Prove logic formulas ... to prove programs

```
fun nth :: "nat => 'a list => 'a"
  where
  "nth 0 (x#_)=x" |
  "nth x (y#ys)= (nth (x - 1) ys)"

fun index :: "'a => 'a list => nat"
  where
  "index x (y#ys)= (if x=y then 1 else 1+(index x ys))"

lemma nth_index: "nth (index e l) l= e"
```

How to prove the lemma \texttt{nth\_index}? (Recall that everything is logic!)

What we are going to prove is thus a formula of the form:

\[
\text{Theory of lists} \land \text{Equations for nth} \land \text{Equations for index} \rightarrow \text{nth\_index}
\]

Finding counterexamples

\textbf{Why?} because «90% of the theorems we write are false!»

- Because this is not what we want to prove!
- Because the formula is imprecise
- Because the function is false
- Because there are typos...

\textbf{Before starting a proof, always first search for a counterexample!}

Isabelle/HOL offers two counterexample finders:

- \texttt{nitpick}: uses finite model enumeration
  - Works on any logic formula, any type and any function
  - Rapidly exhausted on large programs and properties
- \texttt{quickcheck}: uses random testing, exhaustive testing and narrowing
  - Does not covers all formula and all types
  - Scales well even on large programs and complex properties
Nitpick

To build an interpretation \( I \) such that \( I \not\models \phi \) (or \( I \models \neg \phi \)) .... nitpick

nitpick principle: build an interpretation \( I \models \neg \phi \) on a finite domain \( D \)
- Choose a cardinality \( k \)
- Enumerate all possible domains \( D \) of size \( k \) for all types \( \tau \) in \( \neg \phi \)
- Build all possible interpretations of functions in \( \neg \phi \) on all \( D \).
- Check if one interpretation satisfy \( \neg \phi \) (this is a counterexample for \( \phi \))
- If not, there is no counterexample on a domain of size \( k \) for \( \phi \)

nitpick algorithm:
- Search for a counterexample to \( \phi \) with cardinalities 1 upto \( n \)
- Stops when \( I \) such that \( I \models \neg \phi \) is found (counterex. to \( \phi \)), or
- Stops when maximal cardinality \( n \) is reached (10 by default), or
- Stops after 30 seconds (default timeout)

Exercise 1

By hand, iteratively check if there is a counterexample of cardinality 1, 2, 3 for the formula \( \phi \), where \( \phi \) is \( \text{length } la <= 1 \)

Remark 1

- The types occurring in \( \phi \) are \( 'a \) and \( 'a \) list
- One possible domain \( D_a \) of cardinality 1: \( \{a_1\} \)
- One possible domain \( D_a \) list of cardinality 1: \( \{[[]]\} \)

Domains have to be subterm-closed, thus \( \{[a_1]\} \) is not valid
- One possible domain \( D_a \) of cardinality 2: \( \{a_1, a_2\} \)
- Two possible domains \( D_a \) list of cardinality 2: \( \{[], [a_1]\} \) and \( \{[], [a_2]\} \)
- One possible domain \( D_a \) of cardinality 3: \( \{a_1, a_2, a_3\} \)
- Twelve possible domains \( D_a \) list of cardinality 3: \( \{[], [a_1], [a_1, a_1]\}, \{[], [a_1], [a_1, a_2]\}, \{[], [a_1], [a_3, a_1]\}, \ldots \)

Nitpick (II)

Nitpick options:
- \text{timeout=t}, set the timeout to \( t \) seconds (timeout=none possible)
- \text{show_all}, displays the domains and interpretations for the counterex.
- \text{expect=s}, specifies the expected outcome where \( s \) can be none (no counterexample) or genuine (a counterexample exists)
- \text{card=i-j}, specifies the cardinalities to explore

For instance:

nitpick [timeout=120, show_all, card=3-5]

Exercise 2

- Explain the counterexample found for \( \text{rev } l = 1 \)
- Is there a counterexample to the lemma \( \text{nth_index} \)?
- Correct the lemma and definitions of \( \text{index} \) and \( \text{nth} \)
- Is the lemma append_commute true? really?

