Experiments in Verification

SS 2011

Christian Sternagel

Computational Logic
Institute of Computer Science
University of Innsbruck

April 15, 2011

Today's Topics

- Sets and Relations
- Inductively Defined Sets
- Evaluation
- Projects

Sets and Relations

Sets in Isabelle

- type

```
(* characteristic function. *)
type_synonym 'a set = "('a ⇒ bool)"
```

- x is member of set S if characteristic function returns `True`
- lemma `mem_def: "x ∈ S ≡ S x"`

Basic Operations on Sets – Intersection

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$
- `IntD2`: $c \in A \cap B \Longrightarrow c \in B$

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$
- `IntD2`: $c \in A \cap B \Longrightarrow c \in B$

Basic Operations on Sets – Union

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$
- `IntD2`: $c \in A \cap B \Longrightarrow c \in B$

Basic Operations on Sets – Union

- notation: $A \cup B$ (ASCII: A `Un` B)

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$
- `IntD2`: $c \in A \cap B \Longrightarrow c \in B$

Basic Operations on Sets – Union

- notation: $A \cup B$ (ASCII: A `Un` B)
- `UnI1`: $c \in A \Longrightarrow c \in A \cup B$

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$
- `IntD2`: $c \in A \cap B \Longrightarrow c \in B$

Basic Operations on Sets – Union

- notation: $A \cup B$ (ASCII: A `Un` B)
- `UnI1`: $c \in A \Longrightarrow c \in A \cup B$
- `UnI2`: $c \in B \Longrightarrow c \in A \cup B$

Basic Operations on Sets – Intersection

- notation: $A \cap B$ (ASCII: A `Int` B)
- `IntI`: $[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$
- `IntD1`: $c \in A \cap B \Longrightarrow c \in A$
- `IntD2`: $c \in A \cap B \Longrightarrow c \in B$

Basic Operations on Sets – Union

- notation: $A \cup B$ (ASCII: A `Un` B)
- `UnI1`: $c \in A \Longrightarrow c \in A \cup B$
- `UnI2`: $c \in B \Longrightarrow c \in A \cup B$
- `UnE`: $[\![c \in A \cup B; c \in A \Longrightarrow P; c \in B \Longrightarrow P]\!] \Longrightarrow P$

Basic Operations on Sets – Complement and Difference

Basic Operations on Sets – Complement and Difference

- complement: $-A$

Basic Operations on Sets – Complement and Difference

- complement: $-A$
- `Compl_iff`: $(c \in -A) = (c \notin A)$

Basic Operations on Sets – Complement and Difference

- complement: $-A$
- `Compl_iff`: $(c \in -A) = (c \notin A)$
- difference: $A - B$

Basic Operations on Sets – Complement and Difference

- complement: $-A$
- `Compl_iff`: $(c \in -A) = (c \notin A)$
- difference: $A - B$

Basic Operations on Sets – Subsets

Basic Operations on Sets – Complement and Difference

- complement: $-A$
- `Compl_iff`: $(c \in -A) = (c \notin A)$
- difference: $A - B$

Basic Operations on Sets – Subsets

- notation: $A \subseteq B$ (ASCII: A `<=` B)

Basic Operations on Sets – Complement and Difference

- complement: $-A$
- `Compl_iff`: $(c \in -A) = (c \notin A)$
- difference: $A - B$

Basic Operations on Sets – Subsets

- notation: $A \subseteq B$ (ASCII: A `<=` B)
- `subsetI`: $(\bigwedge x.\ x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$

Basic Operations on Sets – Complement and Difference

- complement: $-A$
- `Compl_iff`: $(c \in -A) = (c \notin A)$
- difference: $A - B$

Basic Operations on Sets – Subsets

- notation: $A \subseteq B$ (ASCII: A `<=` B)
- `subsetI`: $(\bigwedge x.\ x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$
- `equalityI`: $[\![A \subseteq B\,;B \subseteq A]\!] \Longrightarrow A = B$

