Isabelle Cheatsheet (2016-1)

定理・定義

⟨*theory*⟩ ::=

- **theorem** theorem_name:
  fixes x_1 :: "type 1" **and** ⋯ **and** x_ℓ :: "type ℓ"
  assumes ass_name_1: "ass 1" **and** ⋯ **and** ass_name_m: "ass m"
  shows sub_name_1: "claim 1" **and** ⋯ **and** sub_name_n: "claim n"
  ⟨*proof*⟩
- 同義語：**lemma** = **proposition** = **corollary** = **theorem**
- **definition** f :: "type" **where** def_name: "f x_1 ⋯ x_n ≡ right"
- **abbreviation** f :: "type" **where** "f x_1 ⋯ x_n ≡ right"
- **fun** f_1 :: "type 1" **and** ⋯ **and** f_n :: "type n"
  where name_1: "left_1 = right_1"
  | ⋯
  | name_m: "left_m = right_m"
- **function** f :: "type"
  where name_1: "left_1 = right_1"
  | ⋯
  | name_n: "left_n = right_n"
  ⟨*proof*⟩
  termination ⟨*proof*⟩
  ⋯ **fun** で自動証明ができない場合に使用.
- **datatype** ('a1, ..., 'an) t =
  Const_1 ⟨*type*⟩* | ⋯ | Const_n ⟨*type*⟩*
- **declare** ⟨*fact*⟩ [⟨*attr*⟩*]
  既知の fact の属性を変更.

補助コマンド

- **thm** ⟨*fact*⟩+ ⋯ fact を表示
- **find_theorems** ⟨*pattern*⟩ ⋯ パターンにマッチする fact を検索
- **print_theorems** ⋯ 直前のコマンドで作られた fact を表示

ロケール，クラス

- **context**
  fixes x_1 :: "type 1" **and** ⋯ **and** x_n :: "type n"
  assumes name_1: "ass 1" **and** ⋯ **and** name_m: "ass m"
  begin ⟨*theory*⟩* **end**
- **locale** locale_name = ⟨*locale expression*⟩ +
  fixes x_1 :: "type 1" **and** ⋯ **and** x_n :: "type n"
  assumes name_1: "ass 1" **and** ⋯ **and** name_m: "ass m"
  begin ⟨*theory*⟩* **end**
- **class** class_name = ⟨*class expression*⟩ +
  fixes x_1 :: "type 1" **and** ⋯ **and** x_n :: "type n"
  assumes name_1: "ass 1" **and** ⋯ **and** name_m: "ass m"
  begin ⟨*theory*⟩* **end**
- **interpretation** label: locale_name "arg 1" ⋯ "arg n"
  rewrites name_1: "l_1 = r_1" **and** ⋯ **and** name_m: "l_m = r_m"
  ⟨*proof*⟩
- **sublocale** label: locale_name "arg 1" ⋯ "arg n" ⟨*proof*⟩
- **subclass** class_name ⟨*proof*⟩
- **instance** t :: (class_1, ⋯, class_n) class_name ⟨*proof*⟩
- **instantiation** t :: (class_1, ⋯, class_n) class_name
  begin
  ⟨*theory*⟩*
  instance ⟨*proof*⟩
  end
- **context** name **begin** ⟨*theory*⟩* **end**
  定義済みのロケール，クラスに言明を追加.
- 同義語：**theorem** (**in** name) ⋯ =
  context name **begin** **theorem** ⋯ **end**
  同様の記法が **definition**，**subclass** などにも適用可.

補助コマンド

- **class_deps** ⟨*class*⟩? ⟨*class*⟩? ⋯ クラスの依存関係をグラフ化

証明

⟨*proof*⟩ ::=

- **by** ⟨*method*⟩
- **apply** ⟨*method*⟩ ⟨*proof*⟩
- **proof** ⟨*method*⟩ ⟨*isar*⟩* **qed** ⟨*method*⟩?

同義語

- `.` = **by** `assumption`
- `..` = **by** `standard`
- **done** = **by**-
- **using** ⟨*fact*⟩+ ≒ **apply** (**insert** ⟨*fact*⟩+)
- **unfolding** ⟨*fact*⟩+ ≒ **apply** (**unfold** ⟨*fact*⟩+)
- **proof** ⟨*isar*⟩* = **proof** `standard` ⟨*isar*⟩*
- **by** ⟨*method*⟩ ⟨*method*⟩ = **proof** ⟨*method*⟩ **qed** ⟨*method*⟩