Quickcheck

To build an interpretation \( I \) such that \( I \models \neg \phi \) (or \( I \models \neg \phi \)) .... quickcheck

quickcheck principle: test \( \phi \) with automatically generated values of size \( k \)

Either with a generator
- Random: values are generated randomly (Haskell’s QuickCheck)
- Exhaustive: (almost) all values of size \( k \) are generated
- Narrowing: like exhaustive but taking advantage of symbolic values

No exhaustiveness guarantee!! with any of them

quickcheck algorithm:
- Export Haskell code for functions and lemmas
- Generate test values of size 1 upto \( n \) and, test \( \phi \) using Haskell code
- Stops when a counterexample is found, or
- Stops when max. size of test values has been reached (default 5), or
- Stops after 30 seconds (default timeout)
Quickcheck (II)

quickcheck options:
- `timeout=t`, set the timeout to `t` seconds
- `expect=s`, specifies the expected outcome where `s` can be `no_counterexample`, `counterexample` or `no_expectation`
- `tester=tool`, specifies generator to use where `tool` can be `random`, `exhaustive` or `narrowing`
- `size=i`, specifies the maximal size of testing values

For instance: quickcheck [tester=narrowing,size=6]

Exercise 3 (Using `quickcheck`)
- find a counterexample on TP0 (`solTP0.thy`, `CM4_TP0`)

Remark 2
Quickcheck first generates values and then does the tests. As a result, it may not run the tests if you choose bad values for size and timeout.

How do proofs look like?
A formula of the form $A_1 \land \ldots \land A_n$ is represented by the proof goal:

```
goal (n subgoals):
1. $A_1$
...  
n. $A_n$
```

Where each subgoal to prove is either a formula of the form

- $A \land \ldots \land A_n \implies B$ meaning prove $B$, or
- $\neg A \land \ldots \land A_n \implies B$ meaning prove $B \implies C$, or
- $\neg A \land \ldots \land A_n \land B \implies \ldots \land B_n \implies C$ meaning prove $B_1 \land \ldots \land B_n \implies C$

and $\land x_1 \ldots x_n$ means that those variables are local to this subgoal.

Example 1 (Proof goal)

```
goal (2 subgoals):
1. member [] e \implies nth (index e []) [] = e
2. \forall a 1. e \neq a \implies member (a # l) e \implies 
   \neg member l e \implies nth (index e l) l = e
```

Proof by cases
... possible when the proof can be split into a finite number of cases

Proof by cases on a formula $F$
Do a proof by cases on a formula $F$ ..............apply (case_tac "F")
Splits the current goal in two: one with assumption $F$ and one with $\neg F$

Example 2 (Proof by case on a formula)
With apply (case_tac "F::bool")
goal (1 subgoal): becomes goal (2 subgoals):
1. $A \implies B$
2. $\neg F \implies A \implies B$

Exercise 4
Prove that for any natural number $x$, if $x < 4$ then $x \times x < 10$. 
Proof by induction (II)

Proof by cases (II)

To prove \( P \) on \( \cdot \) of an enumerated type of size \( n \)

Proof by cases on a variable \( x \) of an enumerated type

Do a proof by cases on a variable \( x \) ............... apply (case_tac "x")

Splits the current goal into \( n \) goals, one for each case of \( x \).

Example 3 (Proof by case on a variable of an enumerated type)

In Course 3, we defined datatype color = Black | White | Grey

With apply (case_tac "x")

\[
\begin{align*}
\text{goal (1 subgoal):} & \quad \text{becomes} \\
1. P (x::color) & \quad \text{goal (3 subgoals):} \\
2. x = \text{Black} \implies P x & \quad 1. x = \text{Black} \implies P x \\
3. x = \text{White} \implies P x & \quad 2. x = \text{White} \implies P x \\
3. x = \text{Grey} \implies P x & \quad 3. x = \text{Grey} \implies P x
\end{align*}
\]

Exercise 5

On the color enumerated type or course 3, show that for all color \( x \) if the
notBlack \( x \) is true then \( x \) is either white or grey.

