Formalized Soundness and Completeness of Natural Deduction for First-Order Logic

Scandinavian Logic Symposium 2018

Andreas Halkjær From
Introduction

Several benefits to formalizing proofs:

- Less room for error (if any).
- No parts left as *exercise for the reader*.
- Proof can be explored interactively.
- It’s fun!

At DTU we are interested in natural deduction for teaching purposes (NaDeA).

Abstract referred to my extension of Berghofer’s work. For continuity with previous talk I will use NaDeA here.
Agenda

• Isabelle & NaDeA
• Soundness
• Closed Formulas
• Open Formulas
• Conclusion
LCF-style theorem prover based on Higher-Order Logic (HOL). All proofs go through (small) kernel of axioms and inference rules. Like functional programming with datatypes for First-Order Logic. Proofs are written in the declarative language Isar.
Syntax

LCF-style theorem prover based on Higher-Order Logic (HOL). All proofs go through (small) kernel of axioms and inference rules. Like functional programming with datatypes for First-Order Logic. Proofs are written in the declarative language Isar.

Terms

type-synonym id = char list

datatype tm = Var nat | Fun id tm list

Formulas

datatype fm = Falsity | Pre id tm list | Imp fm fm | Dis fm fm | Con fm fm | Exi fm | Uni fm
Type variable 'a encodes domain.
Environment e :: nat ⇒ 'a.
Function denotation: f :: id ⇒ 'a list ⇒ 'a.
Predicate denotation: $g :: \text{id} \Rightarrow 'a \text{ list} \Rightarrow \text{bool}$.

**Formulas**

```plaintext
primrec
  semantics :: \((\text{nat} \Rightarrow 'a) \Rightarrow (\text{id} \Rightarrow 'a \text{ list} \Rightarrow 'a) \Rightarrow (\text{id} \Rightarrow 'a \text{ list} \Rightarrow \text{bool}) \Rightarrow \text{fm} \Rightarrow \text{bool})
where
  semantics e f g Falsity = False |
  semantics e f g (\text{Pre} \ i \ l) = g \ i \ (\text{semantics-list} \ e \ f \ l) | 
  semantics e f g (\text{Imp} \ p \ q) = (\text{if} \ \text{semantics} \ e \ f \ g \ p \ \text{then} \ \text{semantics} \ e \ f \ g \ q \ \text{else} \ \text{True}) | 
  semantics e f g (\text{Dis} \ p \ q) = (\text{if} \ \text{semantics} \ e \ f \ g \ p \ \text{then} \ \text{True} \ \text{else} \ \text{semantics} \ e \ f \ g \ q) | 
  semantics e f g (\text{Con} \ p \ q) = (\text{if} \ \text{semantics} \ e \ f \ g \ p \ \text{then} \ \text{semantics} \ e \ f \ g \ q \ \text{else} \ \text{False}) | 
  semantics e f g (\text{Exi} \ p) = (∃x. \ \text{semantics} \ (\lambda n. \ \text{if} \ n = 0 \ \text{then} \ x \ \text{else} \ e (n - 1)) \ f \ g \ p) | 
  semantics e f g (\text{Uni} \ p) = (∀x. \ \text{semantics} \ (\lambda n. \ \text{if} \ n = 0 \ \text{then} \ x \ \text{else} \ e (n - 1)) \ f \ g \ p)
```

6 DTU Compute Formalized Soundness and Completeness of Natural Deduction for First-Order Logic 2018–06–12
\( \phi \in \Gamma \) \[ \frac{}{\Gamma \vdash \phi} \text{ assum} \]
Isabelle & NaDeA

