Chapter 1

Library abstract_completeness

This text provides self contained formalized proofs of generalization of the completeness theorem of propositional and first order predicate calculus. It follows roughly https://www.irif.fr/~krivine/articl although for the first order part the generalization rule we use is slightly different but is the one that can be found in most presentations of proof theory based on Hilbert systems.

The most striking aspect (in the opinion of the author) of the completeness theorem presented in the above article and here is the fact that the proof is intuitionistic and that we can extract programs from it. No additional axioms were used in the following coq file.

The plan we follow differs from the article. All the sets of formulas involved are assumed to be countable. First we prove the Lindenbaum lemma (which is actually the propositional completeness theorem) which asserts that for any formula F, any set of formulas which doesn't proves F using classical propositional calculus is contained in a maximal consistent set formulas which doesn't either proves F with these rules. When classical reasoning is available this theorem is obvious (the set is built by induction). A slight modification of this argument gives the intuitionistic case. Then we prove the completeness theorem for predicate calculus by first order logic, by extending the theory adding Henkin constants and then exploiting the propositional case on the theory obtained.

This method allows us to get models where witnesses are available for any universally quantified formula, which is a stronger form of the well known completeness theorem. Because of this, convenient applications are possible (e.g. easy proofs of skolemization).

```
Section General_type_tools.
```

The sets of formulas that we'll consider further in the text are of the form "Formula -> Type" rather than "Formula -> Prop" in order to allow full program extraction. The following tools are introduced in order to manipulate these objects.

```
Variable T:Type.

Section Addition_of_a_new_item_to_a_type_valued_set.

Variable P: T → Type.
Variable n:T.

Inductive add_extra_item: T → Type:=
```

|aei_base: $\forall\ x{:}T,\ P\ x \to$ **add_extra_item** x
|aei_new: **add_extra_item** n.

End Addition_of_a_new_item_to_a_type_valued_set.

Section Inclusion_of_type_valued_sets.

Definition type_valued_inclusion $(P\ Q{:}\ T \to$ Type$):= \forall\ x{:}T,\ P\ x \to Q\ x$.

Definition tvi_identity: $\forall\ P{:}\ T \to$ Type, type_valued_inclusion $P\ P$.

Definition tvi_transitivity: $\forall\ A\ B\ C{:}T \to$ Type,
type_valued_inclusion $A\ B \to$ type_valued_inclusion $B\ C \to$ type_valued_inclusion A C.

End Inclusion_of_type_valued_sets.

End General_type_tools.

Section A_classical_Hilbert_proof_system.

Variable *Sentence*: Type.
Variable *s_implies*: *Sentence* $\to$ *Sentence* $\to$ *Sentence*.

Notation "a -o b":= $(s\_implies\ a\ b)$ (right associativity, at level 41).

Section Definition_of_proofs.

Variable *Specific_axiom*: *Sentence* $\to$ Type.

Inductive **classical_propositional_implicative_theorem**: *Sentence* $\to$ Type:=
|cpit_ax: $\forall\ x{:}Sentence,\ Specific\_axiom\ x \to$
classical_propositional_implicative_theorem x
|cpit_k: $\forall\ x\ y{:}Sentence$, **classical_propositional_implicative_theorem** (x -o y -o x)
|cpit_s: $\forall\ x\ y\ z{:}Sentence$,
classical_propositional_implicative_theorem ((x -o y -o z) -o (x -o y) -o x -o z)
|cpit_Peirce: $\forall\ x\ y{:}Sentence$,
classical_propositional_implicative_theorem (((x -o y) -o x) -o x)
|cpit_modus_ponens: $\forall\ x\ y{:}Sentence$,
classical_propositional_implicative_theorem (x -o y) $\to$
classical_propositional_implicative_theorem x $\to$
classical_propositional_implicative_theorem y.

Definition cpit_i: $\forall\ x{:}Sentence$, **classical_propositional_implicative_theorem** (x -o x).

