Temporal Resolution

[Simplified Approach]

Michael Fisher

Department of Computer Science, University of Liverpool, UK

[MFisher@liverpool.ac.uk]
Making it Simpler

In clausal temporal resolution, the key inference rule is the temporal resolution rule which, as we have seen, generates quite a complex resolvent.

It turns out that, if we take a simplified original problem then we do not need such a complex clausal temporal resolution rule. So, if an SNF problem

1. has only one eventuality (the single eventuality case), and

2. the sometime SNF-clause that contains that eventuality only has true on its left-hand side (the unconditional eventuality case),

then we only need a simplified clausal temporal resolution rule.
Simplified Temporal Rule

\[\begin{align*}
L_0 & \Rightarrow \bigcirc R_0 \\
\ldots & \Rightarrow \ldots \\
L_n & \Rightarrow \bigcirc R_n \\
\text{true} & \Rightarrow \Diamond l \\
\text{true} & \Rightarrow \bigwedge_{i=0}^{n} \neg L_i
\end{align*}\]

Notice, here, that the resolvent is actually just a non-temporal formula!
Simplified Example

Let us begin with the simple set of SNF clauses

1. \( a \Rightarrow \Diamond b \)
2. \( a \Rightarrow \Diamond \neg m \)
3. \( b \Rightarrow \Diamond a \)
4. \( b \Rightarrow \Diamond \neg m \)
5. \( \text{true} \Rightarrow \Diamond m \)

We can see that one loop is caused by “\( a \lor b \)”, i.e:

6. \( (a \lor b) \Rightarrow \Diamond \Box \neg m \).

Applying the simplified temporal resolution rule between (5) and (6) now gives us

7. \( \text{true} \Rightarrow \neg a \lor \neg b \).
Beyond SNF (1)

The fact that we above deal with only sometime-clauses of the form “true ⇒ ♦ . . .” and generate resolvents that are purely classical (i.e. not temporal) leads us to use an alternative (but equivalent) formulation of the normal form.

This is used not only in more recent papers on clausal temporal resolution, but has been found to be particularly useful in developing clausal resolution for FOTL and is the basis of modern clausal resolution provers for PTL such as TSPASS.

Thus, a temporal problem is comprised of four sets of formulae, $\mathcal{I}$, $\mathcal{U}$, $\mathcal{E}$, and $\mathcal{S}$. 
A temporal problem is comprised of four sets of formulae, $I$, $U$, $E$, and $S$, where

$I$ represents the initial part of the problem, and contains non-temporal formulae that should be satisfied at the first moment in time

$\rightarrow$ where previously we had “start $\Rightarrow p$” we would just add “$p$” to $I$.

$U$ represents the universal part of the problem, and contains non-temporal formulae that are universally true

$\rightarrow$ if ever we want to say that some (non-temporal) property is globally true, e.g. “true $\Rightarrow (q \lor r)$”, then we just add “$q \lor r$” to $U$. 
Beyond SNF (3)

\( \mathcal{E} \) is the *sometime part* of the problem, containing eventualities that must be satisfied infinitely often → since our sometime-clauses will now be of the form “\( \text{true} \Rightarrow \Diamond s \)” we just store “\( \Diamond s \)” in \( \mathcal{E} \).

\( \mathcal{S} \) is the *step part* of the problem, containing step clauses → each element of \( \mathcal{S} \) is an SNF step clause, e.g.

“\( (a \land b) \Rightarrow \lozenge (c \lor d) \)”.

So, the full temporal problem is characterised by the formula

\[
\bigwedge \mathcal{I} \land \Box \bigwedge \mathcal{U} \land \Box \bigwedge \mathcal{S} \land \Box \bigwedge \mathcal{E}
\]

where “\( \bigwedge \)” just means we conjoin together all elements of the subsequent set.
Example

1. `start` \(\Rightarrow\) \(x_1\)
2. `true` \(\Rightarrow\) \(\Diamond p\)
3. \(x_3\) \(\Rightarrow\) \(\Box x_3\)
4. \((x_2 \land p)\) \(\Rightarrow\) \(\Box p\)
5. `true` \(\Rightarrow\) \(\neg x_1 \lor x_3\)
6. `true` \(\Rightarrow\) \(\Box (\neg x_1 \lor x_3)\)
7. `true` \(\Rightarrow\) \(\neg x_3 \lor x_2\)
8. `true` \(\Rightarrow\) \(\Box (\neg x_3 \lor x_2)\)
9. `true` \(\Rightarrow\) \(\Diamond \neg p\)
10. \(x_5\) \(\Rightarrow\) \(\Box x_5\)
11. `true` \(\Rightarrow\) \(\neg x_1 \lor x_5\)
12. `true` \(\Rightarrow\) \(\Box (\neg x_1 \lor x_5)\)
13. `true` \(\Rightarrow\) \(\neg x_5 \lor x_4\)
14. `true` \(\Rightarrow\) \(\Box (\neg x_5 \lor x_4)\)
Temporal Problem Representation

