

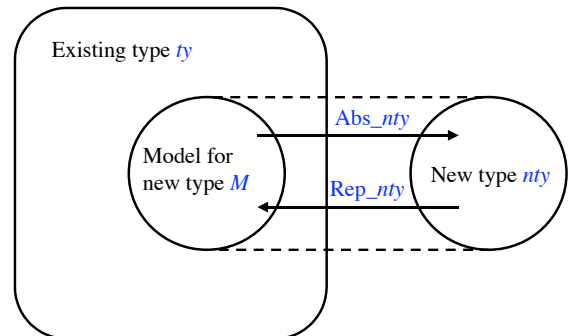
## CS576 Topics in Automated Deduction

Elsa L. Gunter  
 2112 SC, UIUC  
 egunter@illinois.edu  
<http://courses.grainger.illinois.edu/cs576>

Slides based in part on slides by Tobias Nipkow

March 4, 2026

## Introducing new types



Properties of a Defined Type Syntax for `typedef`:

```
typedef nty = "modeling_set"
Proof of  $\exists x. x \in \text{modeling\_set}$ .
```

introduces a new type named `nty`, and functions

```
Abs_nty :: ty  $\Rightarrow$  new_ty
Rep_nty :: new_ty  $\Rightarrow$  ty
```

where `modeling_set::ty`

Properties of a Defined Type Theorems automatically provided:

```
Rep_nty: Rep_nty x  $\in$  modeling_set
Abs_nty_inverse: Abs_nty(Rep_nty x) = x
Rep_nty_inverse:
  y  $\in$  modeling_set  $\implies$  Rep_nty(Abs_nty y) = y
Abs_nty_inject:
  [|x  $\in$  modeling_set : y  $\in$  modeling_set|]  $\implies$ 
  (Abs_nty x = Abs_nty y) = (x = y)
Rep_nty_inject: (Rep_nty x = Rep_nty y) = (x = y)
```

## Records in Isabelle/HOL

- Records in HOL are basically tuples, but...

```
record ('a) graph_sig =
  Vertices :: "'a set"
  Edges :: "('a  $\times$  'a) set"
```

```
definition one :: "(unit) graph_sig" where
  "one  $\equiv$   $\langle$  Vertices = {}, Edges = {(((),()))  $\rangle$ "
```

- Components accessed by field

```
lemma "()  $\in$  Vertices one"
by (simp add: one_def)
```

- Unlike functional programming languages, position of fields matters

```
 $\langle$  Edges = {(((),())) $\rangle$ , Vertices = {}  $\rangle$ "
causes an "Error in record input"
```

## Record Field Update

- Records support field update

```
definition no_edge :: "(unit) graph_sig" where
  "no_edge = one  $\langle$  Edges := {}  $\rangle$ "
```

```
lemma "no_edge =  $\langle$  Vertices = {}, Edges = {}  $\rangle$ "
by (simp add: no_edge_def one_def)
```

## Record Field Update

- Record update is functional

```
definition no_edge :: "(unit) graph_sig" where
  "no_edge = one(Edges := {})"
```

```
lemma "no_edge = (Vertices = {}, Edges = {})"
  by (simp add: no_edge_def one_def)
```

```
lemma "no_edge ≠ one"
  by (simp add: no_edge_def one_def)
```

```
lemma "(Vertices no_edge = Vertices one) ∧
  (Edges no_edge = {})"
  by (simp add: no_edge_def one_def)
```

## record type representations

- Every record type may be given by field syntax:

```
lemma "(one::unit graph_sig) =
  (one::(Vertices::unit set, Edges::(unit × unit) set))"
  by (rule refl)
```

- Every record is extensible
  - Every record `rec_ty` defines a polymorphic type `(‘a) rec_ty_scheme`;  
type `rec_ty` same as `(unit) rec_ty_scheme`

```
term "one::(unit,unit) graph_sig_scheme"
```

- Every record type has a "hidden field" `more`

## New Record Types From Old

- New record types may be created by extending existing record types with additional fields:

```
record ('a,'b) labeled_graph_sig = "'a graph_sig" +
  Label :: "('a × 'a) ⇒ ('b) option"
```

```
definition two :: "(Vertices :: nat set,
  Edges :: (nat × nat) set,
  Label :: nat × nat ⇒ bool option)" where
```

```
"two ≡
  (Vertices = {1,2}, Edges = {(1,2)},
  Label = (λ (m,n). (if (m,n) = (1,2) then Some True else None))"
```

constants

```
two :: "(nat, bool) labeled_graph_sig"
```

## Record Polymorphism

- The `_scheme` types allow for (weak) record polymorphism
- Also referred to as record subtyping
- Generally, input to functions should use `_scheme` type instead of strict record type

```
definition is_graph :: "('a,'b) graph_sig_scheme ⇒ bool" where
  "is_graph G ≡ (∀ e ∈ Edges G. {fst e, snd e} ⊆ Vertices G)"
```

```
lemma shows "is_graph one"
  by (simp add: one_def is_graph_def)
```

```
lemma shows "is_graph two"
  by (simp add: two_def is_graph_def)
```