Appendix to Chapter 2: Automating Classical Propositional Reasoning with OTTER (Introduction)

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

Example: Consider the following set of propositional formulas Σ

$$a \wedge b \wedge c \longrightarrow f \quad (1)$$

$$d \wedge e \longrightarrow f \quad (2)$$

$$a \quad (3)$$

$$b \quad (4)$$

$$c \quad (5)$$

$$d \quad (6)$$

$$e \quad (7)$$

We want to prove that $\Sigma \models f$; equivalently, that $\Sigma \cup \{\neg f\}$ is an inconsistent set of formulas

Otter transforms the initial set of formulas into a logically equivalent set of clauses, i.e. formulas of the form $l_1 \vee l_2 \vee \cdots \vee l_n$, where each l_i is a propositional variable or a negation of one

For example, formula (1) is transformed into: $\neg a \vee \neg b \vee \neg c \vee f$

- Binary resolution: The standard case of a resolution (deductive) step

  Two complementary literals are deleted, one from each clause, without restriction on the clauses

  For example:
  $$\begin{array}{r} p \vee \neg q \vee r \\ s \vee q \vee \neg t \\ - \; - \; - \; - \; - - \\ p \vee r \vee s \vee \neg t \end{array}$$
  (canceling the literals in red)

  It can be set in the Otter file

  Similarly, other forms of (resolution-based) deductive steps can be set or unset (see example below)

We invoke Otter with the following list of formulas:

```
set(binary_res). clear(unit_deletion). clear(factor).

formula_list(usable).
a & b & c -> f.
d & e -> f.
a.
b.
c.
d.
e.
end_of_list.

formula_list(sos). -f. end_of_list.
```

- Binary resolution is assumed by default by Otter

  Assuming or explicitly declaring binary resolution implies the automatic use by Otter of the rules “unit deletion” and “factor”

  In the example above, we do not want to use the last two, because, although O.K., they are “non standard”, “derived” deduction rules

  We want to illustrate standard behavior

  (In particular, with binary resolution only, proofs will be similar to those in logic programming)

  So, with set and clear different modes are turned on and off , resp.

- We want Otter to reach a contradiction

  The clauses have been split (by the user) into the two main sets, the usable and the set of support (`sos`)

- Intuitively, we are forcing Otter to start from the negation of what we want to prove, i.e. $\neg f$, that was placed in the sos

- The lists of formulas in `usable` and `sos` are transformed into clauses:

```
-------> usable clausifies to:
 list(usable).
  1 [] -a| -b| -c|f.
  2 [] -d| -e|f.
  3 [] a.    4 [] b.    5 [] c.
  6 [] d.    7 [] e.
  end_of_list.

-------> sos clausifies to:
 list(sos). 8 [] -f.
 end_of_list.
```

Corresponding to the formulas:

$$\neg a \vee \neg b \vee \neg c \vee f$$
$$\neg d \vee \neg e \vee f$$
$$a \quad b \quad c$$
$$d \quad e \quad \neg f$$

- Before describing the proof by Otter: clauses and literals have weights assigned by the user or by the system (by default)

  By default, Otter assigns weight 1 to each propositional variable; negative literals also get weight 1

  The relationship between the weights of complementary literals is given by the formula $weight(\neg p) = weight(p) + neg\_weight$

  Since the latter flag has not been specified by the user, it is given by default the value 0

The weight of a clause is the sum of the weights of the literals in it; then, e.g. the weight of clause $\neg a \vee \neg b \vee \neg c \vee f$ is 4

Intuitively, heavier clauses are less likely to be dragged into a resolution (lower priority)

In particular, this makes it easier to apply resolution to the “ideal case” of just two complementary literals

Refutation by Otter: (part of)

```
========== start of search ==========
given clause #1: (wt=1) 8 [] -f.
 ** KEPT (wt=2):
       9 [binary,8.1,2.3] -d| -e.
 ** KEPT (wt=3):
      10 [binary,8.1,1.4] -a| -b| -c.
 9 back subsumes 2.
 10 back subsumes 1.
```

In this first step Otter chooses the lightest clause from the sos list, i.e. $\neg f$

With element $\neg f$, all the possible resolutions with elements in the usable list will be performed, in this case, with the clauses 2 and 1 (in that order), generating the new elements:

$$9 \quad \neg d \vee \neg e$$
$$10 \quad \neg a \vee \neg b \vee \neg c$$

which get weights 2 and 3, resp.

