The LogicS of Prior:
Past, Present, and Future

6-7 September 2019

Roskilde University

Using the Isabelle Proof Assistant

Seligman-Style Tableau for Hybrid Logic

Andreas Halkjær From, DTU Compute

Background

- Masters thesis: Hybrid Logic
- Current plan: Formalize in Isabelle/HOL the paper
  - Klaus Frovin Jørgensen, Patrick Blackburn, Thomas Bolander and Torben Braüner. Synthetic Completeness Proofs for Seligman-style Tableau Systems. Advances in Modal Logic 11:302-321 2016.
  - [Patrick Blackburn, Thomas Bolander, Torben Braüner and Klaus Frovin Jørgensen. Completeness and Termination for a Seligman-style Tableau System. Journal of Logic and Computation 27(1): 81-107, 2017.]
- Dates: 19/08 2019 – 19/01 2020
- Supervisors:
  - Jørgen Villadsen
  - Alexander Birch Jensen
  - Patrick Blackburn

Isabelle

- Isabelle is a generic proof assistant.
- It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus.
- The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols.

http://isabelle.in.tum.de/overview.html

- That is, **machine-checked proofs**.

Syntax I

- Encode the syntax as a datatype.
- Automatically generates induction principle, disjointness lemmas and more.

$$\varphi ::= i \mid p \mid \neg\varphi \mid \varphi \vee \psi \mid \Diamond\varphi \mid @_i\varphi.$$

```
datatype ('a, 'b) fm
  = Pro 'a
  | Nom 'b
  | Neg ‹('a, 'b) fm› ("¬ _" [40] 40)
  | Dis ‹('a, 'b) fm› ‹('a, 'b) fm› (infixr "∨" 30)
  | Dia ‹('a, 'b) fm› ("◇ _" 10)
  | Sat 'b ‹('a, 'b) fm› ("@ _ _" 10)
```

Syntax II

- Introduce abbreviations for syntax

```
abbreviation Box ("□ _" 10) where
  ‹□ φ ≡ ¬ (◇ ¬ φ)›
```

Syntax II

- Introduce abbreviations for syntax

```
abbreviation Box ("□ _" 10) where
  ‹□ φ ≡ ¬ (◇ ¬ φ)›
```

- Define substitution. Checked for type safety and totality.

```
primrec sub :: ‹('b ⇒ 'c) ⇒ ('a, 'b) fm ⇒ ('a, 'c) fm› where
  ‹sub _ (Pro x) = Pro x›
| ‹sub f (Nom i) = Nom (f i)›
| ‹sub f (¬ p) = (¬ sub f p)›
| ‹sub f (p ∨ q) = (sub f p ∨ sub f q)›
| ‹sub f (◇ p) = (◇ sub f p)›
| ‹sub f (@i p) = (@ (f i) (sub f p))›
```

Syntax III

- We can try out our definitions.

```
value ‹sub Suc (@ 0 (Pro ''p'' ∨ ¬ Nom 3))›
```

```
"@ 1 Pro ''p'' ∨ ¬ Nom 4"
  :: "(char list, nat) fm"
```

Syntax III

- We can try out our definitions.

```
value ‹sub Suc (@ 0 (Pro ''p'' ∨ ¬ Nom 3))›
```

```
"@ 1 Pro ''p'' ∨ ¬ Nom 4"
  :: "(char list, nat) fm"
```

```
value ‹sub (λn :: nat. n > 2) (@ 0 (Pro ''p'' ∨ ¬ Nom 3))›
```

```
"@ False Pro ''p'' ∨ ¬ Nom True"
  :: "(char list, bool) fm"
```

Semantics I

- ”We interpret the language in models based on frames (W, R), where W is a non-empty set (we call its elements worlds) and R is a binary relation on W (the accessibility relation).”
- ”A model is a triple (W, R, V) where (W, R) is a frame and V (the valuation) maps propositional symbols p to arbitrary subsets of W, and nominals i to singleton subsets of W.”

$$
\begin{array}{ll}
\mathfrak{M}, w \models a & \text{iff } a \text{ is atomic and } w \in V(a) \\
\mathfrak{M}, w \models \neg\varphi & \text{iff } \mathfrak{M}, w \not\models \varphi \\
\mathfrak{M}, w \models \varphi \vee \psi & \text{iff } \mathfrak{M}, w \models \varphi \text{ or } \mathfrak{M}, w \models \psi \\
\mathfrak{M}, w \models \Diamond\varphi & \text{iff for some } w',\ wRw' \text{ and } \mathfrak{M}, w' \models \varphi \\
\mathfrak{M}, w \models @_i\varphi & \text{iff } \mathfrak{M}, w' \models \varphi \text{ and } w' \in V(i).
\end{array}
$$

Semantics II

- 'w is the non-empty type of worlds.
- R is the accessibility relation, V is the valuation on propositions, g maps nominals to worlds.

