An introduction to ORME

Pierre LESCANNE

October 3, 1994

Contents

Chapter 1

Introduction

ORME[1] is both a rewrite rule laboratory for equational theories and a toolbox for building theorem provers or software related to equational theories. A previous version of ORME was first written in CAML and took elements from the *CAML Anthology*[Pro87]. In this manual we present only the rewrite rule laboratory. The version described here is in caml-light a dialect on ML [LM93]. The idea behind the construction of ORME was to use a description of completion algorithm by transition rules and to program them directly by ML functions. The original goal described in [Les89] was later modified to build a tool for AC completion based on the same principle and described in [Les90]. The tool was later rewritten in caml-light and a simple stand-alone program was proposed. It is the use of this simple program that is described here.

In the tradeoff between readability of the source code and efficiency, choices have been made toward readability. It results a non so inefficient readable code. For this reason, it could be used in classroom when teaching rewriting and completion or in prototyping other software for equational reasoning.

Few knowledge on associative and commutative completion may be useful, but is not absolutely necessary (see Bachmair's book [Bac91] or Dershowitz and Jouannaud's survey [DJ90] for an introduction). ORME uses a specific completion based on a specific representation of terms, studied and proved to be correct by Eric Domenjoud [Dom91].

ORME uses the associative unification algorithm of UNIF due to Mohamed Adi and Claude Kirchner [AK92]. The current implementation works only on sun3 and sun4 architectures. This means that if you try AC completion on other architectures you may have problems. We will adapt it to other architectures.

In addition to an interactive order, ORME proposes two kinds of orders, namely lexicographic path order and elementary interpretations. *Lexicographic*

[1] Orme is the French name for elm, a tree that disappeared from the European landscape killed by a disease.

path order is a precedence order to due Kamin and Levy (see [DJ90]). *Elementary interpretations* define an order based on elementary functions, i.e., functions on naturals generated by sums, products and exponentials [Les92].

ORME is public domain and due to contributions of:

Zino Benaissa,

Houda Chabbi,

Pierre Lescanne,

and Jocelyne Rouyer.

ORME is available on ftp.loria.fr:/pub/loria/eureca/ORME_LIGHT.

Bibliography

[AK92] M. Adi and Claude Kirchner. AC-unification race: the system solving approach, implementation and benchmarks. *Journal of Symbolic Computation*, 14(1):51–70, 1992.

[Bac91] L. Bachmair. *Canonical equational proofs.* Computer Science Logic, Progress in Theoretical Computer Science. Birkhäuser Verlag AG, 1991.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[Dom91] E. Domenjoud. *Outils pour la déduction automatique dans les théories associatives-commutatives.* Thèse de Doctorat d'Université, Université de Nancy 1, September 1991.

[Les89] P. Lescanne. Completion procedures as transition rules + control. In M. Diaz and F. Orejas, editors, *TAPSOFT'89*, volume 351 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, 1989.

[Les90] P. Lescanne. Implementation of completion by transition rules + control: ORME. In H. Kirchner and W. Wechler, editors, *Proceedings 2nd International Conference on Algebraic and Logic Programming, Nancy (France)*, volume 463 of *Lecture Notes in Computer Science*, pages 262–269. Springer-Verlag, 1990.

[Les92] P. Lescanne. Termination of rewrite systems by elementary interpretations. In H. Kirchner and G. Levi, editors, *Proceedings 3rd International Conference on Algebraic and Logic Programming, Volterra (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 21–36. Springer-Verlag, September 1992.

[LM93] X. Leroy and M. Mauny. The Caml Light system, release 0.5. Documentation and users' manual. Technical report, INRIA Rocquencourt, September 1993.

[Pro87] Projet FORMEL. *The CAML Anthology*, July 1987. Internal Document.

Chapter 2

The commands

The rewrite rule laboratory ORME is aimed to run associative and commutative completion on a set of identities in order to produce *a confluent and terminating set of rules* (often called convergent and sometimes canonical).

To start ORME be sure that your variable environment `$ORME` is properly set[1]. Then type `orme` at the shell level.