These formulas go into the refreshed sos list (from which $\neg f$ is eliminated)

Before, continuing applying resolution, Otter discovers that these clauses imply (subsume) formulas 2 and 1, resp.

Then, 1 and 2 are eliminated from the usable list, which will now contain only 3 - 7

Now the lightest clause in the sos is chosen: 9

```
given clause #2: (wt=2)
         9 [binary,8.1,2.3] -d| -e.
** KEPT (wt=1):
        11 [binary,9.1,6.1] -e.

----> UNIT CONFLICT at   0.02 sec
   ----> 12 [binary,11.1,7.1] $F.
```

In this second step, Otter checks the new sos list, chooses the lightest clause, i.e. $\neg d \vee \neg e$ with weight 2, and applies resolution

It first resolves with d, obtaining $\neg e$, and it then realizes that a UNIT CONFLICT is produced with the element e: an application of binary resolution to the unit clauses (i.e. literals) e and $\neg e$ leads to the empty clause

The refutation has been successful and the process terminates

After this, Otter presents the complete proof, but pruning out those parts of the search and resolutions that are not needed in the successful refutation

```
---------------- PROOF ----------------
  2 [] -d| -e|f.
  6 [] d.
  7 [] e.
  8 [] -f.
  9 [binary,8.1,2.3] -d| -e.
 11 [binary,9.1,6.1] -e.
 12 [binary,11.1,7.1] $F.
------------ end of proof -------------
```

- Notice that not all the hypothesis were use in the proof

  This can be valuable information since the inconsistency can be “localized”

- Playing with weights can be useful to explore other regions of the search space

  This can be useful if different refutations need to be found, each of them giving different information, e.g. if we want to find possible and different diagnosis for the malfunctioning of a specified system

  Or to get all the answers to a query

  There may be different underlying contradictions between the specification of the system and a set of observations

  The default weights and default strategy will always lead to the same and only diagnosis

  Assigning adequate weights to the literals, we can make Otter follow a specific search strategy (e.g. the one of Prolog)

Example: Same as before, but now with non-default weights

In order to infer f, we want Otter to use the formula

$$a \wedge b \wedge c \longrightarrow f \tag{8}$$

rather than the formula

$$d \wedge e \longrightarrow f \tag{9}$$

We increase the weight of clause (9) wrt that of clause (8) by assigning weight 2 to variables d and e, and these new weights are to be considered when picking up clauses from the sos list

```
set(binary_res).
clear(unit_deletion).
clear(factor).
weight_list(pick_given).
weight(d,2).
weight(e,2).
end_of_list.
```

```
formula_list(usable).
a & b & c -> f.
d & e -> f.
a.   b.   c.   d.   e.
end_of_list.
formula_list(sos). -f.
end_of_list.
```

The preliminary transformations are the same

```
=========== start of search ===========
 given clause #1: (wt=1) 8 [] -f.
 ** KEPT (wt=4):
        9 [binary,8.1,2.3] -d| -e.
 ** KEPT (wt=3):
       10 [binary,8.1,1.4] -a| -b| -c.
 9 back subsumes 2. 10 back subsumes 1.
```

Otter chooses from the sos the lightest clause $\neg f$; next performs all resolutions between $\neg f$ and the clauses in the usable, in this case with clauses 2 and 1, generating

$$9 \quad \neg d \vee \neg e$$

$$10 \quad \neg a \vee \neg b \vee \neg c$$

with weights 4 and 3, resp., because

$$weight(\neg d \vee \neg e) = weight(\neg d) + weight(\neg e),$$
$$weight(\neg x) = weight(x) + neg\_weight$$
$$neg\_weight = 0 \quad \text{(by default)}$$

The execution so far has been as before, but different weights have been generated; which is relevant for the next step

