Temporal Resolution

[Introducing Temporal Proof]

Michael Fisher

Department of Computer Science, University of Liverpool, UK

[MFisher@liverpool.ac.uk]
Temporal Proof

As we have seen, we often want to check whether one temporal formula implies another.

For example, if we have specified a system using the temporal formula, $\psi$, then we can check whether a certain temporal property, say $\varphi$, follows from the specification of the system by establishing that $\varphi$ is implied by $\psi$ i.e., whether

$$\vdash \psi \Rightarrow \varphi$$

Specifically, by proving that one temporal formula implies another.

There is a range mechanisms for deciding proof problems in temporal logic, but our main technique will be a variety of clausal resolution for PTL.
Resolution was first proposed as a proof procedure for classical logics by Robinson in the 1960s.

It was deemed particularly suitable for proofs to be performed by computer, having only one rule of inference that could be applied many times.

Using this approach the validity of a logical formula, \( \varphi \), is checked by first negating it and translating \( \neg \varphi \) into a particular form, _Conjunctive Normal Form_ (CNF):

\[ C_1 \land C_2 \land \ldots \land C_m \]

Here, each \( C_i \), known as a _clause_, is a disjunction of literals and each _literal_ is either a proposition or its negation. Typically, a CNF formula is represented as a set of clauses

\[ \mathcal{C} = \{ C_1, C_2, \ldots, C_m \} \]
The resolution inference rule is then applied repeatedly to the set of CNF clauses derived from $\neg \varphi$ and new inferences ($D_1 \lor D_2$, in the case below) are added to the set ($D_1$ and $D_2$ are disjunctions of literals and $p$ is a proposition):

$$\begin{align*}
D_1 \lor p \\
D_2 \lor \neg p \\
\hline
D_1 \lor D_2
\end{align*}$$

The intuition here is that, since one of $p$ or $\neg p$ must be false, then one of $D_1$ or $D_2$ must be true.

The resolution rule is applied to pairs of clauses from $C$, with new resolvents being added back into $C$, until either an empty resolvent (denoting $\text{false}$) is derived or no further resolvents can be generated.
Refutation

Typically, given some formula, \( \varphi \), that we wish to decide the validity of, we apply the resolution process to \( \neg \varphi \).

If a contradiction (false) is derived then \( \neg \varphi \) is unsatisfiable and so the original formula \( \varphi \) must therefore be valid.

This process of determining the unsatisfiability of the negation of a formula is known as refutation.

The resolution proof procedure is refutation complete for classical logic as, when applied to an unsatisfiable formula, the procedure is guaranteed to (terminate and) produce false.
Resolution Example

If we begin with the set of clauses

\[ \{ p \lor q, p \lor \neg q, \neg p \lor q, \neg p \lor \neg q \} \].

then we can construct a refutation just using the basic resolution rule together with simplification:

1. \( p \lor q \)
2. \( p \lor \neg q \)
3. \( \neg p \lor q \)
4. \( \neg p \lor \neg q \)
5. \( p \) (resolve 1 and 2, simplify \( p \lor p \))
6. \( \neg p \) (resolve 3 and 4, simplify \( \neg p \lor \neg p \))
7. \textbf{false} (resolve 5 and 6)

showing that the original set of clauses is unsatisfiable.
Resolution is Practical

Many deductive systems based on resolution have indeed been developed.

Especially in classical propositional and first-order logics, resolution-based provers have been very successful.

For example, Otter and Vampire are based on resolution:

http://www.cs.unm.edu/~mccune/otter
http://www.vprover.org

The former is a popular, and widely used, automated reasoning system; the latter is currently the fastest such system for analyzing classical first-order logics.
Clausal Temporal Resolution

As the temporal resolution method is *clausal*, we must transform a PTL formula into a clausal form, specifically into SNF that we saw earlier.

We then use three types of resolution rule that are applicable to clauses; two analogous to the classical resolution rule, the other a new *temporal resolution* rule.

Due to the inductive interaction between the ‘\[\square\]’ and ‘\[\bigcirc\]’, the application of the temporal resolution rule is non-trivial, requiring quite specialised algorithms.

It is important to we note that, having been defined, proved correct and extended to more complex logics, the temporal resolution method has been implemented and has been shown to be generally competitive.
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).
Simplification and Subsumption

The following transformation is used for SNF clauses which imply false (where \( L \) is a conjunction of literals).

<table>
<thead>
<tr>
<th>Formula</th>
<th>Translates to</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ⋄(L \Rightarrow \bigcirc \text{false}) )</td>
<td>→</td>
<td>( ⋄(\text{start} \Rightarrow \neg L) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( ⋄(\text{true} \Rightarrow \bigcirc \neg L) )</td>
</tr>
</tbody>
</table>

If, by satisfying \( L \), a contradiction is produced in the next moment, then \( L \) must never be satisfied.