For completing a set of identities, the user needs to provide

- a set of identities,
- an order for orienting identities into rules,
- the associative and commutative operators of the experiment.

ORME can also restart a completion from any state of the computation. ORME files are postfixed by `or`. For instance, `true.or` is an ORME file. ORME proposes twelve commands, namely a, c, d, f, h, i, k, o, q, r, t, v.

a sets the status of an operator to be associative and commutative. In the case of more than one associative and commutative operator, operators should be processed one at a time. Beware that declaring operators to be associative and commutative changes the unification and the matching algorithms and can make the completion slower!

c asks ORME to complete a set of identities with a given order. In silent mode with a non interactive order, ORME just returns the result of the completion or fails if a rule cannot be oriented and saved the current state. In verbose mode, the system returns a trace of the completion i.e., displays states at each basic change during the completion.

[1]It should be set a the path name of the directory where ORME lies.

d displays the data known from ORME before, the completion. It is very useful to check that everything has been properly set. This display has four parts plus an optional one,

- the kind of order which has been chosen,
- a description of the order if necessary,
- the set of identities
- the list of associative and commutative operators,
- the saved current state if this state is not the empty state.

Here is a typical example.

```
The current order is ELEMENTARY INTERPRETATIONS

The current interpretation is
+   |--->  X1  +  X2  +  1
M   |--->  (X1)^(1  +  1)
0   |--->   1  +  1

The current identities are
1. (x1 + M(x1)) = 0
2. (x1 + 0) = x1

The current AC operators are:  +

The current state is

  {AC_ops = [+]
   C_ops  = []
   E_raw  = []
   E_nor  = [(x1 + M(0)) = x1]
   S      = []
   S_ext  = []
   T      = []
   T_ext  = []
   C      = []
   C_ext  = [((x1 + M(x1)) -> 0, (x1 + x2 + M(x2)) -> x1, False)]
   N      = []
   N_ext  = []
   A      = [(x1 + 0) -> x1]
   A_ext  = []
   }
```

f saves (freezes) the current state of a computation for a latter restart. See a description of states in Chapter 3. This command is convenient for long interactive completion one wants to restart later one. See its companion t (*thaw*).

h displays a short description of the command, i.e., the following text.

```
a: propose an AC operator
c: complete the set of identities with the given order
d: display the current system and order
f: freeze a state
h: help (this message)
i: entering a list of identities
k: what kind of order: LPO (l),
                       elementary interpretations (e),
                       two levels of elementary interpretations (e2),
                       three levels of elementary interpretations (e3),
                       interactive order (i)
o: entering a file containing an order
q: quit
r: restart completion from the current state
t: thaw a state
v: switch from verbose to silent completion or vice-versa (default is verbose)
```

i asks for the name (without its suffix or) of a file containing a list of identities. The syntax of identities is as follows.

$$\begin{array}{llll}
\textbf{List} & l & ::= & i^* \\
\textbf{Identities} & i & ::= & \texttt{n.}\ t_1 = t_2 \\
\textbf{Terms} & t & ::= & (t_1\ op\ t_2) \mid F(t_1,\ldots,t_n) \mid x\ \ C \\
\textbf{Operators} & op & ::= & + \mid * \mid \hat{}\ \mid \& \mid @ \mid / \mid = \mid \# \mid ! \mid \mid . \mid \texttt{=>} \\
 & & & \mid - \mid \quad \mid \backslash/ \mid /\backslash \mid V \\
\textbf{Naturals} & \texttt{n} & ::= & \texttt{1} \mid \texttt{2} \mid \ldots
\end{array}$$

Variables (x in the grammar) start with lower case and constants and functions symbols (C and F in the grammar) start with upper case.

k asks for the kind of order, one wants to choose. There are five possibilities.

Lexicographic path order, l it is a precedence based order. In the current version of ORME only left-to-right lexicographic status (i.e., the original Kamin Levy order) are allowed. Therefore only a precedence is required (see **o** command).

Elementary interpretations, e an elementary function is a function generated by sums, products and exponentials. To define this ordering the user will be required to provide an interpretation for each operators. Since elementary functions contains polynomials, this order works also for polynomial interpretations.