From rule to code

\[
\begin{align*}
\phi \in \Gamma & \quad \text{assum} \\
\Gamma \vdash \phi & \\
\end{align*}
\]

\[
\begin{align*}
p \in z & \quad \text{assum} \\
z \vdash p & \\
\end{align*}
\]
\[ \phi \in \Gamma \quad \text{assum} \]
\[
\frac{\Gamma \vdash \phi}{\Gamma} 
\]

\[ p \in z \quad \text{assum} \]
\[
\frac{z \vdash p}{z} 
\]

\[ \text{member } p z \quad \text{Assume} \]
\[
\frac{\text{OK } p z}{OK} 
\]
\[ \frac{\phi \in \Gamma}{\Gamma \vdash \phi} \quad \text{assume} \]

\[ \frac{p \in z}{z \vdash p} \quad \text{assume} \]

\[ \frac{\text{member } p \ in \ z}{\text{OK } p \ in \ z} \quad \text{Assume} \]

Assume: member \( p \ in \ z \) \( \quad \Rightarrow \quad \) OK \( p \ in \ z \)
inductive OK :: fm ⇒ fm list ⇒ bool where
    Assume: member p z ⊢ OK p z |
    Boole: OK Falsity ((Imp p Falsity) # z) ⊢ OK p z |
    Imp-E: OK (Imp p q) z ⊢ OK p z ⊢ OK q z |
    Imp-I: OK q (p # z) ⊢ OK (Imp p q) z |
    Dis-E: OK (Dis p q) z ⊢ OK r (p # z) ⊢ OK r (q # z) ⊢ OK r z |
    Dis-l1: OK p z ⊢ OK (Dis p q) z |
    Dis-l2: OK q z ⊢ OK (Dis p q) z |
    Con-E1: OK (Con p q) z ⊢ OK p z |
    Con-E2: OK (Con p q) z ⊢ OK q z |
    Con-l: OK p z ⊢ OK q z ⊢ OK (Con p q) z |
    Exi-E: OK (Exi p) z ⊢ OK q ((sub 0 (Fun c []) p) # z) ⊢ 
            news c (p # q # z) ⊢ OK q z |
    Exi-l: OK (sub 0 t p) z ⊢ OK (Exi p) z |
    Uni-E: OK (Uni p) z ⊢ OK (sub 0 t p) z |
    Uni-l: OK (sub 0 (Fun c []) p) z ⊢ news c (p # z) ⊢ OK (Uni p) z
lemma soundness':
    OK p z \rightarrow list-all (semantics e f g) z \rightarrow semantics e f g p

Proof by induction over inference rules (for arbitrary function denotation).
Written declaratively:

    case (Uni-I c p z)
    then have \forall x. list-all (semantics e (f(c := \lambda w. x)) g) z
    by simp

    by simp

~100 lines including helper lemmas.
Soundness

Context

Lemma soundness':

\[ \text{OK } p \ z \implies \text{list-all } (\text{semantics } e \ f \ g) \ z \implies \text{semantics } e \ f \ g \ p \]

Proof by induction over inference rules (for arbitrary function denotation).
Written declaratively:

\begin{align*}
\text{case } (\text{Uni-I } c \ p \ z) \\
\text{then have } & \forall x. \text{list-all } (\text{semantics } e \ (f(c := \lambda w. x)) \ g) \ z \quad \text{[c is fresh]} \\
\text{by simp} \\
\text{then have } & \forall x. \text{semantics } e \ (f(c := \lambda w. x)) \ g \ (\text{sub } 0 \ (\text{Fun } c [])) \ p \quad \text{[IH]} \\
\text{using Uni-I by blast}
\end{align*}
Soundness

Proof by induction over inference rules (for arbitrary function denotation).