Definition cpit_syllogism: $\forall\ x\ y\ z{:}Sentence$,
classical_propositional_implicative_theorem (x -o y) $\to$
classical_propositional_implicative_theorem (y -o z) $\to$
classical_propositional_implicative_theorem (x -o z).

Definition cpit_b: $\forall\ x\ y\ z{:}Sentence$,
classical_propositional_implicative_theorem ((y -o z) -o (x -o y) -o x -o z).

End Definition_of_proofs.

Definition cpit_subtheory: ∀ (A B: *Sentence* → Type),
type_valued_inclusion *Sentence* A B → ∀ p:*Sentence*,
classical_propositional_implicative_theorem A p →
classical_propositional_implicative_theorem B p.

Section The_deduction_theorem.

Variable *base_theory*: *Sentence* → Type.
Variable *new_sentence*: *Sentence*.

Definition cpit_deduction_theorem: ∀ (f: *Sentence*),
classical_propositional_implicative_theorem
(**add_extra_item** *Sentence* *base_theory* *new_sentence*) f →
classical_propositional_implicative_theorem *base_theory* (*new_sentence* -o f).

End The_deduction_theorem.

Definition cpit_triple_syllogism (*theory*: *Sentence* → Type) (x a b c:*Sentence*):
classical_propositional_implicative_theorem
theory
((a -o b -o c) -o (x -o a) -o (x -o b) -o (x -o c)).

Section A_classical_tautology.

Variable *theory*: *Sentence* → Type.
Variable a b c:*Sentence*.

This theorem is not valid in intuitionistic logic

Definition cpit_classical_cases_analysis:
classical_propositional_implicative_theorem *theory* (a -o c) →
classical_propositional_implicative_theorem *theory* ((a -o b) -o c) →
classical_propositional_implicative_theorem *theory* c.

End A_classical_tautology.

Section Propositional_Completeness.

Variable *indexation*: **nat** → *Sentence*.
Variable *countable*: ∀ f:*Sentence*, {n:**nat** | *indexation* n = f}.

Notation "T |- p" := (**classical_propositional_implicative_theorem** T p) (at level 42).

Notation add_hypothesis := (**add_extra_item** *Sentence*).

Section A_special_one_step_extension_of_a_theory.

This paragraph is the only part where our text actually differs from usual proofs of the Lindenbaum's lemma, in order to accomodate with coq intuitionistic constraints. the rest of the text is tedious but probably straightforward if the reader knows how to prove completeness according to the route we've chosen i.e. Lindenbaum -> propositional case ->