Two levels of elementary interpretations, e2 some examples can only work with two levels of orders. If this is the case **e2** is chosen. In this case, the user will be asked to provide two files of the same format that contain each an interpretation for the operators.

Three levels of elementary interpretations, e3 the same as for two levels.

Interactive order, i in this case the user will be prompted for each identity that has to be oriented into a rule. The prompt has the following form:

```
=========================================================

M(x1 + x2)  =  (M(x1) + M(x2))  ?
```

- The answer $>$ means that the orientation `M(x1 + x2) > M(x1) + M(x2)` is proposed, then the rule `M(x1 + x2) -> M(x1) + M(x2)` will be created.
- The answer $<$ means that the orientation `M(x1 + x2) < M(x1) + M(x2)` is proposed, then the rule `M(x1) + M(x2) -> M(x1 + x2)` will be created.
- The answer $=$ means that no orientation is proposed and that the intended ordering would say that the two terms are equivalent. Usually, this leads to a failure and should not be regarded as really different from `#`.
- The answer `#` means that no orientation is proposed because the terms are non comparable for the intended ordering. Usually this leads to a failure that can be used to save the current state.
- The answer `q` means that one wanst to leave the completion. This leads to a failure that can be used to save the current state.
- The answer `?` displays a list of the possible answers.

The default ordering is Interactive ordering.

o reads a file containing an order, i.e., an interpretation if **e** has been chosen with the **k** command, two interpretations if **e2** has been chosen with the **k** command, three interpretations if **e3** has been chosen with the **k** command, a precedence if **l** has been chosen with the **k** command. A precedence is a list of pairs $op_1 < op_2$, $p_1 > op_2$ or $op_1 = op_2$ (where op_1 and op_2 are operators or symbols functions) separated by semi-columns. Files are entered without their or postfix. If the file does not contain what is expected ORME complains. Check with **d** whether the correct interpretation has been accepted. See examples in directory EXAMPLES for samples of inputs.

q for quitting ORME.

r restarts a completion starting from the current state. The used ordering is the current one. I may be incompatible (inconsistent) with the one used in the beginning of the completion for reaching the current state. In this case, the behavior is unpredictable. A state is usually read form a file using command `t`. The statistics displayed at the end of a successful are only for this part of the computation.

t restores (thaws) a state in order to make it the current state for a latter restart of the completion. This command is convenient for a long interactive completion one wants to restart later one. See its companion `f` (*freeze*). it is not advisable to type in a state in a file. It is rather suggested to read a frozen state. One should not forget also to set the ordering if one does not want to use the interactive ordering or the current ordering.

v In verbose mode ORME displays, during the completion, all the states when they are to be changed by a transition rule and, at the end of the computation, statistics on critical pairs, unifications and matchings. In silent mode, ORME displays only the result, i.e., a state, at the end of the completion. **v** switches from one mode to the other. The default mode is VERBOSE.

Chapter 3

States and transition rules

The procedure is basically designed as a completion machine that has states and transition rules. The completion procedure provides a strategy to invoke transition rules. This chapter describes the states and the transition rules.

States

A state has fourteen components.

AC_ops is the set of associative and commutative operators

C_ops is the set of commutative operators (this feature is not yet implemented).

E_raw is the set of identities before any simplification and deletion of trivial identities.

E_nor is the set of identities after simplification, standardization (renaming of variables in increasing order from left to right) and deletion of trivial identities.

S is a container that contains at most one rule called a *simplifier*. When there is a rule in **S**, this rule is used to simplify every left-hand side of rules in the state. Simplified rules become raw identities and go into **E_raw**.

S_ext is a container that at most contains one extended rule and its extension. Its role is similar to that of **S**.

T is a waiting room where rules stay before being processed furthermore.

T_ext is a waiting room where extended rules and there extensions stay before being processed furthermore.

C is a container where a rule stays during the process of computing its critical pairs with other rules.

C_ext the same as **S**, but contains an extended rule, its extension and a boolean. The boolean allows controlling the superpositions between **C** and **N**. This way one can interleave more simplifications between computations of critical pairs.

N is a set of rules whose critical pairs have <u>n</u>ot been computed yet with rule in **C** or rules in **C_ext**.

