

- nitpick
- quickcheck
- **2** Proving true formulas
 - Proof by cases: apply (case_tac x)
 - Proof by induction: apply (induct x)
 - Combination of decision procedures: apply auto and apply simp
 - Solving theorems in the Cloud: sledgehammer

Acknowledgements: some material is borrowed from T. Nipkow's lectures and from <u>Concrete Semantics</u> by Nipkow and Klein, Springer Verlag, 2016.

More details (in french) about those proof techniques can be found in:

- http://people.irisa.fr/Thomas.Genet/ACF/TPs/pc.thy
- CM4 video and "Principes de preuve avancés" video

T. Genet (ISTIC/IRISA)

3 / 27

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 nth_index? (Recall that everything is logic!)

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



Finding counterexamples

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...

T. Genet (ISTIC/IRISA)

Before starting a proof, always first search for a counterexample!

Isabelle/HOL offers two counterexample finders:

- nitpick: uses finite model enumeration
 - + Works on any logic formula, any type and any function
 - Rapidly exhausted on large programs and properties
- 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

<code>nitpick</code> principle: build an interpretation $I\models\neg\phi$ on a finite domain D

- Choose a cardinality k
- Enumerate all possible domains $D_{ au}$ of size ${\it k}$ for all types au in $\neg\phi$
- Build all possible interpretations of functions in $\neg\phi$ on all D_τ
- Check if one interpretation satisfy $\neg \phi$ (this is a counterexample for ϕ)
- If not, there is no counterexample on a domain of size ${\it k}$ for ϕ

nitpick algorithm:

- Search for a counterexample to ϕ with cardinalities 1 upto n
- Stops when I such that $I \models \neg \phi$ is found (counterex. to ϕ), or
- Stops when maximal cardinality *n* is reached (10 by default), or

ACF-4

• Stops after 30 seconds (default timeout)

T. Genet (ISTIC/IRISA)

5 / 27

Nitpick (III)

nitpick options:

- timeout=t, set the timeout to t seconds (timeout=none possible)
- show_all, displays the domains and interpretations for the counterex.
- expect=s, specifies the expected outcome where s can be none (no counterexample) or genuine (a counterexample exists)

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 rev 1 = 1
- Is there a counterexample to the lemma nth_index?
- Correct the lemma and definitions of index and nth
- Is the lemma append_commut true? really?

7 / 27

Nitpick (II)

Exercise 1

By hand, iteratively check if there is a counterexample of cardinality 1, 2, 3 for the formula ϕ , where ϕ is length la <= 1.

Remark 1

- The types occurring in ϕ are 'a and 'a list
- One possible domain D_{i_a} of cardinality 1: $\{a_1\}$
- One possible domain $D_{a \ list}$ of cardinality 1: {[]} {[a_1]} Domains have to be subterm-closed, thus {[a_1]} is not valid
- **One** possible domain D_{i_a} of cardinality 2: $\{a_1, a_2\}$
- Two possible domains $D_{a \ list}$ of cardinality 2: {[],[a₁]} and {[],[a₂]}
- **One** possible domain D_{a} of cardinality 3: $\{a_1, a_2, a_3\}$
- **Twelve** possible domains $D_{a \ list}$ of cardinality 3: {[], $[a_1]$, $[a_1, a_1]$ }, {[], $[a_1]$, $[a_2]$ }, {[], $[a_1]$, $[a_3, a_1]$ }, ... $\frac{\{[], [a_1], [a_3, a_2]\}}{\{[], [a_1], [a_3, a_2]\}}$ (Demo!)

Quickcheck

To build an interpretation I such that $I \not\models \phi$ (or $I \models \neg \phi$) quickcheck quickcheck principle: test ϕ with automatically generated values of size kEither with a generator

- Random: values are generated randomly (Haskell's QuickCheck)
- Exhaustive: (almost) all values of size k are generated (TP4bis)
- Narrowing: like exhaustive but taking advantage of symbolic values

No exhautiveness guarantee!! with any of them

T. Genet (ISTIC/IRISA) ACF-4

quickcheck algorithm:

- Export Haskell code for functions and lemmas
- Generate test values of size 1 upto n and, test ϕ using Haskell code
- $\bullet\,$ Stops when a counterexample is found, or
- $\bullet\,$ 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 TPO (solTPO.thy, CM4_TPO)
- find a counterexample for length_slice

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.

ACE-4

```
T. Genet (ISTIC/IRISA)
```

9 / 27

What to do next?

When no counterexample is found what can we do?

- Increase the timeout and size values for nitpick and quickcheck?
- ... go for a proof!