Henkin extension of a first order theory to exploit the propositional result. All these further part are usually already constructive.

```
Variable base_theory: Sentence → Type.
Variable addendum target: Sentence.

Inductive inflate: Sentence → Type:=
|infl_base: ∀ x:Sentence, base_theory x → inflate x
|infl_new:
   ((add_hypothesis base_theory addendum ⊢ target) → base_theory ⊢ target)
   → inflate addendum.

Lemma cpit_inflate_dichotomy: ∀ q:Sentence,
    inflate ⊢ q →
                sum
                  ((add_hypothesis base_theory addendum ⊢ target)
                   → base_theory ⊢ target)
                  (base_theory ⊢ q).

Definition cpit_inflate_conservation:
  inflate ⊢ target → base_theory ⊢ target.

Definition cpit_inflate_already_proven_sentence:
  base_theory ⊢ addendum → inflate addendum.

End A_special_one_step_extension_of_a_theory.

Section The_Lindenbaum_maximal_construction.

Variable ground_theory: Sentence → Type.
Variable target: Sentence.

Fixpoint consistent_theory_sequence (n:nat) {struct n}: Sentence → Type:=
  match n with
  | 0 ⇒ ground_theory
  | S m ⇒ inflate (consistent_theory_sequence m) (indexation m) target
  end.

Inductive Lindenbaum_maximal_theory: Sentence → Type:=
|lmt_intro: ∀ (k:nat) (f: Sentence),
    consistent_theory_sequence k f → Lindenbaum_maximal_theory f.

Fixpoint cts_conservation (n:nat) {struct n}:
  consistent_theory_sequence n ⊢ target → ground_theory ⊢ target.

Section Two_arithmetical_lemmas_about_the_max_of_integers.

  Lemma nat_max_symmetry: ∀ p q:nat, max p q = max q p.

  Lemma nat_max_succ: ∀ p q:nat,
      sum (max p (S q) = max p q) (max p (S q) = S (max p q)).

End Two_arithmetical_lemmas_about_the_max_of_integers.
```

```
Definition cts_increasing:
  ∀ p q:nat,
    type_valued_inclusion Sentence
                                   (consistent_theory_sequence q)
                                   (consistent_theory_sequence (max p q)).

Definition cpit_lmt_to_cts_index (f:Sentence) (pr: Lindenbaum_maximal_theory
⊢ f):
  {n:nat & consistent_theory_sequence n ⊢ f}.

Definition lmt_conservation:
  Lindenbaum_maximal_theory ⊢ target → ground_theory ⊢ target.

Definition lmt_counter_model:
  Lindenbaum_maximal_theory target → ground_theory ⊢ target.

Definition lmt_belonging_criterion (x:Sentence):
  (Lindenbaum_maximal_theory ⊢ x -o target → Lindenbaum_maximal_theory
⊢ target) →
  Lindenbaum_maximal_theory x.

Definition lmt_modus_ponens_stability: ∀ x y:Sentence,
    Lindenbaum_maximal_theory (x -o y) →
    Lindenbaum_maximal_theory x →
    Lindenbaum_maximal_theory y.

Definition lmt_reverse_modus_ponens: ∀ x y:Sentence,
    (Lindenbaum_maximal_theory x → Lindenbaum_maximal_theory y) →
    Lindenbaum_maximal_theory (x -o y).

Definition lmt_proof_stability (f:Sentence) (pr: Lindenbaum_maximal_theory ⊢
f):
  Lindenbaum_maximal_theory f.

End The_Lindenbaum_maximal_construction.

Section Concise_reformulation_of_results.

  Variable theory: Sentence → Type.

  Definition classical_propositional_model (M: Sentence → Type):=
    prod (type_valued_inclusion Sentence theory M)
         (prod (∀ x y:Sentence, M (((x -o y) -o x) -o x))
               (∀ x y:Sentence,
                   prod (M (x -o y) → M x → M y)
                        ((M x → M y) → M (x -o y))
               )
         ).

  Definition classical_propositional_soundness_theorem:
    ∀ (M: Sentence → Type) (f: Sentence),
```

```
      classical_propositional_model M →
      theory ⊢ f → M f.

    Definition constructive_classical_propositional_completeness_theorem:
      ∀ h:Sentence,
        {M : Sentence → Type & prod (classical_propositional_model M)
                                     (M h → theory ⊢ h)}.

    Definition classical_propositional_completeness_theorem:
      ∀ h:Sentence,
        prod (theory ⊢ h →
                    ∀ M:Sentence → Type, classical_propositional_model M → M
h)
             ((∀ M:Sentence → Type, classical_propositional_model M → M h) →
              theory ⊢ h).

   End Concise_reformulation_of_results.

  End Propositional_Completeness.

End A_classical_Hilbert_proof_system.

Section Abstract_classical_first_order_systems.

  Variable General_Property: Type.
  Variable Term:Type.
  Variable gp_implies: General_Property → General_Property → General_Property.
  Variable gp_specialize: General_Property → Term → General_Property.
  Variable gp_fresh_variable: General_Property → General_Property → Term.
  Variable gp_theorem: General_Property → Type.

  Notation "a -o b" := (gp_implies a b) (right associativity, at level 41).
  Notation "|- a" := (gp_theorem a) (at level 44).

  Variable gp_k: ∀ x y:General_Property, ⊢ x -o y -o x.
  Variable gp_s: ∀ x y z:General_Property, ⊢ (x -o y -o z) -o (x -o y) -o x -o z.
  Variable gp_Peirce: ∀ x y: General_Property, ⊢ ((x -o y) -o x) -o x.
  Variable gp_special_case: ∀ (f:General_Property) (t:Term), ⊢ f -o (gp_specialize f t).
  Variable gp_modus_ponens: ∀ x y:General_Property, ⊢ x -o y → ⊢ x → ⊢ y.
  Variable gp_generalization_rule: ∀ p q:General_Property,
      ⊢ p -o gp_specialize q (gp_fresh_variable p q) →
      ⊢ p -o q.
```

