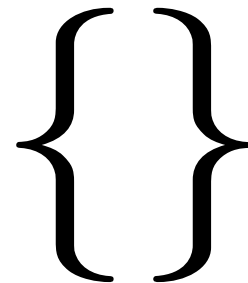

COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray, Rafal Kolanski



Content

- Intro & motivation, getting started [1]

- Foundations & Principles
 - Lambda Calculus, natural deduction [1,2]
 - Higher Order Logic [3]
 - Term rewriting [4^a]

- Proof & Specification Techniques
 - Inductively defined sets, rule induction [5]
 - Datatypes, recursion, induction [6^b, 7]
 - Code generation, type classes [7]
 - Hoare logic, proofs about programs, refinement [8,9^c,10^d]
 - Isar, locales [11,12]

^a a1 due; ^b a2 due; ^c session break; ^d a3 due

Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
- Congruence rules
- AC Rules
- Knuth-Bendix Completion (Waldmeister)
- Orthogonal Rewrite Systems

SPECIFICATION TECHNIQUES: SETS

Sets in Isabelle

Type 'a **set**: sets over type 'a

- $\{\}, \{e_1, \dots, e_n\}, \{x. P\ x\}$
- $e \in A, A \subseteq B$
- $A \cup B, A \cap B, A - B, -A$
- $\bigcup x \in A. B\ x, \bigcap x \in A. B\ x, \bigcap A, \bigcup A$
- $\{i..j\}$
- $\text{insert} :: \alpha \Rightarrow \alpha\ \text{set} \Rightarrow \alpha\ \text{set}$
- $f' A \equiv \{y. \exists x \in A. y = f\ x\}$
- ...

Proofs about Sets

Natural deduction proofs:

- equality: $\llbracket A \subseteq B; B \subseteq A \rrbracket \implies A = B$
- subset: $(\bigwedge x. x \in A \implies x \in B) \implies A \subseteq B$
- ... (see Tutorial)

Bounded Quantifiers

→ $\forall x \in A. P x \equiv \forall x. x \in A \longrightarrow P x$

→ $\exists x \in A. P x \equiv \exists x. x \in A \wedge P x$

→ ball: $(\bigwedge x. x \in A \implies P x) \implies \forall x \in A. P x$

→ bspec: $\llbracket \forall x \in A. P x; x \in A \rrbracket \implies P x$

→ bexI: $\llbracket P x; x \in A \rrbracket \implies \exists x \in A. P x$

→ bexE: $\llbracket \exists x \in A. P x; \bigwedge x. \llbracket x \in A; P x \rrbracket \implies Q \rrbracket \implies Q$

DEMO: SETS

The Three Basic Ways of Introducing Types

→ **typedecl**: by name only

Example: **typedecl** names

Introduces new type *names* without any further assumptions

→ **type_synonym**: by abbreviation

Example: **type_synonym** α rel = " $\alpha \Rightarrow \alpha \Rightarrow bool$ "

Introduces abbreviation *rel* for existing type $\alpha \Rightarrow \alpha \Rightarrow bool$