Written declaratively:

\[
\text{case } (\text{Uni-}\text{I } c p z) \\
\text{then have } \forall x. \text{list-all } (\text{semantics } e (f(c := \lambda w. x)) g) z \quad [c \text{ is fresh}] \\
\text{by simp} \\
\text{then have } \forall x. \text{semantics } e (f(c := \lambda w. x)) g (\text{sub 0 } (\text{Fun } c [])) p \quad [\text{IH}] \\
\text{using } \text{Uni-}\text{I by blast} \\
\text{then have } \forall x. \text{semantics } (\text{put } e 0 x) (f(c := \lambda w. x)) g p \quad [\text{subst. lemma}] \\
\text{by simp}
\]
lemma soundness':

\[ \text{OK } p \ z \rightarrow \text{list-all } (\text{semantics } e \ f \ g) \ z \rightarrow \text{semantics } e \ f \ g \ p \]

Proof by induction over inference rules (for arbitrary function denotation). Written declaratively:

case (Uni-I c p z)
then have \( \forall x. \text{list-all } (\text{semantics } e (f(c := \lambda w. x)) \ g) \ z \) \hfill [c is fresh]
by simp
then have \( \forall x. \text{semantics } e (f(c := \lambda w. x)) \ g (\text{sub } 0 \ (\text{Fun } c \ [])) \ p \) \hfill [IH]
using Uni-I by blast
then have \( \forall x. \text{semantics } (\text{put } e \ 0 \ x) \ (f(c := \lambda w. x)) \ g \ p \) \hfill [subst. lemma]
by simp
then have \( \forall x. \text{semantics } (\text{put } e \ 0 \ x) \ f \ g \ p \) \hfill [c is fresh again]
lemma soundness':

$$\text{OK } p \ z \implies \text{list-all } (\text{semantics } e \ f \ g) \ z \implies \text{semantics } e \ f \ g \ p$$

Proof by induction over inference rules (for arbitrary function denotation).
Written declaratively:

```plaintext
case (Uni-I c p z)
then have $\forall \ x. \ \text{list-all } (\text{semantics } e (f(c := \lambda w. \ x)) \ g) \ z$  \hspace{1cm} [c is fresh]
    by simp
then have $\forall \ x. \ \text{semantics } e (f(c := \lambda w. \ x)) \ g \ (\text{sub } 0 \ (\text{Fun } c []) \ p)$  \hspace{1cm} [IH]
    using Uni-I by blast
then have $\forall \ x. \ \text{semantics } (\text{put } e \ 0 \ x) \ (f(c := \lambda w. \ x)) \ g \ p$  \hspace{1cm} [subst. lemma]
    by simp
then have $\forall \ x. \ \text{semantics } (\text{put } e \ 0 \ x) \ f \ g \ p$  \hspace{1cm} [c is fresh again]
    using news c (p ≠ z) by simp
then show $\text{semantics } e \ f \ g \ (\text{Uni } p)$  \hspace{1cm} [exactly semantics for Uni]
    by simp
```

∼100 lines including helper lemmas.
Closed Formulas

Completeness

Proof by Fitting in *First-Order Logic and Automated Theorem Proving*. Formalized by Berghofer for different natural deduction proof systems.

**Dependent on semantics (~1500 lines)**

- Consistency property, $C$, on sets of formulas
- Alternative consistency, $C^+$
- Finite character, $C^*$
- Maximal extension, $H$, is Hintikka, sentences in $H$ have Herbrand model
Closed Formulas

Completeness

Proof by Fitting in *First-Order Logic and Automated Theorem Proving*. Formalized by Berghofer for different natural deduction proof system.

Dependent on semantics (~1500 lines)

- Consistency property, \( C \), on sets of formulas
- Alternative consistency, \( C^+ \)
- Finite character, \( C^* \)
- Maximal extension, \( H \), is Hintikka, sentences in \( H \) have Herbrand model

Dependent on inference rules (~350 lines)

- Show consistency of formulas from which false cannot be derived.
Closed Formulas

Completeness

Proof by Fitting in *First-Order Logic and Automated Theorem Proving*. Formalized by Berghofer for different natural deduction proof system.

**Dependent on semantics (~1500 lines)**

- Consistency property, $C$, on sets of formulas
- Alternative consistency, $C^+$
- Finite character, $C^*$
- Maximal extension, $H$, is Hintikka, sentences in $H$ have Herbrand model

**Dependent on inference rules (~350 lines)**

- Show consistency of formulas from which false cannot be derived.

