Theorems / lemmas are of the form:

```
theorem theorem_name:
  fixes x::"type"
  ... assumes "assm1" and "assm2" ...
  shows "statement"
  ...
```

(Grey parts are optional. Assumptions and the (goal)statement must be of type bool.)
Structural induction proofs have the form:

\[
\text{proof (induct } x \text{ rule: rule\_name)} \\
\text{case (Name1 } y_1\ldots) \\
\ldots \\
\text{next} \\
\text{case (Name2 } y_2\ldots) \\
\ldots \\
\text{qed}
\]

(For each term constructor there is a case.)
Each case is of the form:

case (Name x...) 
   have n1: "statement1" by justification 
   have n2: "statement2" by justification 
   ... 
   show "statement" by justification

Useful, but less readable: show ?case

Justifications can be: using ... by ...
Justifications

- Omitting proofs
  - sorry

- Automated proofs
  - by simp: simplification (equations)
  - by auto: simplification & proof search
  - by blast: proof search
  ...
Any questions about yesterday's lecture?
Any questions about yesterday's lecture?

lemma lookup_test:
assumes "(x,y) ∈ set xs"
shows "lookup x xs = Some y"
Any questions about yesterday's lecture?

```ml
lemma lookup_test:
  assumes "(x, y) ∈ set xs"
  shows "lookup x xs = Some y"
```

Bruce Schneier on “The Security Mindset”:

SmartWater is a liquid with a unique identifier linked to a particular owner. The idea is for me to paint this stuff on my valuables as proof of ownership. — I think a better idea would be for me to paint it on your valuables, and then call the police. ;o)
Any questions about yesterday's lecture?

Today:

- another look at inductions
- inductive predicates
List Difference

Define a function that deletes the elements of the second list from the first.

Then prove

\textbf{lemma} \quad \textasciitilde \text{rev} (x_{s} - y_{s}) = (\text{rev} x_{s}) - y_{s}
List Difference

Define a function that deletes the elements of the second list from the first.

Then prove

\textbf{lemma} “\texttt{rev (xs - ys) = (rev xs) - ys}”

The point is that the induction is not so simple (do it by hand first).
List Difference

fun
del :: "'a list ⇒ 'a ⇒ 'a list"
where
  "del [] x = []"
| "del (y#xs) x = (if x=y then del xs x else y#(del xs x))"

fun
delete :: "'a list ⇒ 'a list ⇒ 'a list" ("- - -")
where
  "ys - [] = ys"
| "ys - (x#xs) = (del ys x) - xs"

lemma
shows "[1,2,3] - [1,3] = [(2::nat)]" by simp
Proofs

lemma del_append:
  shows "(del (xs@ys) x) = (del xs x)@(del ys x)"
by (induct xs) (auto)

lemma del_rev:
  shows "rev (del xs x) = del (rev xs) x"
by (induct xs) (auto simp add: del_append)

lemma delete_rev:
  shows "\( \forall xs. \ rev (xs - ys) = (rev xs) - ys \)"
by (induct ys) (simp_all add: del_rev)

The induction only goes through if we generalise over \( xs \).
Distinctness Function

- We defined yesterday:

```plaintext
fun "distinct" :: "a list ⇒ bool"
where
  "distinct [] = True"
| "distinct (x#xs) = (x ∉ set xs ∧ distinct xs)"
```

distinct [3,4] expands to

```
3 ∉ set [4] ∧ 4 ∉ set [] ∧ True
```
An Inductive Definition

inductive
  distinct' :: "a list ⇒ bool"
where
  d1: "distinct' []"
| d2: "[x ∉ set xs; distinct' xs] ⟷ distinct' (x#xs)"

axiom: conclusion

in general: prem₁ ... premₙ conclusion
Induction Principle

Inductive Definitions:

\[ \text{prem}_1 \ldots \text{prem}_n \]
\[ \text{concl} \]

Rule Inductions:

1.) Assume the property for the premises. Also assume side-conditions.
2.) Show the property for the conclusion.

In other words: the property has to be preserved by the rules.
A Proof by Rule Induction

lemma
  assumes a: "distinct' xs"
  shows "remdups xs = xs"
using a
proof (induct)
  case d1
  show "remdups [] = []" by simp
next
  case (d2 x xs)
  have b: "x \notin set xs" by fact
  have ih: "remdups xs = xs" by fact
  show "remdups (x#xs) = (x#xs)" using b ih by simp
qed

d1 and d2 are the names of the introduction rules.
Point to Take Home

Inductive Definitions:

\[ \text{prem}_1 \ldots \text{prem}_n \]
\[ \text{concl} \]

Rule Inductions:

1.) Assume the property for the premises. Also assume side-conditions.
2.) Show the property for the conclusion.

\[ x \notin \text{set } xs \quad \text{distinct'} xs \]
\[ \text{distinct'} (x \neq xs) \]