Ott: Effective Tool Support for the Working Semanticist

Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, Rok Strniša

Defining real programming languages rigorously is a rarely achieved challenge, partly due to scale. Ott (ICFP'07) is a tool to ease the task. It translates concise and readable ASCII source into LaTeX, Coq, HOL, and Isabelle/HOL. Ott can be used for substantial developments, e.g. OCaml$_{\text{light}}$ (Owens, ESOP'08) and Java Module Systems (Strniša, OOPSLA'07), and also to quickly formalise small calculi.

Ott source for an untyped λ-calculus

```
metavar termvar, x ::=   {{ com  term variable }}
{{ hol string}} {{ isa string}} {{ coq nat}} {{ coq-equality }}
{{ ocaml int}} {{ lex alphanum}} {{ tex \mathit{[[termvar]]} }}

grammar
t :: 't_' ::=
  | x              ::   :: Var
  | \ x . t        ::   :: Lam          (+ bind x in t +)
  | t t'           ::   :: App
  | ( t )          :: S :: Paren        {{ icho [[t]]  }}
  | { t / x } t' :: M :: Tsub         {{ icho (tsubst_t [[t]] [[x]] [[t']])}}

v :: 'v_' ::=
  | \ x . t        ::   :: Lam

terminals :: 'terminals_' ::=
  | \              ::   :: lambda       {{ tex \lambda }}
  | -->            ::   :: red          {{ tex \longrightarrow }}

subrules
  v <:: t

substitutions
  single t x :: tsubst

defns
Jop :: '' ::=

 defn
t1 --> t2 :: ::reduce::'' {{ com [[t1]] reduces to [[t2]]}} by


    -------------------------  :: ax_app
    (\x.t12) v2 -->  {v2/x}t12

    t1 --> t1'
    -------------- :: ctx_app_fun
    t1 t --> t1' t

    t1 --> t1'
    -------------- :: ctx_app_arg
    v t1 --> v t1'
```

binding specifications

translations of meta-syntax, for provers

translations for typesetting

generating substitutions (fully concrete rep.)

[also free variable fns, subgrammar predicates, fancy binding structures, OCaml boilerplate, parsing and filtering, list forms, context grammars, simple modular semantics, ...]

concrete and abstract syntax

subgrammars

inductive relations

readable rules!

Generated LaTeX

$termvar, x$ term variable

t ::=
- x
- $\lambda x\,.\,t$ bind x in t
- $t\ t'$

v ::=
- $\lambda x\,.\,t$

$\boxed{t_1 \longrightarrow t_2}$ t_1 reduces to t_2

$$\frac{}{(\lambda x\,.\,t_{12})\ v_2 \longrightarrow \{v_2/x\}t_{12}}\ \text{AX\_APP}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t \longrightarrow t_1'\ t}\ \text{CTX\_APP\_FUN}$$

$$\frac{t_1 \longrightarrow t_1'}{v\ t_1 \longrightarrow v\ t_1'}\ \text{CTX\_APP\_ARG}$$

Generated HOL

```
(* generated by Ott 0.10.14 from: ../tests/non_super_tabular.ott ../tests/test10_poster.ott *)
(* to compile: Holmake test10_posterTheory.uo   *)
(* for interactive use:
  app load ["pred_setTheory","finite_mapTheory","stringTheory","containerTheory","ottLib"];
*)

open HolKernel boolLib Parse bossLib ottLib;
infix THEN THENC |-> ## ;
local open arithmeticTheory stringTheory containerTheory pred_setTheory listTheory
  finite_mapTheory in end;

val _ = new_theory "test10_poster";

(** syntax *)
val _ = type_abbrev("termvar", ``:string``);
val _ = Hol_datatype '
t =
   t_Var of termvar
 | t_Lam of termvar => t
 | t_App of t => t
';


(** subrules *)
val _ = ottDefine "is_v_of_t" '
    ( is_v_of_t (t_Var x) = F)
/\  ( is_v_of_t (t_Lam x t) = (T))
/\  ( is_v_of_t (t_App t t') = F)
';

(** substitutions *)
val _ = ottDefine "tsubst_t" '
    ( tsubst_t t5 x5 (t_Var x) = (if x=x5 then t5 else (t_Var x)))
/\  ( tsubst_t t5 x5 (t_Lam x t) = t_Lam x (if MEM x5 [x] then t else (tsubst_t t5 x5 t)))
/\  ( tsubst_t t5 x5 (t_App t t') = t_App (tsubst_t t5 x5 t) (tsubst_t t5 x5 t'))
';

(** definitions *)
(* defns Jop *)

val (Jop_rules, Jop_ind, Jop_cases) = Hol_reln '
(* defn reduce *)

( (* ax_app *)  !(x:termvar) (t12:t) (v2:t) . (clause_name "ax_app") /\
((is_v_of_t v2))
 ==>
( ( reduce (t_App  (t_Lam x t12)  v2)  (tsubst_t  v2   x   t12 )  )))

/\ ( (* ctx_app_fun *)  !(t1:t) (t:t) (t1':t) . (clause_name "ctx_app_fun") /\
(( ( reduce t1 t1' )))
 ==>
( ( reduce (t_App t1 t) (t_App t1' t) )))

/\ ( (* ctx_app_arg *)  !(v:t) (t1:t) (t1':t) . (clause_name "ctx_app_arg") /\
((is_v_of_t v) /\
( ( reduce t1 t1' )))
 ==>
( ( reduce (t_App v t1) (t_App v t1') )))

';


val _ = export_theory ();
```

A more typical rule (from OCaml$_{\text{light}}$), in Ott and LaTeX

```
E |- e1 : t1 ... E |- en : tn
E |- field_name1 : t->t1 ... E |- field_namen : t->tn
t = (t1', ..., tl') typeconstr_name
E |- typeconstr_name gives typeconstr_name:kind {field_name1'; ...; field_namem'}
field_name1...field_namen PERMUTES field_name1'...field_namem'
length (e1)...(en)>=1
------------------------------------------------------------------------------ :: record_constr
E |- {field_name1=e1; ...; field_namen=en} : t
```

$$\frac{\begin{array}{l} E \vdash e_1 : t_1 \quad \ldots \quad E \vdash e_n : t_n \\ E \vdash field\_name_1 : t \rightarrow t_1 \quad \ldots \quad E \vdash field\_name_n : t \rightarrow t_n \\ t = (t_1', \ldots, t_l')\ typeconstr\_name \\ E \vdash typeconstr\_name \ \triangleright \ typeconstr\_name : kind\,\{field\_name_1' ; \ldots ; field\_name_m'\} \\ field\_name_1 \ldots field\_name_n \ \textbf{PERMUTES}\ field\_name_1' \ldots field\_name_m' \\ \textbf{length}\,(e_1)\ldots(e_n) \geq 1 \end{array}}{E \vdash \{field\_name_1 = e_1 ; \ldots ; field\_name_n = e_n\} : t}\ \text{JTE\_RECORD\_CONSTR}$$

University of Cambridge
INRIA

http://www.cl.cam.ac.uk/users/pes20/ott