\( \mathcal{I} = \{x_1\} \)

\( \mathcal{U} = \{\neg x_1 \lor x_3, \neg x_3 \lor x_2, \neg x_1 \lor x_5, \neg x_5 \lor x_4\} \)

\( \mathcal{E} = \{\lozenge \neg p, \lozenge p\} \)

\( \mathcal{S} = \{x_3 \Rightarrow \lozenge x_3, (x_2 \land p) \Rightarrow \lozenge p, x_5 \Rightarrow \lozenge x_5\} \)

The two representations are equivalent, but the latter is clearly more concise.
Applying Simplified Resolution

If we look at the new clausal temporal resolution rule introduced above, we can see why this problem form is convenient.

In applying clausal temporal resolution, we
- take several elements from $S$,
- one element from $E$ (there may only be one) and
- generate a new element in $U$.

With this style of calculus we are required to carry out quite a lot of proof within the non-temporal sets $I$ and $U$, which explains why modern clausal temporal resolution provers are based on classical tools, such as Otter, Vampire and SPASS.
As stated above, if our problem has only one eventuality (the *single eventuality* case) and the sometime SNF-clause that contains that eventuality only has “true” on its left-hand side (the *unconditional eventuality* case), then we can use a simplified clausal resolution rule.

Happily, we can transform *any* SNF problem (and, thus, any PTL problem) into this unconditional, single eventuality case:

1. moving from *conditional* to *unconditional* eventualities; and
2. moving from *many* eventualities to *one*.

Let us see how.
Conditional to Unconditional

Recall what $p \Rightarrow \Diamond q$ means.

Whenever $p$ becomes true then we can see a sequence of states in which $q$ is false before we see a state where $q$ is true (and we will definitely see such a state).

So, consider the following SNF clauses incorporating the new proposition $\text{wait\_for\_q}$.

\[
(p \land \neg q) \Rightarrow \text{wait\_for\_q} \\
\text{wait\_for\_q} \Rightarrow \Box (q \lor \text{wait\_for\_q}) \\
\text{true} \Rightarrow \Diamond \neg \text{wait\_for\_q}
\]

It turns out that the above clauses are satisfied in exactly the same situations that $p \Rightarrow \Diamond q$ is satisfied at, and so we can replace $p \Rightarrow \Diamond q$ by the above.
Further Intuition

\[(p \land \neg q) \Rightarrow \text{wait}_\text{for}_\text{q}\]

\[\text{wait}_\text{for}_\text{q} \Rightarrow \bigcirc(q \lor \text{wait}_\text{for}_\text{q})\]

\[\text{true} \Rightarrow \Diamond \neg \text{wait}_\text{for}_\text{q}\]

states that \text{wait}_\text{for}_\text{q} must be \textit{false} infinitely often.

To ensure \text{wait}_\text{for}_\text{q} is false then, either \(p\) has not occurred or, if it has, \(q\) has subsequently been satisfied.

The only way for \(\Box(\text{true} \Rightarrow \Diamond \neg \text{wait}_\text{for}_\text{q})\) to fail is if, after some point, \text{wait}_\text{for}_\text{q} is always true. But for \text{wait}_\text{for}_\text{q} to be true, we must have had a \(p\) but not yet reached a \(q\).

As we originally had \(\Box(p \Rightarrow \Diamond q)\), then this situation can’t occur and so \text{wait}_\text{for}_\text{q} can’t go on being true forever.
From Many Eventualities to One

Now, how do we go from a problem with many eventualities to a problem with only one?

Basic idea: given several eventualities, for example $\Diamond a$, $\Diamond b$, and $\Diamond c$, then there will be a point where $a$, $b$ and $c$ have all been satisfied.

We just need to make sure that this point is eventually reached.

