Logic for Partial Functions in SML

Roger Bishop Jones
ICL Defence Systems

ABSTRACT

This document consists of a transcription into Standard ML of the proof theory of the Logic for Partial Functions due to Jen Huan Cheng.

1. INTRODUCTION

This is a transcription of the proof theory of (first order) lpf (without equality) into sml (standard ml). The material is transcribed from "A Logic for Partial Functions", UMCS-86-7-1, by Jen Huan Cheng, Department of Computer Science, University of Manchester. I have made some adjustments, for technical convenience, which I do not, but some might, consider material. SML is described in ECS-LFCS-86-2 (University of Edinburgh, Laboratory for the Foundations of Computer Science).

Concrete syntax is not considered. An sml datatype defines the abstract syntax of lpf, and an sml abstract datatype defines the type of provable sequents, as a proof checker in the LCF style. The constructors of the abstract datatype correspond to the axioms and inference rules of lpf.

The main areas in which this transcription differs from the original are:

formality

The presentation is more formal and is given in SML (this is a "literate script", the sections marked by a vertical bar on the left have been type checked by the sml compiler)

quantification

The unusual treatment of quantification adopted by Cheng has been dropped, since it seemed to me to clutter the syntax to no effect.

substitution

There is a full specification of substitution. The apparent increase in complexity here is not caused by the change in handling of quantifiers, it arises simply from spelling out the details where Cheng finds this unnecessary.

identifiers

The sets of identifiers used for propositional and individual variables, and for individual and function constants are not assumed to be disjoint. In the abstract syntax presented no ambiguity arises from this, and the proof theory should be sound. Similarly the same function name may be used at various arities, and will be treated as if it denoted a distinct function at each arity. We do not address disambiguation in
concrete syntax, which might be achieved by the use of disjoint sets of identifiers.

sets
Where Cheng uses sets in his metalanguage, I use lists. The effect is that inference rules normally operate only on the propositions at the head of the lists of propositions forming the sequents. In the weakening rules, which are formalised so as to permit arbitrary reordering of the lists, this weakness is remedied.

syntax
The abstract syntax is simplified, first by discarding those elements which are not primitive and do not occur in the proof rules, secondly by reducing the number of non-terminal symbols to two, ‘formula’ and ‘term’.

Despite these ‘adjustments’, the formal statement of the axioms and rules follows very closely the presentation on pp 54-56 of UMCS-86-7-1.

2. ABSTRACT SYNTAX

```sml
infix 6
infix 7 App;

type prop_var = string;
type ind_var = string;
type ind_cons = string;
type fun_cons = string;

datatype formula = uu |
| P of (prop_var * term list) |
| E of term |
| ¬ of formula |
| Δ of formula |
| δ of formula |
| op of (formula * formula) |
| ∃ of (ind_var * formula) |

and term = Constant of ind_cons |
| Variable of ind_var |
| op App of (fun_cons * term list);```
3. SUBSTITUTION

3.1. Set Theoretic Preliminaries

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

First, membership:

\[
\text{infix 8 } \in; \\
\text{fun } x \in [] = \text{false} | x \in (y::z) = x = y \text{ or else } x \in z;
\]

‘set’ inclusion:

\[
\text{infix 8 } \subseteq; \\
\text{fun } [] \subseteq y = \text{true} | (h::t) \subseteq y = h \in y \text{ and also } t \subseteq y;
\]

Union of a set of sets, and removal of an element from a set:

\[
\text{fun } \bigcup [] = [] | \bigcup (a::b) = a @ (\bigcup b); \\
\text{infix 8 } \rightarrow; \\
\text{fun } [] \rightarrow a = [] | (h::t) \rightarrow a = \text{if } h=a \text{ then } t \rightarrow a \text{ else } h::(t \rightarrow a);
\]

3.2. Free Variables

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

\[
\text{fun } \text{term_freevars (Constant } c) = [] | \text{term_freevars (Variable } v) = [v] | \text{term_freevars (f App } \text{term_list}) = \text{term_list_freevars term_list} \\
\text{and } \text{term_list_freevars term_list} = \bigcup (\text{map term_freevars term_list});
\]

\[
\text{fun } \text{freevars uu} = [] | \text{freevars (P (pv,term_list))} = \text{term_list_freevars term_list} | \text{freevars (E term)} = \text{term_freevars term} | \text{freevars (¬ formula)} = \text{freevars formula} | \text{freevars (Δ formula)} = \text{freevars formula} | \text{freevars (δ formula)} = \text{freevars formula} | \text{freevars (form1 form2)} = \bigcup [\text{freevars form1, freevars form2}] | \text{freevars (∃ (v,form))} = \text{freevars form } \rightarrow v;
\]
3.3. Permissible Substitutions

A substitution of a term for a variable in a formula is permissible only when no variable free in the term will become bound after the substitution. We make this condition precise here.