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

primrec semantics
  :: ‹('w, 'a) model ⇒ ('b ⇒ 'w) ⇒ 'w ⇒ ('a, 'b) fm ⇒ bool›
  ("_, _, _ ⊨ _" [50, 50, 50] 50) where
  ‹(M, _, w ⊨ Pro x) = V M w x›
| ‹(_, g, w ⊨ Nom i) = (w = g i)›
| ‹(M, g, w ⊨ ¬ p) = (¬ M, g, w ⊨ p)›
| ‹(M, g, w ⊨ (p ∨ q)) = ((M, g, w ⊨ p) ∨ (M, g, w ⊨ q))›
| ‹(M, g, w ⊨ ◇ p) = (∃v ∈ R M w. M, g, v ⊨ p)›
| ‹(M, g, _ ⊨ @ i p) = (M, g, g i ⊨ p)›
```

Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›
```

```
lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›


lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›


lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›


lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›


lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›


lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

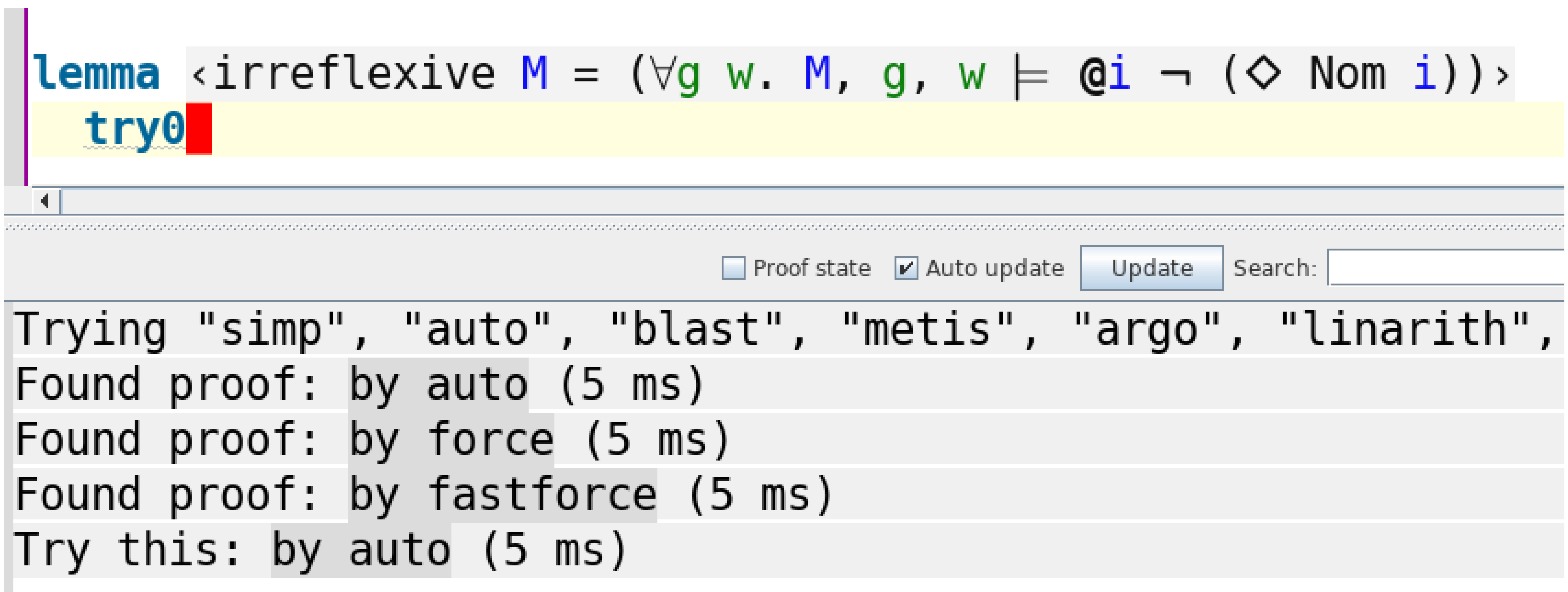
Example Proof I

