ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Dipl.-Inf. Achim D. Brucker
Dr. Burkhart Wolff

Computer-supported Modeling and Reasoning
http://www.infsec.ethz.ch/education/permanent/csmr/
(rev. 16814)
Submission date: –

Naïve Set Theory

In this exercise, we will study a particular version of a set theory, called *Naïve Set Theory*, originally proposed by Frege and still implicitly used by many mathematicians. We introduce some of its axioms, notation and, properties. At the end, we use a Paradox due to Russel in order to show that Naive Set Theory is inconsistent.

1. More on Isabelle

1.1. Backward Proof Control Structures

Revising our first proof scripts, it becomes clear that proof-scripts contain considerable repetition. Thus, more automation can be achieved by introducing control structures in the ISAR-language. These are:

control structures
ISAR

1. M,M' *sequential composition*: try tactic M; if it succeeds try tactic M'. — sequential composition (,)
2. M|M *alternative*: try tactic M; if it fails try tactic M'. — alternative (|)
3. M? *option*: try tactic M; if it fails report success. — option (?)
4. M+ *repetition*: try tactic M and repeat as long as no failure occurs. — repetition (+)

For example, instead of:

apply(*rule* X)
apply(*erule* Y)

we may write:

apply(*rule* X, *rule* Y)

Further, instead of:

apply(*drule* mp)
apply(*assumption*)
apply(*assumption*)
apply(*erule* disjE)
apply(*drule* mp)
apply(*drule* conjunct1)

we may write:

apply(*drule* mp,(*assumption*|*erule* disjE | *drule* conjunct1))+

1.2. Proof-State Massage

defer n
prefer n

The standard **apply**-command usually effects only the first subgoal. Thus, it may be desirable to rotate the list of subgoals in a proof state. The **defer** n or **prefer** n commands move a subgoal to the last or the first position.

rotate_tac n

For the choice of unifiers, the order of assumptions in a subgoal may be relevant. *rotate_tac* n rotates the assumptions of the first subgoal by n positions: from right to left if n is positive, and from left to right otherwise. The default value is one.

2. Naïve Set Theory

`naive_set.thy`

(Naïve) set theory has been formalized in Isabelle in the theory `naive_set.thy` (see allso Appendix A). Tab. 1 shows some hints on the syntax used in this theory.

In lecture "Naïve Set Theory", we have seen four elementary rules of set theory

$$\frac{P(t)}{t \in \{x \mid P(x)\}}\ \in\text{-}I \qquad \frac{t \in \{x \mid P(x)\}}{P(t)}\ \in\text{-}E$$

$$\frac{\forall x.x \in A \leftrightarrow x \in B}{A = B}\ =\text{-I} \qquad \frac{A = B}{\forall x.x \in A \leftrightarrow x \in B}\ =\text{-E}$$

Usual notation	Isabelle
$\{1, 2, 3\}$	{1,2,3}
$\in$	:
$\notin$	~:
$\{y \mid P(y)\}$	{y. P(y)}
$A \cap B$	A Int B

Usual notation	Isabelle
$A \cup B$	A Un B
$A \subseteq B$	A <= B
$A \setminus B$	A Minus B
$\mathcal{P}(A)$	Pow(A)

Table 1: Notations used in `naive_set.thy`

In the provided Isabelle theory, instead of those inference rules, we have two axioms `ext` and `Collect` which have been encoded and derived in Isabelle.

ext
Collect

For this exercise, download the `http://www.infsec.ethz.ch/education/permanent/csmr/material/naive_set.thy` file and install it in the same directory where you store your work. Use the following template as starting point for your own Isabelle file:

naive_set.thy

theory exercise2 = naive_set.thy:

end

3. Exercises

3.1. Exercise 11

Prove in Isabelle that the subset relation is a partial order, i.e., it is reflexive, transitive and antisymmetric.

3.2. Exercise 12

Prove $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ in Isabelle.