The new constraints generated effectively represent \( ⋄\neg L \) since \( \text{start} \Rightarrow \neg L \) ensures that \( L \) is false at the initial state and \( \text{true} \Rightarrow \bigcirc \neg L \) ensures that \( L \) is false at any next moment.
Assume we have one SNF clause, namely

1. \( \text{push\_the\_button} \implies \Box \text{false} \)

We can apply our simplification rule to replace this by two new SNF clauses:

2. \( \text{start} \implies \neg \text{push\_the\_button} \)
3. \( \text{true} \implies \Box \neg \text{push\_the\_button} \)

Thus, if “\( \text{push\_the\_button} \)” leads to a contradiction, then we can never have this, either at the start

\( \text{start} \implies \neg \text{push\_the\_button} \)

or anywhere else

\( \text{true} \implies \Box \neg \text{push\_the\_button} \)
Clause Removal

Certain SNF clauses can be removed during simplification as they represent valid sub-formulae and therefore cannot directly contribute to the generation of a contradiction.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Translates to</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐(false ⇒ ☐R)</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>☐(L ⇒ ☐true)</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

The first clause is valid as false can never be satisfied.
The second is valid as ☐true is always satisfied.
Subsumption

Subsumption also forms part of the simplification process. As in classical resolution, an SNF clause may be removed from the clause-set if it is subsumed by an existing clause. Subsumption may be expressed as the following operation.

\[
\begin{align*}
\square (L_1 \Rightarrow R_1) & \quad \text{Translates to} \quad \square (L_2 \Rightarrow R_2) \\
\square (L_2 \Rightarrow R_2) & \quad \frac{\vdash L_1 \Rightarrow L_2 \quad \vdash R_2 \Rightarrow R_1}{\vdash L_1 \Rightarrow L_2 \quad \vdash R_2 \Rightarrow R_1} \\
\end{align*}
\]

The side conditions \( \vdash L_1 \Rightarrow L_2 \) and \( \vdash R_2 \Rightarrow R_1 \) must hold before this subsumption operation can be applied.

In this case, the clause \( L_1 \Rightarrow R_1 \) can be deleted without losing information.
Subsumption Example

If we have

1. \texttt{start} \Rightarrow a
2. \texttt{start} \Rightarrow a \lor b
3. z \Rightarrow \bigcirc y
4. (w \land z) \Rightarrow \bigcirc (y \lor x)

then this clause set simplifies to

1. \texttt{start} \Rightarrow a
3. z \Rightarrow \bigcirc y

\textbf{Because:} “\texttt{start} \Rightarrow a” provides more information than “\texttt{start} \Rightarrow (a \lor b)” and,
if “w \land z” is true then z must also be true, so “z \Rightarrow \bigcirc y” provides more information than “(w \land z) \Rightarrow \bigcirc (y \lor x)”
Step Resolution

Pairs of initial or step SNF clauses may be resolved using the following (resolution) operations (where $R_1$ and $R_2$ are disjunctions of literals, $L_1$ and $L_2$ are conjunctions of literals and $p$ is a proposition).

\[
\begin{align*}
\text{start} & \Rightarrow R_1 \lor p \\
\text{start} & \Rightarrow R_2 \lor \neg p \\
\hline
\text{start} & \Rightarrow R_1 \lor R_2
\end{align*}
\]

\[
\begin{align*}
L_1 & \Rightarrow \bigcirc (R_1 \lor p) \\
L_2 & \Rightarrow \bigcirc (R_2 \lor \neg p) \\
(L_1 \land L_2) & \Rightarrow \bigcirc (R_1 \lor R_2)
\end{align*}
\]

The first operation is called initial resolution; the second is termed step resolution.

See how these rules resemble the classical resolution rule.
Step Resolution Example

We can now prove an instance of one of the PTL axioms using only step resolution and initial resolution, namely

$$\vdash \Box (a \Rightarrow b) \Rightarrow (\Box a \Rightarrow \Box b).$$

To establish validity, we negate, transform to SNF, and apply the resolution procedure. If \textbf{false} is generated, then the negated formula is unsatisfiable, and so the original formula is valid. Negated formula in SNF is

1. \textbf{start} \Rightarrow f
2. f \Rightarrow \Box x
3. \textbf{start} \Rightarrow (\neg x \lor \neg a \lor b)
4. \textbf{true} \Rightarrow \Box (\neg x \lor \neg a \lor b)
5. f \Rightarrow \Box a
6. f \Rightarrow \Box \neg b
Step Resolution Example (2)

