Roger’s Set Theory

Roger Bishop Jones
ICL Defence Systems

1. INTRODUCTION
This document provides a formal description of a variant of classical set theory axiomatised in
first order logic.
The set theory is roughly NBG with a complete hierarchy of closure operators. It is adapted from
DBC/RBJ/085 primarily by introduction of the primitive $C$.
The specification is written in the language SML (standard ML).
SML is described in ECS-LFCS-86-2 (University of Edinburgh, Laboratory for Foundations of
Computer Science).

2. ABSTRACT SYNTAX

infix 4 $\leftrightarrow$
infix 5 $\leftrightarrow$
infix 6
infix 7 $\land$
infix 8 $==\in$
infix 9 inn;
type var = string;

datatype
  formula = $\neg$ of formula |
  op $\leftrightarrow$ of (formula * formula) |
  $\forall$ of (var * formula) |
  op $==$ of (term * term) |
  op $\in$ of (term * term)

These constructs have their normal meaning, viz. negation, material implication, existential quan-
tification, equality and membership respectively.
and \( \text{term} = \) Var of var |
            comp of (var * formula) |
            Cl of (term * term) |
            \( \mu \) of term;

The meanings of these term constructors are:

- **Var** Variables
- **comp** comprehension, "comp(a,b)" is more usually written \{a | b\}
  comprehension rather than separation is allowed but does not necessarily yield a set.
  It yields the class of sets which satisfy the predicate \( \lambda a.b \).
- **Cl** Cl \((x,y)\) is a "universe" containing \( y \) satisfying closure condition \( x \).
  Closure condition \( x \) is obtained by iterating the closure conditions for every set \( z \in x \).

3. SUBSTITUTION

3.1. Set Theory in the Metalanguage (sml)

First a few definitions which enable us to talk about lists as if they were sets.

```sml
fun \( x \in [] \) = false |
   \( x \in (y::z) \) = \( x=y \) orelse \( x \in z \);

fun \( m \cup [] = [] \) |
   \( m \cup (a::b) = a \cup (m \cup b) \);

fun \[ \] -- a = \[ \] |
   (h::t) -- a = if \( h=a \) then t -- a else h::(t -- a);
```

3.2. Free Variables

We define the free variables in a formula or term as follows:

```sml
fun term_freevars (Var v) = [v] |
   term_freevars (comp (v,f)) = freevars f -- v |
   term_freevars (Cl (x,y)) = term_freevars x @ (term_freevars y) |
   term_freevars (\( \mu \) t) = term_freevars t

and term_list_freevars term_list = m\( \cup \) (map term_freevars term_list)
```
and freevars \( \neg f \) = freevars f

freevars \( f_1 \land f_2 \) = freevars f_1 @ (freevars f_2)

freevars \( \forall (v,f) \) = freevars f -- v

freevars \( t_1 ::= t_2 \) = term_freevars t_1 @ (term_freevars t_2)

freevars \( t_1 \in t_2 \) = term_freevars t_1 @ (term_freevars t_2);

3.3. Substitutions

First define substitutions into terms.

\[
\begin{align*}
\text{fun primed nv varl} &= \text{if } \text{nv} \notin \text{varl} \\
&\quad \text{then primed (nv""""""") varl} \\
&\quad \text{else } \text{nv};
\end{align*}
\]

\[
\begin{align*}
\text{fun free v terml} &= \text{primed v (term_list_freevars terml)};
\end{align*}
\]

\[
\begin{align*}
\text{datatype ('a,'b)inc = op inn of ('a * 'b);}
\end{align*}
\]

\[
\begin{align*}
\text{fun term t_subs_for i var inn (V ar v)} &= \text{if } \text{var} = v \\
&\quad \text{then term} \\
&\quad \text{else } \text{V ar v} \\
&\quad \text{term t_subs_for i var inn (comp(v,f))} \\
&= \text{if } \text{var} = v \\
&\quad \text{then comp(v,f)} \\
&\quad \text{else } \text{let val nv = primed v (term_freevars term @} \\
&\quad \quad \text{(freevars f -- v))} \\
&\quad \quad \text{val nf = term_subs_for i var inn} \\
&\quad \quad \text{((V ar nv) subs_for i var inn f) in} \\
&\quad \quad \text{comp(nv,nf)) end}
\end{align*}
\]

\[
\begin{align*}
\text{term t_subs_for i var inn (Cl (x,y))} &= \text{Cl (term t_subs_for i var inn x, term t_subs_for i var inn y)}
\end{align*}
\]

\[
\begin{align*}
\text{term t_subs_for i var inn (\mu t)} &= \mu (\text{term t_subs_for i var inn t})
\end{align*}
\]

and \( \text{curried_t_subs term i var term'} = \text{term t_subs_for i var inn term'} \)