We explain what the objects above and especially "gp_specialize" and "gp_fresh_variable" maps are intended to mean. A "general property" is simply a first order formula over a given signature. Let f be such a formula and t a term of the language. if f is "forall x, g" where g is another formula, then "gp_specialize f t" is nothing but g<x := t>, i.e. the the result of the capture-free substitution of x by t in g. In every other cases, gp_specialize f t is f itself. This justifies the name "General_Property" we've picked. If n -> v_n is the sequence of all the variables of the language (which will be assumed to be countable in the further developments), then for every formulas a,b, "gp_fresh_variable a b" is v_m where m is the smallest integer such that v_m has no free occurences in a or in b (if you don't know what that means: take "v_m doesn't appears in a or in b", this would give the same results at the end). gp_theorem is the proof system used and the properties "gp_special_case" and "gp_generalization_rule" are obvious (wether f and p are of the form "forall x g" or not). The program offered is general enough so that hopefully it can be used in any reasonable implementation of first order logic. The only real constraint is that the set of formulas is countable.

No soundness theorem is provided (such a result would depend on how gp_theorem is actually defined), the user will have to provide one (which is usually easy).

```
Section Basic_proof_theoretic_results.

  Ltac mp t := apply gp_modus_ponens with (x:= t).

  Section propositional_proof_to_gp_theorem.

    Variable T: General_Property → Type.
    Variable t_incl: type_valued_inclusion General_Property T gp_theorem.

    Definition cpit_inclusion_to_gp: ∀ p:General_Property,
        classical_propositional_implicative_theorem General_Property gp_implies T
p → ⊢ p.

  End propositional_proof_to_gp_theorem.

  Definition cpit_to_gp: ∀ p:General_Property,
      classical_propositional_implicative_theorem General_Property gp_implies gp_theorem
p
      → ⊢ p:= cpit_inclusion_to_gp gp_theorem (tvi_identity General_Property gp_theorem).

  Definition gp_syllogism (x y z:General_Property): ⊢ x -o y → ⊢ y -o z → ⊢ x -o z.

  Ltac syl t := apply gp_syllogism with (y:= t).

  Definition gp_permutation_insertion: ∀ x y z:General_Property,
    ⊢ x -o y -o z → ⊢ y → ⊢ x -o z.

  Definition gp_universal_witness_property: ∀ g h:General_Property,
    ⊢ (gp_specialize g (gp_fresh_variable (h -o g) g) -o g) -o h → ⊢ h.

End Basic_proof_theoretic_results.

Section Abstract_first_order_completeness.

  Variable indexation: nat → General_Property.
  Variable countable: ∀ f:General_Property, {n:nat | indexation n = f}.
  Variable target: General_Property.

  Section The_associated_Henkin_extension_of_a_theory.

    Fixpoint gp_Henkin_auxiliary_formula (n:nat) {struct n}: General_Property:=
      match n with
      | 0 ⇒ target
```

```
  | S m ⇒
    (gp_specialize (indexation m)
                      (gp_fresh_variable
                          ((gp_Henkin_auxiliary_formula m) -o indexation m)
                          (indexation m)) -o (indexation m)
    )
    -o gp_Henkin_auxiliary_formula m
  end.

Notation gp_Henkin_universal_constant :=
  (fun (k:nat) ⇒
     gp_fresh_variable
       ((gp_Henkin_auxiliary_formula k) -o indexation k) (indexation k)).

Definition gp_universal_witness (f: General_Property): Term:=
  gp_Henkin_universal_constant (proj1_sig (countable f)).

Notation critical_formula:=
  (fun n:nat ⇒ (gp_specialize (indexation n) (gp_Henkin_universal_constant n))