N_ext is a set of extended rules and their extensions whose critical pairs have <u>n</u>ot been computed yet with rule in **C** or rules in **C_ext**.

A is a set of rules whose critical pairs have been <u>a</u>lready computed with rule in **C** or rules in **C_ext**.

A_ext is a set of extended rules and their extensions whose critical pairs have been <u>a</u>lready computed with rule in **C** or rules in **C_ext**.

Transitions rules

ORME describes a completion by transition rules.

Cleaning of E_raw, the first transition of a completion, it means there is something in *E_raw*. This transition rule transfers rules from *E_raw* into *E_nor* after standardization, deletion of trivial identities and simplification by the rules of the current state.

New loop, this rule indicates that one restarts a new loop for computing critical pairs. In this case *E_raw*, *E_nor*, *S*, *C* and *C_ext* are empty.

Simplification, means that there is a rule in **S** and ORME will simplify everything with that rule.

S_ext Simplification, means that there is a rule and its extension in **S_ext** and ORME will simplify everything with those rules.

Orientation of E_nor, takes a identity in **E_nor** and orients it into a rule according to the given order.

Deductions, according to which of **N**, **N_ext**, **C** and **C_ext** is chosen, there are four rules *N_C_Deduction*, *N_ext_C_Deduction*, *N_C_ext_Deduction* and *N_ext_C_ext_Deduction*.

Internal Deduction, announces a computation of critical pairs from superposition of the rule in **C** or **C_ext** on itself or themselves. There is two rules *C_internal_Deduction* or *C_ext_internal_Deduction* according to whether **C** or *C_ext* is involved. We call such generated critical pairs, *self critical pairs*.

Chapter 4

A commented example

As an illustration on how ORME works we propose to run an example. We choose a very classical case of associative and commutative completion, namely abelian groups. This example lies in directory EXAMPLES/ABELIAN. There are three files in this directory namely README, eq.or and in.or. eq.or contains

```
1. (x + M(x)) = 0
2. (x + 0) = x
```

and in.or contains

```
+ |--> X1 + X2 + 1
M |--> (X1)^(2)
0 |--> 2
```

Here is documented script. Commands typed by the user are underlined. Comments added are in italics

% orme
Welcome to ORME the rewrite rule laboratory
Copyright (C) 1993, EURECA Group CRIN (CNRS) & INRIA
Type 'h' (followed by a carriage return) for help
Starting ORME.
ORME1: i
Input a file with list of identities (default is ".or"): eq

Entering a list of identities.
ORME2: k
Choose an order (l, e, e2, e3, i): e

Choosing the elementary interpretation order.

ORME3: o

Order input file (default is ".or"): in

Declaring + to be associative and commutative.

ORME4: a

+

Displaying what has been set.

ORME5: d

The current order is ELEMENTARY INTERPRETATIONS

The current interpretation is
+ ——> X1 + X2 + 1
M ——> (X1)(1 + 1)
O ——> 1 + 1

The current identities are
1. (x1 + M(x1)) = O
2. (x1 + O) = x1

The current AC operators are: +

ORME6: c

Let us start a completion.

— Cleaning of E_raw — step 1

```
AC_ops = ["+"]
C_ops = []
E_raw = [(x1 + M(x1)) = O;
         (x1 + O) = x1]
E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = []
```

```
A_ext = []


— Orientation of E_nor — step 2

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = [(x1 + O) = x1;
          O = (x1 + M(x1))]
S = []
S_ext = []
T = []
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = []
A_ext = []


— Simplification — step 3

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = [O = (x1 + M(x1))]
S = [(x1 + O) -> x1]
S_ext = []
T = []
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = []
A_ext = []


— Orientation of E_nor — step 4

AC_ops = ["+"]
C_ops = []
```

E_raw = []
E_nor = [O = (x1 + M(x1))]
S = []
S_ext = []
T = [(x1 + O) -> x1]
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = []
A_ext = []

— S_ext Simplification — step 5

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = [((x1 + M(x1)) -> O, (x2 + x1 + M(x1)) -> x2)]
T = [(x1 + O) -> x1]
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = []
A_ext = []