**Completeness via contradiction (~40 lines)**

- Assume $p$ is (closed and) valid but not derivable
- Then $\{\neg p\} \in C$ (no contradiction without $p$), has a model
Closed Formulas

Completeness

Proof by Fitting in *First-Order Logic and Automated Theorem Proving*. Formalized by Berghofer for different natural deduction proof system.

Dependent on semantics (~1500 lines)

- Consistency property, \( C \), on sets of formulas
- Alternative consistency, \( C^+ \)
- Finite character, \( C^* \)
- Maximal extension, \( H \), is Hintikka, sentences in \( H \) have Herbrand model

Dependent on inference rules (~350 lines)

- Show consistency of formulas from which false cannot be derived.

Completeness via contradiction (~40 lines)

- Assume \( p \) is (closed and) valid but not derivable
- Then \( \{\neg p\} \in C \) (no contradiction without \( p \)), has a model

(+ ~200 lines for Löwenheim-Skolem, ~200 lines for any countably infinite universe)
Open Formulas

Standard trick

One trick for open formulas: Just universally close it!

\[ x \rightarrow x \quad \leadsto \quad \forall x. x \rightarrow x \]

Then we obtain a derivation for a syntactically different formula.

Open formulas are well-defined in our formalization. We should treat them properly.

This might teach students something about environments etc.
Open Formulas

Strategy

How to reuse completeness proof for sentences.

Starting point

\[ p \vdash x \rightarrow p \]
How to reuse completeness proof for sentences.

Starting point

Premises to implications

\[
p \vdash x \rightarrow p
\]

\[
\vdash p \rightarrow x \rightarrow p
\]
Strategy

How to reuse completeness proof for sentences.

Starting point

Premises to implications

Universally close formula

\[ p \vdash x \rightarrow p \]

\[ p \vdash p \rightarrow x \rightarrow p \]

\[ \vdash \forall x. p \rightarrow x \rightarrow p \]
How to reuse completeness proof for sentences.

Starting point

Premises to implications

Universally close formula

Obtain proof

\[ \vdash \forall x. p \rightarrow x \rightarrow p \]
**Strategy**

How to reuse completeness proof for sentences.

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Starting point</strong></td>
<td>$p \vdash x \rightarrow p$</td>
</tr>
<tr>
<td><strong>Premises to implications</strong></td>
<td>$\vdash p \rightarrow x \rightarrow p$</td>
</tr>
<tr>
<td><strong>Universally close formula</strong></td>
<td>$\vdash \forall x. p \rightarrow x \rightarrow p$</td>
</tr>
<tr>
<td><strong>Obtain proof</strong></td>
<td>$\vdash \forall x. p \rightarrow x \rightarrow p$</td>
</tr>
<tr>
<td><strong>Eliminate quantifiers with constants</strong></td>
<td>$\vdash p \rightarrow c \rightarrow p$</td>
</tr>
</tbody>
</table>
Strategy

How to reuse completeness proof for sentences.

Starting point

Premises to implications

Universally close formula

Obtain proof

Eliminate quantifiers with constants

Substitute constants with variables
Open Formulas

Strategy

How to reuse completeness proof for sentences.

Starting point

Premises to implications

Universally close formula

Obtain proof

Eliminate quantifiers with constants

Substitute constants with variables

Implications back to premises

~1100 additional lines.
Premises to implications

Turn premises into chain of implications:

\textbf{primrec} \textit{put-imps} :: \textit{fm} \Rightarrow \textit{fm list} \Rightarrow \textit{fm} \textbf{where}

\begin{align*}
\text{put-imps} \ p \ [\ ] & = p \\
\text{put-imps} \ p \ (q \ # \ z) & = \text{Imp} \ q \ (\text{put-imps} \ p \ z)
\end{align*}
Premises to implications

