Artifacts for Semantics: An OCaml Experiment

Daniel Patterson and Gabriel Scherer

June 2, 2017

Northeastern University

Goals for Semantic Artifacts

Our paper:

FunTAL: Reasonably Mixing a Functional Language with Assembly.
Daniel Patterson, Jamie Perconti, Christos Dimoulas, and Amal Ahmed.
To appear in *PLDI 2017*.

We wanted an artifact that was:

Goals for Semantic Artifacts

Our paper:

FunTAL: Reasonably Mixing a Functional Language with Assembly.
Daniel Patterson, Jamie Perconti, Christos Dimoulas, and Amal Ahmed.
To appear in *PLDI 2017*.

We wanted an artifact that was:

- easy for researchers to write.

Goals for Semantic Artifacts

Our paper:

FunTAL: Reasonably Mixing a Functional Language with Assembly.
Daniel Patterson, Jamie Perconti, Christos Dimoulas, and Amal Ahmed.
To appear in *PLDI 2017*.

We wanted an artifact that was:

- easy for researchers to write.
- didn't require reader to install software.

Goals for Semantic Artifacts

Our paper:

FunTAL: Reasonably Mixing a Functional Language with Assembly.
Daniel Patterson, Jamie Perconti, Christos Dimoulas, and Amal Ahmed.
To appear in *PLDI 2017*.

We wanted an artifact that was:

- easy for researchers to write.
- didn't require reader to install software.
- matched syntax of paper.

Non-goals (though okay if happen)

We weren't trying to:

Non-goals (though okay if happen)

We weren't trying to:

- increase trust in results (i.e., machine assisted proofs).

Non-goals (though okay if happen)

We weren't trying to:

- increase trust in results (i.e., machine assisted proofs).
- aid in experimenting with language semantics.

Our approach

Our approach

- Single step interpreter.

Our approach

- Single step interpreter.
- Syntax directed typechecker.

Our approach

- Single step interpreter.
- Syntax directed typechecker.
- Parser / Pretty Printer that matches paper (modulo super/subscripts, greek letters).

Our approach

- Single step interpreter.
- Syntax directed typechecker.
- Parser / Pretty Printer that matches paper (modulo super/subscripts, greek letters).
- Web frontend with an editor, all examples from paper, forwards/backwards stepper.

Our approach: Interpreter

Translation reduction relation to OCaml.

1.1.18 Reduction Relation

1.1.19 Instruction Sequence Reduction Relation $\boxed{\langle M \mid I\rangle \longrightarrow \langle M' \mid I'\rangle}$

$$\langle (H, R, S) \mid \texttt{aop}\ r_d, r_s, u; I\rangle \longrightarrow \langle (H, R[r_d \mapsto \delta(aop, R(r_s), \hat{R}(u))], S) \mid I\rangle$$

$$\langle (H, R, S) \mid \texttt{bnz}\ r, u; I\rangle \longrightarrow \langle (H, R, S) \mid I\rangle \quad \text{if}$$

$$\langle (H, R, S) \mid \texttt{bnz}\ r, u; I\rangle \longrightarrow \langle (H, R, S) \mid I'[\overline{\omega}/\Delta]\rangle \quad \text{if } R(r) =$$
$$\text{where } \hat{R}(u) = \ell[\overline{\omega}] \text{ and } H(\ell) = \texttt{code}[\Delta]\{\chi; \sigma\}^{q}.I'$$

$$\langle (H, R, S) \mid \texttt{ld}\ r_d, r_s[i]; I\rangle \longrightarrow \langle (H, R[r_d \mapsto w_i], S) \mid I\rangle$$
$$\text{where } R(r_s) = \ell \text{ and } H(\ell) = \langle w_0, \ldots, w_i, \ldots, w_n\rangle$$

$$\langle (H, R, S) \mid \texttt{st}\ r_d[i], r_s; I\rangle \longrightarrow \langle (H[\ell \mapsto \langle w_0, \ldots, w', \ldots, w_n\rangle], R, S) \mid I\rangle$$
$$\text{where } R(r_s) = w', R(r_d) = \ell, \text{and } H(\ell) = \langle w_0, \ldots, w_i,$$

$$\langle (H, R, \overline{w} :: S) \mid \texttt{ralloc}\ r_d, n; I\rangle \longrightarrow \langle (H[\ell \mapsto \langle \overline{w}\rangle], R[r_d \mapsto \ell], S) \mid I\rangle \quad \text{if } \ell \notin \mathrm{dom}(H), \text{le}$$

$$\langle (H, R, \overline{w} :: S) \mid \texttt{balloc}\ r_d, n; I\rangle \longrightarrow \langle (H[\ell \mapsto \langle \overline{w}\rangle], R[r_d \mapsto \ell], S) \mid I\rangle \quad \text{if } \ell \notin \mathrm{dom}(H), \text{le}$$

$$\langle (H, R, S) \mid \texttt{mv}\ r_d, u; I\rangle \longrightarrow \langle (H, R[r_d \mapsto \hat{R}(u)], S) \mid I\rangle$$

