

4 / 19

2/19

Terminating Recursive Functions

In Isabelle/HOL, all the recursive functions have to be terminating!

How to guarantee the termination of a recursive function? (practice)

- Needs at least one base case (non recursive case)
- Every recursive case must go towards a base case
- ... or every recursive case «decreases» the size of one parameter

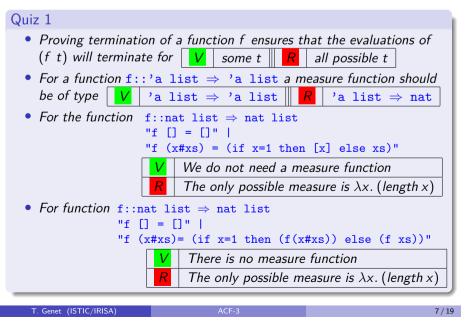
How to guarantee the termination of a recursive function? (theory)

- If $f::\tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow \tau$ then define a measure function $g::\tau_1 \times \ldots \times \tau_n \Rightarrow \mathbb{N}$
- Prove that the measure of all recursive calls is decreasing To prove termination of f $f(t_1) \rightarrow f(t_2) \rightarrow \dots$ Prove that $g(t_1) > g(t_2) > \dots$
- The ordering > is well founded on ℕ
 i.e. no infinite decreasing sequence of naturals n₁ > n₂ > ...

ACF-3

T. Genet (ISTIC/IRISA)

Proving termination with measure – the quiz



Terminating Recursive Functions (II)

How to guarantee the termination of a recursive function? (theory)

- If $f::\tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow \tau$ then define a measure function $g::\tau_1 \times \ldots \times \tau_n \Rightarrow \mathbb{N}$
- Prove that the measure of all recursive calls is decreasing To prove termination of $f(t_1) \rightarrow f(t_2) \rightarrow \dots$ Prove that $g(t_1) > g(t_2) > \dots$

Example 1 (Proving termination using a measure)

- "contains e [] = False" |
 "contains e (x#xs)= (if e=x then True else (contains e xs))"
- **1** We define the measure $g = \lambda(x, y)$. (*length* y)
- **2** We prove that $\forall e x xs. g(e, (x#xs)) > g(e, xs)$

T. Genet (ISTIC/IRISA)

6/19

8/19

Terminating Recursive Functions (III)

How to guarantee the termination of a recursive function? (Isabelle/HOL)

- Define the recursive function using fun
- Isabelle/HOL automatically tries to build a measure¹
- If no measure is found the function is rejected
- If it is not terminating, make it terminating!
- Try to modify it so that its termination is easier to show

Otherwise

- Re-define the recursive function using **function** (sequential)
- Manually give a measure to achieve the termination proof

¹Actually, it tries to build a termination ordering but it has the same objective. T. Genet (ISTIC/IRISA) ACF-3

Terminating Recursive Functions (IV)

Example 2 A definition of the contains function using function is the following: function (sequential) contains::"'a \Rightarrow 'a list \Rightarrow bool" where "contains e [] = False" | "contains e (x#xs)= (if e=x then True else (contains e xs))" Prove that the function is "complete" apply pat completeness *i.e.* patterns cover the domain apply auto done Prove its termination using the measure proposed in Example 1 termination contains apply (relation "measure $(\lambda(x,y))$. (length y))") apply auto done

9

Terminating Recursive Functions (VI)

Automatic termination proofs (fun definition) are generally enough

• Covers 90% of the functions commonly defined by programmers

• Otherwise, it is generally possible to adapt a function to fit this setting Most of the functions are terminating by construction (primitive recursive)

Definition 3 (Primitive recursive functions: primrec)

Functions whose recursive calls «peels off» exactly one constructor

```
Example 4 (contains can be defined using primrec instead of fun)
primrec contains:: "'a => 'a list => bool"
where
"contains e [] = False" |
"contains e (x#xs)= (if e=x then True else (contains e xs))"
```

For instance, in List.thy:

- 26 "fun", 34 "primrec" with automatic termination proofs
- 3 "function" needing measures and manual termination proofs.