```
abbreviation irreflexive :: ‹('w, 'b) model ⇒ bool› where
  ‹irreflexive M ≡ ∀w. w ∉ R M w›


lemma ‹irreflexive M ⟹ M, g, w ⊨ @i ¬ (◇ Nom i)›
proof -
  assume ‹irreflexive M›
  then have ‹g i ∉ R M (g i)›
    by simp
  then have ‹¬ (∃v ∈ R M (g i). g i = v)›
    by simp
  then have ‹¬ M, g, g i ⊨ ◇ Nom i›
    by simp
  then have ‹M, g, g i ⊨ ¬ (◇ Nom i)›
    by simp
  then show ‹M, g, w ⊨ @i ¬ (◇ Nom i)›
    by simp
qed
```

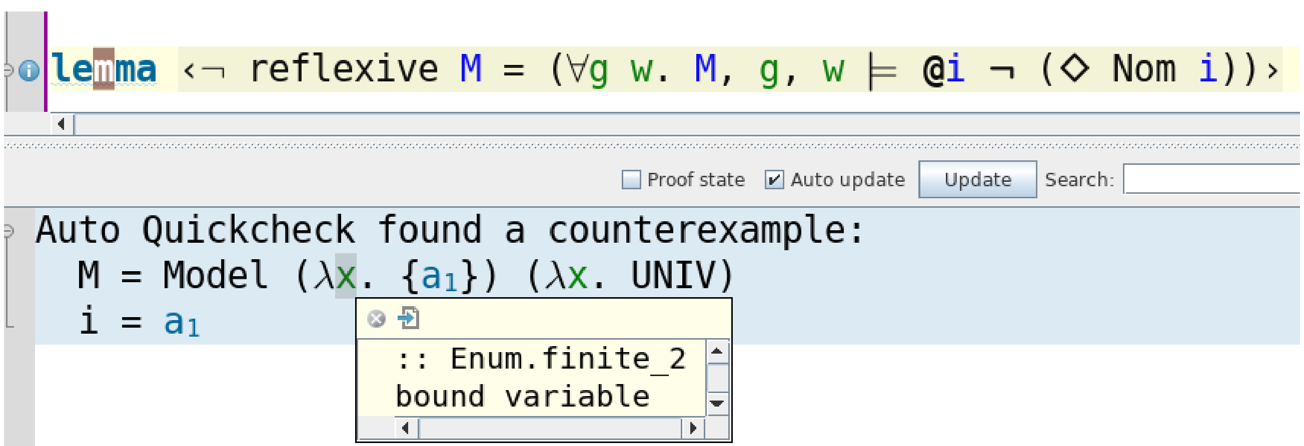
Example Proof II

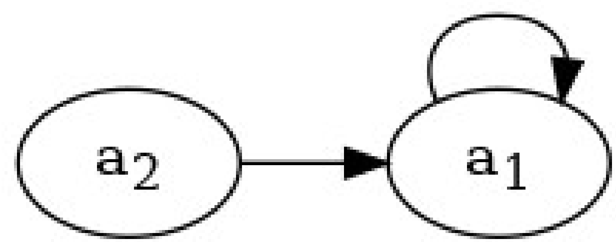
- Isabelle has powerful proof search.



Counterexample Search

- Goals are checked for counter-examples automatically or on demand
- E.g. if we use nonreflexive instead of irreflexive in the previous example.





Calculus I

1	$\neg(@_i j \wedge @_j \varphi \to @_i \varphi)$	
2	$@_i j \wedge @_j \varphi$	$(\neg \to)$ on 1
3	$\neg @_i \varphi$	$(\neg \to)$ on 1
4	$@_i j$	$(\wedge)$ on 2
5	$@_j \varphi$	$(\wedge)$ on 2
6	i	GoTo
7	j	$(@)$ on 4,6
8	φ	$(@)$ on 5,7
9	$\neg \varphi$	$(\neg @)$ on 3,6
	$\times$	

Calculus II

- We inductively define the tableau rules.
- Definition over a single branch. Whole tree is implicit.

```
inductive ST :: ‹('a, 'b) branch ⇒ bool› ("⊢ _" [50] 50) where
```

Calculus II

- We inductively define the tableau rules.
- Definition over a single branch. Whole tree is implicit.