Type abbreviations are immediately expanded internally

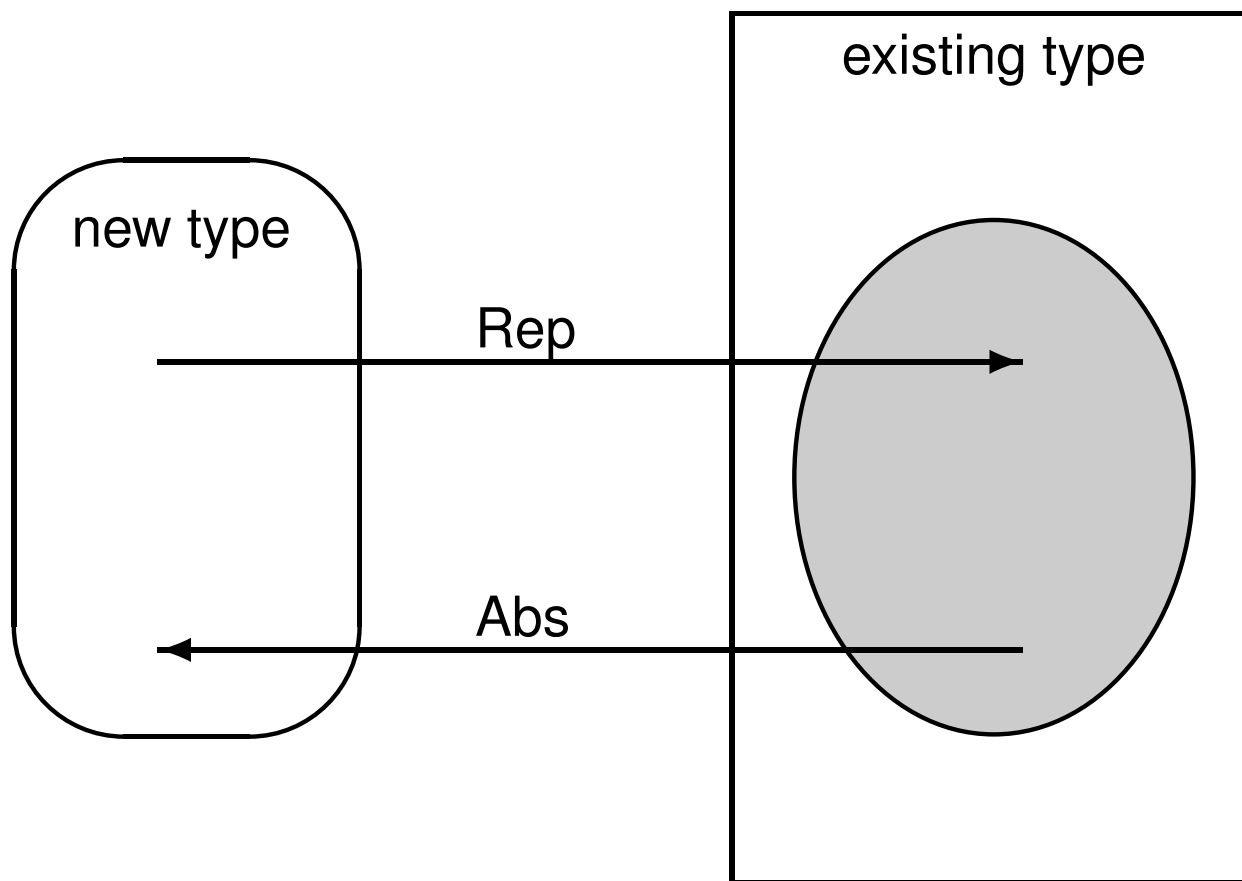
→ **typedef**: by definition as a set

Example: **typedef** new_type = "{some set}" <proof>

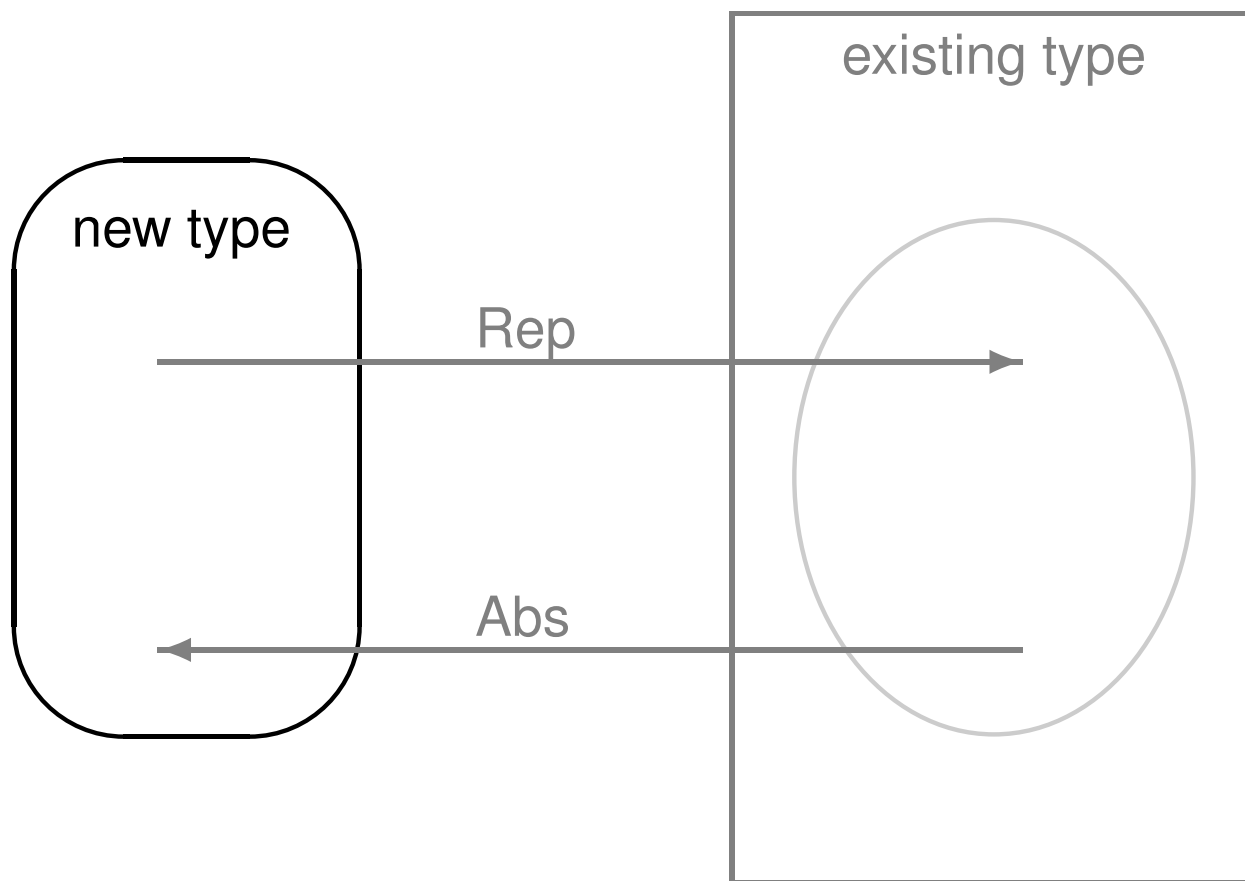
Introduces a new type as a subset of an existing type.

