CalcCheck Structured Proofs

Simple Induction

```
By induction on `var : Ty`:
  Base case:
    ?
  Induction step:
      ?
    ... Induction hypothesis ...
      ?
```

Making base case, induction step, and induction hypothesis explicit:

```
By induction on `var : Ty`:
  Base case `?`:
    ?
  Induction step `?`:
      ?
    ... Induction hypothesis `?` ...
      ?
```

Remember that in nested inductions, induction hypotheses always need to be made explicit!

Induction pattern for sequences (choose x wisely!):

```
Theorem: P
Proof:
  By induction on `xs : Seq A`:
    Base case `P[xs ≔ ϵ]`:
        ?
    Induction step `∀ x : A • P[xs ≔ x ◃ xs]`:
      For any `x`:
        ?
```

These can also be used for proving theorems of shape

$$\forall\ \texttt{var}\ :\ \texttt{Ty} \bullet P$$

by induction on precisely that universally-quantified variable, that is, “on `var : Ty`:”.
The induction hypothesis is then P.

Example for sequences:

```
Theorem: ∀ xs : Seq A • P
Proof:
  By induction on `xs : Seq A`:
    Base case `P[xs ≔ ϵ]`:
        ?
    Induction step `∀ x : A • P[xs ≔ x ◃ xs]`:
      For any `x`:
        ?
```

Facts that can be shown by “Evaluation”

Only where enabled (and never can contain variables):
Fact \`$6 \cdot 7 = 42$\`, Fact \`$6 > 7 \equiv \mathit{false}$\`

Assuming the Antecedent

```
Assuming `p`, `q`:
    ?
  ... Assumption `p` ...
    ?
```

```
Assuming `p` and using with ...:
    ?
  ... Assumption `p` ...
    ?
```

Assuming a Witness

Assuming witness \`*x*\` satisfying \`*P*\`:
Proof for Q using **Assumption** \`*P*\`

proves “$(\mathit{exists}\ x \bullet P) \Rightarrow Q$” (if $\neg \mathit{occurs}(`x', `P')$).

Assuming witness \`*x*\` satisfying \`*P*\` by *Hint*:
Proof for Q using **Assumption** \`*P*\`

proves “Q” if the hint proves “$(\mathit{exists}\ x \bullet P)$” (if $\neg \mathit{occurs}(`x', `P')$).

Proving Universal Quantifications

Proving $(\forall\ v : \mathbb{N} \bullet P)$:

For any \`$v : \mathbb{N}$\`:
Proof for P

Proving $(\forall\ v : \mathbb{N}\ \mid\ R \bullet P)$:

For any \`$v : \mathbb{N}$\` **satisfying** \`*R*\`:
Proof for P using **Assumption** \`*R*\`

Case Analysis

```
By cases: `p`, `q`, `r`
  Completeness:
    ?
  Case `p`:
      ?
    ... Assumption `p` ...
      ?
  ...
```

Subproofs

```
  ?
≡⟨ Subproof for `...`:
     ⟪ proof indented as far as needed
       to avoid parse error! ⟫
  ⟩
  ?
```

```
   ⟪ Calculation ending in `P` ⟫
Proof for this:
  ⟪ Proof for `P` ⟫
```

is the same, **but with different indentation!**, as:

```
   ⟪ Calculation ending in `P` ⟫
≡⟨ Subproof for `P`:
     ⟪ Proof for `P` ⟫
  ⟩
  true
```

Theorems Used as Proof Methods (Example)

```
Using “Mutual implication”:
  Subproof for `...  ⇒  ...`:
    ?
  Subproof for `...  ⇒  ...`:
    ?
```

Side Proofs

```
Side proof for `P`:
  ?
Continuing with goal `?`:
    ?
  ... local property `P` ...
    ?
```

(Multiple side proofs at the same indentation are possible, and can use any previously-established local property.)

Disabling Hints Producing Time-outs

Add “?, ” at the beginning of the hint:

```
    ≡⟨ ?, “Golden rule” ⟩
```

Selected CalcCheckWeb Key Bindings

(See Getting Started with CalcCheckWeb for the complete listing.)

The following key bindings work the same in **both edit and command modes**:

`Ctrl-Enter` performs a syntax check on the contents of all code cells before and up to the current cell.

`Ctrl-Alt-Enter` performs proof checks (if enabled) on the contents of all code cells before and up to the current cell. **During Midterm 1:** Same as `Ctrl-Enter`.

