Temporal Resolution

[Loop Search]

Michael Fisher

Department of Computer Science, University of Liverpool, UK

[MFisher@liverpool.ac.uk]

An Introduction to Practical Formal Methods Using Temporal Logic
Review: Resolution Method

To check a PTL formula, say $A$, for unsatisfiability we perform the following steps.

1. *Transform* $A$ into SNF, giving a set of clauses $A_S$.

2. Perform *step resolution* on clauses from $A_S$ until either:
   (a) a contradiction is derived, in which case $A$ is *unsatisfiable*; or
   (b) no new resolvents are generated, in which case we continue to step (3).

3. Select an eventuality from within $A_S$, and perform *temporal resolution* with respect to this — if any new formulae are generated, go back to step (2).

4. If all eventualities have been resolved, then $A$ is *satisfiable*, otherwise go back to step (3).
Merged SNF

To apply the temporal resolution rule, it is often convenient to combine one or more step clauses. Consequently, a variant on SNF called *merged-SNF (or SNF*$_m$*)*, can also be defined.

Any SNF step clause is also an SNF*$_m$* clause. Any two SNF*$_m$* clauses in SNF*$_m$* may be combined to produce a further SNF*$_m$* clause as follows.

\[
L_1 \Rightarrow \bigcirc R_1 \\
L_2 \Rightarrow \bigcirc R_2 \\
(L_1 \land L_2) \Rightarrow \bigcirc (R_1 \land R_2)
\]

Thus, any possible conjunctive combination of SNF step-clauses can be represented in SNF*$_m$*.
So the full temporal resolution operation essentially applies between a *sometime* clause and a *set* of $\text{SNF}_m$ clauses that together imply $A \Rightarrow \bigcirc \: \square \neg l$:

\[
\begin{align*}
L_0 & \Rightarrow \bigcirc R_0 \\
\ldots & \Rightarrow \ldots \\
L_n & \Rightarrow \bigcirc R_n \\
C & \Rightarrow \lozenge l
\end{align*}
\]

\[
C \Rightarrow \left[ \bigwedge_{i=0}^{n} \neg L_i \right] W l
\]

again, with the various side conditions on $R_j$. 
Loops

The set of $\text{SNF}_m$ PTL-clauses $L_i \Rightarrow \bigcirc R_i$ that satisfy these side conditions are together known as

\[ a \text{ loop in } \neg l. \]

The disjunction of the left hand side of this set of $\text{SNF}_m$ PTL-clauses, i.e.

\[ \bigvee_{i} L_i \]

is known as a

\[ \text{loop formula for } \neg l. \]

Once $\bigvee_{i} L_i$ is satisfied, the loop is guaranteed.
Aside: Temporal Resolvent

The resolvent is of the form $C \Rightarrow (\neg A)\ W\ \!l$.

Generating $C \Rightarrow (\neg A)\ U\!l$ as a resolvent would also be sound.

However as

$$\neg A)\ U\!l \equiv ((\neg A)\ W\!l) \land \lozenge \!l$$

the resolvent

$$C \Rightarrow (\neg A)\ U\!l$$

would be equivalent to the pair of resolvents

$$C \Rightarrow (\neg A)\ W\!l \quad \text{and} \quad C \Rightarrow \lozenge \!l.$$  

The latter is subsumed by the sometime PTL-clause we have resolved with. So this leaves only the ‘$\mathcal{W}$’ formula.
Example (1)

Assume we wish to show that the following set of PTL-clauses (already translated into SNF) is unsatisfiable.

1. \( \text{start} \implies f \)
2. \( \text{start} \implies a \)
3. \( \text{start} \implies p \)
4. \( f \implies \Diamond \neg p \)
5. \( f \implies 
\Box a \)
6. \( a \implies 
\Box (b \lor x) \)
7. \( b \implies 
\Box a \)
8. \( b \implies 
\Box p \)
9. \( a \implies 
\Box p \)
10. \( a \implies 
\Box \neg x \)
Example (2)

Step resolution occurs as follows.

