OTTER Documentation

OTTER stands for “Organized Techniques for Theorem-proving and Effective Research”

1 Running OTTER

Suppose input-file is the file containing your first-order logical sentences (contained within a formula-list/end-of-list block, as described below). OTTER reads input from the standard input and writes to the standard output; thus if you type

```
    otter < input-file > output-file
```

at the UNIX prompt, OTTER will read the input-file and send output to the output-file.¹ No command line arguments are accepted; all options are given in the input-file using OTTER commands.

2 Syntax

OTTER recognizes two basic types of statement: clauses and formulas. Clauses are simple disjunctions all of whose variables are implicitly universally quantified. OTTER searches for proofs by operating on clauses. Formulas are first-order statements without free variables--all variables are explicitly quantified. When formulas are input, OTTER immediately translates them into clauses.

Function symbols and predicate symbols are sometimes referred to as functors when the distinction is not important.

2.1 Comments

Comments can be placed in the input file by using the symbol %. All characters from the first % on a line to the end of the line are ignored. Comments can occur within terms. Comments are not echoed to the output file.

¹ The Makefile will automatically concatenate your kb.in and query.in files to create a suitable input-file. That's why you don't put an end-of-list statement in your kb.in file, but you DO put it in the various query.in files.
2.2 Names for Variables, Constants, Functions, and Predicates

Three kinds of character string, collectively referred to as names, can be used for variables, constants, function symbols, and predicate symbols:

- An ordinary name is a string of alphanumerics, $, and _.
- A special name is a string of characters in the set * + - / \ < > = ' ~ : ? @ & ! ; # (and sometimes |).
- A quoted name is any string enclosed in two quotation marks of the same type, either " or '. We have no trick for including a quotation mark of the same type in a quoted name.

(The reason for separating ordinary and special names has to do with infix, prefix, and postfix operators.) Although out of place here, for completeness we list the meanings of the remaining printable characters.

- . (period) -- terminates input expressions.
- $ -- starts a comment (which ends with the end of the line).
- , () [] {} (and sometimes |) -- are punctuation and grouping symbols.

Variables. Determining whether a simple term is a constant or a variable depends on the context of the term. If it occurs in a clause, the symbol determines the type: the default rule is that a simple term is a variable if it starts with u, v, w, x, y or z. If the flag prolog_style_variable is set, a simple term is a variable if and only if it starts with an upper case letter or with _. (Therefore, variables in clauses must be ordinary names.) A simple term in a formula is a variable if and only if it is bound by a quantifier.

Reserved and Built-in-Names. Names that start with $ are reserved for special purposes, including evaluable functions and predicates, answer literals and terms, and some internal system names. The name = and any name that starts with eq, EQ or Eq when used as a binary predicate symbol, is recognized as an equality predicate by the demodulation and paramodulation processes. And some names, when they occur in clauses or formulas, are recognized as logic symbols.

Overloaded Symbols. The user can use a name for more than one purpose, for example, as a constant and as a 5-ary predicate symbol. When the flag check arity is set (the default), the user is warned about such uses. Some built-in names are also overloaded; for example, | is used both for disjunction and as Prolog-style list punctuation, and although ~ is built in as logical negation, it is generally used for both unary and binary minus as well.

2.3 Terms and Atoms

Recall that, when interpreted, terms are evaluated as objects in some domain, and atoms are evaluated as truth values. Constants and variables are terms. An n-ary function symbol applied to n terms is also a term. An n-ary predicate symbol applied to n terms is an atom. A nullary predicate symbol (also referred to as a propositional variable) is also an atom.
The pure way of writing complex terms and atoms is with *standard application*: the function or predicate symbol, opening parenthesis, arguments separated by commas, then closing parenthesis, for example, \( f(a, b, c) \) and \( (f(x, e), x) \). If all subterms of a term are written with standard application, the term is in *pure prefix form*. Whitespace (spaces, tabs, newlines, and comments) can appear in standard application terms anywhere *except* between a function or predicate symbol and its opening parenthesis. If the flag `display_terms` is set, OTTER will output terms in pure prefix forms.

**Infix Equality.** Some binary functors can be written in infix form; the most important is \( = \). In addition, a negated equality, \(-(a=b)\) can be abbreviated \( a \neq b \).