Isar

⟨*isar*⟩ ::=

- **have** `name_1: "claim 1" and ⋯ and name_`n`: "claim `n`"` ⟨*proof*⟩
  `"claim `i`"` を証明し `name_`i という名前を付ける．
- **show** `name_1: "claim 1" and ⋯ and name_`n`: "claim `n`"` ⟨*proof*⟩
  have と同様だがゴール（のうち n 個）を証明して discharge.
- **note** `name [`⟨*attr*⟩+`] =` ⟨*fact*⟩+
  既知の fact を言明.
- **assume** `name_1: "ass 1" and ⋯ and name_`n`: "ass `n`"`
  `"ass `i`"` を仮定し `name_`i という名前を付ける．以下の形のゴールの証明に使用.
  `"ass 1 ⟹ ⋯ ass `n` ⟹ claim"`
- **fix** `x_1 :: "type 1" and ⋯ and x_`n` :: "type `n`"`
  `x_`i を，型 `"type `i`"` の任意の値とする．以下の形のゴールの証明に使用.
  `"⋀x_1 ⋯ x_`n`. claim"`
- **define** `f :: "type" where name: "f x_1 ⋯ x_`n` ≡ right"`
  証明中でのみ有効な **definition**.
- **interpret** `locale_name "arg 1" ⋯ "arg `n`"` ⟨*proof*⟩
  証明中でのみ有効な **interpretation**.
- **obtain** `x_1 :: "type 1" and ⋯ and x_`n` :: "type `n`"`
  where `name_1: "claim 1" and ⋯ and name_`m`: "claim `m`"`
  ⟨*proof*⟩
  `"claim 1" ⋯ "claim `m`"`を満たす `x_1 ⋯ x_`n が存在することを証明し，以降の証明で利用.
- **case** `name: (Case_Name x_1 ⋯ x_`n`)`
  場合分け・帰納法中，`Case_Name` と名付けられたケースを考える．場合分けを表す仮定，帰納法の仮定が `name` (省略した場合 `Case_Name`) で参照可.
- **next**
  これまでに証明・仮定した fact などをすべてリセット．前提の異なるゴールの証明，別の場合分けに移る際に使用.
- `{` ⟨*isar*⟩+ `}`
  ローカルスコープ．スコープ最後の言明に，スコープ内で **assume** した仮定を前提とし，**fix** した変数を ⋀ で束縛をした命題を **have** で示したのと同じ.
- **let** `?v = "expression"`
  メタ変数 `?v` を定義.
- ⟨*isar*⟩ **also** ⋯ **also** ⟨*isar*⟩ **finally** ⟨*isar*⟩
  推移性を持つ関係や等式を並べてひとつの結論にまとめる．直前の言明の右辺を表す `"..."` が便利．$>, \geq$ などは左右が逆になるので注意.
- ⟨*isar*⟩ **moreover** ⋯ **moreover** ⟨*isar*⟩ **ultimately** ⟨*isar*⟩
  複数の言明をまとめて最後の言明の前提に加える.

同義語

- **from** ⟨*fact*⟩$^+$ **have** ⋯ ⟨*proof*⟩ = **have** ⋯ **using** ⟨*fact*⟩$^+$ ⟨*proof*⟩
  ただし前者のみ，直前の言明を `this` として参照できる．
- **then** = **from** `this`
- **with** ⟨*fact*⟩$^+$ = **from** `this` ⟨*fact*⟩$^+$
- **hence** = **then have**
- **thus** = **then show**

証明メソッド

⟨*method*⟩ ::=

非自動的，堅牢

- `rule` ⟨*fact*⟩$^+$
  1 つ目のゴールの結論を fact（のうち，適用可能などれか）の結論として導く．適用された fact の前提が新たなゴールに加わる．
- `fact` ⟨*fact*⟩$^+$
  1 つ目のゴールを fact（のうち，適用可能などれか）から直接導いて discharge.
- `insert` ⟨*fact*⟩$^+$
  すべてのゴールの前提に既知の fact を加える．
- `assumption`
  1 つ目のゴールの結論を前提から直接導いて discharge.
- `intro` ⟨*fact*⟩$^+$
  適用できる限り `(rule` ⟨*fact*⟩$^+$`)` を適用（1 回も適用できなければ失敗）．
- `elim` ⟨*fact*⟩$^+$
  適用できる限り除去規則を適用（1 回も適用できなければ失敗）．
- `cases "expression"`
  `"expression"` のデータ構造に応じた場合分け．
- `cases "expression 1"` ⋯ `"expression` n`" rule:` ⟨*fact*⟩
  場合分け規則を指定した場合分け．
- `induct x arbitrary: y_1` ⋯ `y_`n `rule:` ⟨*fact*⟩
  変数 `x` に関する帰納法を適用．帰納法の仮定において，`y_1` ⋯ `y_`n は任意となる．`rule:`を省略すれば構造帰納法．
  帰納法内では元の `x` は無関係となるため，`x` に関する必要な fact は事前に `insert` で前提に含めてから `induct` を適用する．
- `subst (abs)`$^?$ `(`n ⋯`)` ⟨*fact*⟩
  現在のゴールに書換え規則を強制適用．
- `unfold` ⟨*fact*⟩$^+$
  現在のゴールに 1 回以上，適用できる限り書換え規則を適用．
- `fold` ⟨*fact*⟩$^+$
  `unfold` の逆 (ただし，なぜかより保守的)．
- `atomize (full)`
  すべてのゴールを 1 つの HOL レベルの論理式にまとめる．複数のゴールに帰納法などを適用する場合に便利．