11. \( a \Rightarrow \Diamond b \) \hspace{1cm} [6, 10 \text{ Step Resolution}]

By merging PTL-clauses (9) and (11), and (7) and (8) into \( \text{SNF}_m \) we obtain the following loop in \( p \) (in \( \text{SNF}_m \) form)

\[
\begin{align*}
    a & \Rightarrow \Diamond (b \land p) \hspace{1cm} [9, 11 \text{ SNF}_m] \\
    b & \Rightarrow \Diamond (a \land p) \hspace{1cm} [7, 8 \text{ SNF}_m]
\end{align*}
\]

for resolution with PTL-clause (4).

Effectively, this gives us \( (a \lor b) \Rightarrow \Diamond \Box p \).

The resolvents after temporal resolution are PTL-clauses 12–20 below (expansion of \( f \Rightarrow (\neg a \land \neg b) \Diamond \neg p \)).
Example (3)

12. $\textbf{start} \Rightarrow \neg f \vee \neg p \vee \neg a \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
13. $\textbf{true} \Rightarrow \Box (\neg f \vee \neg p \vee \neg a) \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
14. $\textbf{start} \Rightarrow \neg f \vee \neg p \vee \neg b \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
15. $\textbf{true} \Rightarrow \Box (\neg f \vee \neg p \vee \neg b) \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
16. $\textbf{start} \Rightarrow \neg f \vee \neg p \vee t \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
17. $\textbf{true} \Rightarrow \Box (\neg f \vee \neg p \vee t) \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
18. $t \Rightarrow \Box (\neg p \vee \neg a) \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
19. $t \Rightarrow \Box (\neg p \vee \neg b) \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$
20. $t \Rightarrow \Box (l \vee t) \quad [4, 7, 8, 9, 11 \text{ Temp. Res.}]$

where $t$ is a new proposition. Then proceed with:

21. $\textbf{start} \Rightarrow \neg f \vee \neg a \quad [3, 12 \text{ Initial Resolution}]$
22. $\textbf{start} \Rightarrow \neg f \quad [2, 21 \text{ Initial Resolution}]$
23. $\textbf{start} \Rightarrow \text{false} \quad [1, 22 \text{ Initial Resolution}]$
Once we have applied the clausal temporal resolution rule, the resolvent must be translated into SNF before any further resolution steps are applied. Recall that formulae such as

\[ C \Rightarrow ( \bigwedge_{i=0}^{n} \neg L_i ) \ W \]

are transformed into “\( C \Rightarrow w \)” where the new proposition ‘\( w \)’ is constrained by

\[ \square ( w \Rightarrow ( l \lor ( \bigwedge_{i=0}^{n} \neg L_i ) ) ) \]

\[ \square ( w \Rightarrow ( l \lor \Box w ) ) \]
Resolvents (2)

However, a translation to the normal form is given below that avoids the renaming of the sub-formula \( \bigwedge_{i=0}^{n} \neg L_i \) where \( t \) is a new proposition symbol and \( i = 0, \ldots, n \).

Thus, for each of the PTL-clauses (1), (2) and (5) there are \( n + 1 \) copies, one for each \( L_i \).

\[
\begin{align*}
\text{start} &\Rightarrow \neg C \lor l \lor \neg L_i & (1) \\
\text{true} &\Rightarrow \Box (\neg C \lor l \lor \neg L_i) & (2) \\
\text{start} &\Rightarrow \neg C \lor l \lor t & (3) \\
\text{true} &\Rightarrow \Box (\neg C \lor l \lor t) & (4) \\
t &\Rightarrow \Box (l \lor \neg L_i) & (5) \\
t &\Rightarrow \Box (l \lor t) & (6)
\end{align*}
\]
Resolvents (3)

1. **start** ⇒ \( \neg C \lor I \lor \neg L_i \)
2. **true** ⇒ \( \bigcirc (\neg C \lor I \lor \neg L_i) \)
3. **start** ⇒ \( \neg C \lor I \lor t \)
4. **true** ⇒ \( \bigcirc (\neg C \lor I \lor t) \)
5. \( t \) ⇒ \( \bigcirc (I \lor \neg L_i) \)
6. \( t \) ⇒ \( \bigcirc (I \lor t) \)

Only resolvents (1), (2) and (5) depend on the loop being resolved with, i.e. contain a reference to \( L_i \).