                   -o indexation n).

Inductive gp_Henkin_theory: General_Property → Type:=
|gpht_base: ∀ x:General_Property, gp_theorem x → gp_Henkin_theory x
|gpht_critical_formula: ∀ n:nat,
     gp_Henkin_theory (critical_formula n).

Definition gpuw_critical_belongs_to_Henkin: ∀ (f: General_Property),
     gp_Henkin_theory ((gp_specialize f (gp_universal_witness f)) -o f).

Fixpoint gp_add_to_tail (head: General_Property) (n:nat) {struct n}:General_Property:=
  match n with
  | 0 ⇒ head
  | S m ⇒ (critical_formula m) -o (gp_add_to_tail head m)
  end.

Fixpoint gp_att_k (x:General_Property) (n:nat) {struct n}:
  gp_theorem (x -o (gp_add_to_tail x n)).

Definition nat_max_sum (p:nat): ∀ q:nat, (p - q) + q = max p q.

Definition gp_att_max: ∀ (x:General_Property) (p q:nat),
     gp_theorem (gp_add_to_tail x p) → gp_theorem (gp_add_to_tail x (max q p)).

Definition gp_att_modus_ponens: ∀ (n:nat) (x y:General_Property),
     ⊢ gp_add_to_tail (x -o y) n →
     ⊢ gp_add_to_tail x n →
     ⊢ gp_add_to_tail y n.

Lemma gp_att_proof: ∀ (f: General_Property),
     classical_propositional_implicative_theorem
```

```
        General_Property gp_implies (gp_Henkin_theory) f →
      {n: nat & ⊢ gp_add_to_tail f n}.

  Definition gp_Henkin_auxiliary_formula_elimination: ∀ n:nat,
      ⊢ gp_Henkin_auxiliary_formula n → ⊢ target.

  Definition gp_att_haf_equality: ∀ n:nat,
      gp_add_to_tail target n = gp_Henkin_auxiliary_formula n.

  Definition gp_Henkin_theory_conservation:
    classical_propositional_implicative_theorem
      General_Property gp_implies (gp_Henkin_theory) target →
  ⊢ target.

End The_associated_Henkin_extension_of_a_theory.

Definition abstract_first_order_model (M: General_Property → Type):=
  prod (type_valued_inclusion General_Property gp_theorem M)
       (prod (∀ x y:General_Property, M (((x -o y) -o x) -o x))
             (prod (∀ x y:General_Property,
                        prod (M (x -o y) → M x → M y)
                             ((M x → M y) → M (x -o y)))
                   (∀ f:General_Property,
                        prod
                          (M f → ∀ t:Term, M (gp_specialize f t))
                          ((∀ t:Term, M (gp_specialize f t)) → M f)

                   )
       )).
```

The following maps build the "drinker" in the "drinker's paradox", when f means "everyone drinks". if g is any first order formula, a witnessing function w, when applied to some "forall x, g", will reliably provide a counterexample for g whenever there is one i.e. forall x, g is not true.

```
Definition abstract_first_order_witnessing_function
          (M: General_Property → Type) (w: General_Property → Term):=
  ∀ f: General_Property, M (gp_specialize f (w f) -o f).
```

Not only we construct a model, but one where there is a witnessing function `Definition`
constructive_abstract_first_order_completeness_theorem_with_witnessing_function:

```
  {M: General_Property
      → Type
        &
        {f: General_Property
            → Term
              &
              prod
```

```
                    (abstract_first_order_model M )
                    (prod (abstract_first_order_witnessing_function M f)
                          (M target → ⊢ target))
    }}.

  Definition abstract_first_order_completeness_theorem_with_witnessing_function:
    (∀ (M: General_Property → Type) (w: General_Property → Term),
      abstract_first_order_model M → abstract_first_order_witnessing_function M w →
      M target) → ⊢ target.

  End Abstract_first_order_completeness.

End Abstract_classical_first_order_systems.
```