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

T. Genet (ISTIC/IRISA)

11 / 19

Terminating Recursive Functions (V)

Exercise 1

Define the following functions, see if they are terminating. If not, try to modify them so that they become terminating.

fun f::"nat => nat"
where
"f x=f (x - 1)"

fun f2::"int => int"
where
"f2 x = (if x=0 then 0 else f2 (x - 1))"

fun f3::"nat => nat => nat" where "f3 x y= (if x >= 10 then 0 else f3 (x + 1) (y + 1))"

T. Genet (ISTIC/IRISA)

10/19

Recursive functions, exercises

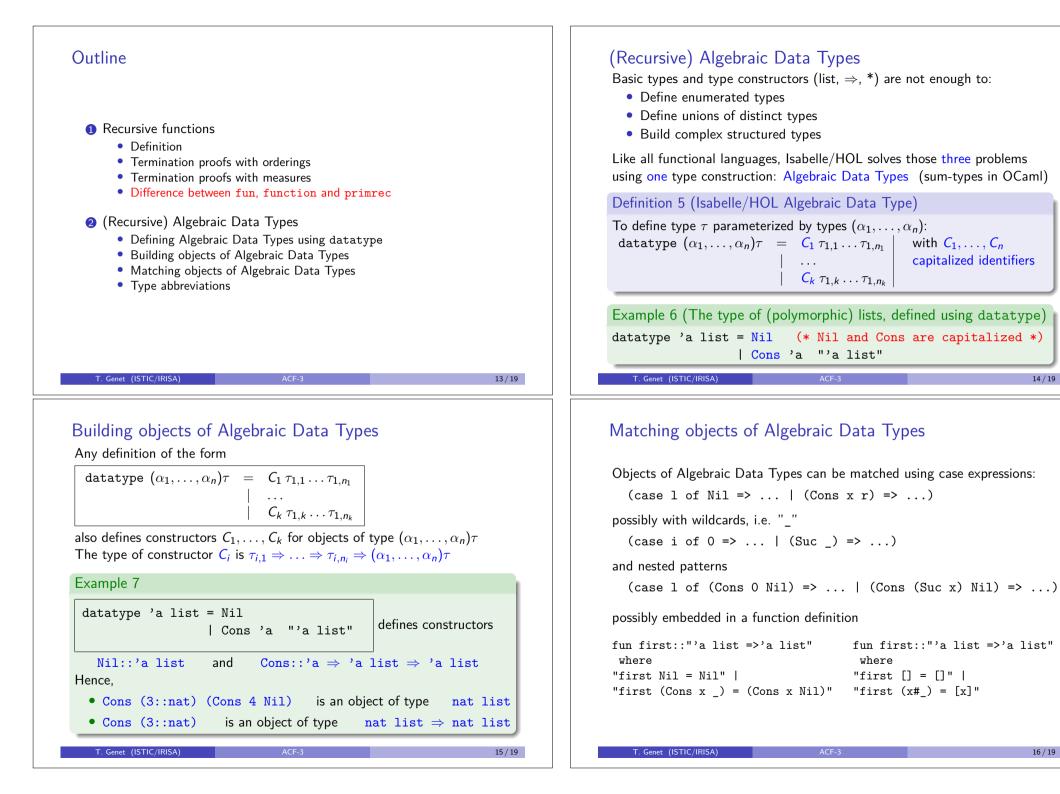
Exercise 2

T. Genet (ISTIC/IRISA)

Define the following recursive functions

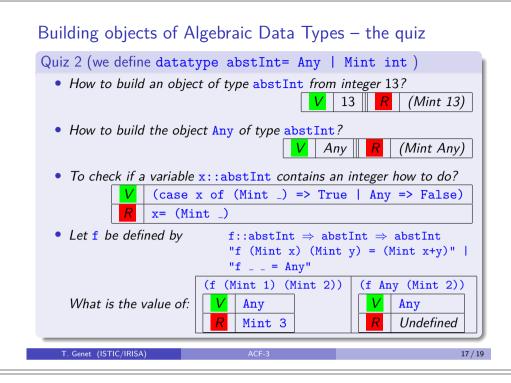
- A function sumList computing the sum of the elements of a list of naturals
- A function sumNat computing the sum of the n first naturals
- A function makeList building the list of the n first naturals

State and verify a lemma relating sumList, sumNat and makeList



14 / 19

16 / 19



Type abbreviations

In Isabelle/HOL, it is possible to define abbreviations for complex types To introduce a type abbreviationtype_synonym

For instance:

- type_synonym name="(string * string)"
- type_synonym ('a,'b) pair="('a * 'b)"

Using those abbreviations, objects can be explicitly typed:

- value "(''Leonard'',''Michalon'')::name"
- value "(1,''toto'')::(nat,string)pair"
- \ldots though the type synonym name is ignored in Isabelle/HOL output \odot

Algebraic Data Types, exercises

Exercise 3

Define the following types and build an object of each type using value

- The enumerated type color with possible values: black, white and grey
- The type token union of types string and int
- The type of (polymorphic) binary trees whose elements are of type 'a

Define the following functions

T. Genet (ISTIC/IRISA)

- A function notBlack that answers true if a color object is not black
- A function sumToken that gives the sum of two integer tokens and 0 otherwise
- A function merge::color tree ⇒ color that merges all colors in a color tree (leaf is supposed to be black)

18/19