1. \texttt{start} \Rightarrow f
2. \texttt{f} \Rightarrow \bigcirc x
3. \texttt{start} \Rightarrow (\neg x \lor \neg a \lor b)
4. \texttt{true} \Rightarrow \bigcirc (\neg x \lor \neg a \lor b)
5. \texttt{f} \Rightarrow \bigcirc a
6. \texttt{f} \Rightarrow \bigcirc \neg b
7. \texttt{f} \Rightarrow \bigcirc (\neg x \lor \neg a) \quad [4, 6 \text{ Step Resolution}]
8. \texttt{f} \Rightarrow \bigcirc \neg x \quad [5, 7 \text{ Step Resolution}]
9. \texttt{f} \Rightarrow \bigcirc \texttt{false} \quad [2, 8 \text{ Step Resolution}]
10. \texttt{start} \Rightarrow \neg f \quad [9 \text{ Simplification}]
11. \texttt{true} \Rightarrow \bigcirc \neg f \quad [9 \text{ Simplification}]
12. \texttt{start} \Rightarrow \texttt{false} \quad [1, 10 \text{ Initial Resolution}]

A contradiction is obtained, meaning the negated formula is unsatisfiable and therefore the original formula is valid.
We would also *like* to have a resolution rule such as

\[
\begin{align*}
L_1 &\Rightarrow \lozenge l \\
L_2 &\Rightarrow \square \neg l \\
\neg (L_1 \land L_2)
\end{align*}
\]

However, in translating to SNF, we have removed clauses such as \( L_2 \Rightarrow \square \neg l \).

So, part of the (necessary) complexity of the temporal resolution operation comes from having to reconstruct \( \square \neg l \) from a set of step clauses.

This is possible, for example recall the *induction axiom*:

\[
\vdash \square (\varphi \Rightarrow \lozenge \varphi) \Rightarrow (\varphi \Rightarrow \square \varphi)
\]
Temporal Resolution: Intuition (2)

So, we essentially try to collect together a set of step clauses

\[ L_0 \Rightarrow \bigcirc R_0 \]
\[ \ldots \Rightarrow \ldots \]
\[ L_n \Rightarrow \bigcirc R_n \]

where each \( R_j \) will ensure \( \neg l \) and ‘trigger’ at least one \( L_i \)

\[ \bigvee_j R_j \Rightarrow \neg l \]
\[ \bigvee_j R_j \Rightarrow \bigvee_i L_i \]

**Intuitively:** once one of the “\( L_i \Rightarrow \bigcirc R_i \)” clauses is ‘fired’ then \( \neg l \) will be required and another clause in the set will be ‘fired’ at the next step.

Thus, together, these clauses ensure that \( \neg l \) occurs forever.
So, once we have \( \neg l \land \bigvee_j R_j \)
in one state it will repeat in the next state. Thus

\[
\Box\left( (\neg l \land \bigvee_j R_j) \Rightarrow \Diamond (\neg l \land \bigvee_j R_j) \right).
\]

Applying the PTL induction axiom to this, we get

\[
(\neg l \land \bigvee_j R_j) \Rightarrow \Box (\neg l \land \bigvee_j R_j)
\]

This then gives us

\[
\bigvee_i L_i \Rightarrow \Diamond \Box (\neg l \land \bigvee_j R_j)
\]

which simplifies to

\[
\bigvee_i L_i \Rightarrow \Diamond \Box \neg l
\]

We can then resolve the above with any \( C \Rightarrow \Diamond l \) clause.
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$_m$), is also defined.

Any SNF step clause is also a step clause in SNF$_m$. Any two step clauses in SNF$_m$ may be combined to produce a further step clause in SNF$_m$ 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 PTL-clauses can be represented in SNF$_m$. 
Detailed Resolution Rule

So the full temporal resolution operation essentially applies between a *sometime* clause and a *set* of $\text{SNF}_m$ clauses that together imply $A \Rightarrow \Box \Box \neg l$:

\[
L_0 \Rightarrow \Box R_0 \\
\ldots \Rightarrow \ldots \\
L_n \Rightarrow \Box R_n \\
C \Rightarrow \Diamond I \\
\]

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

again, with the various side conditions on $R_j$. 
Intuition about the Resolvent

Consider the formulae

\[
\text{gambler} \Rightarrow \bigcirc \bigboxdot \neg \text{rich}
\]
\[
\text{ambitious} \Rightarrow \bigdiamond \text{rich}
\]

In English, we might say that a gambler will never be rich in the future, while someone who is ambitious will eventually be rich.

Applying the clausal temporal resolution rule to the above gives us

\[
\text{ambitious} \Rightarrow (\neg \text{gambler}) \mathcal{W} \text{rich}
\]

So, someone who is ambitious should not be a gambler unless they become rich.
Correctness

The key formal results concerning the temporal resolution process include:

- **soundness**
  i.e. that if a contradiction is generated via clausal temporal resolution, then the original PTL formula was unsatisfiable;

- **completeness**
  i.e. that if a PTL formula is unsatisfiable, then a contradiction will be derived by applying clausal temporal resolution to it; and

- **termination**
  i.e. that any clausal temporal resolution process will terminate, regardless of its initial PTL formula.