```sml
infix 8 free_for;
infix 9 inn;
datatype ('a,''a) pair = op inn of ('a * ''a);

fun term free_for ind_var inn uu = true |
term free_for ind_var inn (P (pv,tl)) = true |
term free_for ind_var inn (E term’) = true |
term free_for ind_var inn (¬ form) = term free_for ind_var inn form |
term free_for ind_var inn (Δ form) = term free_for ind_var inn form |
term free_for ind_var inn (δ form) = term free_for ind_var inn form |
term free_for ind_var inn (form1 form2) = term free_for ind_var inn form1 |
andalso term free_for ind_var inn form2 |
term free_for ind_var inn (∃ (ind_var’,form)) = not(ind_var’ ∈ term_freevars term) |
orelse not(ind_var ∈ freevars form);
```

3.4. Substitutions

First define substitutions into terms.

```sml
infix 8 t_subs_for;
fun term t_subs_for i var inn (Constant c) = (Constant c) |
term t_subs_for i var inn (Variable v) = if i var = v then term |
else Variable v |
term t_subs_for i var inn (f App t_list) = f App (map (curried_t_subs term i var) t_list)
and curried_t_subs term i var term’ = term t_subs_for i var inn term’;
```

Next substitutions into formulae.
infix 8 subs_for;

fun term_subs_for ind_var inn uu
    = uu |
    term_subs_for ind_var inn (P (pv,tl))
    = P (pv, map (curried_t_subs term ind_var) tl) |
    term_subs_for ind_var inn (E term’)
    = E (term t_subs_for ind_var inn term’) |
    term_subs_for ind_var inn (¬ form)
    = (¬ (term_subs_for ind_var inn form)) |
    term_subs_for ind_var inn (Δ form)
    = Δ (term_subs_for ind_var inn form) |
    term_subs_for ind_var inn (δ form)
    = δ (term_subs_for ind_var inn form) |
    term_subs_for ind_var inn (form1 form2)
    = (term_subs_for ind_var inn form1)
        (term_subs_for ind_var inn form2) |
    term_subs_for ind_var inn (∃ (ind_var’,form))
    = (∃ (ind_var’,term_subs_for ind_var inn form)) |
and curried_subs a b c = a subs_for b inn c;

4. THE SEQUENT CALCULUS

infix →
datatype sequent = o p of (formula list * formula list);

4.1. The Abstract Data Type of Sequent-theorems

abstype sequent_theorem = of sequent with

4.2. The Axiom Schemata

fun axiom_1 A = ([A] -> [A])
fun axiom_2 A = ([A, ¬ A] -> [])
val axiom_3 = ([uu] -> [])
val axiom_4 = ([¬ uu] -> [])
fun axiom_5 c = ([] -> [E (Constant c)])
fun axiom_6 y = ([] -> [E (Variable y)])
fun axiom_7 t = ([¬ (E t)] -> [])
4.3. The Propositional Inference Rules

$\Delta \rightarrow$:  

fun rule_1 (( (¬A::Γ) → Σ)) 
( ((A’::Γ’)→ Σ’))  
= if A=A’ andalso Γ=Γ” andalso Σ=Σ’  
then ((A::Γ)→ Σ)  
else axiom_3  
rule_1 st1 st2 = axiom_3

$\rightarrow \Delta$:  

fun rule_2a ( (Γ→ (A::Σ)))  
= ( (Γ→ (Δ A::Σ)))  
rule_2a st1 = axiom_3

fun rule_2b ( (Γ→ (¬ A::Σ)))  
= ( (Γ→ (Δ A::Σ)))  
rule_2b st1 = axiom_3

$\neg \Delta \rightarrow$:  

fun rule_3a ( (Γ→ (A::Σ)))  
= ( ( (¬(Δ A)::Γ)→ Σ))  
rule_3a st1 = axiom_3

fun rule_3b ( (Γ→ (¬ A::Σ)))  
= ( ( (¬(Δ A)::Γ)→ Σ))  
rule_3b st1 = axiom_3

$\rightarrow \neg \Delta$:  