Clauses (3), (4) and (6) do not depend on the loop and are relevant for any resolution with \( C \Rightarrow \diamondsuit I \).

So, typically, (3), (4) and (6) are added earlier in the resolution process.
Example

Resolving \((a \lor b \lor c) \Rightarrow \Box \neg m\) and \(d \Rightarrow \Diamond m\) to give \(d \Rightarrow (\neg a \land \neg b \land \neg c) W m\)

rewriting this (\(x\) is new) gives:

1. **start** \(\Rightarrow \neg d \lor m \lor \neg a\)
2. **start** \(\Rightarrow \neg d \lor m \lor \neg b\)
3. **start** \(\Rightarrow \neg d \lor m \lor \neg c\)
4. **true** \(\Rightarrow \Box (\neg d \lor m \lor \neg a)\)
5. **true** \(\Rightarrow \Box (\neg d \lor m \lor \neg b)\)
6. **true** \(\Rightarrow \Box (\neg d \lor m \lor \neg c)\)
7. **start** \(\Rightarrow \neg d \lor m \lor x\)
8. **true** \(\Rightarrow \Box (\neg d \lor m \lor x)\)
9. \(x\) \(\Rightarrow \Box (m \lor \neg a)\)
10. \(x\) \(\Rightarrow \Box (m \lor \neg b)\)
11. \(x\) \(\Rightarrow \Box (m \lor \neg c)\)
12. \(x\) \(\Rightarrow \Box (m \lor x)\)
Basic Loop Search (1)

We have the following set of clauses and want to find a loop in $\neg r$, e.g. to resolve with $C \Rightarrow \Diamond r$.

1. $f \Rightarrow \Box \neg r$
2. $(q \land \neg r) \Rightarrow \Box \neg r$
3. $q \Rightarrow \Box p$
4. $p \Rightarrow \Box (f \lor q)$
5. $p \Rightarrow \Box \neg r$
6. $p \Rightarrow \Box g$
7. $(f \land g) \Rightarrow \Box (p \lor q)$

Consider 3 possible propositional configurations:

- config1 . . . . . . . . $f$ and $g$ are true, but $r$ is false;
- config2 . . . . . . . . $q$ is true, but $r$ is false; and
- config3 . . . . . . . . $p$ is true, but $r$ is false.
Basic Loop Search (2)

Now, assume we begin in config1.

Clauses (1) and (7) show us that we either move to config2 or config3. If we are in config2, then clauses (2) and (3) show us that we next move to config3. If we are in config3, then clauses (4), (5), and (6) force us to move to either config1 or config2. Thus, once we are in any of these configurations, we remain within them forever.

1. \( f \Rightarrow \Diamond \neg r \)
2. \( (q \land \neg r) \Rightarrow \Diamond \neg r \)
3. \( q \Rightarrow \Diamond p \)
4. \( p \Rightarrow \Diamond (f \lor q) \)
5. \( p \Rightarrow \Diamond \neg r \)
6. \( p \Rightarrow \Diamond g \)
7. \( (f \land g) \Rightarrow \Diamond (p \lor q) \)
Basic Loop Search (3)

$\text{SNF}_m$ is useful for representing transitions between such configurations. Merging clauses (1) and (7) gives

$$(f \land g) \Rightarrow \Box (\neg r \land (p \lor q))$$

merging clauses (2) and (3) gives

$$(q \land \neg r) \Rightarrow \Box (p \land \neg r)$$

while merging clauses (4), (5) and (6) gives

$$p \Rightarrow \Box ((g \land f \land \neg r) \lor (g \land q \land \neg r))$$.

Since whichever node we are in, then $r$ must be false, then the above clauses effectively represent

$$((f \land g \land \neg r) \lor (q \land \neg r) \lor (p \land \neg r)) \Rightarrow \Box \square \neg r$$
Basic Loop Search (4)

So, this gives us a simple way to find appropriate loop formulae:

1. combine all SNF step clauses together to get a large set of SNF\(_m\) clauses;
2. represent the SNF\(_m\) clauses as transitions on a graph such as above;
3. search the graph for strongly connected components throughout which our target literal is true.