Next substitutions into formulae.
and  \( \text{term subs_for ivar inn } (\neg \text{form}) \)
    \[ = \neg (\text{term subs_for ivar inn form}) \]
\( \text{term subs_for ivar inn } (\text{form1 } \equiv \text{form2}) \)
    \[ = (\text{term subs_for ivar inn form1}) \equiv (\text{term subs_for ivar inn form2}) \]
\( \text{term subs_for ivar inn } (\forall (\text{ivar}',\text{form})) \)
    \[ = (\forall (\text{ivar}',\text{term subs_for ivar inn form}) \]
\( \text{term subs_for ivar inn } (\text{t1} \implies \text{t2}) \)
    \[ = (\text{term t_subs_for ivar inn t1}) \implies (\text{term t_subs_for ivar inn t2}) \]
\( \text{term subs_for ivar inn } (\text{t1} \epsilon \text{t2}) \)
    \[ = (\text{term t_subs_for ivar inn t1}) \epsilon (\text{term t_subs_for ivar inn t2}) \]

\[ \text{and curried_subs a b c = a subs_for b inn c; } \]

4. DERIVED CONSTRUCTORS

fun op (a,b) = (\neg a) \equiv b;
fun op \& (a,b) = \neg (\neg a) \land (\neg b) ;
fun op \equiv (a,b) = (a \equiv b) \land (b \equiv a) ;
fun \exists (v,f) = \neg (\exists (v,\neg f)) ;
fun op \subseteq (a,b) = let val x = freev "x" [a,b]
        in \forall (x,(\text{Var x } \epsilon \text{a}) \equiv (\text{Var x } \epsilon \text{b}))
        end;

fun set a = let val x = (freev "x" [a])
        in \exists (x, a \epsilon (\text{Var x}))
        end;

val \emptyset = \text{comp("x",\neg (\text{Var "x" } \implies \text{Var "x"})});

fun pair a b = let val x = (freev "x" [a,b])
        in \text{comp (x,(\text{Var x}==\text{a}) \land (\text{Var x}==\text{b}))}
        end;

fun opair a b = pair a (pair a b);
fun unit a = pair a a;

fun Π a = let val x = (free v"x" [a])
in   comp (x,(Var x ⊆ a))
end;

fun ∪ a =
    let val x = (free v"x" [a]);
        val y = (free v"y" [a])
in   comp (x,∃ (y,(Var x ∈ (Var y)) ∧ (Var y ∈ a)))
end;

fun ∩ a =
    let val x = (free v"x" [a]);
        val y = (free v"y" [a])
in   comp (x,∀ (y,(Var y ∈ a) ∨ (Var x ∈ (Var y))))
end;

fun sv a = let val x = free v"x" [a]
        and y = free v"y" [a]
        and z = free v"z" [a]
in ∀(x,∀(y,∀(z,
            (opair (Var x) (Var y) ∈ a) ∧
            (opair (Var x) (Var z) ∈ a) ⇔ ((Var y) == (Var z)))))
end;

fun dom f = let val x = free v"x" [f]
        and y = free v"y" [f]
in   comp (x,∃(y,opair (Var x) (Var y) ∈ f))
end;

fun ran f = let val x = free v"x" [f]
        and y = free v"y" [f]
in   comp (x,∃(y,opair (Var y) (Var x) ∈ f))
end;

5. THE ABSTRACT DATA TYPE THEOREM

The set theory is formalised as a hilbert style axiom system by defining an abstract data type of theorems, where theorems are represented by formulae.

5.1. Inference Rules

Inference rules are formalised as functions from theorems to theorems, and axioms schemata as functions on any type yeilding theorems.
abstype theorem = of formula

with

5.1.1. Modus Ponens

fun MP (A) ((B ⊃ C)) = if A=B then C else A |
      MP x y = x;

5.1.2. Generalisation

fun UI (A) x = (∀x, A);

5.2. Propositional Axioms

fun P1 A B = ((A ⊃ B) ⊃ A);
fun P2 A B C = ((A ⊃ B ⊃ C) ⊃ (A ⊃ B) ⊃ (A ⊃ C));
fun P3 A B = (¬A ⊃ ¬B) ⊃ (B ⊃ A);

5.3. Axioms of Quantification

fun Q1 A x t = (∀x, A) ⊃ (t subs_for x in A);
fun Q2 A x = (A ⊃ (A ⊃ (∀Ax)));
fun Q3 A B x = (∀x, A ⊃ B) ⊃ (∀x, A) ⊃ (∀x, B);