fun rule_4 (( (¬A::Γ) → Σ)) 
( ((A’::Γ’)→ Σ’))  
= if A=A’ andalso Γ=Γ” andalso Σ=Σ’  
then (Γ→ (¬(Δ A)::Σ))  
else axiom_3  
rule_4 st1 st2 = axiom_3

$\neg \neg \rightarrow$:  

fun rule_5 ( ((A::Γ) → Σ))  
= ( (¬(¬ A)::Γ)→ Σ)  
rule_5 st1 = axiom_3
\(\rightarrow \neg \neg:\)

\[
\begin{align*}
\text{fun rule}_6 \quad & (\Gamma \rightarrow (A::\Sigma)) \\
= \quad & (\Gamma \rightarrow (\neg(\neg A)::\Sigma)) \\
\text{rule}_6 \quad & \text{st1 = axiom}_3
\end{align*}
\]

\(\rightarrow:\)

\[
\begin{align*}
\text{fun rule}_7 \quad & (((A::\Gamma) \rightarrow \Sigma)) \\
& ((B::\Gamma') \rightarrow \Sigma') \\
= \quad & \text{if } \Gamma = \Gamma' \text{ andalso } \Sigma = \Sigma' \\
& \text{then } (((A \: B)::\Gamma) \rightarrow \Sigma) \\
& \text{else } \text{axiom}_3 \\
\text{rule}_7 \quad & \text{st1 st2 = axiom}_3
\end{align*}
\]

\(\rightarrow:\)

\[
\begin{align*}
\text{fun rule}_8a \ B \quad & (\Gamma \rightarrow (A::\Sigma)) \\
= \quad & (\Gamma \rightarrow ((A \: B)::\Sigma)) \\
\text{rule}_8a \ B \quad & \text{st1 = axiom}_3 \\
\text{fun rule}_8b \ A \quad & (\Gamma \rightarrow (B::\Sigma)) \\
= \quad & (\Gamma \rightarrow ((A \: B)::\Sigma)) \\
\text{rule}_8b \ A \quad & \text{st1 = axiom}_3
\end{align*}
\]

\(\neg \rightarrow:\)

\[
\begin{align*}
\text{fun rule}_9a \ B \quad & (((\neg A)::\Gamma) \rightarrow \Sigma) \\
= \quad & ((\neg (A \: B)::\Gamma) \rightarrow \Sigma) \\
\text{rule}_9a \ B \quad & \text{st1 = axiom}_3 \\
\text{fun rule}_9b \ A \quad & (((\neg B)::\Gamma) \rightarrow \Sigma) \\
= \quad & ((\neg (A \: B)::\Gamma) \rightarrow \Sigma) \\
\text{rule}_9b \ A \quad & \text{st1 = axiom}_3
\end{align*}
\]

\(\rightarrow \neg:\)

\[
\begin{align*}
\text{fun rule}_10 \quad & (\Gamma \rightarrow (\neg A::\Sigma)) \\
& (\Gamma' \rightarrow (\neg B::\Sigma')) \\
= \quad & \text{if } \Gamma = \Gamma' \text{ andalso } \Sigma = \Sigma' \\
& \text{then } (\Gamma \rightarrow (\neg (A \: B)::\Sigma)) \\
& \text{else } \text{axiom}_3 \\
\text{rule}_10 \quad & \text{st1 st2 = axiom}_3
\end{align*}
\]
4.4. The Quantification Rules

\[\exists \rightarrow:\]

\[
\text{fun rule}_11 \ x \ (\ ((A :: \Gamma) \rightarrow \Sigma)) = \begin{cases} 
\text{not}(x \in (\cup (\text{map freevars} \ (\Gamma \odot \Sigma)))) & \text{if} \\
(\exists(x,A) :: \Gamma) \rightarrow \Sigma & \text{then} \\
\text{else axiom}_3 & \text{else}
\end{cases}
\]

\[
\text{rule}_11 \ x \ \text{st}1 = \text{axiom}_3
\]

\[\rightarrow \exists:\]

\[
\text{fun rule}_12 \ x \ A \ (\ (\Gamma \rightarrow (E \ t :: \Sigma))) = \begin{cases} 
\Gamma = \Gamma' \land \Sigma = \Sigma' & \text{if} \\
\text{andalso \ free\_for \ x \ inn \ A} & \text{andalso} \\
\text{andalso \ A' = t \ subs\_for \ x \ inn \ A} & \text{andalso not}(x \in (\cup (\text{map freevars} \ (\Gamma \odot \Sigma)))) \\
(\exists(x,A) :: \Sigma) & \text{then} \\
\text{else axiom}_3 & \text{else}
\end{cases}