A strongly connected component is a sub-graph where each transition leads to other nodes in the sub-graph.

In the case above, our target literal is “\(\neg r\)” and we see that the diagram above describes a strongly connected component in which the target literal occurs at every node.
Dixon’s Breadth-First Search (1)

While the use of the above SNF\(_m\) based search is possible, it is inefficient and unwieldy.

Although searching for strongly connected components will be quite quick, the number of potential SNF\(_m\) clauses, and hence the graph, is huge.

An alternative approach, and one that has led to the mechanisms used in contemporary clausal temporal resolution systems, was developed by Clare Dixon.

Rather than giving the full algorithm, we will simply provide some intuition of how this approach works by tackling the above problem.

We begin by recognising that, since we are looking for \(\bigcirc \square \neg r\), the literal we need at every next state is \(\neg r\).
So, let us identify the SNF clauses within our original set that imply $\Diamond \neg r$, namely:

1. $f \Rightarrow \Diamond \neg r$
2. $(q \land \neg r) \Rightarrow \Diamond \neg r$
5. $p \Rightarrow \Diamond \neg r$

Thus, for $\neg r$ to be satisfied in the next state, we require

$$f \lor (q \land \neg r) \lor p$$

to be satisfied in the current state. Let us describe this graphically, as follows.

$$f \lor (q \land \neg r) \lor p$$
Dixon’s Breadth-First Search (3)

So, if we are to continue in a loop we know that we need one of $f$, $q \land \neg r$, or $p$ to be satisfied.

In addition, we still need $\neg r$ to be satisfied.

So, how can we generate either $f \land \neg r$, $q \land \neg r$, or $p \land \neg r$?

Well, we can examine SNF$_m$ combinations to see if we can achieve these. Specifically $f \land \neg r$, $q \land \neg r$, or $p \land \neg r$ all occur on the right-hand sides of the following combinations:

\[
\begin{align*}
2 + 3 & : \quad (q \land \neg r) \Rightarrow \quad \Box (\neg r \land p) \\
4 + 5 + 6 & : \quad p \Rightarrow \quad \Box ((f \land g \land \neg r) \lor (q \land g \land \neg r)) \\
1 + 7 & : \quad (f \land g) \Rightarrow \quad \Box (\neg r \land (p \lor q))
\end{align*}
\]
So, we extend our graph:

$$f \lor (q \land \neg r) \lor p$$

We continue now with the same procedure, checking what clauses together imply the above disjunctions while also implying $\neg r$. 
We again construct another node:

\[ f \lor (q \land \neg r) \lor p \]

\[ (f \land g) \lor (q \land \neg r) \lor p \]

\[ (f \land g) \lor (q \land \neg r) \lor p \]

Once we have seen the same node twice we can stop, having detected a loop. Thus, if any of \( f \land g \), \( q \land \neg r \), or \( p \) occur then \( \lozenge \Box \neg r \) is guaranteed.
Consequently, we can apply temporal resolution to

\[
\begin{align*}
\text{true} & \Rightarrow \lozenge r \\
((f \land g) \lor (q \land \neg r) \lor p) & \Rightarrow \lozenge \square \neg r
\end{align*}
\]

Dixon’s BFS has been defined, proved correct and implemented.

It carries out a *directed* search for clauses that are truly relevant, rather than looking at all combinations of clauses.

The whole process is guaranteed to terminate, and often requires only a few nodes to be constructed before an appropriate loop is found.
Dixon’s BFS is used in most contemporary clausal temporal resolution provers.

Crucially, most of the steps in identifying nodes, generating combinations and recognising termination could be carried out by utilising classical automated reasoning systems.

Thus, the Otter system was used for this, and so clausal temporal resolution can now be implemented directly on top of a classical resolution system.

We will see that TSPASS is built on top of the SPASS automated reasoning system.

Note: as PTL is significantly more complex than classical propositional logic, quite a lot of extra work must potentially be carried out within the classical reasoning system.