Imperial College London - Department of Computing

# A Brief Introduction to Otter

Krysia Broda and Silvana Zappacosta

November 2009

## What is Otter?

Otter could stand for Organized Techniques for Theorem-proving and Effective Research. It is a resolution based theorem prover that incorporates binary resolution, factoring, hyper-resolution and paramodulation. It was developed at Argonne by William McCune, Larry Wos and others and is written in C.

For a complete reference see the reference manual for Otter 3.3. The manual can be retrieved either from the directory `/usr/share/doc/otter` or from the Otter web page at location `http://www-unix.mcs.anl.gov/AR/otter`.

Otter is not automatic. The user sets various parameters and options and then Otter applies a simple loop to look for a proof. If it cannot find one then the user can alter the settings.

There are two main lists of clauses used by Otter the `sos` list (set-of-support) and `usable` list (there are two more, but see the manual - the `passive` list can be especially useful). The `sos` list is used to control the refutation, in that every resolvent must either have a parent in `sos` or one derived from it. This is not a restriction, for every clause can initially be in `sos` if you wish.

The main loop is –

```
While (sos is not empty and no refutation found yet) do
      select given-clause from sos (the 'lightest')
      move given-clause to usable
      infer and process new clauses using the inference rules in effect;
          each new clause has the given-clause as one of its parents
          and one of the usable clauses as the other parent;
          new clauses that are kept are added to sos
End while.
```

Various processing is applied to new clauses, such as factoring and subsumption tests if these are indicated by the options, called *flags*.

## Using Otter

The specification of the Computer-Based Coursework (CBC) No. 1 contains the necessary guidelines for running Otter under Linux.

All **input** is taken from a prepared input file and **output** is written to an output file. It is not particularly convenient to use, but it does work!

**Comments** can be inserted into the input file using %. All input on a line beyond % is ignored.

**Variables** start with the lower case letters in the range `u` to `z`. If you prefer, setting the flag `prolog_style_variables` says that variables start with upper case letters.

Prolog style list notation can be used – `[]` is the empty list, `[`$a, b, c, d$`]` represents the list with the four elements $a$, $b$, $c$ and $d$ and `[`$h \mid t$`]` represents the list with head $h$ and tail $t$.

Spaces tabs and newlines can occur anywhere in complex terms except within names and between a function or predicate symbol and the opening parenthesis.

A **clause** is a sequence of literals separated with |. If $a$ is an atom then $a$ and $-a$ are literals. A clause is ended by a full stop (not considered part of the clause).

### Commands

There are commands that indicate that a list of clauses follows, commands that set and clear flags and commands to assign parameters to various values. We only give a few here to get you going.

To input a list of clauses use either `list(usable)` or `list(sos)`. Each command and each clause is ended by a full stop. The whole list is ended by `end_of_list` (followed by a full stop).

There are ways to control the order in which Otter selects clauses from `sos` to be the given-clause, by weighting clauses in various ways. Or, it is possble to control the order interactively. The default weight of a term is 1+ the sum of the weights of the arguments. The weight of a constant or variable is just 1. The default weight of an atom is similarly 1+ the sum of the weights of the arguments. Thus the default weight of a clause is the number of symbols it contains. It is possible to change the weight of an atom or term to be 1+ the maximum weight of the arguments.

The commands for that are `set(atom_wt_max_args)` and `set( term_wt_max_args)` respectively. To make interactive selection of givens, use `set(interactive_given)`.

### Various flags

The default setting of most flags is clear. If not, it is indicated. Flags are set by the command `set(flag)` and cleared by `clear(flag)`.

`input_sos_first` If set, the input clauses (initial `sos` ) are selected as givens first and then the lightest clauses are chosen. Otherwise, the lightest clauses are always chosen first from `sos`.

`sos_queue` If set, `sos` operates as a queue.

`sos_stack` If set, `sos` operates as a stack.

`print_given` If set, clauses are output when they become given.

`binary_res` If set, binary resolution is available as an inference rule (as well as any others).

`hyper_res` If set, hyper-resolution is available as an inference rule.

`ur_res` If set, UR-resolution is available. This rule generates unit clauses (from other units and clauses).

`neg_hyper_res` If set, negative hyper-resolution is available. This is hyper-resolution in which the roles of positive and negative literals are reversed – all-negative clauses are electrons and positive literals in nucleii are resolved away. It can (more-or-less) be used to simulate Prolog execution. Try it and see what happens for a set of Horn clauses.

`for_sub` default is set; if set apply forward subsumption to newly generated clauses (delete clause if subsumed by a clause in `sos` or `usable`).

`unit_deletion` If set apply unit deletion to newly generated clauses; i.e. remove a literal if it is the negation of an instance of a unit clause in `sos` or `usable`.

`back_sub` default is set; if set apply back subsumption to newly generated clauses.

`factor` If set, factor newly kept clauses. All factors are generated. Safe factors replace the original.

`very_verbose` If set, lots of information is printed!

`print_kept` default is set; if set new clauses that are kept are output.

`print_proofs` default is set; if set print all proofs; if not set print no proofs.

`print_back_sub` default is set; if set output clauses that are backward subsumed.

## Some Parameters

`report` default is -1, range is $[-1..MAX-INT]$. If $n > 0$ then output statistics about every $n$ seconds . $n = 30$ is a good start.

`max_gen` default is -1, range is $[-1..MAX-INT]$. If $n$ is not -1 then terminate after $n$ clauses have been generated. There are other limits, see manual.

`max_proofs` default is 1, range is $[0..MAX-INT]$. If $n = 1$ OTTER will stop if it finds a proof. If $n > 1$ OTTER will search until it finds $n$ proofs. If $n = 0$ OTTER will find as many proofs as it can.

`stats_level` default is 2, range is $[0..3]$. This controls the level of detail of statistics pronted at the end. If $n = 0$ there are no statistics printed. If $n = 1$ a few important statistics are printed. If $n = 2$ most relevant statistics are printed. If $n = 3$ subsumption counts are output as well.

For using equality you will have to look at the manual. There are lots and lots of flags to set or clear.