3.3. Exercise 13

Prove $\mathcal{P}(A) \subseteq \mathcal{P}(B) \leftrightarrow A \subseteq B$ in Isabelle

3.4. Exercise 14

Show that `NSet` is inconsistent, i.e., that $\bot$ can be derived in it. You should start like this:

```
lemma ex14: "False"
  apply( rule_tac P = "{A. A ∉A} ∈{A. A ∉A}" in notE)
```

The rule *classical_dual* will be useful.

A. Encoding Naïve Set Theory in Isabelle

```
1   theory naive_set = FOL:
2
3   nonterminals  i
4
5   types set = i
6
7   arities  i  ::  "term"
8
9   consts
10    "0"       ::  i                       ("0")
11    "1"       ::  i                       ("1")
12    "{}"      ::  set                     ("{}")
13     insert   ::  "[i, set] ⇒ set"
14    "op :"    ::  "[i, set] ⇒ o"          ("(_/ : _)" [50, 51] 50)
15    "<="      ::  "[set, set] ⇒ o"        (infixl 50)
16    Collect   ::  "[i ⇒ o] ⇒ set"
17    INTER     ::  "[set, i ⇒ set] ⇒ set"
18    UNION     ::  "[set, i ⇒ set] ⇒ set"
19    Minus     ::  "[set, set] ⇒ set"      (infixl 65)
20    Int       ::  "[set, set] ⇒ set"      (infixl 70)
21    Un        ::  "[set, set] ⇒ set"      (infixl 65)
22    Compl     ::  "set ⇒ set"
23    Pow       ::  "set ⇒ set"
24    UNIV      ::  set
25    Ball      ::  "[set, i ⇒ o] ⇒ o"
26    Bex       ::  "[set, i ⇒ o] ⇒ o"
27
28  syntax
29    "op :"       ::  "[i,set] ⇒ o"              ("op :")
30    "op ~:"      ::  "[i,set] ⇒ o"              ("(_/ ~: _)" [50, 51] 50)
31    "op ~:"      ::  "[i,set] ⇒ o"              ("op ~:")
32    "@Collect"   ::  "[pttrn, o] ⇒ set"         ("(1{_./ _})")
33    "@Finset"    ::  "args => set"              ("{(_)}")
34    "@INTER"     ::  "[pttrn, set, set] ⇒ set"  ("(3INT _:_./ _)" 10)
35    "@UNION"     ::  "[pttrn, set, set] ⇒ set"  ("(3UN _:_./ _)" 10)
36    "*Ball"      ::  "[pttrn, set, o] ⇒ o"      ("(3ALL _:_./ _)" [0, 0, 10] 10)
37    "*Bex"       ::  "[pttrn, set, o] ⇒ o"      ("(3EX _:_./ _)" [0, 0, 10] 10)
38
39  translations
40    "UNIV"       == "Compl({})"
41    "x ~: y"     == "~ (x : y)"
42    "{x, xs}"    == "insert(x, {xs})"
43    "{x}"        == "insert(x, {})"
44    "{x. P}"     == "Collect(λx. P)"
45    "INT x:A. B" == "INTER(A, (λx. B))"
46    "UN x:A. B"  == "UNION(A, (λx. B))"
47    "ALL x:A. P" == "Ball(A, (λx. P))"
48    "EX x:A. P"  == "Bex(A, (λx. P))"
49
50  axioms
51    ext:       "A = B ⟷(∀x. ((x:A) ⟷(x:B)))"
52    Collect:   "(t : { x. P(x) }) ⟷(P(t))"
53
54  syntax ("" output)
55    "_setle"       ::  "[set, set] ⇒ o"        ("op <=")
56    "_setle"       ::  "[set, set] ⇒ o"        ("(_/ <= _)" [50, 51] 50)
57    "_setless"     ::  "[set, set] ⇒ o"        ("op <")
58    "_setless"     ::  "[set, set] ⇒ o"        ("(_/ < _)" [50, 51] 50)