Set Notation

Set Notation

- the empty set: `{}`

Set Notation

- the empty set: `{}`
- the universal set: `UNIV`

Set Notation

- the empty set: `{}`
- the universal set: `UNIV`
- a singleton set: `{`x`}`

Set Notation

- the empty set: `{}`
- the universal set: `UNIV`
- a singleton set: `{`x`}`
- insertion (`insert_is_Un`): `insert` x A `=` `{`x`}` $\cup$ A

Set Notation

- the empty set: `{}`
- the universal set: `UNIV`
- a singleton set: `{`x`}`
- insertion (`insert_is_Un`): `insert` x A `=` `{`x`}` $\cup A$
- finite sets, e.g., `{`a, b, c, d`}`

An Example Proof

```
lemma "A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)"
```

An Example Proof

```
lemma "A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)"
```

Proof

Isabelle

A Shorter Proof – The `blast` Method

- applies introduction and elimination rules automatically
- suitable for many goals concerning logical and/or set operations

```
lemma "A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)" by blast
```

Set Comprehension by Example

Mathematics	Isabelle
$\{x \mid P(x)\}$	`{x. P x}`
$\{(x, y) \mid x \in A, y \in B\}$	`{(x,y)	x y. x ∈ A ∧ y ∈ B}`

Binding Operators for Sets

- bounded universal quantifier: $\forall x \in A.\ P\ x$

Binding Operators for Sets

- bounded universal quantifier: $\forall x \in A.\ P\ x$
- bounded existential quantifier: $\exists x \in A.\ P\ x$

Binding Operators for Sets

- bounded universal quantifier: $\forall x \in A.\ P\ x$
- bounded existential quantifier: $\exists x \in A.\ P\ x$
- `ballI`: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \Longrightarrow \forall x \in A.\ P\ x$

Binding Operators for Sets

- bounded universal quantifier: $\forall x \in A.\ P\ x$
- bounded existential quantifier: $\exists x \in A.\ P\ x$
- `ballI`: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \Longrightarrow \forall x \in A.\ P\ x$
- `bspec`: $[\![\forall x \in A.\ P\ x; x \in A]\!] \Longrightarrow P\ x$

Binding Operators for Sets

- bounded universal quantifier: $\forall x \in A.\ P\ x$
- bounded existential quantifier: $\exists x \in A.\ P\ x$
- `ballI`: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \Longrightarrow \forall x \in A.\ P\ x$
- `bspec`: $[\![\forall x \in A.\ P\ x; x \in A]\!] \Longrightarrow P\ x$
- `bexI`: $[\![P\ x; x \in A]\!] \Longrightarrow \exists x \in A.\ P\ x$

Binding Operators for Sets

- bounded universal quantifier: $\forall x \in A.\ P\ x$
- bounded existential quantifier: $\exists x \in A.\ P\ x$
- `ballI`: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \Longrightarrow \forall x \in A.\ P\ x$
- `bspec`: $[\![\forall x \in A.\ P\ x; x \in A]\!] \Longrightarrow P\ x$
- `bexI`: $[\![P\ x; x \in A]\!] \Longrightarrow \exists x \in A.\ P\ x$
- `bexE`: $[\![\exists x \in A.\ P\ x; \bigwedge x.\ [\![x \in A; P\ x]\!] \Longrightarrow Q]\!] \Longrightarrow Q$

Relations – Basics

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: $\texttt{Id} \equiv \{p.\ \exists x.\ p = (x, x)\}$

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: $\texttt{Id} \equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: $r \ O \ s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \land (y, z) \in s\}$

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: $\mathtt{Id} \equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: $r\ \mathtt{O}\ s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \wedge (y, z) \in s\}$
- converse: $((a, b) \in r\mathtt{\hat{}{-1}}) = ((b, a) \in r)$

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: `Id` $\equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: r `O` $s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \land (y, z) \in s\}$
- converse: $((a, b) \in r$`^-1`$) = ((b, a) \in r)$

