator provided by this package, and they cannot remain symbolic – a predicate that cannot be evaluated to a Boolean value causes a normal REDUCE error.

The `evalb` operator provides a convenient shorthand for an `if` statement designed purely to display the value of any Boolean expression (not only predicates defined in this package). It has some similarity with the `evalb` function in Maple, except that the values returned by `evalb` in REDUCE (the identifiers `true` and `false`) have no significance to REDUCE itself. Hence, in REDUCE, use of `evalb` is *never* necessary.

```
if a = a then true else false;

true

evalb(a = a);

true

if a = b then true else false;

false
```

```
evalb(a = b);

false

evalb 1;

true

evalb 0;

false
```

I will use the `evalb` operator in preference to an explicit `if` statement for purposes of illustration.

7.1 Set membership

Set membership is tested by the predicate `member`. Its left operand is regarded as a potential set element and its right operand *must* evaluate to an explicit set. There is currently no sense in which the right operand could be an implicit set; this would require a mechanism for declaring implicit set membership (akin to implicit variable dependence) which is currently not implemented. Set membership testing works like this:

```
evalb(1 member {1,2,3});

true

evalb(2 member {1,2} intersect {2,3});

true

evalb(a member b);

***** b invalid as list
```

7.2 Set inclusion

Set inclusion is tested by the predicate `subset_eq` where `a subset_eq b` is true if the set a is either a subset of or equal to the set b; strict inclusion is tested by the predicate `subset` where `a subset b` is true if the set a is *strictly* a subset of the set b and is false is a is equal to b. These predicates provide some support for symbolic set expressions, but this is not yet correct as indicated below. Here are some examples:

```
evalb({1,2} subset_eq {1,2,3});

true

evalb({1,2} subset_eq {1,2});

true

evalb({1,2} subset {1,2});

false


evalb(a subset a union b);

true

evalb(a\b subset a);

true

evalb(a intersect b subset a union b);  %%% BUG

false
```

An undecidable predicate causes a normal REDUCE error, e.g.

```
evalb(a subset_eq {b});

***** Cannot evaluate a subset_eq {b} as Boolean-valued set
 expression
```

```
evalb(a subset_eq b);  %%% BUG

false
```

7.3 Set equality

As explained above, equality of two sets in canonical form can be reliably tested by the standard REDUCE equality predicate (=). This package also provides the predicate `set_eq` to test equality of two sets not represented canonically. The two predicates behave identically for operands that are symbolic set expressions because these are always evaluated to canonical form (although currently this is probably strictly true only in simple cases). Here are some examples:

```
evalb({1,2,3} = {1,2,3});

true

evalb({2,1,3} = {1,3,2});

false

evalb(mkset{2,1,3} = mkset{1,3,2});

true

evalb({2,1,3} set_eq {1,3,2});

true

evalb(a union a = a\{});

true
```

8 Installation

The source file `sets.red` can be read into REDUCE when required using `IN`. If the "professional" version is being used this should be done with `ON COMP` set, but it is much better to compile the code as a `FASL` file using `FASLOUT` and then load it with `LOAD_PACKAGE` (or `LOAD`). See the REDUCE manual and implementation-specific guide for further details.

This package has to redefine the REDUCE internal procedure `mk!*sq` and a warning about this can be expected and ignored. I believe (and hope!) that this redefinition is safe and will not have any unexpected consequences for the rest of REDUCE.

9 Possible future developments

- Unary union/intersection to implement repeated union/intersection on a set of sets.
- More symbolic set algebra, canonical forms for set expressions, more complete simplification.
- Better support for Boolean variables via a version (evalb10?) of `evalb` that returns 1/0 instead of `true`/`false`, or predicates that return 1/0 directly.