59
60  syntax (symbols)
61    "_setle"       ::  "[set, set] ⇒ o"        ("op ⊆")
62    "_setle"       ::  "[set, set] ⇒ o"        ("(_/ ⊆ _)" [50, 51] 50)
63    "_setless"     ::  "[set, set] ⇒ o"        ("op ⊂")
64    "_setless"     ::  "[set, set] ⇒ o"        ("(_/ ⊂ _)" [50, 51] 50)
65    "op Int"       ::  "[set, set] ⇒ set"      (infixl "∩" 70)
66    "op Un"        ::  "[set, set] ⇒ set"      (infixl "∪" 65)
67    "op :"         ::  "['a, set] ⇒ o"         ("op ∈")
68    "op :"         ::  "['a, set] ⇒ o"         ("(_/ ∈ _)" [50, 51] 50)
69    "op ~:"        ::  "['a, set] ⇒ o"         ("op ∉")
70    "op ~:"        ::  "['a, set] ⇒ o"         ("(_/ ∉ _)" [50, 51] 50)
71    "UN "          ::  "[idts, o] ⇒ o"         ("(3⋃ _./ _)" 10)
72    "INT "         ::  "[idts, o] ⇒ o"         ("(3⋂ _./ _)" 10)
73    "@UNION1"      ::  "[pttrn, set] ⇒ set"     ("(3⋃ _./ _)" 10)
```

```
    "@INTER1"   :: "[pttrn, set] ⇒ set"        ("(3⋂ _./ _)" 10)
    "@UNION"    :: "[pttrn, set, set] ⇒ set"   ("(3⋃ _∈_./ _)" 10)
    "@INTER"    :: "[pttrn, set, set] ⇒ set"   ("(3⋂ _∈_./ _)" 10)
    "_Ball"         :: "[pttrn, set, o] ⇒ o"       ("(3∀ _∈_./ _)" [0, 0, 10] 10)
    "_Bex"          :: "[pttrn, set, o] ⇒ o"       ("(3∃ _∈_./ _)" [0, 0, 10] 10)

translations
  "op ⊆" == "op <= :: [_ set, _ set] => o"


defs
    subset_def: "A <= B          ≡∀x. x∈A ⟶x∈B"
    empty_def:  "{}              ≡{x. False}"
    Minus_def:  "A Minus B       ≡{x. x ∈ A ∧x ∉ B}"
    Un_def:     "A Un B          ≡{x. x ∈ A ∨x ∈ B}"
    Int_def:    "A Int B         ≡{x. x ∈ A ∧x ∈ B}"
    Ball_def:   "Ball(A, P)      ≡(∀ x. x∈A ⟶P(x))"
    Bex_def:    "Bex(A, P)       ≡(∃ x. x∈A ∧P(x))"
    Compl_def:  "Compl(A)        ≡{x. ¬x:A}"
    INTER_def: "INTER(A, B)   ≡{y. ALL x:A. y:B(x)}"
    UNION_def: "UNION(A, B)   ≡  {y. EX x:A. y:B(x)}"
    insert_def: "insert (a, B)   ≡{x. x=a} Un B"
    Pow_def:    "Pow(A)          ≡{B. B <= A}"


lemma inI: "(P(t)) ⟹(t ∈ { x. P(x) })"
  apply(rule iffD2)
  apply(rule Collect)
  apply(assumption)
  done

lemma "inE2": "(t ∈{ x. P(x) }) ⟹P(t)"
  apply(rule iffD1)
  apply(rule Collect)
  apply(assumption)
  done

lemma inE: assumes p1: "(t ∈{ x. P(x) })" assumes p2: "P(t) ⟹R" shows "R"
  apply(rule p2)
  apply(rule inE2)
  apply(rule p1)
  done

lemma equalsI: "(∀x. x∈A ⟷x∈B) ⟹A = B"
  apply(rule iffD2)
  apply(rule ext)
  apply(assumption)
  done

lemma equalsE2: "A = B ⟹(∀ x. x∈A ⟷x∈B)"
  apply(rule iffD1)
  apply(rule ext)
  apply(assumption)
  done

lemma equalsE: assumes p1: "A = B" assumes p2: "(∀ x. x∈A ⟷x∈B) ⟹R"
  shows "R"
  apply(rule p2)
  apply(rule equalsE2)
  apply(rule p1)
  done

end
```