Relations – Reflexive and Transitive Closure

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: `Id` $\equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: r `O` $s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \land (y, z) \in s\}$
- converse: $((a, b) \in r$`^-1`$) = ((b, a) \in r)$

Relations – Reflexive and Transitive Closure

- reflexive and transitive closure: r`^*`

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: $\texttt{Id} \equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: $r\ \texttt{O}\ s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \wedge (y, z) \in s\}$
- converse: $((a, b) \in r\texttt{\^{}-1}) = ((b, a) \in r)$

Relations – Reflexive and Transitive Closure

- reflexive and transitive closure: $r\texttt{\^{}*}$
- transitive closure: $r\texttt{\^{}+}$

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: `Id` $\equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: r `O` $s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \wedge (y, z) \in s\}$
- converse: $((a, b) \in r$`^-1`$) = ((b, a) \in r)$

Relations – Reflexive and Transitive Closure

- reflexive and transitive closure: r`^*`
- transitive closure: r`^+`
- `rtrancl_refl`: $(a, a) \in r$`^*`

Relations – Basics

- a relation is a set of pairs (`('a × 'b) set`)
- identity relation: `Id` $\equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: r `O` $s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \wedge (y, z) \in s\}$
- converse: $((a, b) \in r$`^-1`$) = ((b, a) \in r)$

Relations – Reflexive and Transitive Closure

- reflexive and transitive closure: r`^*`
- transitive closure: r`^+`
- `rtrancl_refl`: $(a, a) \in r$`^*`
- `r_into_rtrancl`: $p \in r \Longrightarrow p \in r$`^*`

Relations – Basics

- a relation is a set of pairs (('a × 'b) set)
- identity relation: $\mathtt{Id} \equiv \{p.\ \exists x.\ p = (x, x)\}$
- composition: $r\ \mathtt{O}\ s \equiv \{(x, z).\ \exists y.\ (x, y) \in r \wedge (y, z) \in s\}$
- converse: $((a, b) \in r\mathtt{\hat{}{-1}}) = ((b, a) \in r)$

Relations – Reflexive and Transitive Closure

- reflexive and transitive closure: $r\mathtt{\hat{}*}$
- transitive closure: $r\mathtt{\hat{}+}$
- rtrancl_refl: $(a, a) \in r\mathtt{\hat{}*}$
- r_into_rtrancl: $p \in r \Longrightarrow p \in r\mathtt{\hat{}*}$
- rtrancl_trans: $[\![(a, b) \in r\mathtt{\hat{}*}; (b, c) \in r\mathtt{\hat{}*}]\!] \Longrightarrow (a, c) \in r\mathtt{\hat{}*}$

Example

```
lemma "(r O s)^-1 = s^-1 O r^-1"
```

Example

```
lemma "(r O s)^-1 = s^-1 O r^-1"
```

Proof

Isabelle

Inductively Defined Sets

An Introductory Definition – Even Numbers

```
inductive_set even :: "nat set" where
  zero[intro!]: "0 ∈ even"
| step[intro!]: "n ∈ even ⟹ Suc (Suc n) ∈ even"
```

An Introductory Definition – Even Numbers

```
inductive_set even :: "nat set" where
  zero[intro!]: "0 ∈ even"
| step[intro!]: "n ∈ even ⟹ Suc (Suc n) ∈ even"
```

Remarks

An Introductory Definition – Even Numbers

```
inductive_set even :: "nat set" where
  zero[intro!]: "0 ∈ even"
| step[intro!]: "n ∈ even ⟹ Suc (Suc n) ∈ even"
```

Remarks

- `intro`: declares a lemma as introduction rule (for blast/auto)

An Introductory Definition – Even Numbers

```
inductive_set even :: "nat set" where
  zero[intro!]: "0 ∈ even"
| step[intro!]: "n ∈ even ⟹ Suc (Suc n) ∈ even"
```

Remarks

- `intro`: declares a lemma as introduction rule (for blast/auto)
- `elim`: declares a lemma as elimination rule (for blast/auto)