$$\langle (H, R, S) \mid \texttt{unpack}\ \langle \alpha, r_d\rangle\ u; I\rangle \longrightarrow \langle (H, R[r_d \mapsto w], S) \mid I[\tau'/\alpha]\rangle$$
$$\text{where } \hat{R}(u) = \texttt{pack}\langle \tau', w\rangle \text{ as } \exists\alpha.\tau$$

$$\langle (H, R, S) \mid \texttt{unfold}\ r_d, u; I\rangle \longrightarrow \langle (H, R[r_d \mapsto w], S) \mid I\rangle \quad \text{where } \hat{R}(u) = \text{f}$$

$$\langle (H, R, S) \mid \texttt{salloc}\ n; I\rangle \longrightarrow \langle (H, R, \overline{()} :: S) \mid I\rangle \quad \text{le}$$

$$\langle (H, R, \overline{w} :: S) \mid \texttt{sfree}\ n; I\rangle \longrightarrow \langle (H, R, S) \mid I\rangle \quad \text{le}$$

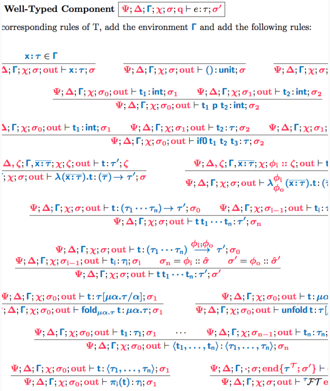
$$\langle (H, R, S) \mid \texttt{sld}\ r_d, i; I\rangle \longrightarrow \langle (H, R[r_d \mapsto w_i], S) \mid I\rangle \quad \text{where } S = w_0 :: \cdots$$

$$\langle (H, R, S) \mid \texttt{sst}\ i, r_s; I\rangle \longrightarrow \langle (H, R, S') \mid I\rangle$$
$$\text{where } R(r_s) = w',$$
$$S = w_0 :: \cdots :: w_i :: S_0, \text{ and } S' = w_0 :: \cdots :: w' :: S_0,$$

```ocaml
let reduce (c : mem * instr list) =
  match c with
  | ((hm,rm,sm), Iaop (_, op, rd, rs, u)::is) ->
    ((hm, replace rm rd (delta op (List.Assoc.find_exn rm rs) (ru rm u)), sm), is)
  | ((hm,rm,sm), Ibnz (_, r,u)::is) ->
    begin match List.Assoc.find rm r with
      | Some (WInt (_, 0)) -> ((hm,rm,sm), is)
      | Some (WInt _) ->
        let hc os l =
          match List.Assoc.find hm l with
          | Some (_, (HCode (delt,ch,s,qr,is))) ->
            instrs_sub delt os is
          | _ -> raise (Failure "branching to missing or non-code")
        in
        begin match ru rm u with
          | WLoc (_, l) -> ((hm,rm,sm), hc [] l)
          | WApp (_, WLoc (_, l), os) -> ((hm,rm,sm), hc os l)
          | _ -> raise (Failure "branching to non-loc")
        end
      | _ -> raise (Failure "branching to on missing or non-int")
    end
  | ((hm,rm,sm), Ild (_, rd,rs,i)::is) ->
    begin match List.Assoc.find_exn rm rs with
      | WLoc (_, l) ->
        begin match List.Assoc.find hm l with
          | Some (_, HTuple ws) when List.length ws > i ->
            ((hm, replace rm rd (List.nth_exn ws i), sm), is)
          | Some (_, HTuple _) -> raise (Failure "ld: tuple index out of bounds")
          | _ -> raise (Failure "ld: trying to load from missing or non-tuple")
        end
      | _ -> raise (Failure "ld: trying to load from non-location")
    end
  | ((hm,rm,sm), Ist (_, rd,i,rs)::is) ->
    begin match List.Assoc.find rm rd with
      | Some (WLoc (_, l)) ->
        begin match List.Assoc.find hm l with
          | Some (Ref, HTuple ws) when List.length ws > i ->
            (((replace hm l (Ref, HTuple (list_replace i ws (List.Assoc.find_exn rm rs)))),
          | Some (Box, HTuple ws) ->
            raise (Failure "st: can't write to immutable tuple")
          | Some (_, HTuple _) -> raise (Failure "st: tuple index out of bounds")
          | _ -> raise (Failure "st: trying to store to missing or non-tuple")
        end
      | _ -> raise (Failure "st: trying to store to missing or non-location")
    end
  | ((hm,rm,sm), Iralloc (l',rd,n)::is) when List.length sm >= n ->
    let l = gen_sym () in (((l, (Ref, HTuple (List.take sm n))) :: hm, replace rm rd (WLoc (
  | ((hm,rm,sm), Iballoc (l',rd,n)::is) when List.length sm >= n ->
    let l = gen_sym () in (((l, (Box, HTuple (List.take sm n))) :: hm, replace rm rd (WLoc (
  | ((hm,rm,sm), Imv (_,rd,u)::is) ->
    ((hm, replace rm rd (ru rm u), sm), is)
```