— New loop — step 6

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = [(x1 + O) -> x1]
T_ext = [((x1 + M(x1)) -> O, (x2 + x1 + M(x1)) -> x2)]
C = []
C_ext = []

N = []
N_ext = []
A = []
A_ext = []

— C_internal_Deduction — step 7

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = []
T_ext = [((x1 + M(x1)) -> O, (x2 + x1 + M(x1)) -> x2)]
C = [(x1 + O) -> x1]
C_ext = []
N = []
N_ext = []
A = []
A_ext = []

— New loop — step 8

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = []
T_ext = [((x1 + M(x1)) -> O, (x2 + x1 + M(x1)) -> x2)]
C = []
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1]
A_ext = []

— N_C_ext_Deduction — step 9

```
AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = []
C_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1, True)]
N = [(x1 + O) -> x1]
N_ext = []
A = []
A_ext = []
```

— Cleaning of E_raw — step 10

```
AC_ops = ["+"]
C_ops = []
E_raw = [(x1 + M(O)) = x1;
         (x1 + y2 + M(O + y2)) = x1;
         (x2 + M(x2)) = O;
         (y3 + x2 + M(x2)) = (O + y3)]
E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = []
C_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1, False)]
N = []
N_ext = []
A = [(x1 + O) -> x1]
A_ext = []
```

⋮

Let us drop computations.

⋮

— Cleaning of E_raw — step 22

```
AC_ops = ["+"]
```

C_ops = []
E_raw = [(x2 + x1) = (x1 + x2)]
E_nor = [M(x1 + M(x1 + x2)) = x2;
M(x1) = (x2 + M(x1 + x2));
(x1 + M(x1 + x2)) = M(x2);
(x1 + M(x1 + M(x2))) = x2;
(x1 + M(x2)) = (x1 + x3 + M(x2 + x3));
(x1 + M(x1 + x2 + M(x2 + x3))) = x3;
(x1 + x2 + M(x2 + x3)) = (x1 + M(x3));
(x1 + x2 + M(x2 + M(x3))) = (x1 + x3);
(x1 + M(x2 + M(x2 + x3))) = (x1 + x3);
(x1 + M(x1 + x2)) = (x3 + M(x2 + x3));
(x1 + x2 + M(x2 + x3 + M(x3 + x4))) = (x1 + x4);
(x1 + x2 + M(x2 + x3)) = (x1 + x4 + M(x3 + x4))]
S = []
S_ext = []
T = [M(O) -> O;
M(M(x1)) -> x1]
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]

— Orientation of E_nor — step 23

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = [M(x1 + M(x1 + x2)) = x2;
M(x1) = (x2 + M(x1 + x2));
(x1 + M(x1 + x2)) = M(x2);
(x1 + M(x1 + M(x2))) = x2;
(x1 + M(x2)) = (x1 + x3 + M(x2 + x3));
(x1 + M(x1 + x2 + M(x2 + x3))) = x3;
(x1 + x2 + M(x2 + x3)) = (x1 + M(x3));
(x1 + x2 + M(x2 + M(x3))) = (x1 + x3);
(x1 + M(x2 + M(x2 + x3))) = (x1 + x3);
(x1 + M(x1 + x2)) = (x3 + M(x2 + x3));
(x1 + x2 + M(x2 + x3 + M(x3 + x4))) = (x1 + x4);
(x1 + x2 + M(x2 + x3)) = (x1 + x4 + M(x3 + x4))]

```
S = []
S_ext = []
T = [M(O) -> O;
      M(M(x1)) -> x1]
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]
```

— Simplification — step 24

```
AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = [M(x1) = (x2 + M(x1 + x2));
      (x1 + M(x1 + x2)) = M(x2);
      (x1 + M(x1 + M(x2))) = x2;
      (x1 + M(x2)) = (x1 + x3 + M(x2 + x3));
      (x1 + M(x1 + x2 + M(x2 + x3))) = x3;
      (x1 + x2 + M(x2 + x3)) = (x1 + M(x3));
      (x1 + x2 + M(x2 + M(x3))) = (x1 + x3);
      (x1 + M(x2 + M(x2 + x3))) = (x1 + x3);
      (x1 + M(x1 + x2)) = (x3 + M(x2 + x3));
      (x1 + x2 + M(x2 + x3 + M(x3 + x4))) = (x1 + x4);
      (x1 + x2 + M(x2 + x3)) = (x1 + x4 + M(x3 + x4))]
S = [M(x1 + M(x1 + x2)) -> x2]
S_ext = []
T = [M(O) -> O;
      M(M(x1)) -> x1]
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]
```

⋮

Let us drop again computations.