Any proof is faster than an infinite time nitpick or quickcheck

Any proof is more reliable than an infinite time nitpick or quickcheck

(They make approximations or assumptions on infinite types)

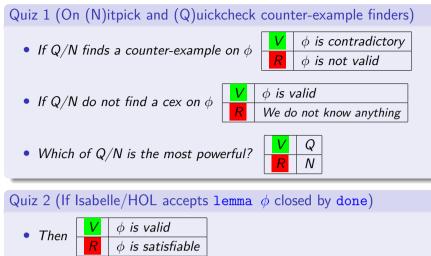
The five proof tools that we will focus on:

- apply case_tac
- 2 apply induct
- 3 apply auto
- 4 apply simp
- 5 sledgehammer

T. Genet (ISTIC/IRISA)

11 / 27

Counter-example finders – the quiz



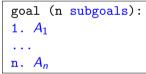
• There may remain some counter-example

How do proofs look like?

A formula of the form $A_1 \wedge \ldots \wedge A_n$ is represented by the proof goal:

True

False



T. Genet (ISTIC/IRISA)

Where each subgoal to prove is either a formula of the form $\bigwedge x_1 \dots x_n$. B meaning prove B, or $\bigwedge x_1 \dots x_n$. $B \Longrightarrow C$ meaning prove $B \longrightarrow C$, or $\bigwedge x_1 \dots x_n$. $B_1 \Longrightarrow \dots B_n \Longrightarrow C$ meaning prove $B_1 \land \dots \land B_n \longrightarrow C$

and $\bigwedge x_1 \dots x_n$ means that those variables are local to this subgoal.

Example 1 (Proof goal) goal (2 subgoals): 1. contains e [] \implies nth (index e []) [] = e 2. \land a 1. e \neq a \implies contains e (a # 1) \implies \neg contains e 1 \implies nth (index e 1) 1 = e

T. Genet (ISTIC/IRISA)

AC

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 $\ldots\ldots$ 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): become 1. A \Longrightarrow B

becomes goal (2 subgoals): 1. $F \implies A \implies B$ 2. $\neg F \implies A \implies B$

Exercise 4

Prove that for any natural number x, if x < 4 then x * x < 10.

T. Genet (ISTIC/IRISA)

13/2

Proof by induction

«Properties on recursive functions need proofs by induction»

Recall the basic induction principle on naturals:

 $P(0) \land \forall x \in \mathbb{N}. \ (P(x) \longrightarrow P(x+1)) \longrightarrow \forall x \in \mathbb{N}. \ P(x)$

All recursive datatype have a similar induction principle, *e.g.* 'a lists:

 $P([]) \land \forall e \in \texttt{'a. } \forall l \in \texttt{'a list.} (P(l) \longrightarrow P(e \# l)) \longrightarrow \forall l \in \texttt{'a list.} P(l)$

Etc...

Example 4

datatype 'a binTree= Leaf | Node 'a "'a binTree" "'a binTree"

 $\begin{array}{l} P(\texttt{Leaf}) \land \forall e \in \texttt{'a. } \forall t1 \ t2 \in \texttt{'a binTree.} \\ (P(t1) \land P(t2) \longrightarrow P(\texttt{Node e } t1 \ t2)) \longrightarrow \forall t \in \texttt{'a binTree.} P(t) \end{array}$

T. Genet (ISTIC/IRISA)

15 / 27

Proof by cases (II)

Proof by cases on a variable x of an enumerated type of size nDo a proof by cases on a variable xapply (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)		
<pre>In Course 3, we defined datatype color= Black White Grey With apply (case_tac "x")</pre>		
11 5		goal (3 subgoals):
goal (1 subgoal):	becomes	1. x = Black \implies P x
1. P (x::color)		2. x = White \implies P x
		3. x = Grey \implies P x

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.

T. Genet (ISTIC/IRISA)

14 / 27

Proof by induction (II) $P(I) \land \forall e \in A \forall I \in A$ list $(P(I) \longrightarrow P(e\#I))$

 $P([\]) \land \forall e \in \texttt{'a. } \forall l \in \texttt{'a list.} (P(l) \longrightarrow P(e \# l)) \implies \forall l \in \texttt{'a list.} P(l)$

Example 5 (Proof by induction on lists) Recall the definition of the function append: (1) append [] 1 = 1 (2) append (x#xs) 1 = x#(append xs 1) To prove $\forall l \in 'a \ list. (append l[]) = l$ by induction on l, we prove: **1** append [][] = [], proven by the first equation of append **2** $\forall e \in 'a. \forall l \in 'a \ list.$ (append l[]) = l \longrightarrow (append (e#l) []) = (e#l) using the second equation of append, it becomes (append l[]) = l $\longrightarrow e#(append l[]) = (e#l)$ using the (induction) hypothesis, it becomes (append l[]) = l $\longrightarrow e#l = (e#l)$

T. Genet (ISTIC/IRISA)

ACF-4

Proof by induction: apply (induct x)

To apply induction principle on variable xapply (induct x)

Conditions on the variable chosen for induction (induction variable):

- The variable x has to be of an inductive type (nat, datatypes, ...) Otherwise apply (induct x) fails
- The terms built by induction cases should easily be reducible!