\]

\[
\text{rule}_12 \ x \ A \ \text{st}1 \ \text{st}2 = \text{axiom}_3
\]

\[\neg \exists \rightarrow:\]

\[
\text{fun rule}_13 \ x \ A \ (\ (\Gamma \rightarrow (E \ t :: \Sigma))) = \begin{cases} 
(\rightarrow (A' :: \Gamma') \rightarrow \Sigma') & \text{if} \\
\Gamma = \Gamma' \land \Sigma = \Sigma' & \text{andalso} \\
\text{andalso \ free\_for \ x \ inn \ A} & \text{andalso} \\
\text{andalso \ A' = t \ subs\_for \ x \ inn \ A} & \text{andalso not}(x \in (\cup (\text{map freevars} \ (\Gamma \odot \Sigma)))) \\
(\neg (\exists(x,A) :: \Gamma) \rightarrow \Sigma) & \text{then} \\
\text{else axiom}_3 & \text{else}
\end{cases}
\]

\[
\text{rule}_13 \ x \ A \ \text{st}1 \ \text{st}2 = \text{axiom}_3
\]

\[\rightarrow \neg \exists:\]

\[
\text{fun rule}_14 \ x \ (\ (\Gamma \rightarrow (\neg A :: \Sigma))) = \begin{cases} 
\text{not}(x \in (\cup (\text{map freevars} \ (\Gamma \odot \Sigma)))) & \text{if} \\
(\Gamma \rightarrow (\neg (\exists(x,A) :: \Sigma)) & \text{then} \\
\text{else axiom}_3 & \text{else}
\end{cases}
\]

\[
\text{rule}_14 \ x \ \text{st}1 = \text{axiom}_3
\]
4.5. Structural Rules

Since I use lists rather than sets a formulation of the weakening rules which allows re-ordering is given.

fun rule_15 \( \Gamma' \subseteq \Gamma \) andalso \( \Sigma' \subseteq \Sigma \)
then \( (\Gamma' \rightarrow \Sigma') \)
else axiom_3

rule_15 \( \Gamma' \rightarrow \Sigma' \) st1 = axiom_3

4.6. Cut Rule

fun rule_16 \( (\Gamma_1 \rightarrow (A::\Sigma_1)) \)
(\((A'::\Gamma_2) \rightarrow \Sigma_2)\)
then \( ((\Gamma_1 @ \Gamma_2) \rightarrow (\Sigma_1 @ \Sigma_2)) \)
else axiom_3

rule_16 st1 st2 = axiom_3

end;
5. TYPES

The following types are inferred by the SML compiler from the above script:

> type prop_var = string
> type ind_var = string
> type ind_cons = string
> type fun_cons = string
> datatype term = App of fun_cons * (term list) | Constant of ind_cons |
  Variable of ind_var
  datatype formula = E of term | P of prop_var * (term list) | ∃ of ind_var |
    * formula | Δ of formula | δ of formula | of formula * formula | uu | ¬ |
      of formula
  con Variable = fn : ind_var → term
  con Constant = fn : ind_cons → term
  con App = fn : (fun_cons * (term list)) → term
  con → = fn : formula → formula
  con uu = uu : formula
  con = fn : (formula * formula) → formula
  con δ = fn : formula → formula
  con Δ = fn : formula → formula
  con ∃ = fn : (ind_var * formula) → formula
  con P = fn : (prop_var * (term list)) → formula
  con E = fn : term → formula
  > val ∈ = fn : ("a * ("a list)) → bool
  > val ≤ = fn : (("a list) * ("a list)) → bool
  > val ∪ = fn : (("a list) list) → ("a list)
  > val → = fn : (("a list) * "a) → ("a list)
  > val term_list_freevars = fn : (term list) → (ind_var list)
  val term_freevars = fn : term → (ind_var list)
  > val freevars = fn : formula → (ind_var list)
  > datatype ("a,"b) pair = inn of "a * "b