Turn premises into chain of implications:

\[
\textbf{primrec} \quad \text{put-imps} :: \quad \text{fm} \Rightarrow \text{fm list} \Rightarrow \text{fm} \quad \text{where} \\
\text{put-imsps} \ p \ [] = p | \\
\text{put-imsps} \ p \ (q \neq z) = \text{Imp} \ q \ (\text{put-imsps} \ p \ z)
\]

This behaves as expected with regards to the semantics:

\[
\textbf{lemma} \quad \text{semantics-put-imsps}:
\quad \text{list-all} \ (\text{semantics} \ e \ f \ g) \ z \rightarrow \text{semantics} \ e \ f \ g \ p = \\
\quad \text{semantics} \ e \ f \ g \ (\text{put-imsps} \ p \ z) \\
\quad \text{by} \ (\text{induct} \ z) \ \text{auto}
\]
Open Formulas

Universal closure

Put a number of universal quantifiers in front:

\[
\text{primrec } \text{put-unis} :: \text{nat } \Rightarrow \text{fm } \Rightarrow \text{fm} \text{ where }
\]

\[
\begin{align*}
\text{put-unis } 0 \ p &= \ p \\
\text{put-unis } (\text{Suc } m) \ p &= \text{Uni} \ (\text{put-unis } m \ p)
\end{align*}
\]

This preserves validity:

\[
\text{lemma } \text{valid-put-unis} :: \\
\forall (e :: \text{nat } \Rightarrow \text{fm} \Rightarrow \text{fm}). \text{semantics} \ e \ f \ g \ p = \Rightarrow \text{semantics} \ (e :: \text{nat } \Rightarrow \text{fm} \Rightarrow \text{fm}) \ (\text{put-unis } m \ p)
\]

by \text{(induct} m \ \text{arbitrary:} \ e \text{)} \ \text{simp-all}

The universal closure exists:

\[
\text{lemma } \text{ex-closure} :: \\
\exists m. \text{sentence} \ (\text{put-unis } m \ p)
\]

using \text{ex-closed closed-put-unis} by \text{simp}
Universal closure

Put a number of universal quantifiers in front:

```haskell
primrec put-unis :: nat ⇒ fm ⇒ fm where
  put-unis 0 p = p |
  put-unis (Suc m) p = Uni (put-unis m p)
```

This preserves validity:

```haskell
lemma valid-put-unis: ∀ (e :: nat ⇒ 'a) f g. semantics e f g p ⇒⇒
  semantics (e :: nat ⇒ 'a) f g (put-unis m p)
by (induct m arbitrary: e) simp-all
```
Open Formulas

Universal closure

Put a number of universal quantifiers in front:

\[
\text{primrec } \text{put-unis} :: \text{nat} \Rightarrow \text{fm} \Rightarrow \text{fm} \text{ where }
\begin{align*}
\text{put-unis} \ 0 \ p &= p \mid \\
\text{put-unis} \ (\text{Suc} \ m) \ p &= \text{Uni} \ (\text{put-unis} \ m \ p)
\end{align*}
\]

This preserves validity:

\[
\text{lemma } \text{valid-put-unis}: \forall (e :: \text{nat} \Rightarrow 'a) \ f \ g. \ \text{semantics} \ e \ f \ g \ p \Rightarrow \\
\text{semantics} \ (e :: \text{nat} \Rightarrow 'a) \ f \ g \ (\text{put-unis} \ m \ p)
\]

\text{by } (\text{induct } m \text{ arbitrary: } e) \ \text{simp-all}

The universal closure exists:

\[
\text{lemma } \text{ex-closure}: \exists m. \ \text{sentence} \ (\text{put-unis} \ m \ p)
\]

\text{using } \text{ex-closed} \ \text{closed-put-unis by simp}
We can combine the above to obtain our derivation:

```
let ?p = put-imps p (rev z)

have ℹ️: ∀ (e :: nat ⇒ 'a) f g. semantics e f g ?p
  using assms semantics-put-imps by fastforce
obtain m where ℹ️: sentence (put-unis m ?p)
  using ex-closure by blast
moreover have ∀ (e :: nat ⇒ 'a) f g. semantics e f g (put-unis m ?p)
  using ℹ️ valid-put-unis by blast
ultimately have OK (put-unis m ?p) []
  using assms sentence-completeness by blast
```

Next step: Work within proof system to derive open formula from this.
Open Formulas

Direct closure elimination

Tricky to eliminate quantifiers directly with de Bruijn indices.

