Hybrid Logic in the Isabelle Proof Assistant: Benefits, Challenges and the Road Ahead

Asta Halkjær From
Technical University of Denmark

AiML 2020, August 26.
Introduction

Case study: Tableau system $ST^A$ for basic hybrid logic with a formalization in Isabelle/HOL (4900+ lines in the Archive of Formal Proofs):
https://devel.isa-afp.org/entries/Hybrid_Logic.html

Based on $ST^*$ by Blackburn, Bolander, Braüner and Jørgensen.
Hybrid Logic

Modal logic with names for worlds (nominals) and satisfaction operators (@_i_).

\( \phi, \psi ::= x \mid i \mid \neg \phi \mid \phi \lor \psi \mid \Diamond \phi \mid @i \phi \)

Semantics based on Kripke models \(((W, R), V)\) and an assignment \(g\).

- \(W\): underlying set, \(R\): binary relation, \(V\): unary relation, \(g\): map from nominals to worlds.

\[
\begin{align*}
M, g, w \models x & \iff w \in V(x) \\
M, g, w \models i & \iff g(i) = w \\
M, g, w \models \neg \phi & \iff M, g, w \not\models \phi \\
M, g, w \models \phi \lor \psi & \iff M, g, w \models \phi \text{ or } M, g, w \models \psi \\
M, g, w \models \Diamond \phi & \iff \text{for some } w', wRw' \text{ and } M, g, w' \models \phi \\
M, g, w \models @i \phi & \iff M, g, g(i) \models \phi
\end{align*}
\]
Hybrid Logic in Isabelle – Syntax

The syntax of our object logic becomes a `datatype` in the metalogic:

```isabelle
datatype ('a, 'b) fm = Pro 'a 
| Nom 'b 
| Neg (('a, 'b) fm) (\neg \_\_ ) [40] 40 
| Dis (('a, 'b) fm) (('a, 'b) fm) (infixr \land\_\_ ) 30 
| Dia (('a, 'b) fm) (\&\_\_ ) 10 
| Sat 'b (('a, 'b) fm) (@\_\_ ) 10
```

Values of type `(a, b) fm` are built from the above constructors. Propositional symbols are drawn from type variable `a` and nominals from `b`.

We specify the usual infix notation in bold.
Hybrid Logic in Isabelle – Semantics

Type variable ‘w represents the set of worlds:

```isabelle
datatype ('w, 'a) model =
  Model (R: ('w ⇒ 'w set)) (V: ('w ⇒ 'a ⇒ bool))
```

The semantics interprets the object logic in the metalogic (HOL, not English).

```isabelle
primrec semantics :: ('w, 'a) model ⇒ ('b ⇒ 'w) ⇒ 'w ⇒ ('a, 'b) fm ⇒ bool
  (\_, \_, \_ \models \_ \rightarrow [50, 50, 50] 50) where
  \langle M, \_, w \models \text{Pro } x \rangle = \text{V M w x}
  \langle \_, g, w \models \text{Nom } i \rangle = (w = g i)
  \langle M, g, w \models \lnot p \rangle = (\lnot M, g, w \models p)
  \langle M, g, w \models (p \lor q) \rangle = ((M, g, w \models p) \lor (M, g, w \models q))
  \langle M, g, w \models \Diamond p \rangle = (\exists v \in R M w. M, g, v \models p)
  \langle M, g, \_ \models @ i p \rangle = (M, g, g i \models p)
```
Seligman-Style Tableau System – Blocks

Formulas are true relative to a given world.

- Internalized calculus: Satisfaction statements only.
- Seligman-style: Group formulas into blocks named by opening nominals.

Blocks and branches are modelled naturally:

\[
\begin{align*}
@_i & \phi_1 \\
@_i & \phi_2 \\
\vdots \\
@_j & \psi_1 \\
\vdots \\
\end{align*}
\]

We know exactly what a branch is formally. (a) Internalized. (b) Seligman-style.
# Seligman-Style Tableau System – Rules

Horizontal lines indicate that input can appear on an earlier block.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$\phi \lor \psi$</th>
<th>$\neg (\phi \lor \psi)$</th>
<th>$\neg \neg \phi$</th>
<th>$\Diamond \phi$</th>
<th>$\neg \Diamond \phi$</th>
<th>$\Diamond i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>$\phi$</td>
<td>$\psi$</td>
<td>$\neg \phi$</td>
<td>$\phi$</td>
<td>$\Diamond i$</td>
<td>$\neg \Diamond \phi$</td>
<td></td>
</tr>
<tr>
<td>$\Diamond$</td>
<td>$i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Vertical lines indicate extensions / justifications.