Example 6 (Choice of the induction variable)

(1) append [] l = l

(2) append (x#xs) 1 = x#(append xs 1)

To prove $\left| orall l_1 \ l_2 \ \in$ 'a list. (length (append l_1 l_2)) \geq (length l_2) $\right|$

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

- an induction on l₁ will require to prove:
 (length (append (e#l₁) l₂) ≥ (length l₂)
- an induction on l_2 will require to prove: (*length* (*append* l_1 ($e \# l_2$)) \ge (*length* ($e \# l_2$))
- T. Genet (ISTIC/IRISA)

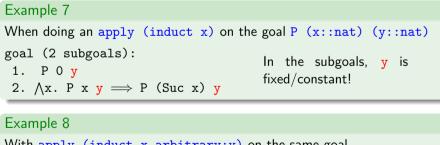
```
17 /
```

19 / 27

Proof by induction: generalize the goals

By defaut apply induct may produce too weak induction hypothesis

ACE-4



goal (2 subgoals):

1. ∧y. P O y

2. $\land x y$. P x y \Longrightarrow P (Suc x) y

The subgoals range over any y

Exercise 8

Prove the sym lemma on the leq function.

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 contains 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

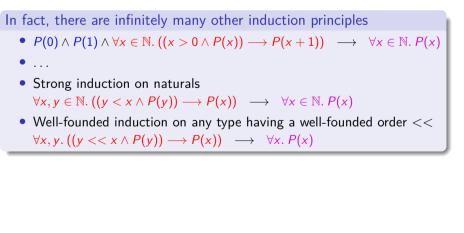
T. Genet (ISTIC/IRISA)

18 / 27

Proof by induction: : induction principles

Recall the basic induction principle on naturals:

 $P(0) \land \forall x \in \mathbb{N}. (P(x) \longrightarrow P(x+1)) \longrightarrow \forall x \in \mathbb{N}. P(x)$



ACF-4

Proof by induction: : induction principles (II)

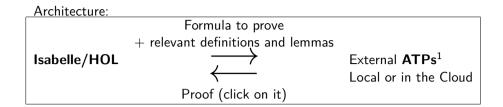
Automatically solve or simplify all subgoalsapply auto apply auto does the following: Apply an induction principle adapted to the function call (f x y z)• Rewrites using equations (function definitions, etc)apply (induct x y z rule:f.induct) • Applies a bit of arithmetic, logic reasoning and set reasoning Apply strong induction on variable x of type nat • On all subgoalsapply (induct x rule:nat_less_induct) • Solves them all or stops when stuck and shows the remaining subgoals Apply well-founded induction on a variable xapply (induct x rule:wf_induct) Automatically simplify **the first subgoal**apply simp Exercise 9 apply simp does the following: Prove the lemma on function divBy2. • 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 T. Genet (ISTIC/IRISA) T. Genet (ISTIC/IRISA) ACF-4 Combination of decision procedures auto and simp (II) Sledgehammer Want to know what those tactics do? • Add the command using [[simp_trace=true]] in the proof script • Look in the output buffer Example 9 Switch on tracing and try to prove the lemma: lemma "(index (1::nat) [3,4,1,3]) = 2" using [[simp trace=true]] apply auto «Sledgehammers are often used in destruction work...»

23 / 27

Combination of decision procedures auto and simp

Sledgehammer

«Solve theorems in the Cloud»



Prove the first subgoal using state-of-the-art² 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 explicitely selected (*e.g.* [provers= z3 spass]
- A proof consists of applications of lemmas or definition using the lsabelle/HOL tactics: metis, smt, simp, fast, etc.

¹Automatic Theorem Provers ²See http://www.tptp.org/CASC/. T. Genet (ISTIC/IRISA)

25 / 27

Hints for building proofs in Isabelle/HOL

When stuck in the proof of prop1, add relevant intermediate lemmas:

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

To build correct theories, do not confuse oops and sorry:

- Always close an unprovable property by oops
- Always close an unfinished proof of a provable property by sorry

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

27 / 27

Sledgehammer (II)

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 nth and index

Exercise 11

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

- Prove that there is no repeated occurrence of elements in the list produced by makeList
- **2** Finish the proof of the property relating those three functions

T. Genet (ISTIC/IRISA)