**Example**

\[
\forall\forall p(0, 1, 2)[2/0] \sim \forall((\forall p(0, 1, 2))[3/1]) \sim \forall\forall p(0, 1, 4)
\]

\[
\forall p(0, 1, 4)[1/0] \sim \forall(p(0, 1, 4)[2/1]) \sim \forall p(0, 2, 3)
\]

\[
p(0, 2, 3)[0/0] \sim p(0, 1, 2)
\]

Previously substituted variables are adjusted by subsequent substitutions.
Open Formulas

Direct closure elimination

Tricky to eliminate quantifiers directly with de Bruijn indices.

Example

\[
(\forall\forall p(0, 1, 2))[2/0] \leadsto \forall((\forall p(0, 1, 2))[3/1]) \leadsto \forall\forall p(0, 1, 2)[4/2]) \leadsto \forall\forall p(0, 1, 4)
\]

\[
(\forall p(0, 1, 4))[1/0] \leadsto \forall (p(0, 1, 4)[2/1]) \leadsto \forall p(0, 2, 3)
\]

\[
p(0, 2, 3)[0/0] \leadsto p(0, 1, 2)
\]

Previously substituted variables are adjusted by subsequent substitutions.

Idea: Eliminate with (fresh) constants instead!

\textbf{fun const-for-unis :: fm } \Rightarrow \textbf{id list } \Rightarrow \textbf{fm where}
\[
\text{const-for-unis (Uni p) (c#cs)} = \text{const-for-unis (sub 0 (Fun c []) p) cs } |
\]
\text{const-for-unis p - = p}
Constant substitution

New type of substitution: $subc \, c \, s \, p$ replaces occurrences of $c$ with $s$ in $p$, adjusting $s$ when passing a quantifier.
Disadvantage: Have to reprove many substitution lemmas for $subc$. 
**Open Formulas**

**Constant substitution**

New type of substitution: \( subc \ c \ s \ p \) replaces occurrences of \( c \) with \( s \) in \( p \), adjusting \( s \) when passing a quantifier.

Disadvantage: Have to reprove many substitution lemmas for \( subc \).

We prove the new rule admissible by induction over the rules:

**lemma** \( OK-subc: OK \ p \ z \implies OK \ (subc \ c \ s \ p) \ (subcs \ c \ s \ z) \)

Trivial for everything except cases with quantifiers, newness, e.g. witness in \( Exi-E \) rule (no assumptions on \( c \) or \( s \)).

Requires renaming:

**lemma** \( OK-psubst: OK \ p \ z \implies OK \ (psubst \ f \ p) \ (map \ (psubst \ f) \ z) \)
Composing closure elimination with constant substitution yields telescoping sequence:

$$subc\ c_0\ (m-1)\ (subc\ c_1\ (m-2)\ (\ldots\ (subc\ c_{m-1}\ 0\ (sub\ 0\ c_{m-1}\ \ldots))\ldots))$$

Each introduced constant is immediately substituted with correct variable. Subsequent substitutions do not adjust previous variables.

**lemma** \textit{vars-for-consts-for-unis}:

\begin{align*}
\text{closed}\ (\text{length}\ cs)\ p &\implies \text{list-all}\ (\lambda c.\ \text{new}\ c\ p)\ cs \implies \text{distinct}\ cs \implies \\
\text{vars-for-consts}\ (\text{consts-for-unis}\ (\text{put-unis}\ (\text{length}\ cs)\ p)\ cs)\ cs &\equiv p
\end{align*}
Composing closure elimination with constant substitution yields telescoping sequence:

\[
\text{subc } c_0 \ (m-1) \ (\text{subc } c_1 \ (m-2) \ (\ldots \ (\text{subc } c_{m-1} \ 0 \ (\text{sub } 0 \ c_{m-1} \ \ldots )))))
\]

Each introduced constant is immediately substituted with correct variable. Subsequent substitutions do not adjust previous variables.