If we require that $\Diamond a$, $\Diamond b$, and $\Diamond c$ keep on occurring, for example because we have $\text{true} \Rightarrow \Diamond a$, $\text{true} \Rightarrow \Diamond b$, and $\text{true} \Rightarrow \Diamond c$ then our “success point” must keep on occurring (infinitely often).
Example (1)

Let us assume that we have the step clauses

\[ \text{true} \Rightarrow \Diamond a \quad \text{true} \Rightarrow \Diamond b \quad \text{true} \Rightarrow \Diamond c \]

and let us define a new proposition, “success”, that describes a point where \( a \), \( b \) and \( c \) have occurred. Then consider the following formulae.

\[
\begin{align*}
\text{start} & \Rightarrow \text{success} \\
\text{success} & \Rightarrow \Diamond \bigg( (\neg \text{success}S_a) \land (\neg \text{success}S_b) \land (\neg \text{success}S_c) \land \text{success} \bigg)
\end{align*}
\]

Note: \( fSg \) means that \( g \) occurred sometime in the past and at all moments between that point and now (but not including either one) then \( f \) must hold.
Example (2)

\[
\text{success} \Rightarrow \Box \Diamond \left( (\neg \text{success} S a) \land (\neg \text{success} S b) \land (\neg \text{success} S c) \land \text{success} \right)
\]

ensures that there is some point in the future where

- \text{success} is satisfied, yet
- \text{success} has not been satisfied since \(a\) became true,
- \text{success} has not been satisfied since \(b\) became true, and
- \text{success} has not been satisfied since \(c\) became true.

\text{success}, marks a point when all of ‘\(a\)’, ‘\(b\)’ and ‘\(c\)’ have occurred; this continues infinitely often in the future:

\[\begin{array}{cccccc}
\bullet & & & & & \\
\ldots & \cdot & \cdot & \cdot & \cdot & \cdot \\
\text{success} & \text{~success} & \text{~success} & \text{~success} & \text{~success} & \text{success} \\
\end{array}\]

\[\begin{array}{cccccc}
\bullet & & & & & \\
\ldots & \cdot & \cdot & \cdot & \cdot & \cdot \\
\neg \text{success} S a & \neg \text{success} S b & \neg \text{success} S c & \end{array}\]
Aside: Translating “$S$”

The fact that we now have past-time operators in the above example might appear problematic, but such operators are easily translated into the normal form as follows.

Consider “$x \Rightarrow (pSq)$”. We translate this into the following set of SNF clauses (again requiring a little further rewriting), introducing a new proposition, in this case “$s$”.

\[
\begin{align*}
\text{start} & \Rightarrow \neg s \\
 x & \Rightarrow s \\
 ((q \lor (p \land s)) & \Rightarrow \bigcirc s \\
 \neg((q \lor (p \land s)) & \Rightarrow \bigcirc \neg s
\end{align*}
\]
Let us begin with “start ⇒ (◊a ∧ ◊b ∧ ◊c)” and reduce to one eventuality. Similar to above, we can first introduce ‘success’ and generate

\[
\begin{align*}
\text{success} & \Rightarrow \neg\text{success}\text{Sa} \\
\text{success} & \Rightarrow \neg\text{success}\text{Sb} \\
\text{success} & \Rightarrow \neg\text{success}\text{Sc} \\
\text{start} & \Rightarrow \Diamond\text{success}
\end{align*}
\]

\(\neg\text{success}\text{Sb}\) means that ‘b’ has occurred in the past and, since that point, success has always been false. The above ensure that, once success becomes true, it marks the first time that a, b, and c have all been true in the past. Now all we do is say that, at some point in the future, success must become true. Thus, we can infer that a, b, and c have all become true by then.
We can use SPASS, or any reasonable resolution-based system, to carry out loop search as well as more obvious resolution operations. The basic idea is as follows.

At each step in the loop search algorithm, we can generate classical (often propositional) logic clauses that can then be fed to our classical prover.

In searching for loops, e.g. $\Diamond \Box \neg r$, we begin by searching for clauses that imply $\Diamond \neg r$. 
Loop Search using SPASS (2)

We can translate all the SNF step clauses, e.g:

\[ f \Rightarrow \Box \neg r \] becomes \[ f \Rightarrow \neg \text{next}_r \]
\[ (q \land \neg r) \Rightarrow \Box \neg r \] becomes \[ (q \land \neg r) \Rightarrow \neg \text{next}_r \]
\[ p \Rightarrow \Box \neg r \] becomes \[ \Rightarrow \neg \text{next}_r \]

To search for clauses that imply \( \Box \neg r \), we add the clause \( \text{found} \Rightarrow \text{next}_r \) and pass all of these to our classical prover.

If we can derive a clause of the form “\( P \lor \neg \text{found} \)” containing no “\( \text{next}_r \)” literals then the SNF clause “\( P \Rightarrow \Box \neg r \)” exists.

And so on.

**TSPASS** works in a more sophisticated way, but still essentially translates the loop search process into a refutation process within **SPASS**.