5.4. Equality and Membership

fun EQ a b x =
  let
    val nx = primed x (term_list_freevars[a,b])
    in
    ((a==b) ⇔ (∀nx, (Var nx ∈ a ⇔ Var nx ∈ b)))
  end;

fun EXT a b x =
  let
    val nx = primed x (term_list_freevars[a,b])
    in
    ((a==b) ⇔ (∀nx, (a ∈ (Var nx) ⇔ b ∈ (Var nx))))
  end;
5.5. Comprehension

\[
\text{fun COM } x \, p = (\forall (x, \text{set } (\text{Var } x) \land p \iff (\text{Var } x) \in \text{comp}(x, p)));
\]

5.6. Closure

We have a closure operator Cl which forms closures of varying strength. The first parameter is the strength, and the second an arbitrary set. Cl(u,v) then denotes a set containing v closed under all weaker closure operations.

\[
\text{fun Cl } \emptyset \, x = (\text{set } \emptyset));
\]

\[
\text{fun Cl mem } u \, v = (\text{set } v \iff (\text{Cl } (u, v)) \land v \subseteq (\text{Cl } (u, v)));
\]

\[
\text{fun ClC } u \, v \, w \, x = (x \in \text{Cl}(u, v) \land w \in u \Rightarrow \text{Cl}(w, x) \in \text{Cl}(u, v));
\]

5.7. Choice

\[
\text{fun CH } x = (\text{set } (\mu x) \iff (\neg (x == \emptyset)) \iff (\mu x) \in x);
\]

5.8. Foundation

\[
\text{fun FO } x = \begin{aligned}
\text{let } \text{val } y = \text{primed } \text{"y" } (\text{term_freevars } x) \\
\text{in } (x == \emptyset \\
\exists (y, \text{Var } y \in x \land (\cap \text{pair } x (\text{Var } y)) == \emptyset))
\end{aligned}
\]

5.9. End of Abstract Data Type

end;

The types inferred by the SML compiler were:
type var = string  
datatype term = Cl of term * term | Var of var | µ of term | comp of var * 
  formula 
datatype formula = ∀ of var * formula | == of term * term | ≡ of formula * 
  formula | ∈ of term * term | ¬ of formula 
con comp = fn : (var * formula) → term 
con µ = fn : term → term 
con Var = fn : var → term 
con Cl = fn : (term * term) → term 
con ¬ = fn : formula → formula 
con ∈ = fn : (term * term) → formula 
con ≡ = fn : (formula * formula) → formula 
con ∀ = fn : (term * term) → formula 
val e = fn : ('"a * ('"a list)) → bool 
val m = fn : (('a list) list) → ('"a list) 
val -- = fn : ("'a list) * '"a) → ('"a list) 
val freevars = fn : formula → (var list) 
val term_list_freevars = fn : (term list) → (var list) 
val term_freevars = fn : term → (var list) 
val primed = fn : string → ((string list) → string) 
val freev = fn : string → ((term list) → string) 
val datatype ('a,'b) inc = inn of 'a * 'b 
con inn = fn : ('a * 'b) → (('a,'b) inc) 
val curried_subs = fn : term → (var → (formula → formula)) 
val subs_for = fn : (term * ((var,formula) inc)) → formula 
val curried_t_subs = fn : (var → (term → term)) 
val t_subs_for = fn : (term * ((var,term) inc)) → term 
val = fn : (formula * formula) → formula 
val ∧ = fn : (formula * formula) → formula 
val ↔ = fn : (formula * formula) → formula 
val k = fn : ('a * formula) → formula 
val ⊆ = fn : (term * term) → formula 
val set = fn : term → formula 
val ∅ = comp ('"x",¬((= (Var "x",Var "x")))) : term 
val pair = fn : term → (term → term) 
val opair = fn : term → (term → term) 
val unit = fn : term → term 
val P = fn : term → term 
val ∪ = fn : term → term 
val ∩ = fn : term → term 
val sv = fn : term → formula 
val dom = fn : term → term 
val ran = fn : term → term 
type theorem 
val FO = fn : term → theorem 
val CH = fn : term → theorem
val CIC = fn : term → (term → (term → (term → theorem)))
val Clmem = fn : term → (term → theorem)
val ClØ = fn : 'a → theorem
val COM = fn : var → (formula → theorem)
val EXT = fn : term → (term → (string → theorem))
val EQ = fn : term → (term → (string → theorem))
val Q3 = fn : formula → (formula → (var → theorem))
val Q2 = fn : formula → (string → theorem)
val Q1 = fn : formula → (var → (term → theorem))
val P3 = fn : formula → (formula → theorem)
val P2 = fn : formula → (formula → (formula → theorem))
val P1 = fn : formula → (formula → theorem)
val UI = fn : theorem → (var → theorem)
val MP = fn : theorem → (theorem → theorem)