Our approach: Typechecker

Translate typing judgments to OCaml.



```
let rec tc (context:context) e = match e with
  | FC exp -> begin
      let tc' e = tc context (FC e) in
      let open F in
      match exp, get_ret context with
      | EVar (l,i), TAL.QOut ->
        begin match List.Assoc.find (get_env context) i with
          | Some v -> (FT v, get_stack context)
          | None -> raise (TypeError ("Variable '" ^ i ^ "' not in scope.", l))
        end
      | EUnit l, TAL.QOut  -> (FT TUnit, get_stack context)
      | EInt _, TAL.QOut -> (FT TInt, get_stack context)
      | EBinop (l,e1,o,e2), TAL.QOut ->
        begin match tc' e1 with
          | (FT TInt, s1) ->
            begin match tc (set_stack context s1) (FC e2) with
              | (FT TInt, s2) -> (FT TInt, s2)
              | (FT t, _) -> raise (TypeError (Pretty.F.show_binop o ^ ": second argument has type " ^
              | _ -> raise (TypeError (Pretty.F.show_binop o ^ ": Uh-oh, got something I didn't underst
            end
          | (FT t, _) -> raise (TypeError (Pretty.F.show_binop o ^ ": first argument has type " ^ Prett
          | _ -> raise (TypeError (Pretty.F.show_binop o ^ ": Uh-oh, got something I didn't understand.
        end
      | EIf0 (l,c,e1,e2), TAL.QOut ->
        begin match tc' c with
          | FT TInt, s1 ->
            begin match tc (set_stack context s1) (FC e1) with
              | FT t1, s2 ->
                begin match tc (set_stack context s2) (FC e2) with
                  | FT t2, s3 -> if F'.t_eq t1 t2 && TAL'.s_eq s2 s3 then (FT t1, s2) else
                      raise (TypeError ("if: then branch has type " ^ Pretty.F.show t1 ^ " but else bra
                  | _ -> raise (TypeError ("if: Uh-oh, got something I didn't understand.", l))
                end
              | _ -> raise (TypeError ("if: Uh-oh, got something I didn't understand.", l))
            end
          | _ -> raise (TypeError ("if: test not an int.", l))
        end
      | ELam (l,ps,b), TAL.QOut ->
        let zeta = TAL.SAbstract ([], gen_sym ~pref:"z" ()) in
        begin match tc (set_stack (set_env context (List.append ps (get_env context))) zeta)
                  (FC b) with
         | (FT t, zeta') when zeta = zeta' -> (FT (TArrow (List.map ~f:snd ps, t)), get_stack context)
         | (FT _, _) -> raise (TypeError ("lam: body does not preserve stack.", l))
         | _ -> raise (TypeError ("lam: Uh-oh, got something I didn't understand.", l))
        end
      | ELamMod (l,ps,sin,sout,b), TAL.QOut ->
        let z = gen_sym ~pref:"z" () in
        let zeta = TAL.SAbstract (sin, z) in
        let zeta_out = TAL.SAbstract (sout, z) in
        begin match tc (set_stack (set_env context (List.append ps (get_env context))) zeta)
                  (FC b) with
         | (FT t, zeta') when zeta_out = zeta' -> (FT (TArrowMod (List.map ~f:snd ps, sin, sout, t)), ge
         | (FT _, zeta') -> raise (TypeError ("lam: body manipulates stack in invalid way. Expected " ^
         | _ -> raise (TypeError ("lam: Uh-oh, got something I didn't understand.", l))
        end
```

Made syntax directed with annotations & local inference.

Our approach: Parser / Printer

- Use `Menhir` to write grammar, with custom error messages for parse failures. These work really well!

Our approach: Parser / Printer

- Use `Menhir` to write grammar, with custom error messages for parse failures. These work really well!
- `PPrint` for pretty printer. Low effort for quite good printing!

Our approach: Web

- One html page with CodeMirror editor.

Our approach: Web

- One html page with CodeMirror editor.
- `js_of_ocaml` for the UI and to compile parser, pretty printer, interpreter, typechecker to Javascript.

Our approach: Web

- One html page with CodeMirror editor.
- `js_of_ocaml` for the UI and to compile parser, pretty printer, interpreter, typechecker to Javascript.
- 42 lines of hand-written javascript, for syntax highlighting (9 lines) and type error highlighting.

Demo

Take our work!

A member of our lab re-used the code for a gradual typing paper to appear in ICFP17.

Take our work!

A member of our lab re-used the code for a gradual typing paper to appear in ICFP17.

- Made artifact in a week (*may* have decided to create after acceptance).

Take our work!

A member of our lab re-used the code for a gradual typing paper to appear in ICFP17.

- Made artifact in a week (*may* have decided to create after acceptance).
- Was able to re-use overall architecture and most of the web frontend.

Take our work!

A member of our lab re-used the code for a gradual typing paper to appear in ICFP17.

- Made artifact in a week (*may* have decided to create after acceptance).
- Was able to re-use overall architecture and most of the web frontend.
- Good feedback — other researchers excited to play around with examples.

Questions?

https://dbp.io/artifacts/funtal

https://github.com/dbp/funtal

https://dbp.io/pubs/2017/funtal.pdf