**Lemma** \( \text{vars-forconsts-forunis} \):

\[
\text{closed} \ (\text{length} \ cs) \ p \implies \text{list-all} \ (\lambda c. \ \text{new} \ c \ p) \ cs \implies \text{distinct} \ cs \implies \\
\text{vars-forconsts} \ (\text{consts-forunis} \ (\text{put-unis} \ (\text{length} \ cs) \ p) \ cs) \ cs = p
\]

**Theorem** \( \text{remove-unis} \): \( \text{OK } (\text{put-unis} \ m \ p) \ [] \implies \text{OK } p \ [] \)
For $p \rightarrow q$, weaken assumptions with $p$, then use modus ponens.

**lemma** shift-imp-assum:

- assumes $OK \ (\text{Imp} \ p \ q) \ z$
- shows $OK \ q \ (p \# \ z)$

**proof**

- have $\text{set} \ z \subseteq \text{set} \ (p \# z)$
  - by auto
- then have $OK \ (\text{Imp} \ p \ q) \ (p \# z)$
  - using assms weaken-assumptions by blast
- moreover have $OK \ p \ (p \# z)$
  - using Assume by simp
- ultimately show $OK \ q \ (p \# z)$
  - using Imp-E by blast

qed
Weaken assumptions

lemma weaken-assumptions: $OK \ p \ z \implies set \ z \subseteq set \ z' \implies OK \ p \ z'$

Shown by induction over inference rules. Trivial, except for $Exi$-$E$ and $Uni$-$I$, where newness is required: The new constant given by the induction hypothesis is not necessarily new under the bigger premises. Again, renaming is necessary.
Open Formulas

Weaken assumptions

**Lemma** weaken-assumptions: \[ OK \ p \ z \implies \text{set} \ z \subseteq \text{set} \ z' \implies OK \ p \ z' \]

Shown by induction over inference rules.
Trivial, except for *Exi-E* and *Uni-I*, where newness is required: The new constant given by the induction hypothesis is not necessarily new under the bigger premises.
Again, renaming is necessary.

Remove chain of implications by induction:

**Lemma** remove-imps: \[ OK \ (\text{put-imps} \ p \ z) \ z' \implies OK \ p \ (\text{rev} \ z @ z') \]

using shift-imp-assum by \( \text{(induct} \ z \ \text{arbitrary:} \ z') \) simp-all
We can now finish the completeness proof:

```plaintext
let \( ?p = \text{put-imps} \ p \ (\text{rev} \ z) \)

have \( \forall (e :: \text{nat} \Rightarrow \ 'a) \ f \ g. \ \text{semantics} \ e \ f \ g \ ?p \)
  using assms semantics-put-imps by fastforce
obtain \( m \) where \( \forall: \text{sentence} (\text{put-unis} \ m \ ?p) \)
  using ex-closure by blast
moreover have \( \forall (e :: \text{nat} \Rightarrow \ 'a) \ f \ g. \ \text{semantics} \ e \ f \ g \ (\text{put-unis} \ m \ ?p) \)
  using \( \forall \) valid-put-unis by blast
ultimately have \( O\!K \ (\text{put-unis} \ m \ ?p) \) []
  using assms sentence-completeness by blast
then have \( O\!K \ ?p \) []
  using \( \forall \) remove-unis by blast
then show \( O\!K \ p \ z \)
  using remove-imps by fastforce
```
• NaDeA is sound and complete.
• Also for open formulas.
  • Standard results like renaming, weakening, deduction theorem arise naturally in proof.
• Formalization ensures tricky cases are treated properly.
• Formalization may also introduce complexity, e.g. de Bruijn indices.
References