<table>
<thead>
<tr>
<th>T. Genet (ISTIC/IRISA)</th>
<th>ACF-4</th>
<th>13 / 26</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proof by induction (II)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P([[]) \land \forall e \in \cdot a. \forall l \in \cdot a \ \text{list}. (P(l) \implies P(e#l)) \implies \forall l \in \cdot a \ \text{list}.P(l) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 5 (Proof by induction on lists)

Recall the definition of the function append:

1. \( \text{append \ }[[]\ ] \ l \ = \ l \)
2. \( \text{append \ }([x\#xs]) \ l \ = \ x\#(\text{append \ }xs \ l) \)

To prove \( \forall l \in \cdot a \ \text{list}.(\text{append \ }([[]) = l) \) by induction on \( l \), we prove:

1. \( \forall e \in \cdot a. \forall l \in \cdot a \ \text{list}. (\text{append \ }([[]) = l) \), proven by the first equation of append
2. \( \forall e \in \cdot a. \forall l \in \cdot a \ \text{list}. (\text{append \ }([[]) = l \implies (\text{append \ }e\#l([[]) = (e\#l)) \)

An induction proof on \( l \), instead of \( l’ \), is more likely to succeed:

- an induction on \( l \) will require to prove:
  \( l \geq (\text{length \ } (\text{append \ }e\#l)) \)
- an induction on \( l’ \) will require to prove:
  \( l’ \geq (\text{length \ } (\text{append \ }e\#l)) \)

Example 6 (Choice of the induction variable)

(1) \( \text{append \ }[[]\ ] \ l \ = \ l \)
(2) \( \text{append \ }([x\#xs]) \ l \ = \ x\#(\text{append \ }xs \ l) \)

To prove \( \forall l_1 l_2 \in \cdot a \ \text{list}.(\text{length \ } (\text{append \ }l_1 l_2)) \geq (\text{length \ } l_2) \)

An induction proof on \( l_1 \), instead of \( l_2 \), is more likely to succeed:

- an induction on \( l_1 \) will require to prove:
  \( (\text{length \ } (\text{append \ }e\#l)) \geq (\text{length \ } l_2) \)
- an induction on \( l_2 \) will require to prove:
  \( (\text{length \ } (\text{append \ }l_1 e\#l_2)) \geq (\text{length \ } (e\#l_2)) \)

Proof by induction (II)

«Properties on recursive functions need proofs by induction»

Recall the basic induction principle on naturals:

\[
P(0) \land \forall x \in \mathbb{N}. (P(x) \implies P(x+1)) \implies \forall x \in \mathbb{N}. P(x)
\]

All recursive datatype have a similar induction principle, e.g. \( \cdot a \) lists:

\[
P([[]]) \land \forall e \in \cdot a. \forall l \in \cdot a \ \text{list}.(P(l) \implies P(e\#l)) \implies \forall l \in \cdot a \ \text{list}.P(l)
\]

Example 4

datatype \( \cdot a \ \text{binTree} = \text{Leaf} \mid \text{Node} \cdot a \ \text{"a binTree}\ " \cdot a \ \text{binTree} \)

\[
P(\text{Leaf}) \land \forall e \in \cdot a. \forall t1 t2 \in \cdot a \ \text{binTree}. (P(t1) \land P(t2) \implies P(\text{Node} e t1 t2)) \implies \forall t \in \cdot a \ \text{binTree}.P(t)
\]

Example 5 (Proof by induction on lists)

Recall the basic induction principle on naturals:

\[
P(0) \land \forall x \in \mathbb{N}. (P(x) \implies P(x+1)) \implies \forall x \in \mathbb{N}. P(x)
\]