`Shift-Alt-RightArrow` enlarges the width of the current code cell entry area by a small amount

`Ctrl-Shift-Alt-RightArrow` enlarges the width of the current code cell entry area by a large amount

`Shift-Alt-LeftArrow` reduces the width of the current code cell entry area by a small amount

`Ctrl-Shift-Alt-LeftArrow` reduces the width of the current code cell entry area by a large amount

`Ctrl-Shift-v` (for visible spaces) toggles display of initial spaces on each line as "␣" characters.

ONLY if you are logged in via Avenue:

`Ctrl-Shift-s` saves the notebook on the server.
To be safest, use in command mode, e.g. after clicking on the area of a code box where the line number would be displayed.
Check the pop-up whether it is the CalcCheck-Web pop-up saying "…Notebook saved to …".
(Links for reloading the last three saved versions are displayed when you view the notebook again.)

In **edit mode**, you have the following **key bindings**:

`Esc` enters command mode

`Alt-i` ***or*** `Alt-SPACE` inserts one space in the current line and in all non-empty lines below it, until a line is encountered that is not indented more than to the cursor position.

`Alt-BACKSPACE` deletes **only a space character** to the left of the current cursor position, and also from lines below it, until a line is encountered that is not indented at least to the cursor position.

`Alt-DELETE` deletes **only a space character** to the right of the current cursor position, and also from lines below it, until a line is encountered that is not indented more than to the cursor position.

The last three bindings also work with `Shift`.

Some important symbols:

Symbol	Key sequence(s)
≡	`\equiv`, `\==`
≢	`\nequiv`
¬	`\lnot`
∧	`\land`
∨	`\lor`
⇒	`\implies`, `\=>`
⇐	`\follows`
≠	`\neq`
∀	`\forall`
∃	`\exists`
∑	`\sum`
∏	`\product`
❙	`\with`
•	`\spot`
↓	`\min`
↑	`\max`
𝔹	`\BB`, `\bool`
ℕ	`\NN`, `\nat`
ℤ	`\ZZ`, `\int`
⨾	`\;;`
∈	`\in`
ℙ	`\PP`, `\powerset`
~	`~`
∪	`\union`
∩	`\intersection`
⋃	`\bigunion`
⋂	`\bigintersection`
⊥	`\bot`
⊤	`\top`
⇨	`\pseudocompl`
⊆	`\subseteq`, `\(=`
⊇	`\supseteq`, `\)=`
⊂	`\subset`
⊃	`\supset`
U	`\universe`

Symbol	Key sequence(s)
×	`\times`
↔	`\rel`
⦗	`\lrel`, `\((`
⦘	`\rrel`, `\))`
⨾	`\rcomp`, `\fcomp`, `\;;`
˘	`\converse`, `\u{}`
⁺	`\^+`
*	`*`
╱	`\lres`
╲	`\rres`
◁	`\drestr`
⩤	`\ndrestr`
▷	`\rrestr`
⩥	`\nrrestr`
⦇	`\limg`
⦈	`\rimg`
⊕	`\oplus`

Symbol	Key sequence(s)
ϵ	`\eps`, `\emptyseq`
◃	`\cons`
▹	`\snoc`
⌢	`\catenate`
⟷	`\Rel`
→	`\tfun`
⇸	`\pfun`
↣	`\tinj`
⤔	`\pinj`
↠	`\tsurj`
⤀	`\psurj`
⤖	`\tbij`
⤗	`\pbij`
⦃	`\lbag`
⦄	`\rbag`
⋿	`\inbag`
[-	`\[-`
-]	`\]-`
: =	`:=` (assignment commands)
:=	`\:=`, `\becomes` (substitutions)

Table of Precedences

- [x := e] (textual substitution) **(highest precedence)**
- _⦇_⦈ _! _*
- unary prefix operators +, −, ¬, #, ~, ℙ, suc_
- __ (function application), @
- **
- · / ÷ **mod** **gcd**
- ⨾ (relation composition) ╱ ╲
- + − ∪ ∩ × ∘ ⊕ ⇨ ◁ ⩤ ▷ ⩥
- ↔ (relation type)
- → (function type)
- ↓ ↑
- #
- ◃ ▹ ⌢
- = ≠ < > ∈ ⊂ ⊆ ⊃ ⊇ | _⦗_⦘_ **(conjunctional)**
- ∨ ∧
- ⇒ ⇏ ⇐ ⇍ _⇒[-_-]_ _[-_-]⇐_
- ≡ ≢
- := (assignment command)
- ⨾ (command sequencing) **(lowest precedence)**

All non-associative binary infix operators associate to the left, except **, ◃, ⇒, →, which associate to the right.