  con inn = fn : ("a * "b) → ("a,"b) pair
  > val free_for = fn : (term * ((ind_var,formula) pair)) → bool
  > val curried_t_subs = fn : term → (ind_var → (term → term))
    val t_subs_for = fn : (term * ((ind_var,term) pair)) → term
  > val curried_subs = fn : term → (ind_var → (formula → formula))
    val subs_for = fn : (term * ((ind_var,formula) pair)) → formula
  > datatype sequent = → of (formula list) * (formula list) → sequent
  con → = fn : ((formula list) * (formula list)) → sequent
  > type sequent_theorem
  val rule_16 = fn : sequent_theorem → (sequent_theorem → sequent_theorem)
  val rule_15 = fn : (formula list) → ((formula list) → (sequent_theorem → sequent_theorem))
  > val rule_14 = fn : ind_var → (sequent_theorem → sequent_theorem)
  val rule_13 = fn : ind_var → (formula → (sequent_theorem → (sequent_theorem → sequent_theorem)))
  val rule_12 = fn : ind_var → (formula → (sequent_theorem → (sequent_theorem → (sequent_theorem)))))
sequent_theorem \rightarrow \text{sequent_theorem})))
\begin{align*}
\text{val rule}_{11} &= \text{fn : ind\_var} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{10} &= \text{fn : sequent\_theorem} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{9b} &= \text{fn : formula} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{9a} &= \text{fn : formula} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{8b} &= \text{fn : formula} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{8a} &= \text{fn : formula} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{7} &= \text{fn : sequent\_theorem} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{6} &= \text{fn : sequent\_theorem} \rightarrow \text{sequent\_theorem} \\
\text{val rule}_{5} &= \text{fn : sequent\_theorem} \rightarrow \text{sequent\_theorem} \\
\text{val rule}_{4} &= \text{fn : sequent\_theorem} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val rule}_{3b} &= \text{fn : sequent\_theorem} \rightarrow \text{sequent\_theorem} \\
\text{val rule}_{3a} &= \text{fn : sequent\_theorem} \rightarrow \text{sequent\_theorem} \\
\text{val rule}_{2b} &= \text{fn : sequent\_theorem} \rightarrow \text{sequent\_theorem} \\
\text{val rule}_{2a} &= \text{fn : sequent\_theorem} \rightarrow \text{sequent\_theorem} \\
\text{val rule}_{1} &= \text{fn : sequent\_theorem} \rightarrow (\text{sequent\_theorem} \rightarrow \text{sequent\_theorem}) \\
\text{val axiom}_{7} &= \text{fn : term} \rightarrow \text{sequent\_theorem} \\
\text{val axiom}_{6} &= \text{fn : ind\_var} \rightarrow \text{sequent\_theorem} \\
\text{val axiom}_{5} &= \text{fn : ind\_cons} \rightarrow \text{sequent\_theorem} \\
\text{val axiom}_{4} &= \text{- : sequent\_theorem} \\
\text{val axiom}_{3} &= \text{- : sequent\_theorem} \\
\text{val axiom}_{2} &= \text{fn : formula} \rightarrow \text{sequent\_theorem} \\
\text{val axiom}_{1} &= \text{fn : formula} \rightarrow \text{sequent\_theorem}
\end{align*}