All recursive datatype have a similar induction principle, e.g. \( \cdot a \) lists:

\[
P([[]]) \land \forall e \in \cdot a. \forall l \in \cdot a \ \text{list}.(P(l) \implies P(e\#l)) \implies \forall l \in \cdot a \ \text{list}.P(l)
\]

Example 4

datatype \( \cdot a \ \text{binTree} = \text{Leaf} \mid \text{Node} \cdot a \ \text{"a binTree}\ " \cdot a \ \text{binTree} \)

\[
P(\text{Leaf}) \land \forall e \in \cdot a. \forall t1 t2 \in \cdot a \ \text{binTree}. (P(t1) \land P(t2) \implies P(\text{Node} e t1 t2)) \implies \forall t \in \cdot a \ \text{binTree}.P(t)
\]

Example 5 (Proof by induction on lists)

Recall the basic induction principle on naturals:

\[
P(0) \land \forall x \in \mathbb{N}. (P(x) \implies P(x+1)) \implies \forall x \in \mathbb{N}. P(x)
\]

All recursive datatype have a similar induction principle, e.g. \( \cdot a \) lists:

\[
P([[]]) \land \forall e \in \cdot a. \forall l \in \cdot a \ \text{list}.(P(l) \implies P(e\#l)) \implies \forall l \in \cdot a \ \text{list}.P(l)
\]

Example 4

datatype \( \cdot a \ \text{binTree} = \text{Leaf} \mid \text{Node} \cdot a \ \text{"a binTree}\ " \cdot a \ \text{binTree} \)

\[
P(\text{Leaf}) \land \forall e \in \cdot a. \forall t1 t2 \in \cdot a \ \text{binTree}. (P(t1) \land P(t2) \implies P(\text{Node} e t1 t2)) \implies \forall t \in \cdot a \ \text{binTree}.P(t)
\]
Proof by induction: apply (induct x) (II)

Exercise 6
Recall the datatype of binary trees we defined in lecture 3. Define and prove the following properties:

1. If member x t, then there is at least one node in the tree t.
2. Relate the fact that x is a sub-tree of y and their number of nodes.

Exercise 7
Recall the functions sumList, sumNat and makeList of lecture 3. Try to state and prove the following properties:

1. Relate the length of list produced by makeList i and i
2. Relate the value of sumNat i and i
3. Give and try to prove the property relating those three functions

Proof by induction: generalize the goals
By default apply induct may produce too weak induction hypothesis

Example 7
When doing an apply (induct x) on the goal P (x::nat) (y::nat) goal (2 subgoals):
1. P 0 y
2. \( \forall x. P x y \Rightarrow P (\text{Suc} x) y \)

Example 8
With apply (induct x arbitrary:y) on the same goal goal (2 subgoals):
1. \( \forall y. P 0 y \)
2. \( \forall x y. P x y \Rightarrow P (\text{Suc} x) y \)

Exercise 8
Prove the sym lemma on the leq function.

Proof by induction: induction principles
Recall the basic induction principle on naturals:
\[ P(0) \land \forall x \in \mathbb{N}. (P(x) \rightarrow P(x + 1)) \rightarrow \forall x \in \mathbb{N}. P(x) \]

In fact, there are infinitely many other induction principles

- \( P(0) \land P(1) \land \forall x \in \mathbb{N}. ((x > 0 \land P(x)) \rightarrow P(x + 1)) \rightarrow \forall x \in \mathbb{N}. P(x) \)
- ...
- Strong induction on naturals
  \( \forall x, y \in \mathbb{N}. ((y < x \land P(y)) \rightarrow P(x)) \rightarrow \forall x \in \mathbb{N}. P(x) \)
- Well-founded induction on any type having a well-founded order \(<<\)
  \( \forall x, y. ((y << x \land P(y)) \rightarrow P(x)) \rightarrow \forall x. P(x) \)