The proof shows that the set on the rhs is non-empty.

How typedef works



How typedef works



Example: Pairs

(α, β) Prod

① Pick existing type: $\alpha \Rightarrow \beta \Rightarrow \text{bool}$

② Identify subset:

$$(\alpha, \beta) \text{ Prod} = \{f. \exists a b. f = \lambda(x :: \alpha) (y :: \beta). x = a \wedge y = b\}$$

③ We get from Isabelle:

- functions Abs_Prod, Rep_Prod
- both injective
- $\text{Abs_Prod} (\text{Rep_Prod } x) = x$

④ We now can:

- define constants Pair, fst, snd in terms of Abs_Prod and Rep_Prod
- derive all characteristic theorems
- forget about Rep/Abs, use characteristic theorems instead

DEMO: INTRODUCING NEW TYPES

INDUCTIVE DEFINITIONS

Example

$$\frac{}{\langle \text{skip}, \sigma \rangle \longrightarrow \sigma} \quad \frac{[[e]]\sigma = v}{\langle x := e, \sigma \rangle \longrightarrow \sigma[x \mapsto v]}$$

$$\frac{\langle c_1, \sigma \rangle \longrightarrow \sigma' \quad \langle c_2, \sigma' \rangle \longrightarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \longrightarrow \sigma''}$$

$$\frac{[[b]]\sigma = \text{False}}{\langle \text{while } b \text{ do } c, \sigma \rangle \longrightarrow \sigma}$$

$$\frac{[[b]]\sigma = \text{True} \quad \langle c, \sigma \rangle \longrightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \longrightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \longrightarrow \sigma''}$$

What does this mean?

- $\langle c, \sigma \rangle \longrightarrow \sigma'$ fancy syntax for a relation $(c, \sigma, \sigma') \in E$
- relations are sets: $E :: (\text{com} \times \text{state} \times \text{state}) \text{ set}$
- the rules define a set inductively

But which set?

Simpler Example

$$\frac{}{0 \in N} \quad \frac{n \in N}{n + 1 \in N}$$

- N is the set of natural numbers \mathbb{N}
- But why not the set of real numbers? $0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R}$
- \mathbb{N} is the **smallest** set that is **consistent** with the rules.

Why the smallest set?

- Objective: **no junk**. Only what must be in X shall be in X .
- Gives rise to a nice proof principle (rule induction)
- Alternative (greatest set) occasionally also useful: coinduction

Rule Induction

$$\frac{}{0 \in N} \quad \frac{n \in N}{n + 1 \in N}$$

induces induction principle

$$\llbracket P\ 0; \bigwedge n. P\ n \implies P\ (n + 1) \rrbracket \implies \forall x \in X. P\ x$$

DEMO: INDUCTIVE DEFINITIONS

We have learned today ...

- Sets
- Type Definitions
- Inductive Definitions