**List Notation.** Prolog-style list notation can be used to write terms that represent lists. Table 1 gives some example terms in list notation and the corresponding pure prefix form. Of course, lists can contain complex terms, including other lists.

<table>
<thead>
<tr>
<th>Table 1: List Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
</tr>
<tr>
<td>[x</td>
</tr>
<tr>
<td>[x,y]</td>
</tr>
<tr>
<td>[a,b,c,d]</td>
</tr>
<tr>
<td>[a,b,c</td>
</tr>
</tbody>
</table>

**2.4 Literals and Clauses**

A literal is either an atom or the negation of an atom. A clause is a disjunction of literals. The built-in symbols for negation and disjunction are \( - \) and \( | \), respectively. Although clauses can be written in pure prefix form, with \( - \) as a unary symbol and \( | \) as a binary symbol, they are rarely written that way. Instead, they are almost always written in infix form, without parentheses. For example, the following is a clause in both forms.

- Pure prefix: \( |(-(a), |=(b1,b2), -=(c1,c2)) \)
- Infix (abbreviated): \( -a \mid b1=b2 \mid c1!=c2 \)

OTTER accepts both forms.
2.5 Formulas

Table 2 lists the built-in logic symbols for constructing formulas.

<table>
<thead>
<tr>
<th>Table 2: Logic Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
</tr>
<tr>
<td>disjunction</td>
</tr>
<tr>
<td>conjunction</td>
</tr>
<tr>
<td>implication</td>
</tr>
<tr>
<td>equivalence</td>
</tr>
<tr>
<td>existential quantification</td>
</tr>
<tr>
<td>universal quantification</td>
</tr>
</tbody>
</table>

Formulas in Pure Prefix Form. Although the practice is rarely done, formulas can be written in pure prefix form. Quantification is the only tricky part: there is a special variable-arity functor, $\exists x y z$, for quantified formulas. For example, $\forall x y z (P(x,y,z) \lor Q(x,z))$ is represented by

$$\exists x y z (P(x,y,z) \lor Q(x,z)).$$

If the flag display_terms is set, the formulas (and everything else) will be displayed in pure prefix form.

Abbreviated Formulas. Formulas are usually abbreviated in a natural way. Here are some examples.

<table>
<thead>
<tr>
<th>standard usage</th>
<th>OTTER syntax (abbreviated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x P(x)$</td>
<td>all x P(x)</td>
</tr>
<tr>
<td>$\forall x y z(P(x,y,z) \lor Q(x,z))$</td>
<td>all x y exists z (P(x,y,z)</td>
</tr>
<tr>
<td>$\forall x (P(x) \land Q(x) \land R(x) \rightarrow S(x))$</td>
<td>all x (P(x) &amp; Q(x) &amp; R(x) -&gt; S(x))</td>
</tr>
</tbody>
</table>

Note that if a formula has a string of identical quantifiers, all but the first can be dropped. For example, all x all y all z p(x,y,z) can be shortened to all x y z p(x,y,z). In expressions involving the associative operations & and |, extra parentheses can be dropped. Moreover, a default precedence on the logic symbols allows us to drop more parentheses: $\leftrightarrow$ has the same precedence as $\rightarrow$, and the rest in decreasing order are $\rightarrow$, $\land$, $\lor$ and $\neg$. Greater precedence means closer to the root of the term (i.e., larger scope). For example, $p \lor \neg q \land \neg r \rightarrow \neg s \land \top$ represents $(p \mid (-q \land r)) \rightarrow (-s \mid \top)$, or in pure prefix form, $\rightarrow ((p, \land (-q, r)), (-s, \top))$.

When in doubt about how a particular string will be phrased, one can simply add additional parentheses and/or test the string by having OTTER read it and then display it in pure prefix form. The following input file can be used to test the preceding example.
Assign(stats_level, 0).
set(display_terms).
formula_list(usable).
p| -q&r-> -s|t. %This formula has minimum whitespace.
end_of_list.

In general, whitespace is required around all and exists and to the left of -. otherwise, whitespace around the logic symbols can be removed.