9 and 10 are the new clauses in sos, from which $\neg f$ has been eliminated

The subsumed clauses 2 and 1 are eliminated from usable

```
given clause #2: (wt=3)
      10 [binary,8.1,1.4] -a| -b| -c.
** KEPT (wt=2):
      11 [binary,10.1,3.1] -b| -c.
** KEPT (wt=2):
      12 [binary,10.2,4.1] -a| -c.
** KEPT (wt=2):
      13 [binary,10.3,5.1] -a| -b.
11 back subsumes 10.
```

In the second step, Otter checks the sos and chooses the lighter clause 10, i.e. $\neg a \vee \neg b \vee \neg c$, and performs all possible resolutions with the set usable, i.e. with clauses 3, 4 and 5, generating the clauses:

$$11 \quad \neg b \vee \neg c$$
$$12 \quad \neg a \vee \neg c$$
$$13 \quad \neg a \vee \neg b,$$

which go, together with clause 9, into the new sos

In the third step, Otter checks sos again, searching for the lightest clause

Since there are three clauses with the lowest weight 2, Otter chooses according to the order in which they are stored, in this case, first 11

These are the steps originated from this element:

```
given clause #3: (wt=2)
   11 [binary,10.1,3.1] -b| -c.
** KEPT (wt=1):
   14 [binary,11.1,4.1] -c.

----> UNIT CONFLICT at   0.11 sec
    ----> 15 [binary,14.1,5.1] $F.
```

Otter performed a resolution with literal b, obtaining $\neg c$; and next, it discovers that there is a conflict between this literal and the element c in the usable list

The empty clause is reached and the refutation is finished; and the simplified version of the proof is returned:

```
---------------- PROOF ----------------

 1 [] -a| -b| -c|f.
 3 [] a.
 4 [] b.
 5 [] c.
 8 [] -f.
 10 [binary,8.1,1.4] -a| -b| -c.
 11 [binary,10.1,3.1] -b| -c.
 14 [binary,11.1,4.1] -c.
 15 [binary,14.1,5.1] $F.

------------ end of proof -------------
```

Other Examples and Issues:

- Using Unit Deletion

```
set(binary_res).
   dependent: set(unit_deletion).
% File given by the user
 list(usable).
 -p | -q | -r.
  p.   q.
 end_of_list.
 list(sos).
  r.
 end_of_list.
% Result returned by Otter
given clause #1: (wt=1)
  4 [] r.
** KEPT (wt=0):
  5 [binary,4.1,1.3,unit_del,2,3] $F.
-----> EMPTY CLAUSE at 0.00 sec
  ---->  5 [binary,4.1,1.3,unit_del,2,3] $F.
```

Unit resolution is applied to two clauses, and at least one of them is a unit clause (a single literal)

Unit deletion is applied to a resolvent (that will then go into the sos) from which one eliminates the literals that are complementary to literals in the usable or sos lists (because the same could be achieved by a sequence of unary resolutions, but making things longer)

- Elimination of Unit Deletion (same example)

```
 set(binary_res). clear(unit_deletion).
% Result returned by Otter
 given clause #1: (wt=1)
      4 [] r.
 ** KEPT (wt=2):
      5 [binary,4.1,1.3] -p| -q.
 5 back subsumes 1.

 given clause #2: (wt=2)
      5 [binary,4.1,1.3] -p| -q.
 ** KEPT (wt=1):
      6 [binary,5.1,2.1] -q.
----> UNIT CONFLICT at 0.02 sec
  ----> 7 [binary,6.1,3.1] $F.
```

- Elimination of repeated literals

```
 list(usable).
  q | p | r.
 end_of_list.
 list(sos).
  p | -q.
 end_of_list.

% Part of result returned by Otter
given clause #1: (wt=2)
  2 [] p| -q. ** KEPT (wt=2):
  3 [binary,2.2,1.1,factor_simp] p|r.
```