Proof by induction: induction principles (II)
Apply an induction principle adapted to the function call \((f x y z)\)
\[ \ldots \ldots \ldots \ldots \ldots \ldots \cdot \text{apply (induct x y z rule:f.induct)} \]
Apply strong induction on variable \(x\) of type \(\mathbb{N}\)
\[ \ldots \ldots \ldots \ldots \ldots \ldots \cdot \text{apply (induct x rule:nat_less_induct)} \]
Apply well-founded induction on a variable \(x\)
\[ \ldots \ldots \ldots \ldots \ldots \ldots \cdot \text{apply (induct x rule:wf_induct)} \]

Exercise 9
Prove the lemma on function \(\text{div2}\).
Combination of decision procedures auto and simp

Automatically solve or simplify all subgoals ................. apply auto
apply auto does the following:
- Rewrites using equations (function definitions, etc)
- Applies a bit of arithmetic, logic reasoning and set reasoning
- On all subgoals
  - Solves them all or stops when stuck and shows the remaining subgoals

Automatically simplify the first subgoal ................. apply simp
apply simp does the following:
- Rewrites using equations (function definitions, etc)
- Applies a bit of arithmetic
- on the first subgoal
  - Solves it or stops when stuck and shows the simplified subgoal

Example 9
Switch on tracing and try to prove the lemma:
```plaintext
lemma "(index (1::nat) [3,4,1,3]) = 2"
using [[simp_trace=true]]
apply auto
```

Combination of decision procedures auto and simp (II)

Want to know what those tactics do?
- Add the command using [[simp_trace=true]] in the proof script
- Look in the output buffer

Sledgehammer

«Sledgehammers are often used in destruction work...»

Sledgehammer

«Solve theorems in the Cloud»

Architecture:

```
Isabelle/HOL  + relevant definitions and lemmas  External ATPs
          ------------>                     1
                        Proof (click on it)
```

Prove the first subgoal using state-of-the-art ATPs ....... sledgehammer
- Call to local or distant ATPs: SPASS, E, Vampire, CVC4, Z3, etc.
- Succeeds or stops on timeout (can be extended, e.g. [timeout=120])
- Provers can be explicitly selected (e.g. [provers= z3 spass])
- A proof consists of applications of lemmas or definition using the Isabelle/HOL tactics: metis, smt, simp, fast, etc.

1Automatic Theorem Provers
2See http://www.tptp.org/CASC/.
Remark 3
By default, sledgehammer does not use all available provers. But, you can remedy this by defining, once for all, the set of provers to be used:

```
sledgehammer_params [provers=cvc4 spass z3 e vampire]
```

Exercise 10
Finish the proof of the property relating \( \text{nth} \) and \( \text{index} \)

Exercise 11
Recall the functions \texttt{sumList}, \texttt{sumNat} and \texttt{makeList} of lecture 3. Try to state and prove the following properties:

1. Prove that there is no repeated occurrence of elements in the list produced by \texttt{makeList}
2. Finish the proof of the property relating those three functions

Hints for building proofs in Isabelle/HOL

When stuck in the proof of \texttt{prop1}, add relevant intermediate lemmas:

1. In the file, define a lemma \texttt{before} the property \texttt{prop1}
2. Name the lemma (say \texttt{lem1}) (to be used by sledgehammer)
3. Try to find a counterexample to \texttt{lem1}
4. If no counterexample is found, close the proof of \texttt{lem1} by \texttt{sorry}
5. Go back to the proof of \texttt{prop1} and check that \texttt{lem1} helps
6. If it helps then prove \texttt{lem1}. If not try to guess another lemma

To build correct theories, do not confuse \texttt{oops} and \texttt{sorry}:

1. Always close an unprovable property by \texttt{oops}
2. Always close an unfinished proof of a provable property by \texttt{sorry}

Example 10 (Everything is provable using contradictory lemmas)
We can prove that \( 1 + 1 = 0 \) using a false lemma.