CS208 2020/21 Logic : Week 05 Tutorial

Try to complete the proofs below before the tutorials on Friday 23rd and Monday 26th.

How to use the proof editor

Proof commands

The blue boxes represent parts of the proof that are unfinished. The comment (in green) tells you what the current goal is: either the goal is unfocused: *{ goal: <some formula> }*, or it has a focus: *{ focus: <formula1>; goal: <formula2> }*. The commands that you can use differ according to which mode you are in. The commands correspond directly to the proof rules given in the Week 04 videos.

Unfocused mode

These rules can be used when the comment in the blue part looks like *{ goal: <formula> }*. These rules either act on the conclusion, or switch to focused mode (`use`).

- `introduce` *H* : can be used when the goal is an implication 'P → Q'. The name *H* is used to give a name to the new assumption P. The proof then continues proving Q with this new assumption. A green comment is inserted to say what the new named assumption is.
- `split` : can be used when the goal is a conjunction

'P ∧ Q'. The proof will split into two sub-proofs, one to prove the first half of the conjunction P, and one to prove the other half Q.

- `true` : can be used when the goal to prove is 'T' (true). This will finish this branch of the proof.
- `left` : can be used when the goal to prove is a disjunction 'P ∨ Q'. A new sub goal will be created to prove 'P'.
- `right` : can be used when the goal to prove is a disjunction 'P ∨ Q'. A new sub goal will be created to prove 'Q'.
- `not-intro` *H* : can be used when the goal is a negation '¬P'. The name *H* is used to give a name to the new assumption P. The proof then continues proving F (i.e. False) with this new assumption. A green comment is inserted to say what the new named assumption is.
- `use` *H* : can be used whenever there is no current focus. *H* is the name of some assumption that is available on this branch of the proof. Named assumptions come from uses of `introduce H`, `cases H1 H2`, `not-intro H`, and `unpack y H`.

Focused mode

These rules apply when there is a formula in focus. In this case, the comment in the blue part looks like: *{ focus: <formula1>; goal: <formula2> }*. These rules either act upon the formula in focus, or finish the proof when the focused formula is the same as the goal.

- `done` : can be used when the formula in focus is exactly the same as the goal formula.
- `apply` : can be used when the formula in focus is an implication 'P → Q'. A new subgoal to prove 'P' is generated, and the focus becomes 'Q' to continue the proof.

- `first` : can be used when the formula in focus is a conjunction ‘P ∧ Q’. The focus then becomes ‘P’, the first part of the conjunction, and the proof continues.
- `second` : can be used when the formula in focus is a conjunction ‘P ∧ Q’. The focus then becomes ‘Q’, the second part of the conjunction, and the proof continues.
- `cases` *`H1 H2`* : can be used then the formula in focus is a disjunction ‘P ∨ Q’. The proof will split into two halves, one for ‘P’ and one for ‘Q’. The two names *H1* and *H2* are used to name the new assumption on the two branches. Green comments are inserted to say what the new named assumptions are.
- `false` : can be used when the formula in focus is ‘F’ (false). The proof finishes at this point.
- `not-elim` : can be used when the formula in focus is a negation ‘¬P’. A new subgoal is generated to prove ‘P’.

Show / Hide proof tree

Theorem: A ∧ B ∧ C ⊢ (A ∧ B) ∧ C

Proof

{ assuming 'A ∧ B ∧ C' with name 'H' }

reset split:

- reset split:
  - reset use *H*,
    reset first,
    reset done.
  - reset use *H*,
    reset second,
    reset first,
    reset done.
- reset use *H*,
  reset second,
  reset second,
  reset done.

Proof Complete.

Show / Hide proof tree

Theorem: ⊢ (A ∨ B) → (A → C) → (B → C) → C

Proof

reset introduce *a*;

{ assuming 'A ∨ B' with name 'a' }

reset introduce *b*;

{ assuming 'A → C' with name 'b' }

reset introduce *c*;

{ assuming 'B → C' with name 'c' }

reset use *a*,

reset cases (1) *a* or (2) *b*:

cases (1) *a* or (2) *b*:

1. *{ assuming 'A' with name 'a' }*
   use *b*,
   apply with:
     use *a*,
     done.
   done.

2. *{ assuming 'B' with name 'b' }*
   use *a*,
   cases (1) *a* or (2) *b*:

   1. *{ assuming 'A' with name 'a' }*
      use *c*,
      apply with:
        use *b*,
        done.
      done.

   2. *{ assuming 'B' with name 'b' }*
      use *c*,
      apply with:
        use *b*,
        done.
      done.

Proof Complete.

Show / Hide proof tree

Theorem: $\vdash (A \vee B) \to (A \to C) \to (B \to D) \to (C \vee D)$

Proof

reset introduce *a*;

{ assuming 'A ∨ B' with name 'a' }

reset introduce *b*;

{ assuming 'A → C' with name 'b' }

reset introduce *c*;

{ assuming 'B → D' with name 'c' }

reset use *a*,

reset cases (1) *a* or (2) *b*:

1. *{ assuming 'A' with name 'a' }*
   reset left;
   reset use *b*,
   reset apply with:
   - reset use *a*,
   - reset done.

   reset done.

2. *{ assuming 'B' with name 'b' }*
   reset right;
   reset use *c*,
   reset apply with:
   - reset use *b*,
   - reset done.

   reset done.

Proof Complete.

Show / Hide proof tree

Theorem: ⊢ ((A ∨ B) → C) → ((A → C) ∧ (B → C))

Proof

reset introduce *h*;

{ assuming '(A ∨ B) → C' with name 'h' }

reset split:

- reset introduce *b*;
  { assuming 'A' with name 'b' }
  reset use *h*,
  reset apply with:
  reset left;
  reset use *b*,
  reset done.
  reset done.

- reset introduce *c*;
  { assuming 'B' with name 'c' }
  reset use *h*,
  reset apply with:
  reset right;
  reset use *c*,
  reset done.
  reset done.

Proof Complete.

Show / Hide proof tree

Theorem: A ∨ (B ∧ C) ⊢ (A ∨ B) ∧ (A ∨ C)

Proof

{ assuming 'A ∨ (B ∧ C)' with name 'H' }

reset use *H*,
reset cases (1) *a* or (2) *b*:

1. *{ assuming 'A' with name 'a' }*
   reset split:
   - reset left;
     reset use *a*,
     reset done.
   - reset left;
     reset use *a*,
     reset done.
2. *{ assuming 'B ∧ C' with name 'b' }*
   reset split:
   - reset right;
     reset use *b*,
     reset first,
     reset done.
   - reset right;
     reset use *b*,
     reset second,
     reset done.

Proof Complete.

Show / Hide proof tree

Theorem: ⊢ A → ¬¬A

Proof

reset introduce *h*;

{ assuming 'A' with name 'h' }

reset not-intro *a*;

{ assuming '¬A' with name 'a' }

reset use *a*,

reset refuted by:

- reset use *h*,
- reset done.

Proof Complete.

Show / Hide proof tree

Theorem: ⊢ (A ∨ ¬A) → ¬¬A → A

Proof

reset introduce *h*;

{ assuming 'A ∨ ¬A' with name 'h' }

reset introduce *k*;

{ assuming '¬¬A' with name 'k' }

reset use *h*,

reset cases (1) *a* or (2) *b*:

1. *{ assuming 'A' with name 'a' }*
   reset use *a*,
   reset done.

2. *{ assuming '¬A' with name 'b' }*
   reset use *k*,
   reset refuted by:
   - reset use *b*,
   - reset done.

Proof Complete.