- Elimination of a clause for having too many literals

```
 list(usable).
 a | -b | -g.
 b | -c | -h.
 c | -d | -i.
 d | -e | -j.
 e | -f | -k.  end_of_list.
list(sos).
 -a.
end_of_list.
% Result returned if max_literals
% has not been defined
given clause #1: (wt=1)
  6 [] -a. ** KEPT (wt=2):
  7 [binary,6.1,1.1] -b| -g.
        7 back subsumes 1.
given clause #2: (wt=2)
  7 [binary,6.1,1.1] -b| -g.
** KEPT (wt=3):
  8 [binary,7.1,2.1] -g| -c| -h.
given clause #3: (wt=3)
  8 [binary,7.1,2.1] -g| -c| -h.
```

```
** KEPT (wt=4):
  9 [binary,8.2,3.1] -g| -h| -d| -i.
given clause #4: (wt=4)
  9 [binary,8.2,3.1] -g| -h| -d| -i.
** KEPT (wt=5):
 10 [binary,9.3,4.1]
          -g| -h| -i| -e| -j.
given clause #5: (wt=5)
 10 [binary,9.3,4.1]
          -g| -h| -i| -e| -j.
** KEPT (wt=6):
 11 [binary,10.4,5.1]
          -g| -h| -i| -j| -f| -k.
given clause #6: (wt=6)
 11 [binary,10.4,5.1]
          -g| -h| -i| -j| -f|-k.

% Result returned with max_literals=4
assign(max_literals,4).

given clause #1: (wt=1)
  6 [] -a.
** KEPT (wt=2):
```

```
  7 [binary,6.1,1.1] -b| -g.
      7 back subsumes 1.
given clause #2: (wt=2)
  7 [binary,6.1,1.1] -b| -g.
** KEPT (wt=3):
  8 [binary,7.1,2.1] -g| -c| -h.
given clause #3: (wt=3)
  8 [binary,7.1,2.1] -g| -c| -h.
** KEPT (wt=4):
  9 [binary,8.2,3.1] -g| -h| -d| -i.
given clause #4: (wt=4)
  9 [binary,8.2,3.1] -g| -h| -d| -i.
```

- Elimination of tautologies

```
 list(usable).   a | -b .    end_of_list.
 list(sos).  b | -a.  end_of_list.

% Result returned by Otter
given clause #1: (wt=2)
    2 [] b| -a.
Search stopped because sos empty.
```

- Elimination of formulas implied by clauses in sos or usable

```
 list(usable).
  -a|b|c.
   b.
 end_of_list.
 list(sos).
  a.
 end_of_list.

% Result returned by Otter
given clause #1: (wt=1)
     3 [] a.
Search stopped because sos empty.
```

- Specification of weights by the user

  The weight_list(pick_given) is the main specification of weight; when weights are printed, they refer to these weights

  With weight_list(purge_gen) one can specify weight for variables and they will be considered when deciding if a resolvent is going to be kept; heavy resolvents do not go into the sos

  pick_given- and purge_gen - weights may declared different for the same variable

  With weight_list(pick_and_purge) the same weight can be assigned to a variable for the two purposes

```
set(binary_res).
assign(max_weight,4).
weight_list(pick_given).
 weight(a,2).
 weight(b,1).
end_of_list.

weight_list(purge_gen).
 weight(c,3).
 weight(d,3).
end_of_list.

list(usable).
 -b|c|d.
 -a.
end_of_list.
list(sos).
 a.   b.
end_of_list.
```

```
% Answer by Otter
 list(usable).
  1 [] -b|c|d.
  2 [] -a.
 end_of_list.
 list(sos).
  3 [] a.
  4 [] b.
 end_of_list.
====== end of input processing ======
========== start of search ==========
given clause #1: (wt=1)
       4 [] b.
given clause #2: (wt=2)
       3 [] a.
** KEPT (wt=0):
       5 [binary,3.1,2.1] $F.

-----> EMPTY CLAUSE at 0.10 sec
   ----> 5 [binary,3.1,2.1] $F.

Length of proof is 0.
            Level of proof is 0.
```

```
---------------- PROOF -------------
 2 [] -a.
 3 [] a.
 5 [binary,3.1,2.1] $F.

------------ end of proof ----------
```