Above each input formula is the nominal of its block.

<table>
<thead>
<tr>
<th>$b$</th>
<th>$b$</th>
<th>$i$</th>
<th>$i$</th>
<th>$b$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\neg \phi$</td>
<td>$@_a \phi$</td>
<td>$\neg @_a \phi$</td>
</tr>
<tr>
<td>$i$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\neg \phi$</td>
<td></td>
</tr>
</tbody>
</table>

Multiple inputs are written next to each other.

- **GoTo**
- **Nom**
- **Closing**

The set $A$ is fixed in advance (e.g. the root nominals)

1. $i$ is fresh, $i \notin A$ and $\phi$ is not a nominal.
2. $i$ is not fresh.
3. If $\phi = i$ or $\phi = \Diamond i$ for some nominal $i$ then $i \in A$. 

Seligman-Style Tableau System – Restricting GoTo

Rule out repeated applications of GoTo: GoTo cannot be applied twice in a row.

Easy to respect informally when proving e.g. substitution lemma: if Θ has a closing tableau then so does Θσ.

Induction in Isabelle “mimics” the construction of the closing tableau for Θ but this may be a problem:

We have to handle detours explicitly:

- New induction principle
- Substitution + pruning
- Weaker lemma
- Different restriction

\[
\begin{align*}
\text{GoTo} & \quad \sigma \\
\phi & \quad a
\end{align*}
\]

\[
\begin{align*}
\text{GoTo} & \quad \psi \sigma \\
\phi & \quad a
\end{align*}
\]
Seligman-Style Tableau System – Potential

Handle detours by assuming more starting “potential”:

R4 The GoTo rule consumes one potential. The remaining rules add one unit of potential and we are allowed to start from any amount.

Allows for local detours.

Still prevents repeated GoTo.

Easy to formalize.
Seligman-Style Tableau System – Formalization

The turnstile predicate holds if the branch can be closed (wrt. $A$ and $n$). E.g.

| Neg: |
| $(\neg \neg p)$ at $a$ in $(ps, a)$ # branch $\implies$
| new $p$ at $(ps, a)$ # branch $\implies$
| $A, Suc n \vdash (p # ps, a)$ # branch $\implies$
| $A, n \vdash (ps, a)$ # branch |

We can then machine verify results like soundness and completeness:

```agda
def soundness_fresh:
  assumes \(\vdash A, n \vdash [([-p], i)]\) \(i \notin \text{nominals p}\)
  shows \(\vdash M, g, w \vdash p\)
```

```agda
def completeness:
  fixes $p : \langle 'a : \text{countable}, 'b : \text{countable} \rangle$ fm
  assumes
    inf: \(\text{infinite (UNIV : 'b set)}\) \(\text{and}\)
    valid: \(\forall (M : ('b \text{ set}, 'a \text{ model}) g w. M, g, w \vdash p)\)
  shows \(\vdash \text{nominals p, } 1 \vdash [([-p], i)]\)
```
Formalizing Termination

We formalized an *inductive* predicate \( \vdash \): does a *finite* branch close? Now we need a *coinductive* well-formedness predicate on finitely branching, possibly *infinite* trees:

\[
\text{codatatype} \ (\text{labels: } \text{'a}) \text{ tree } = \text{Node} \ (\text{getLabel: } \text{'a}) \ (\text{getSubs: } <\text{'a tree list}>)
\]

That is, a codatatype may have *turtles constructors all the way down*.

Current work:

\[
\text{theorem} \ \langle \text{twf A tab } \implies \text{fin tab} \rangle
\]

*owps*

\text{twf} (coinductive): all rules are applied correctly.  
\text{fin} (inductive): the tableau is finite.
Concluding Remarks

“PROOF. This is easily seen by going through each of the tableau rules.”
Not always as easy in Isabelle!

But! We can do our metatheory in higher-order logic instead of natural language. No omissions, no ambiguity.

Verify the definitions ⇒ trust the results.

Future work:

- Formalize termination.
- Produce an executable decision procedure.
References