自動的

- `simp` {{`add` | `del` | `only`}: ⟨*fact*⟩$^+$}*
- `simp_all` {{`add` | `del` | `only`}: ⟨*fact*⟩$^+$}*
- `auto` {{`simp` [`del`] | {`intro`[`!`] | `elim`[`!`] | `dest`[`!`] | `rule del`}: ⟨*fact*⟩$^+$}*
- `force`, `fast`, `fastforce`, `blast`, `meson`, `metis`
  1 つ目のゴールを自動証明（証明できなければ失敗）．**sledgehammer** 等が見つけてきた場合のみ使い分ければよい．
- `linarith`, `arith`, `presburger`
  1 つ目のゴールに算術系に強い自動証明を適用．同上．
- `rule`
  現在のゴールに “標準” の導出規則を 1 回適用．メンテナンス上，適用される規則を明示する方がお勧め．
- `standard`
  現在のゴールに “標準” の証明メソッドを適用．同上．

合成

- (⟨*method*⟩)

  ⟨*method*⟩ が単語でない場合に **by** (⟨*method*⟩) などの形で必要.

- ⟨*method*⟩, ⟨*method*⟩

  ⟨*method*⟩ を順に適用.

- ⟨*method*⟩; ⟨*method*⟩

  1 つ目の ⟨*method*⟩ を適用して生成されたゴールすべてに 2 つ目の ⟨*method*⟩ を適用.

- ⟨*method*⟩? / ⟨*method*⟩* / ⟨*method*⟩+

  ⟨*method*⟩ を 1 回以下 / 0 回以上 / 1 回以上適用. 何回適用する意図なのか分からないため, 保守性が非常に悪い.

属性

`name: "claim"` の形の言明は `name [`⟨*attr*⟩$^+$`]: "claim"` として属性を指定可.
⟨*attr*⟩ ::=

- `simp` [`del`]
- `intro`[`!`|`?`]
- `elim`[`!`|`?`]
- `dest`[`!`|`?`]
- `code`

  プログラムに変換される等式を指定する. 仮定を持つコンテキストやロケール内で定義した関数は, コンテキスト外で `[code]` 宣言する必要がある (Isabelle 2016-1).

- `code_unfold`

  プログラム変換を行う前処理として適用する書換え規則を指定.

- `unfolded` ⟨*fact*⟩$^+$

  参照中の fact に書換えを適用した結果を表す.

- `folded` ⟨*fact*⟩$^+$

  `unfolded` の逆.

- `simplified`

  参照中の fact に `[simp]` 属性の書換え規則を適用した結果を表す.

- `symmetric`

  `fact_name` の結論が等式のとき, `fact_name[symmetric]` は等式の左辺と右辺を交換したもの.

- `OF fact_1` ⋯ `fact_`n

  参照中の fact が `"ass 1` $\Longrightarrow$ ⋯ `ass` n $\Longrightarrow$ `claim"` の形のとき, 各 `fact_`i が表す fact を `"ass` i`"` にマッチさせた結果の `"claim"` を表す. 参照中の fact の形に依存するため, 可変性に難が出る.

- `of "expr 1"` ⋯ `"expr` n`"`

  参照中の fact 中の schematic 変数を, 順に `"expr` i`"` で置き換えた結果を表す. 同上.

Isabelle/jEdit ショートカット

Ctrl-b	completion ヒント. tab で決定
Ctrl-e ↓	subscript
Ctrl-e ↑	superscript
`==>`	$\Longrightarrow$
`!!`	$\bigwedge$
`=>`	$\Rightarrow$
`-->`	$\longrightarrow$
`~`	$\neg$
`\noteq`	$\neq$
`>=`	$\geq$
`<=`	$\leq$
`->`	$\rightarrow$
`.>`	種々の左矢印
`<.`	種々の右矢印
`\forall`	$\forall$
`\exists`	$\exists$