⋮

— N_ext_C_Deduction — step 69

```
AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = [M(x1 + x2) -> (M(x1) + M(x2))]
C_ext = []
N = []
N_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1, True)]
A = [(x1 + O) -> x1;
        M(M(x1)) -> x1;
        M(O) -> O]
A_ext = []
```

— Cleaning of E_raw — step 70

```
AC_ops = ["+"]
C_ops = []
E_raw = [M(x3) = (M(x3) + M(M(x4) + x4));
        M(x1 + y1) = (M(x1) + M(M(x4) + x4 + y1));
        M(x3) = (M(x4) + M(M(x4) + x3));
        M(x3) = (M(x4 + x3) + M(M(x4)));
        M(y2 + y1) = (M(x4 + y2) + M(M(x4) + y1));
        M(x3) = (M(x1) + M(M(x1 + y3) + y3 + x3));
        M(x3) = (M(y4 + x3) + M(M(y4 + y3) + y3));
        M(y2 + y1) = (M(y4 + y2) + M(M(y4 + y3) + y3 + y1));
        M(x3) = (M(M(x4)) + M(x4 + x3));
        M(x3) = (M(M(x4) + x3) + M(x4));
        M(y2 + y1) = (M(M(x4) + y2) + M(x4 + y1));
        M(x3) = (M(M(x4) + x4) + M(x3));
        M(y2 + x2) = (M(M(x4) + x4 + y2) + M(x2));
        M(x3) = (M(M(y4 + y3) + y4) + M(y3 + x3));
        M(x3) = (M(M(y4 + x2) + y4 + x3) + M(x2));
        M(y2 + y1) = (M(M(y4 + y3) + y4 + y2) + M(y3 + y1));
        (x1 + x2 + x3 + M(x1) + M(x2)) = x3]
```

E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = [M(x1 + x2) -> (M(x1) + M(x2))]
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1;
M(M(x1)) -> x1;
M(O) -> O]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]

— C_internal_Deduction — step 71

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = [M(x1 + x2) -> (M(x1) + M(x2))]
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1;
M(M(x1)) -> x1;
M(O) -> O]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]

=====================
Successful Completion
=====================
— RESULT — step 72

AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []

```
S = []
S_ext = []
T = []
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1;
          M(M(x1)) -> x1;
          M(O) -> O;
          M(x1 + x2) -> (M(x1) + M(x2))]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]


=====================
STATISTICS
=====================
72 basic steps
144 successful matchings
1862 unsuccessful matchings
63 successful AC_matchings
1516 unsuccessful AC_matchings
22 calls for AC_unifications
107 critical pairs
35 self-critical pairs
*******************


AC_ops = ["+"]
C_ops = []
E_raw = []
E_nor = []
S = []
S_ext = []
T = []
T_ext = []
C = []
C_ext = []
N = []
N_ext = []
A = [(x1 + O) -> x1;
          M(M(x1)) -> x1;
          M(O) -> O;
```

M(x1 + x2) -> (M(x1) + M(x2))]
A_ext = [((x1 + M(x1)) -> O, (x1 + x2 + M(x2)) -> x1)]

The completion was done in 81 s

The completion succeeded, but nothing has been changed.

ORME7: d

The current order is ELEMENTARY INTERPRETATIONS

The current interpretation is
+ ——> X1 + X2 + 1
M ——> (X1)(1 + 1)
O ——> 1 + 1

The current identities are
1. (x1 + M(x1)) = O
2. (x1 + O) = x1

The current AC operators are: +

ORME8: q

Let us quit ORME.