```
inductive ST :: ‹('a, 'b) branch ⇒ bool› ("⊢ _" [50] 50) where
```

- ”a branch closes either by having φ and ¬φ inside a block, or inside two distinct blocks with the same opening nominal”

```
Close:
‹bl ∈ opens_with i branch ⟹ br ∈ opens_with i branch ⟹
  p ∈ set bl ⟹ (¬ p) ∈ set br ⟹
  ⊢ branch›
```

Calculus III

$$\Diamond \varphi$$

$$\Big| \; (\Diamond)^1$$

$$\Diamond i$$

$$@_i \varphi$$

DiaP:

```
‹block ∈ opens_with (opening_current branch) branch ⟹
  (◇ p) ∈ set block ⟹
  ⊢ (@i p) #. (◇ Nom i) #. branch ⟹
  ∄a. p = Nom a ⟹ i ∉ branch_nominals branch ⟹
  ⊢ branch›
```

Calculus III

$$\Diamond\varphi$$

$$(\Diamond)^1$$

$$\Diamond i$$

$$@_i\varphi$$

- Many modelling choices:
  - Implicit or explicit opening nominals, blocks?
  - Sets or lists for explicit blocks?
  - ...
- The simpler the better.

DiaP:

```
‹block ∈ opens_with (opening_current branch) branch ⟹
  (◇ p) ∈ set block ⟹
  ⊢ (@i p) #. (◇ Nom i) #. branch ⟹
  ∄a. p = Nom a ⟹ i ∉ branch_nominals branch ⟹
  ⊢ branch›
```

Soundness

- Derivation of negation implies validity of original.

```
theorem ‹⊢ Branch [] [¬ p] ⟹ M, g, w ⊨ p›
  using soundness by fastforce
```

Soundness

- Derivation of negation implies validity of original.

```
theorem ‹⊢ Branch [] [¬ p] ⟹ M, g, w ⊨ p›
  using soundness by fastforce
```

Now, the contrapositive of soundness follows from the observation that if a tableau T of the calculus $\mathcal{ST}$ has a branch which is block-wise satisfiable, then the tableau obtained by applying a rule to T also has a branch which is block-wise satisfiable. This can be seen simply by inspecting each rule in $\mathcal{ST}$. ■

- Currently around 285 lines of proof code.
- Lots of case-splitting on whether the current block has an opening nominal.
- Trusting definitions, the tableau is truly sound.

Sledgehammer

- Isabelle has powerful proof search *search*.
- Automatically searches for relevant lemmas, local facts and method to prove current goal.

```
case (DisN block branch p q)
then show ?case
proof (induct branch rule: opening_current.induct)
  case (1 named i nameds init)
  then have ‹∀p ∈ set block. M, g, g i ⊨ p›
    sledgehammer
```

```
Sledgehammering...
Proof found...
"cvc4": Try this: by (metis basic_trans_rules(31) opening_current.simps(1) op
"vampire": Try this: by (metis DisN.hyps(1) DisN.prems basic_trans_rules(31)
```

Substitution I

Lemma 3.2 (Substitution lemma)

(i) ***We can uniformly substitute in a tableau.*** *Suppose* Φ *is a branch in an* STB*-tableau in which the nominals* i *and* j *occur, and* Φ *is extended by applying rule* R *to input* I *to obtain output* O*. Then* $\Phi[i/j]$ *can be extended by applying* R *to input* $I[i/j]$ *to obtain output* $O[i/j]$*.*

(ii) ***Which nominals are used as fresh nominals is irrelevant.*** *Suppose* Θ *is a branch in a finite tableau* T *and let* i *be a nominal which is used somewhere in* Θ *as a fresh nominal. Suppose, furthermore, that* j *is not used at all in* Θ*. We can then uniformly substitute* j *for* i *in* Θ *to obtain a branch* $\Theta[j/i]$*, where rule-applications in* $\Theta[j/i]$ *mimic rule-applications in* Θ*, the only change being that* j *is used instead of* i*.*

Proof. By induction on the construction of Φ and Θ respectively. $\square$

Substitution II

- Cumbersome to specify fresh nominals in formalization.
- Instead, prove a generalized lemma using simultaneous substitutions.

```
theorem ST_sub:
  fixes f :: ‹'b ⇒ 'c›
  assumes ‹infinite (UNIV :: 'c set)›
  shows ‹⊢ branch ⟹ ⊢ sub_branch f branch›
proof (induct branch arbitrary: f rule: ST.induct)
```

- Around 150 lines of proof code.

Applications

- Kepler conjecture (Flyspeck project)
- Gödel’s incompleteness theorems
- Completeness of natural deduction, sequent calculus, resolution for first-order logic
- Algorithms and data structures
- Algebra, analysis, probability theory, graph theory
- ...
- Completeness of System K

Further Reading

- Archive of Formal Proofs
  https://www.isa-afp.org/index.html

- IsaFoL (Isabelle Formalization of Logic)
  https://bitbucket.org/isafol/

- Formalizing 100 Theorems
  http://www.cs.ru.nl/~freek/100/