2.6 Whitespace in Expressions

The reason for separating ordinary names from special names is so that some whitespace (spaces, tabs, newline, and comments) can be removed. We can write a+b+c (instead of having to write a + b + c), because “a+b+c” cannot be a name; that is, it must be parsed into five names.

Caution. There is a deficiency in OTTER’s parser having to do with whitespace between a name and opening parenthesis. The rule to use is: Insert some whitespace if and only if it is not a standard application. For example, the two pieces of whitespace in (a+ (b+c)) = (d+e) are required, and no whitespace is allowed after f or g in f (x, g (x)).

2.7 Examples of Operator Declarations

Group Theory. Suppose we like to see group theory expressions in the form (ab⁻¹c⁻¹⁻¹)⁻¹, in which right association is assumed. We can approximate this for OTTER with (a*b^ *c^ ^)!. (We have to make the group operator explicit; -1 is not a legal OTTER name; the whitespace shown is required). The declarations op(400, xfy, *) and op(350, yf, ^) suffice. Other examples of expressions (with minimum whitespace) using these declarations are (x*y)*z=x*y*z and (y*x)^ =x^ *y^.

3 Commands and the Input File

Input to Otter consists of a small set of commands, some of which indicate that a list of objects (clauses, formulas, or weight templates) follows the command. All lists of objects are terminated with end_of_list. The commands are given in table 3. There are a few other commands for fringe features.
Table 3: Commands

include(file_name).
% read input from another file
op(precedence, type, name(s)).
% declare operator(s)
make_evaluable(sym, eval-sym).
% make a symbol evaluable
set(flag_name).
% set a flag
clear(flag_name).
% clear a flag
assign(parameter_name_integer).
% assign an integer to a parameter
list(list_name).
% read a list of clauses
formula_list(list_name).
% read a list of formulas
weight_list(weight_list_name).
% read weight templates
lex(symbol_list).
% assign an ordering on symbols
skolem(symbol_list).
% identify skolem functions
lrpo_multiset_status(symbol_list).
% status for LRPO

3.1 Input of Options

OTTER recognizes two kinds of option: flags and parameters. Flags are Boolean-valued options; they are changed with the set and the clear commands, which take the name of the flag as the argument. Parameters are integer-valued options; they are changed with the assign command, which takes the name of the parameter as the first argument and an integer as the second. Examples are:

set(binary_res).
% enable binary resolution
clear(back_sub).
% do not use back subsumption
assign(max_seconds, 300).
% stop after about 300 cpu seconds

3.2 Input of Lists of Clauses

A list of clauses is specified with one of the following and is terminated with end_of_list. Each clause is terminated with a period.

list(usable).
list(sos).
list(demodulators).
list(passive).
Example:

\[
\begin{align*}
\text{list(usable).} \\
& x = x. \quad \% \text{reflexivity} \\
& f(e,x) = x. \quad \% \text{left identity} \\
& f(g(x),x) = e. \quad \% \text{left inverse} \\
& f(f(x,y),z) = f(x,f(y,z)). \quad \% \text{associativity} \\
& f(z,x) \neq f(z,y) \mid x = y. \quad \% \text{left cancellation} \\
& f(x,z) \neq f(y,z) \mid x = y. \quad \% \text{right cancellation}
\end{align*}
\]

end_of_list.

If the input contains more than one clause list of the same type, the lists will simply be concatenated.

### 3.3 Input of Lists of Formulas

A list of formulas is specified with one of the following and is terminated with `end_of_list`. Each formula is terminated with a period. (Note that demodulators cannot be input as formulas.)

```
.formula_list(usable).
.formula_list(sos).
.formula_list(passive).
```

Example (analogous to above):

```
formula_list(usable).
  all a (a = a) \quad \% \text{reflexivity} \\
  all a (f(e,a) = a). \quad \% \text{left identity} \\
  all a (f(g(a),a) = e). \quad \% \text{left inverse} \\
  all a b c (f(f(a,b),c) = f(a,f(b,c))). \quad \% \text{associativity} \\
  all a b c (f(c,a) = f(c,b) \rightarrow a = b). \quad \% \text{left cancellation} \\
  all a b c (f(a,c) = f(b,c) \rightarrow a = b). \quad \% \text{right cancellation}
end_of_list.
```