An Introductory Definition – Even Numbers

```
inductive_set even :: "nat set" where
  zero[intro!]: "0 ∈ even"
| step[intro!]: "n ∈ even ⟹ Suc (Suc n) ∈ even"
```

Remarks

- `intro`: declares a lemma as introduction rule (for blast/auto)
- `elim`: declares a lemma as elimination rule (for blast/auto)
- adding a `!` tells the system that a rule is safe (i.e., it can always be applied without making the goal unprovable)

An Introductory Definition – Even Numbers

```
inductive_set even :: "nat set" where
  zero[intro!]: "0 ∈ even"
| step[intro!]: "n ∈ even ⟹ Suc (Suc n) ∈ even"
```

Remarks

- `intro`: declares a lemma as introduction rule (for blast/auto)
- `elim`: declares a lemma as elimination rule (for blast/auto)
- adding a `!` tells the system that a rule is safe (i.e., it can always be applied without making the goal unprovable)
- `even` is the smallest set constructed by finitely many applications of the two rules `zero` and `step` (i.e., it contains only elements that can be added via the rules)

Even Numbers are Divisible by 2

```
lemma even_imp_2_dvd: "n ∈ even ⟹ 2 dvd n"
proof (induct rule: even.induct)
  case zero show ?case by simp
next
  case (step n)
  hence IH: "2 dvd n" by simp
  then obtain k where "n = 2 * k"
    unfolding dvd_def by (rule exE)
  hence "Suc (Suc n) = 2 * (Suc k)" by simp
  thus ?case unfolding dvd_def by (rule exI)
qed
```

Advanced Inductive Sets – Arguments

Advanced Inductive Sets – Arguments

- an inductive definition may take arguments

Advanced Inductive Sets – Arguments

- an inductive definition may take arguments
- hence it is possible to define functions yielding sets inductively

Advanced Inductive Sets – Arguments

- an inductive definition may take arguments
- hence it is possible to define functions yielding sets inductively
- the keyword **`for`** is used to introduce arguments

Advanced Inductive Sets – Arguments

- an inductive definition may take arguments
- hence it is possible to define functions yielding sets inductively
- the keyword **for** is used to introduce arguments

Reflexive Transitive Closure

```
inductive_set
  rtc :: "('a × 'a) set ⇒ ('a × 'a) set"
    ("_*" [1000] 999)
  for r :: "('a × 'a) set"
where
  refl: "(x, x) ∈ r*"
| step: "(x, y) ∈ r ⟹ (y, z) ∈ r* ⟹ (x, z) ∈ r*"
```

Lemma – `rtc` is Transitive

```
lemma rtc_trans:
  assumes "(x, y) ∈ r*" and "(y, z) ∈ r*"
  shows "(x, z) ∈ r*"
```

Lemma – `rtc` is Transitive

```
lemma rtc_trans:
  assumes "(x, y) ∈ r*" and "(y, z) ∈ r*"
  shows "(x, z) ∈ r*"
```

Proof

Isabelle

Evaluation

LVA-Code

703523-0

Additional Questions

a) I can prove simple lemmas in Isabelle/HOL.
b) I would prefer having a final exam instead of a project.
c) The slides were generally helpful.
d) There was too little theory.

Projects

Projects

http://isabelle.in.tum.de/exercises/advanced/sorting/ex.pdf
http://isabelle.in.tum.de/exercises/advanced/mergesort/ex.pdf
http://isabelle.in.tum.de/exercises/advanced/tries/ex.pdf
http://isabelle.in.tum.de/exercises/advanced/interval/ex.pdf
http://isabelle.in.tum.de/exercises/advanced/regmachine/ex.pdf
http://isabelle.in.tum.de/exercises/proj/hanoi/ex.pdf
http://isabelle.in.tum.de/exercises/proj/euclid/ex.pdf
http://isabelle.in.tum.de/exercises/proj/compSE/ex.pdf
http://isabelle.in.tum.de/exercises/proj/bignat/ex.pdf
http://isabelle.in.tum.de/exercises/proj/optComp/ex.pdf